

# iOS 开发者

## 专业人士笔记

**Chapter 4: attributedText in UILabel**

The current styled text that is displayed by the label.

You can add HTML text in **UILabel** using **attributedText** property or customized single [UIAttributedString](#).

### Section 4.1: HTML text in UILabel

```
NSString *htmlString = @"<b>Hello</b><br> Example bold text in HTML </p> <br> the
NSAttributedTextEditing & error = [[NSAttributedTextEditing alloc] initWithData:[NSData
dataUsingEncoding:NSUTF8StringEncoding] options:@{} NSDocumentType:UITextViewType];
NSHTMLTextDocumentType ] message:[attributedTextes.mutableAttributedString];
attributedTextes.mutableAttributedString = error];
```

UILabel \*yourLabel = [UILabel alloc] init];
yourLabel.attributedText = attributedText;

### Section 4.2: Set different property to text in NSAttributedString

The first step you need to perform is to create a **NSMutableAttributedString** instead of **NSAttributedString** because it enables us **NSMutableAttributedStringEditing** instead of **NSMutableAttributedString**.

```
NSMutableAttributedString *attributedText;
attributedText = [[NSMutableAttributedString alloc]
initWithString:@"Hello World"];
NSHTMLTextDocumentType editing = [[NSHTMLTextDocumentType alloc]
initWithString:@"<b>Hello</b><br> Example bold text in HTML </p> <br> the
NSAttributedTextEditing & error = [[NSAttributedTextEditing alloc]
initWithData:[NSData
dataUsingEncoding:NSUTF8StringEncoding] options:@{} NSDocumentType:UITextViewType];
NSHTMLTextDocumentType ] message:[attributedTextes.mutableAttributedString];
attributedTextes.mutableAttributedString = error];
```

// Finding the range of text.
NSRange rangeHello = [attributedText rangeOfString:@"Hello"];
NSRange rangeWorld = [attributedText rangeOfString:@"World"];

```
// Add font style for Hello
[attributedText addAttribute:NSFontAttributeName
value:[UIFont fontWithName:@"Copperplate" size:14]
range:rangeHello];
// Add text color for Hello
[attributedText addAttribute:NSTextColorAttributeName
value:[UIColor blueColor]];
range:rangeHello];
```

// Add font style for World
[attributedText addAttribute:NSFontAttributeName
value:[UIFont fontWithName:@"Chalkboard" size:12]
range:rangeWorld];
// Add text color for world
[attributedText addAttribute:NSTextColorAttributeName
value:[UIColor colorWithRed:(144/255.0) green:
(144/255.0) blue:(144/255.0) alpha:1]
range:rangeWorld];

// Set it to UILabel as an attributedText
UILabel \*yourLabel = [UILabel alloc] initWithFrame:CGRectMake(0, 0, 100, 100);
yourLabel.attributedText = attributedText;
[yourLabel setBounces:YES];
[yourLabel setClipsToBounds:YES];
[yourLabel setEditable:YES];
[yourLabel setSelectable:YES];

Output:

**Chapter 13: Cut a UIImage into a circle - Objective C**

### Section 13.1: Cut a image into a circle - Objective C

```
import UIKit/UIKit.h
```

The code in the viewDidLoad or loadView should look something like this

```
- (void)viewDidLoad {
    [super viewDidLoad];
    UIImage *image = [[UIImage alloc] initWithFrame:CGRectMake(0, 0, 320, 320)];
    self.view.addSubview(imageView);
    UIImage *image = [[UIImage imageNamed:@"Tourist-Image-Photos-Images-Travel-Images-Pictures-00000000.jpg"] circularScaledAndCropImage([UIImage imageNamed:@"Dolphins-Photos-Images-Travel-Tourist-Image-Photos-00000000.jpg"] frame:CGRectMake(0, 0, 320, 320));
}
```

Finally the function which does the heavy lifting circularScaledAndCropImage is as defined below

```
- (UIImage *)circularScaledAndCropImage:(UIImage *)image frame:(CGRect)frame {
    // This function returns a rectangle, based on image, that has been:
    // - scaled to fit in (CGRect) rect
    // - and cropped with a circle of radius: rectWidth/2

    //Create the bitmap graphics context
    UIGraphicsBeginImageContextWithOptions(frame.size.width, frame.size.height, NO);
    CGContextRef context = UIGraphicsGetCurrentContext();

    //Get the width and height
    CGFloat imageWidth = image.size.width;
    CGFloat imageHeight = image.size.height;
    CGFloat rectWidth = frame.size.width;
    CGFloat rectHeight = frame.size.height;

    //Calculate the scale factor
    CGFloat scaleFactorX = rectWidth/(imageWidth);
    CGFloat scaleFactorY = rectHeight/(imageHeight);

    //Calculate the centre of the circle
    CGAffineTransform rotateCenterX = rectWidth/2;
    CGAffineTransform rotateCenterY = rectHeight/2;

    // Create and CLIP to a CIRCULAR Path
    // (This could be replaced with any closed path if you want a different shaped clip)
    CGAffineTransform rotatePath = CGAffineTransformIdentity;
    rotatePath = CGAffineTransformTranslate(rotatePath, rotateCenterX, rotateCenterY);
    rotatePath = CGAffineTransformScale(rotatePath, 1, 1);
    CGPathRef ovalPath = CGPathCreateWithRect(context, &frame);
    CGPathCloseSubpath(ovalPath);
    CGPathAddClip(context, ovalPath, rotatePath);

    //Set the SCALE factor for the graphics context
    //All future draw calls will be scaled by this factor
    CGContextScaleCTM(context, scaleFactorX, scaleFactorY);

    // Draw the IMAGE
    CGContextDrawImage(context, CGRectMake(0, 0, imageWidth, imageHeight));
}
```

© 2012 Apple Inc. All rights reserved.

# Chapter 107: Push Notifications

The screenshot shows the 'AttributedText in UILabel' chapter from the book. It includes:

- Section 4.1: HTML text in UILabel**: A code snippet showing how to set attributedText on a UILabel with HTML-like styling.
- Section 4.2: Set different property to text in NSAttributedString**: A code snippet demonstrating various ways to set text properties using NSAttributedString.
- Output:** A preview of the resulting attributed text with different styles applied.

**er 13: Cut a UIImage into a circle**

### 1: Cut a image into a circle - Objective C

```

- (void) viewDidLoad {
    [super viewDidLoad];
    self.title = @"Cut a Image into a Circle";
}

- (void) loadView {
    [super loadView];
    self.view.backgroundColor = [UIColor lightGrayColor];
}

- (void) drawImage {
    UIImageView *imageView = [[UIImageView alloc] initWithFrame:CGRectMake(0, 50, 320, 320)];
    imageView.image = [UIImage imageNamed:@"Tourist-Photos-Images-Travel-Tourist-Images-100.jpg"];
    imageView.layer.cornerRadius = imageView.frame.size.width / 2;
    imageView.clipsToBounds = YES;
    [self.view addSubview:imageView];
}

- (void) drawCircle {
    CGContextRef context = UIGraphicsGetCurrentContext();
    CGContextSetAllowsAntialiasing(context, YES);
    CGContextSetShouldAntialias(context, YES);

    CGContextSetFillColorWithColor(context, [UIColor redColor].CGColor);
    CGContextFillRect(context, CGRectMake(160, 160, 160, 160));
}

- (void) drawImageWithCircle {
    CGContextRef context = UIGraphicsGetCurrentContext();
    CGContextSetAllowsAntialiasing(context, YES);
    CGContextSetShouldAntialias(context, YES);

    CGContextSetFillColorWithColor(context, [UIColor redColor].CGColor);
    CGContextFillRect(context, CGRectMake(160, 160, 160, 160));

    CGImageRef image = imageView.image.CGImage;
    CGContextDrawImage(context, CGRectMake(0, 0, 320, 320), image);
}

- (void) drawImageWithClippedCircle {
    CGContextRef context = UIGraphicsGetCurrentContext();
    CGContextSetAllowsAntialiasing(context, YES);
    CGContextSetShouldAntialias(context, YES);

    CGContextSetFillColorWithColor(context, [UIColor redColor].CGColor);
    CGContextFillRect(context, CGRectMake(160, 160, 160, 160));

    CGImageRef image = imageView.image.CGImage;
    CGContextDrawImage(context, CGRectMake(0, 0, 320, 320), image);
    CGContextClipToMask(context, CGRectMake(160, 160, 160, 160), image);
}

- (void) drawImageWithClippedImage {
    CGContextRef context = UIGraphicsGetCurrentContext();
    CGContextSetAllowsAntialiasing(context, YES);
    CGContextSetShouldAntialias(context, YES);

    CGContextSetFillColorWithColor(context, [UIColor redColor].CGColor);
    CGContextFillRect(context, CGRectMake(160, 160, 160, 160));

    CGImageRef image = imageView.image.CGImage;
    CGContextDrawImage(context, CGRectMake(0, 0, 320, 320), image);
    CGContextClipToMask(context, CGRectMake(160, 160, 160, 160), image);
}

```

**800+ 页**  
专业提示和技巧

**免责声明**  
这是一本非官方的免费书籍，旨在教育用途，  
与官方iOS®开发者团体或公司无关。  
所有商标和注册商标均为其各自所有者所有。

# 目录

<a href="#">关于</a>	1
<b>第1章：iOS开发入门</b>	2
<a href="#">第1.1节：创建默认的单视图应用程序</a>	2
<a href="#">第1.2节：你好，世界</a>	6
<a href="#">第1.3节：Xcode界面</a>	11
<a href="#">第1.4节：创建你的第一个Swift 3程序</a>	17
<b>第2章：UILabel</b>	22
<a href="#">第2.1节：创建UILabel</a>	22
<a href="#">第2.2节：行数</a>	24
<a href="#">第2.3节：设置字体</a>	25
<a href="#">第2.4节：文本颜色</a>	26
<a href="#">第2.5节：背景颜色</a>	27
<a href="#">第2.6节：适应大小</a>	27
<a href="#">第2.7节：文本对齐</a>	30
<a href="#">第2.8节：计算内容边界（例如动态单元格高度）</a>	30
<a href="#">第2.9节：带标签的文本</a>	32
<a href="#">第2.10节：可点击标签</a>	38
<a href="#">第2.11节：使用约束的可变高度</a>	39
<a href="#">第2.12节：换行模式</a>	39
<a href="#">第2.13节：为文本添加阴影</a>	41
<a href="#">第2.14节：更改现有标签中的文本</a>	41
<a href="#">第2.15节：自动调整标签大小以适应文本</a>	42
<a href="#">第2.16节：严格根据文本和字体获取UILabel的大小</a>	43
<a href="#">第2.17节：高亮和高亮文本颜色</a>	44
<a href="#">第2.18节：文本两端对齐</a>	44
<a href="#">第2.19节：来自未知文本长度的动态标签框架</a>	45
<b>第3章：UILabel文本下划线</b>	47
<a href="#">第3.1节：使用Objective C在UILabel中添加下划线文本</a>	47
<a href="#">第3.2节：使用Swift在UILabel中添加下划线文本</a>	47
<b>第4章：UILabel中的attributedText</b>	48
<a href="#">第4.1节：UILabel中的HTML文本</a>	48
<a href="#">第4.2节：在单个UILabel中为文本设置不同属性</a>	48
<b>第5章：UIButton</b>	50
<a href="#">第5.1节：创建UIButton</a>	50
<a href="#">第5.2节：为按钮绑定方法</a>	50
<a href="#">第5.3节：设置字体</a>	51
<a href="#">第5.4节：设置图像</a>	51
<a href="#">第5.5节：严格根据文本和字体获取UIButton的大小</a>	51
<a href="#">第5.6节：禁用UIButton</a>	52
<a href="#">第5.7节：设置标题</a>	52
<a href="#">第5.8节：设置标题颜色</a>	52
<a href="#">第5.9节：水平对齐内容</a>	53
<a href="#">第5.10节：获取标题标签</a>	53
<a href="#">第5.11节：通过代码（编程方式）为UIButton添加动作</a>	54
<b>第6章：UIDatePicker</b>	55
<a href="#">第6.1节：创建日期选择器</a>	55
<a href="#">第6.2节：设置最小-最大日期</a>	55

# Contents

<a href="#">About</a>	1
<b>Chapter 1: Getting started with iOS Development</b>	2
<a href="#">Section 1.1: Creating a default Single View Application</a>	2
<a href="#">Section 1.2: Hello World</a>	6
<a href="#">Section 1.3: Xcode Interface</a>	11
<a href="#">Section 1.4: Create your first program in Swift 3</a>	17
<b>Chapter 2: UILabel</b>	22
<a href="#">Section 2.1: Create a UILabel</a>	22
<a href="#">Section 2.2: Number of Lines</a>	24
<a href="#">Section 2.3: Set Font</a>	25
<a href="#">Section 2.4: Text Color</a>	26
<a href="#">Section 2.5: Background Color</a>	27
<a href="#">Section 2.6: Size to fit</a>	27
<a href="#">Section 2.7: Text alignment</a>	30
<a href="#">Section 2.8: Calculate Content Bounds (for i.e. dynamic cell heights)</a>	30
<a href="#">Section 2.9: Label Attributed Text</a>	32
<a href="#">Section 2.10: Clickable Label</a>	38
<a href="#">Section 2.11: Variable height using constraints</a>	39
<a href="#">Section 2.12: LineBreakMode</a>	39
<a href="#">Section 2.13: Add shadows to text</a>	41
<a href="#">Section 2.14: Changing Text in an Existing Label</a>	41
<a href="#">Section 2.15: Auto-size label to fit text</a>	42
<a href="#">Section 2.16: Get UILabel's size strictly based on its text and font</a>	43
<a href="#">Section 2.17: Highlighted and Highlighted Text Color</a>	44
<a href="#">Section 2.18: Justify Text</a>	44
<a href="#">Section 2.19: Dynamic label frame from unknown text length</a>	45
<b>Chapter 3: UILabel text underlining</b>	47
<a href="#">Section 3.1: Underlining a text in a UILabel using Objective C</a>	47
<a href="#">Section 3.2: Underlining a text in UILabel using Swift</a>	47
<b>Chapter 4: attributedText in UILabel</b>	48
<a href="#">Section 4.1: HTML text in UILabel</a>	48
<a href="#">Section 4.2: Set different property to text in single UILabel</a>	48
<b>Chapter 5: UIButton</b>	50
<a href="#">Section 5.1: Creating a UIButton</a>	50
<a href="#">Section 5.2: Attaching a Method to a Button</a>	50
<a href="#">Section 5.3: Setting Font</a>	51
<a href="#">Section 5.4: Set Image</a>	51
<a href="#">Section 5.5: Get UIButton's size strictly based on its text and font</a>	51
<a href="#">Section 5.6: Disabling a UIButton</a>	52
<a href="#">Section 5.7: Set title</a>	52
<a href="#">Section 5.8: Set title color</a>	52
<a href="#">Section 5.9: Horizontally aligning contents</a>	53
<a href="#">Section 5.10: Getting the title label</a>	53
<a href="#">Section 5.11: Adding an action to an UIButton via Code (programmatically)</a>	54
<b>Chapter 6: UIDatePicker</b>	55
<a href="#">Section 6.1: Create a Date Picker</a>	55
<a href="#">Section 6.2: Setting Minimum-Maximum Date</a>	55

第6.3节：模式	55
第6.4节：设置分钟间隔	55
第6.5节：倒计时持续时间	56
<b>第7章：UILocalNotification</b>	57
第7.1节：安排本地通知	57
第7.2节：立即显示本地通知	57
第7.3节：使用UUID管理本地通知	58
第7.4节：注册本地通知	59
第7.5节：iOS10中UILocalNotification的新特性	60
第7.6节：响应接收到的本地通知	62
第7.7节：在Swift 3.0 (iOS 10) 中注册和安排本地通知	62
<b>第8章：UIImage</b>	64
第8.1节：创建UIImage	64
第8.2节：比较图像	65
第8.3节：带颜色的渐变图像	66
第8.4节：UIImage与Base64编码的转换	66
第8.5节：对UIView进行快照	67
第8.6节：更改UIImage颜色	67
第8.7节：将UIColor应用于UIImage	67
第8.8节：使用文件内容创建和初始化图像对象	68
第8.9节：带边缘可调整大小的图像	68
第8.10节：用于边界的渐变背景层	69
<b>第9章：将NSAttributedString转换为UIImage</b>	70
第9.1节：NSAttributedString到UIImage的转换	70
<b>第10章：UIImagePickerController</b>	71
第10.1节：UIImagePickerController的通用用法	71
<b>第11章：UIImageView</b>	73
第11.1节：带标签的UIImage遮罩	73
第11.2节：将图像制作成圆形或圆角	73
第11.3节：模式属性如何影响图像	74
第11.4节：为UIImageView制作动画	80
第11.5节：创建UIImageView	81
第11.6节：更改图像颜色	82
第11.7节：为UIImageView分配图像	82
<b>第12章：调整UIImage大小</b>	83
第12.1节：按尺寸和质量调整任意图像大小	83
<b>第13章：将UIImage裁剪成圆形</b>	84
第13.1节：将图像裁剪成圆形 - Objective C	84
第13.2节：SWIFT 3示例	85
<b>第14章：UITableView</b>	87
第14.1节：自适应单元格	87
第14.2节：自定义单元格	87
第14.3节：分隔线	90
第14.4节：代理和数据源	92
第14.5节：创建UITableView	98
第14.6节：滑动删除行	102
第14.7节：展开与折叠UITableView单元格	105
<b>第15章：UITableViewController</b>	108
第15.1节：具有动态属性且使用tableviewCellStyle_basic的表视图	108

Section 6.3: Modes	55
Section 6.4: Setting minute interval	55
Section 6.5: Count Down Duration	56
<b>Chapter 7: UILocalNotification</b>	57
Section 7.1: Scheduling a local notification	57
Section 7.2: Presenting a local notification immediately	57
Section 7.3: Managing local notifications using UUID	58
Section 7.4: Registering for local notifications	59
Section 7.5: what's new in UILocalNotification with iOS10	60
Section 7.6: Responding to received local notification	62
Section 7.7: Register and Schedule Local Notification in Swift 3.0 (iOS 10)	62
<b>Chapter 8: UIImage</b>	64
Section 8.1: Creating UIImage	64
Section 8.2: Comparing Images	65
Section 8.3: Gradient Image with Colors	66
Section 8.4: Convert UIImage to/from base64 encoding	66
Section 8.5: Take a Snapshot of a UIView	67
Section 8.6: Change UIImage Color	67
Section 8.7: Apply UIColor to UIImage	67
Section 8.8: Creating and Initializing Image Objects with file contents	68
Section 8.9: Resizable image with caps	68
Section 8.10: Gradient Background Layer for Bounds	69
<b>Chapter 9: Convert NSAttributedString to UIImage</b>	70
Section 9.1: NSAttributedString to UIImage Conversion	70
<b>Chapter 10: UIImagePickerController</b>	71
Section 10.1: Generic usage of UIImagePickerController	71
<b>Chapter 11: UIImageView</b>	73
Section 11.1: UIImage masked with Label	73
Section 11.2: Making an image into a circle or rounded	73
Section 11.3: How the Mode property affects an image	74
Section 11.4: Animating a UIImageView	80
Section 11.5: Create a UIImageView	81
Section 11.6: Change color of an image	82
Section 11.7: Assigning an image to a UIImageView	82
<b>Chapter 12: Resizing UIImage</b>	83
Section 12.1: Resize any image by size & quality	83
<b>Chapter 13: Cut a UIImage into a circle</b>	84
Section 13.1: Cut a image into a circle - Objective C	84
Section 13.2: SWIFT 3 Example	85
<b>Chapter 14: UITableView</b>	87
Section 14.1: Self-Sizing Cells	87
Section 14.2: Custom Cells	87
Section 14.3: Separator Lines	90
Section 14.4: Delegate and Datasource	92
Section 14.5: Creating a UITableView	98
Section 14.6: Swipe to Delete Rows	102
Section 14.7: Expanding & Collapsing UITableViewCells	105
<b>Chapter 15: UITableViewController</b>	108
Section 15.1: TableView with dynamic properties with tableviewCellStyle_basic	108

<a href="#">第15.2节：自定义单元格的表视图</a>	109	<a href="#">Section 15.2: TableView with Custom Cell</a>	109
<b>第16章：UIRefreshControl 表视图</b>	111	<b>Chapter 16: UIRefreshControl TableView</b>	111
<a href="#">第16.1节：在表视图上设置refreshControl：</a>	111	<a href="#">Section 16.1: Set up refreshControl on tableView:</a>	111
<a href="#">第16.2节：Objective-C 示例</a>	111	<a href="#">Section 16.2: Objective-C Example</a>	111
<b>第17章：UITableViewCell</b>	113	<b>Chapter 17: UITableViewCell</b>	113
<a href="#">第17.1节：UITableViewCell 的Xib文件</a>	113	<a href="#">Section 17.1: Xib file of UITableViewCell</a>	113
<b>第18章：UITableViewCell 选择的自定义方法</b>	114	<b>Chapter 18: Custom methods of selection of UITableViewCells</b>	114
<a href="#">第18.1节：行的单选与双选的区别</a>	114	<a href="#">Section 18.1: Distinction between single and double selection on row</a>	114
<b>第19章：UITableViewCell的自定义选择方法</b>	115	<b>Chapter 19: Custom methods of selection of UITableViewCells</b>	115
<a href="#">第19.1节：行的单选与双选的区别</a>	115	<a href="#">Section 19.1: Distinction between single and double selection on row</a>	115
<b>第20章：UIView</b>	116	<b>Chapter 20: UIView</b>	116
<a href="#">第20.1节：使视图圆角化</a>	116	<a href="#">Section 20.1: Make the view rounded</a>	116
<a href="#">第20.2节：使用IBInspectable和IBDesignable</a>	118	<a href="#">Section 20.2: Using IBInspectable and IBDesignable</a>	118
<a href="#">第20.3节：拍摄快照</a>	121	<a href="#">Section 20.3: Taking a snapshot</a>	121
<a href="#">第20.4节：创建UIView</a>	121	<a href="#">Section 20.4: Create a UIView</a>	121
<a href="#">第20.5节：使视图抖动</a>	121	<a href="#">Section 20.5: Shake a View</a>	121
<a href="#">第20.6节：利用内在内容尺寸</a>	122	<a href="#">Section 20.6: Utilizing Intrinsic Content Size</a>	122
<a href="#">第20.7节：通过编程管理UIView插入和从另一个UIView中删除</a>	124	<a href="#">Section 20.7: Programmatically manage UIView insertion and deletion into and from another UIView</a>	124
<a href="#">第20.8节：使用自动布局创建UIView</a>	125	<a href="#">Section 20.8: Create UIView using Autolayout</a>	125
<a href="#">第20.9节：UIView的动画</a>	127	<a href="#">Section 20.9: Animating a UIView</a>	127
<a href="#">第20.10节：UIView的尺寸和框架属性扩展</a>	127	<a href="#">Section 20.10: UIView extension for size and frame attributes</a>	127
<b>第21章：UIView的快照</b>	129	<b>Chapter 21: Snapshot of UIView</b>	129
<a href="#">第21.1节：获取快照</a>	129	<a href="#">Section 21.1: Getting the Snapshot</a>	129
<a href="#">第21.2节：包含其他标记和文本的子视图快照</a>	129	<a href="#">Section 21.2: Snapshot with subview with other markup and text</a>	129
<b>第22章：UIAlertController</b>	131	<b>Chapter 22: UIAlertController</b>	131
<a href="#">第22.1节：使用UIAlertController的警告视图</a>	131	<a href="#">Section 22.1: AlertViews with UIAlertController</a>	131
<a href="#">第22.2节：使用UIAlertController的操作表</a>	132	<a href="#">Section 22.2: Action Sheets with UIAlertController</a>	132
<a href="#">第22.3节：在UIAlertController中添加文本字段，类似提示框</a>	135	<a href="#">Section 22.3: Adding Text Field in UIAlertController like a prompt Box</a>	135
<a href="#">第22.4节：高亮显示操作按钮</a>	135	<a href="#">Section 22.4: Highlighting an action button</a>	135
<a href="#">第22.5节：显示和处理警告</a>	136	<a href="#">Section 22.5: Displaying and handling alerts</a>	136
<b>第23章：UIColor</b>	141	<b>Chapter 23: UIColor</b>	141
<a href="#">第23.1节：创建UIColor</a>	141	<a href="#">Section 23.1: Creating a UIColor</a>	141
<a href="#">第23.2节：从十六进制数字或字符串创建UIColor</a>	142	<a href="#">Section 23.2: Creating a UIColor from hexadecimal number or string</a>	142
<a href="#">第23.3节：带Alpha组件的颜色</a>	144	<a href="#">Section 23.3: Color with Alpha component</a>	144
<a href="#">第23.4节：未公开的方法</a>	145	<a href="#">Section 23.4: Undocumented Methods</a>	145
<a href="#">第23.5节：从图像模式创建UIColor</a>	146	<a href="#">Section 23.5: UIColor from an image pattern</a>	146
<a href="#">第23.6节：给定UIColor的较浅和较深色调</a>	147	<a href="#">Section 23.6: Lighter and Darker Shade of a given UIColor</a>	147
<a href="#">第23.7节：使用用户定义属性应用CGColor数据类型</a>	148	<a href="#">Section 23.7: Make user defined attributes apply the CGColor datatype</a>	148
<b>第24章：UITextView</b>	149	<b>Chapter 24: UITextView</b>	149
<a href="#">第24.1节：设置富文本</a>	149	<a href="#">Section 24.1: Set attributed text</a>	149
<a href="#">第24.2节：更改字体</a>	149	<a href="#">Section 24.2: Change font</a>	149
<a href="#">第24.3节：自动检测链接、地址、日期等</a>	149	<a href="#">Section 24.3: Auto Detect Links, Addresses, Dates, and more</a>	149
<a href="#">第24.4节：更改文本</a>	150	<a href="#">Section 24.4: Change text</a>	150
<a href="#">第24.5节：更改文本对齐方式</a>	150	<a href="#">Section 24.5: Change text alignment</a>	150
<a href="#">第24.6节：UITextViewDelegate 方法</a>	150	<a href="#">Section 24.6: UITextViewDelegate methods</a>	150
<a href="#">第24.7节：更改文本颜色</a>	151	<a href="#">Section 24.7: Change text color</a>	151
<a href="#">第24.8节：移除多余的填充以适应精确测量的文本</a>	151	<a href="#">Section 24.8: Remove extra paddings to fit to a precisely measured text</a>	151
<a href="#">第24.9节：获取和设置光标位置</a>	151	<a href="#">Section 24.9: Getting and Setting the Cursor Position</a>	151

第24.10节：带有HTML文本的UITextView	153
第24.11节：检查是否为空或nil	153
<b>第25章：UITextField代理</b>	154
第25.1节：用户开始/结束与文本字段交互时的操作	154
第25.2节：UITextField - 限制文本字段输入特定字符	155
<b>第26章：UINavigationController</b>	156
第26.1节：通过编程方式将视图控制器嵌入导航控制器	156
第26.2节：导航控制器中的弹出操作	156
第26.3节：目的	156
第26.4节：将视图控制器推入导航栈	157
第26.5节：创建导航控制器	157
<b>第27章：UIGestureRecognizer</b>	158
第27.1节：UITapGestureRecognizer	158
第27.2节：UITapGestureRecognizer（双击）	159
第27.3节：在界面构建器中添加手势识别器	159
第27.4节：UILongPressGestureRecognizer	160
第27.5节：UISwipeGestureRecognizer	161
第27.6节：UIPinchGestureRecognizer	162
第27.7节：UIRotationGestureRecognizer	163
<b>第28章：UIBarButtonItem</b>	164
第28.1节：在界面构建器中创建UIBarButtonItem	164
第28.2节：创建UIBarButtonItem	167
第28.3节：无色调的导航栏按钮原始图像	167
<b>第29章：UIScrollView</b>	168
第29.1节：启用自动布局的滚动内容	168
第29.2节：创建UIScrollView	171
第29.3节：带自动布局的ScrollView	171
第29.4节：通过代理方法检测UIScrollView滚动结束	176
第29.5节：启用/禁用滚动	176
第29.6节：UIImageView缩放	177
第29.7节：滚动视图内容大小	178
第29.8节：限制滚动方向	178
<b>第30章：UIStackView</b>	179
第30.1节：使用UIStackView居中按钮	179
第30.2节：通过编程方式创建水平堆叠视图	183
第30.3节：通过编程创建垂直堆叠视图	184
<b>第31章：动态更新UIStackView</b>	185
第31.1节：将UISwitch连接到一个动作，我们可以通过该动作在水平或垂直布局的图像视图之间切换并进行动画	
图像视图的布局	185
<b>第32章：带有StackView子视图的UIScrollView</b>	186
第32.1节：ScrollView内的复杂StackView示例	186
第32.2节：防止布局歧义	187
第32.3节：滚动到嵌套StackView内的内容	188
<b>第33章：UIScrollView自动布局</b>	189
第33.1节：可滚动控制器	189
第33.2节：通过Storyboard设置UIScrollView的动态内容大小	193
<b>第34章：UITextField</b>	195
第34.1节：获取键盘焦点并隐藏键盘	195
第34.2节：用户按下回车键时收起键盘	195

Section 24.10: UITextView with HTML text	153
Section 24.11: Check to see if empty or nil	153
<b>Chapter 25: UITextField Delegate</b>	154
Section 25.1: Actions when a user has started/ended interacting with a textfield	154
Section 25.2: UITextField - Restrict textfield to certain characters	155
<b>Chapter 26: UINavigationController</b>	156
Section 26.1: Embed a view controller in a navigation controller programmatically	156
Section 26.2: Popping in a Navigation Controller	156
Section 26.3: Purpose	156
Section 26.4: Pushing a view controller onto the navigation stack	157
Section 26.5: Creating a NavController	157
<b>Chapter 27: UIGestureRecognizer</b>	158
Section 27.1: UITapGestureRecognizer	158
Section 27.2: UITapGestureRecognizer (Double Tap)	159
Section 27.3: Adding a Gesture recognizer in the Interface Builder	159
Section 27.4: UILongPressGestureRecognizer	160
Section 27.5: UISwipeGestureRecognizer	161
Section 27.6: UIPinchGestureRecognizer	162
Section 27.7: UIRotationGestureRecognizer	163
<b>Chapter 28: UIBarButtonItem</b>	164
Section 28.1: Creating a UIBarButtonItem in the Interface Builder	164
Section 28.2: Creating a UIBarButtonItem	167
Section 28.3: Bar Button Item Original Image with no Tint Color	167
<b>Chapter 29: UIScrollView</b>	168
Section 29.1: Scrolling content with Auto Layout enabled	168
Section 29.2: Create a UIScrollView	171
Section 29.3: ScrollView with AutoLayout	171
Section 29.4: Detecting when UIScrollView finished scrolling with delegate methods	176
Section 29.5: Enable/Disable Scrolling	176
Section 29.6: Zoom In/Out UIImageView	177
Section 29.7: Scroll View Content Size	178
Section 29.8: Restrict scrolling direction	178
<b>Chapter 30: UIStackView</b>	179
Section 30.1: Center Buttons with UIStackview	179
Section 30.2: Create a horizontal stack view programmatically	183
Section 30.3: Create a vertical stack view programmatically	184
<b>Chapter 31: Dynamically updating a UIStackView</b>	185
Section 31.1: Connect the UISwitch to an action we can animate switching between a horizontal or vertical layout of the image views	185
<b>Chapter 32: UIScrollView with StackView child</b>	186
Section 32.1: A complex StackView inside Scrollview Example	186
Section 32.2: Preventing ambiguous layout	187
Section 32.3: Scrolling to content inside nested StackViews	188
<b>Chapter 33: UIScrollView AutoLayout</b>	189
Section 33.1: ScrollableController	189
Section 33.2: UIScrollView dynamic content size through Storyboard	193
<b>Chapter 34: UITextField</b>	195
Section 34.1: Get Keyboard Focus and Hide Keyboard	195
Section 34.2: Dismiss keyboard when user pushes the return button	195

第34.3节：隐藏闪烁的插入符号	196	Section 34.3: Hide blinking caret	196
第34.4节：输入辅助视图（工具栏）	196	Section 34.4: Input accessory view (toolbar)	196
第34.5节：UITextView成为第一响应者时滚动视图移动	197	Section 34.5: Moving scroll when UITextView becomes first responder	197
第34.6节：键盘类型	199	Section 34.6: KeyboardType	199
第34.7节：更改占位符颜色和字体	199	Section 34.7: Change placeholder color and font	199
第34.8节：用UIPickerView替换键盘	200	Section 34.8: Replace keyboard with UIPickerView	200
第34.9节：创建UITextField	203	Section 34.9: Create a UITextField	203
第34.10节：获取和设置光标位置	204	Section 34.10: Getting and Setting the Cursor Position	204
第34.11节：收起键盘	205	Section 34.11: Dismiss Keyboard	205
第34.12节：初始化文本字段	208	Section 34.12: Initialize text field	208
第34.13节：自动大写	208	Section 34.13: Autocapitalization	208
第34.14节：设置对齐方式	209	Section 34.14: Set Alignment	209
<b>第35章：自定义UITextField</b>	210	<b>Chapter 35: Custom UITextField</b>	210
第35.1节：用于过滤输入文本的自定义UITextField	210	Section 35.1: Custom UITextField for Filtering Input Text	210
第35.2节：禁止所有操作（如复制、粘贴等）的自定义UITextField	210	Section 35.2: Custom UITextField to Disallow All Actions like Copy, Paste, etc	210
<b>第36章：UIViewController</b>	212	<b>Chapter 36: UIViewController</b>	212
第36.1节：子类化	212	Section 36.1: Subclassing	212
第36.2节：访问容器视图控制器	214	Section 36.2: Access the container view controller	214
第36.3节：以编程方式设置视图	214	Section 36.3: Set the view programmatically	214
第36.4节：从Storyboard实例化	214	Section 36.4: Instantiate from a Storyboard	214
第36.5节：创建一个实例	215	Section 36.5: Create an instance	215
第36.6节：添加/移除子视图控制器	216	Section 36.6: Adding/removing a child view controller	216
<b>第37章：UISwitch</b>	217	<b>Chapter 37: UISwitch</b>	217
第37.1节：设置开启/关闭状态的图片	217	Section 37.1: Set Image for On/Off state	217
第37.2节：设置开启/关闭	217	Section 37.2: Set On / Off	217
第37.3节：设置背景颜色	217	Section 37.3: Set Background Color	217
第37.4节：设置色调颜色	218	Section 37.4: Set Tint Color	218
<b>第38章：UICollectionView</b>	219	<b>Chapter 38: UICollectionView</b>	219
第38.1节：创建UICollectionView	219	Section 38.1: Create a UICollectionView	219
第38.2节：UICollectionView - 数据源	219	Section 38.2: UICollectionView - Datasource	219
第38.3节：Collection View的基本Swift示例	220	Section 38.3: Basic Swift example of a Collection View	220
第38.4节：使用数据源和流布局管理多个Collection View	225	Section 38.4: Manage Multiple Collection view with DataSource and Flowlayout	225
第38.5节：UICollectionViewDelegate设置及项目选择	227	Section 38.5: UICollectionViewDelegate setup and item selection	227
第38.6节：以编程方式创建集合视图	228	Section 38.6: Create a Collection View Programmatically	228
第38.7节：Swift - UICollectionViewDelegateFlowLayout	229	Section 38.7: Swift - UICollectionViewDelegateFlowLayout	229
第38.8节：执行批量更新	229	Section 38.8: Performing batch updates	229
<b>第39章：UISearchController</b>	231	<b>Chapter 39: UISearchController</b>	231
第39.1节：导航栏标题中的搜索栏	231	Section 39.1: Search Bar in Navigation Bar Title	231
第39.2节：表视图头部的搜索栏	234	Section 39.2: Search Bar in Table View Header	234
第39.3节：实现	237	Section 39.3: Implementation	237
第39.4节：Objective-C中的UISerachController	237	Section 39.4: UISerachController in Objective-C	237
<b>第40章：UITabBarController</b>	239	<b>Chapter 40: UITabBarController</b>	239
第40.1节：创建实例	239	Section 40.1: Create an instance	239
第40.2节：带有标签栏的导航控制器	239	Section 40.2: Navigation Controller with TabBar	239
第40.3节：标签栏颜色自定义	240	Section 40.3: Tab Bar color customizing	240
第40.4节：更改标签栏项目标题和图标	240	Section 40.4: Changing Tab Bar Item Title and Icon	240
第40.5节：通过代码创建标签栏控制器，无需Storyboard	241	Section 40.5: Create Tab Bar controller programmatically without Storyboard	241
第40.6节：带自定义颜色选择的UITabBarController	242	Section 40.6: UITabBarController with custom color selection	242
<b>第41章：UIWebView</b>	245	<b>Chapter 41: UIWebView</b>	245

<a href="#">第41章：UIWebView实例</a>	245
<a href="#">第41.2节：确定内容大小</a>	245
<a href="#">第41.3节：加载HTML字符串</a>	245
<a href="#">第41.4节：发起URL请求</a>	246
<a href="#">第41.5节：加载JavaScript</a>	246
<a href="#">第41.6节：停止加载网页内容</a>	247
<a href="#">第41.7节：重新加载当前网页内容</a>	247
<a href="#">第41.8节：加载.pdf、.txt、.doc等文档文件</a>	247
<a href="#">第41.9节：在webView中加载本地HTML文件</a>	247
<a href="#">第41.10节：使UIWebView内的链接可点击</a>	248
<a href="#"><b>第42章：UIActivityViewController</b></a>	250
<a href="#">第42.1节：初始化活动视图控制器</a>	250
<a href="#"><b>第43章：UIButton - 使用块进行事件处理</b></a>	251
<a href="#">第43.1节：介绍</a>	251
<a href="#"><b>第44章：UISplitViewController</b></a>	254
<a href="#">第44.1节：使用Objective C中的代理实现主视图和详细视图交互</a>	254
<a href="#"><b>第45章：UISlider</b></a>	264
<a href="#">第45.1节：UISlider</a>	264
<a href="#">第45.2节：SWIFT示例</a>	264
<a href="#">第45.3节：添加自定义滑块图像</a>	264
<a href="#"><b>第46章：UIStoryboard</b></a>	266
<a href="#">第46.1节：以编程方式获取UIStoryboard实例</a>	266
<a href="#">第46.2节：打开另一个故事板</a>	266
<a href="#"><b>第47章：UIPageViewController</b></a>	267
<a href="#">第47.1节：通过代码创建水平分页的UIPageViewController</a>	267
<a href="#">第47.2节：创建水平分页视图控制器（无限页）的简单方法</a>	268
<a href="#"><b>第48章：UISplitViewController</b></a>	272
<a href="#">第48.1节：使用Objective C中的代理实现主视图和详细视图的交互</a>	272
<a href="#"><b>第49章：UIFont</b></a>	276
<a href="#">第49.1节：声明和初始化UIFont</a>	276
<a href="#">第49.2节：更改标签的字体</a>	276
<a href="#"><b>第50章：UIDevice</b></a>	277
<a href="#">第50.1节：获取iOS设备型号名称</a>	277
<a href="#">第50.2节：识别设备和操作系统</a>	278
<a href="#">第50.3节：获取设备方向</a>	279
<a href="#">第50.4节：获取设备电池状态</a>	280
<a href="#">第50.5节：使用接近传感器</a>	281
<a href="#">第50.6节：获取电池状态和电池电量</a>	281
<a href="#"><b>第51章：使UIView的选定角变圆角</b></a>	282
<a href="#">第51.1节：使用Objective C代码使UIView的选定角变圆角</a>	282
<a href="#"><b>第52章：来自XIB文件的自定义UIView</b></a>	283
<a href="#">第52.1节：布线元件</a>	283
<a href="#">第52.2节：如何使用XIB制作自定义可重用UIView</a>	297
<a href="#"><b>第53章：UIBezierPath</b></a>	298
<a href="#">第53.1节：设计和绘制贝塞尔曲线路径</a>	298
<a href="#">第53.2节：如何为UIBezierPath绘制的矩形应用圆角半径</a>	303
<a href="#">第53.3节：如何将阴影应用于UIBezierPath</a>	305
<a href="#">第53.4节：如何使用UIBezierPath创建简单形状</a>	307

<a href="#">Section 41.1: Create a UIWebView instance</a>	245
<a href="#">Section 41.2: Determining content size</a>	245
<a href="#">Section 41.3: Load HTML string</a>	245
<a href="#">Section 41.4: Making a URL request</a>	246
<a href="#">Section 41.5: Load JavaScript</a>	246
<a href="#">Section 41.6: Stop Loading Web Content</a>	247
<a href="#">Section 41.7: Reload Current Web Content</a>	247
<a href="#">Section 41.8: Load Document files like .pdf, .txt, .doc etc</a>	247
<a href="#">Section 41.9: Load local HTML file in webView</a>	247
<a href="#">Section 41.10: Make links That inside UIWebview clickable</a>	248
<a href="#"><b>Chapter 42: UIActivityViewController</b></a>	250
<a href="#">Section 42.1: Initializing the Activity View Controller</a>	250
<a href="#"><b>Chapter 43: UIControl - Event Handling with Blocks</b></a>	251
<a href="#">Section 43.1: Introduction</a>	251
<a href="#"><b>Chapter 44: UISplitViewController</b></a>	254
<a href="#">Section 44.1: Master and Detail View interaction using Delegates in Objective C</a>	254
<a href="#"><b>Chapter 45: UISlider</b></a>	264
<a href="#">Section 45.1: UISlider</a>	264
<a href="#">Section 45.2: SWIFT Example</a>	264
<a href="#">Section 45.3: Adding a custom thumb image</a>	264
<a href="#"><b>Chapter 46: UIStoryboard</b></a>	266
<a href="#">Section 46.1: Getting an instance of UIStoryboard programmatically</a>	266
<a href="#">Section 46.2: Open another storyboard</a>	266
<a href="#"><b>Chapter 47: UIPageViewController</b></a>	267
<a href="#">Section 47.1: Create a horizontal paging UIPageViewController programmatically</a>	267
<a href="#">Section 47.2: A simple way to create horizontal page view controllers ( infinite pages )</a>	268
<a href="#"><b>Chapter 48: UISplitViewController</b></a>	272
<a href="#">Section 48.1: Interacting Between Master and Detail View using Delegates in Objective C</a>	272
<a href="#"><b>Chapter 49: UIFont</b></a>	276
<a href="#">Section 49.1: Declaring and initializing UIFont</a>	276
<a href="#">Section 49.2: Changing the font of a label</a>	276
<a href="#"><b>Chapter 50: UIDevice</b></a>	277
<a href="#">Section 50.1: Get iOS device model name</a>	277
<a href="#">Section 50.2: Identifying the Device and Operating</a>	278
<a href="#">Section 50.3: Getting the Device Orientation</a>	279
<a href="#">Section 50.4: Getting the Device Battery State</a>	280
<a href="#">Section 50.5: Using the Proximity Sensor</a>	281
<a href="#">Section 50.6: Getting Battery Status and Battery Level</a>	281
<a href="#"><b>Chapter 51: Make selective UIView corners rounded</b></a>	282
<a href="#">Section 51.1: Objective C code to make selected corner of a UIView rounded</a>	282
<a href="#"><b>Chapter 52: Custom UIViews from XIB files</b></a>	283
<a href="#">Section 52.1: Wiring elements</a>	283
<a href="#">Section 52.2: How to make custom reusable UIView using XIB</a>	297
<a href="#"><b>Chapter 53: UIBezierPath</b></a>	298
<a href="#">Section 53.1: Designing and drawing a Bezier Path</a>	298
<a href="#">Section 53.2: How to apply corner radius to rectangles drawn by UIBezierPath</a>	303
<a href="#">Section 53.3: How to apply shadows to UIBezierPath</a>	305
<a href="#">Section 53.4: How to create a simple shapes using UIBezierPath</a>	307

第53.5节：UIBezierPath + 自动布局	309
第53.6节：使用UIBezierPath的饼图视图和柱状图视图	310
<b>第54章：UIPickerView</b>	313
第54.1节：基本示例	313
第54.2节：更改pickerView背景色和文字颜色	314
<b>第55章：UI反馈生成器</b>	315
第55.1节：触发冲击触觉反馈	315
<b>第56章：UI外观</b>	317
第56.1节：设置该类所有实例的外观	317
第56.2节：当包含在容器类中时该类的外观	318
<b>第57章：使用UICollectionView的UIKit动力学</b>	319
第57.1节：使用UIDynamicAnimator创建自定义拖拽行为	319
<b>第58章：UIPheonix——简单、灵活、动态且高度可扩展的UI框架</b>	329
第58.1节：示例UI组件	329
第58.2节：示例用法	331
<b>第59章：UIKit动力学</b>	332
第59.1节：基于手势速度的快速滑动视图	332
第59.2节：“粘性角落”效果，使用UIFieldBehaviors实现	335
第59.3节：由UIDynamicBehavior驱动的自定义过渡	340
第59.4节：使用UIDynamicBehaviors实现具有真实物理效果的遮罩过渡	351
第59.5节：将动态动画位置变化映射到边界	363
第59.6节：下落的正方形	368
<b>第60章：用户界面测试</b>	370
第60.1节：辅助功能标识符	370
第60.2节：向Xcode项目添加测试文件	372
第60.3节：在用户界面测试期间禁用动画	373
第60.4节：执行期间启动和终止应用程序	373
第60.5节：旋转设备	373
<b>第61章：更改状态栏颜色</b>	374
第61.1节：针对非UINavigationBar状态栏	374
第61.2节：针对UINavigationBar状态栏	374
第61.3节：针对ViewController的包含	375
第61.4节：如果无法更改ViewController的代码	375
第61.5节：更改整个应用程序的状态栏样式	375
<b>第62章：UISegmentedControl</b>	377
第62.1节：通过代码创建UISegmentedControl	377
<b>第63章：视图控制器之间传递数据</b>	378
第63.1节：使用代理模式（传递数据回去）	378
第63.2节：使用Segue（传递数据向前）	380
第63.3节：使用unwind segue向后传递数据	381
第63.4节：使用闭包传递数据（传递数据回去）	382
第63.5节：使用回调闭包（block）传回数据	383
第63.6节：通过赋值属性（传递数据）	384
<b>第64章：管理键盘</b>	385
第64.1节：创建自定义应用内键盘	385
第64.2节：通过点击视图关闭键盘	389
第64.3节：使用单例+代理管理键盘	390
第64.4节：键盘出现时视图上下移动	393
第64.5节：显示键盘时滚动UIScrollView/UITableView	394

Section 53.5: UIBezierPath + AutoLayout	309
Section 53.6: pie view & column view with UIBezierPath	310
<b>Chapter 54: UIPickerView</b>	313
Section 54.1: Basic example	313
Section 54.2: Changing pickerView Background Color and text color	314
<b>Chapter 55: UIFeedbackGenerator</b>	315
Section 55.1: Trigger Impact Haptic	315
<b>Chapter 56: UIAppearance</b>	317
Section 56.1: Set appearance of all instances of the class	317
Section 56.2: Appearance for class when contained in container class	318
<b>Chapter 57: UIKit Dynamics with UICollectionView</b>	319
Section 57.1: Creating a Custom Dragging Behavior with UIDynamicAnimator	319
<b>Chapter 58: UIPheonix - easy, flexible, dynamic &amp; highly scalable UI framework</b>	329
Section 58.1: Example UI Components	329
Section 58.2: Example Usage	331
<b>Chapter 59: UIKit Dynamics</b>	332
Section 59.1: Flick View Based on Gesture Velocity	332
Section 59.2: "Sticky Corners" Effect Using UIFieldBehaviors	335
Section 59.3: UIDynamicBehavior Driven Custom Transition	340
Section 59.4: Shade Transition with Real-World Physics Using UIDynamicBehaviors	351
Section 59.5: Map Dynamic Animation Position Changes to Bounds	363
Section 59.6: The Falling Square	368
<b>Chapter 60: UI Testing</b>	370
Section 60.1: Accessibility Identifier	370
Section 60.2: Adding Test Files to Xcode Project	372
Section 60.3: Disable animations during UI Testing	373
Section 60.4: Lunch and Terminate application while executing	373
Section 60.5: Rotate devices	373
<b>Chapter 61: Change Status Bar Color</b>	374
Section 61.1: For non-UINavigationBar status bars	374
Section 61.2: For UINavigationBar status bars	374
Section 61.3: For ViewController containment	375
Section 61.4: If you cannot change ViewController's code	375
Section 61.5: Changing the status bar style for the entire application	375
<b>Chapter 62: UISegmentedControl</b>	377
Section 62.1: Creating UISegmentedControl via code	377
<b>Chapter 63: Passing Data between View Controllers</b>	378
Section 63.1: Using the Delegate Pattern (passing data back)	378
Section 63.2: Using Segues (passing data forward)	380
Section 63.3: Passing data backwards using unwind to segue	381
Section 63.4: Passing data using closures (passing data back)	382
Section 63.5: Using callback closure(block) passing data back	383
Section 63.6: By assigning property (Passing data forward)	384
<b>Chapter 64: Managing the Keyboard</b>	385
Section 64.1: Create a custom in-app keyboard	385
Section 64.2: Dismiss a keyboard with tap on view	389
Section 64.3: Managing the Keyboard Using a Singleton + Delegate	390
Section 64.4: Moving view up or down when keyboard is present	393
Section 64.5: Scrolling a UIScrollView/UITableView When Displaying the Keyboard	394

<b>第65章：检查网络连接</b>	396	<b>Chapter 65: Checking for Network Connectivity</b>	396
第65.1节：创建Reachability监听器	396	Section 65.1: Creating a Reachability listener	396
第65.2节：添加网络变化观察者	396	Section 65.2: Add observer to network changes	396
第65.3节：网络不可用时的警报	396	Section 65.3: Alert when network becomes unavailable	396
第65.4节：连接变为WIFI或蜂窝网络时的警报	396	Section 65.4: Alert when connection becomes a WIFI or cellular network	396
第65.5节：验证是否已连接到网络	397	Section 65.5: Verify if is connected to network	397
<b>第66章：无障碍功能</b>	399	<b>Chapter 66: Accessibility</b>	399
第66.1节：使视图可访问	399	Section 66.1: Make a View Accessible	399
第66.2节：无障碍框架	399	Section 66.2: Accessibility Frame	399
第66.3节：布局更改	399	Section 66.3: Layout Change	399
第66.4节：无障碍容器	399	Section 66.4: Accessibility Container	399
第66.5节：隐藏元素	400	Section 66.5: Hiding Elements	400
第66.6节：屏幕切换	400	Section 66.6: Screen Change	400
第66.7节：公告	400	Section 66.7: Announcement	400
第66.8节：元素排序	400	Section 66.8: Ordering Elements	400
第66.9节：模态视图	401	Section 66.9: Modal View	401
<b>第67章：自动布局</b>	402	<b>Chapter 67: Auto Layout</b>	402
第67.1节：均匀分布视图	402	Section 67.1: Space Views Evenly	402
第67.2节：中心约束	403	Section 67.2: Center Constraints	403
第67.3节：通过编程设置约束	406	Section 67.3: Setting constraints programmatically	406
第67.4节：UILabel及其父视图大小根据UILabel中的文本调整	407	Section 67.4: UILabel & Parentview size According to Text in UILabel	407
第67.5节：UILabel的固有尺寸	412	Section 67.5: UILabel Intrinsic Size	412
第67.6节：视觉格式语言基础：代码中的约束！	422	Section 67.6: Visual Format Language Basics: Constraints in Code!	422
第67.7节：解决UILabel优先级冲突	424	Section 67.7: Resolve UILabel Priority Conflict	424
第67.8节：如何使用自动布局进行动画	426	Section 67.8: How to animate with Auto Layout	426
第67.9节：NSLayoutConstraint：代码中的约束！	427	Section 67.9: NSLayoutConstraint: Constraints in code!	427
第67.10节：比例布局	428	Section 67.10: Proportional Layout	428
第67.11节：自动布局与非自动布局的混合使用	429	Section 67.11: Mixed usage of Auto Layout with non-Auto Layout	429
第67.12节：如何使用自动布局	429	Section 67.12: How to use Auto Layout	429
<b>第68章：MKMapView</b>	431	<b>Chapter 68: MKMapView</b>	431
第68.1节：更改地图类型	431	Section 68.1: Change map-type	431
第68.2节：模拟自定义位置	435	Section 68.2: Simulate a custom location	435
第68.3节：设置地图缩放/区域	435	Section 68.3: Set Zoom/Region for Map	435
第68.4节：使用MKLocalSearch实现本地搜索	436	Section 68.4: Local search implementation using MKLocalSearch	436
第68.5节：OpenStreetMap瓦片覆盖	436	Section 68.5: OpenStreetMap Tile-Overlay	436
第68.6节：滚动到坐标和缩放级别	438	Section 68.6: Scroll to coordinate and zoom-level	438
第68.7节：使用注释	440	Section 68.7: Working With Annotation	440
第68.8节：添加MKMapView	440	Section 68.8: Add MKMapView	440
第68.9节：显示用户位置和用户跟踪示例	441	Section 68.9: Show UserLocation and UserTracking example	441
第68.10节：在地图上添加图钉/点注释	444	Section 68.10: Adding Pin/Point Annotation on map	444
第68.11节：调整地图视图的可见区域以显示所有注释	445	Section 68.11: Adjust the map view's visible rect in order to display all annotations	445
<b>第69章：NSArray</b>	446	<b>Chapter 69: NSArray</b>	446
第69.1节：将数组转换为json字符串	446	Section 69.1: Convert Array into json string	446
<b>第70章：NSAttributedString</b>	447	<b>Chapter 70: NSAttributedString</b>	447
第70.1节：创建具有自定义字距（字母间距）的字符串	447	Section 70.1: Creating a string that has custom kerning (letter spacing)	447
第70.2节：更改单词或字符串的颜色	447	Section 70.2: Change the color of a word or string	447
第70.3节：移除所有属性	448	Section 70.3: Removing all attributes	448
第70.4节：在Swift中追加带属性的字符串和加粗文本	448	Section 70.4: Appending Attributed Strings and bold text in Swift	448
第70.5节：创建带删除线的字符串	448	Section 70.5: Create a string with strikethrough text	448

<b>第71章：HTML与NSAttributedString字符串的相互转换</b>	450
第71.1节：将HTML字符串转换为NSAttributedString及反向转换的Objective C代码	450
<b>第72章：NSTimer</b>	451
第72.1节：创建定时器	451
第72.2节：手动触发定时器	451
第72.3节：定时器频率选项	452
第72.4节：使定时器失效	453
第72.5节：使用定时器传递数据	453
<b>第73章：NSDate</b>	454
第73.1节：NSDateFormatter	454
第73.2节：日期比较	455
第73.3节：从NSDate获取历史时间（例如：5秒前，2分钟前，3小时前）	457
第73.4节：获取Unix纪元时间	458
第73.5节：从JSON日期格式"/Date(1268123281843)"/获取NSDate	458
第73.6节：获取时间周期类型（12小时制或24小时制）	459
第73.7节：获取当前日期	459
第73.8节：获取距离当前日期N秒的NSDate对象	459
第73.9节：带时区的NSDate的UTC时间偏移	460
第73.10节：将仅由小时和分钟组成的NSDate转换为完整的NSDate	460
<b>第74章：NSNotificationCenter</b>	462
第74.1节：移除观察者	462
第74.2节：添加观察者	462
第74.3节：发送带数据的通知	463
第74.4节：为名称添加和移除观察者	463
第74.5节：发送通知	463
第74.6节：观察通知	464
第74.7节：使用Block添加/移除观察者	464
<b>第75章：NSURLSession</b>	465
第75.1节：Objective-C 创建会话和数据任务	465
第75.2节：设置后台配置	465
第75.3节：简单的GET请求	466
第75.4节：使用NSURLSession在Objective-C中发送带参数的POST请求	466
<b>第76章：NSUserDefaults</b>	471
第76.1节：设置值	471
第76.2节：Swift 3中UserDefaults的使用	472
第76.3节：使用管理器保存和读取数据	473
第76.4节：保存值	475
第76.5节：清除NSUserDefaults	475
第76.6节：获取默认值	475
<b>第77章：NSHTTPCookieStorage</b>	477
第77.1节：从NSUserDefaults存储和读取Cookie	477
<b>第78章：NSURLConnection</b>	479
第78.1节：代理方法	479
第78.2节：同步请求	479
第78.3节：异步请求	480
<b>第79章：NSURL</b>	481
第79.1节：如何从NSURL字符串中获取最后的字符串组件	481
第79.2节：如何在Swift中从URL (NSURL) 获取最后的字符串组件	481
<b>第80章：NSData</b>	482

<b>Chapter 71: Convert HTML to NSAttributedString string and vice versa</b>	450
Section 71.1: Objective C code to convert HTML string to NSAttributedString and Vice Versa	450
<b>Chapter 72: NSTimer</b>	451
Section 72.1: Creating a Timer	451
Section 72.2: Manually firing a timer	451
Section 72.3: Timer frequency options	452
Section 72.4: Invalidating a timer	453
Section 72.5: Passing of data using Timer	453
<b>Chapter 73: NSDate</b>	454
Section 73.1: NSDateFormatter	454
Section 73.2: Date Comparison	455
Section 73.3: Get Historic Time from NSDate (eg: 5s ago, 2m ago, 3h ago)	457
Section 73.4: Get Unix Epoch time	458
Section 73.5: Get NSDate from JSON Date format "/Date(1268123281843)/*	458
Section 73.6: Get time cycle type (12-hour or 24-hour)	459
Section 73.7: Get Current Date	459
Section 73.8: Get NSDate Object N seconds from current date	459
Section 73.9: UTC Time offset from NSDate with TimeZone	460
Section 73.10: Convert NSDate that is composed from hour and minute (only) to a full NSDate	460
<b>Chapter 74: NSNotificationCenter</b>	462
Section 74.1: Removing Observers	462
Section 74.2: Adding an Observer	462
Section 74.3: Posting a Notification with Data	463
Section 74.4: Add and remove observer for name	463
Section 74.5: Posting a Notification	463
Section 74.6: Observing a Notification	464
Section 74.7: Adding/Removing an Observer with a Block	464
<b>Chapter 75: NSURLSession</b>	465
Section 75.1: Objective-C Create a Session And Data Task	465
Section 75.2: Setting up background configuration	465
Section 75.3: Simple GET request	466
Section 75.4: Sending a POST Request with arguments using NSURLSession in Objective-C	466
<b>Chapter 76: UserDefaults</b>	471
Section 76.1: Setting values	471
Section 76.2: UserDefaults uses in Swift 3	472
Section 76.3: Use Managers to Save and Read Data	473
Section 76.4: Saving Values	475
Section 76.5: Clearing UserDefaults	475
Section 76.6: Getting Default Values	475
<b>Chapter 77: NSHTTPCookieStorage</b>	477
Section 77.1: Store and read the cookies from UserDefaults	477
<b>Chapter 78: NSURLConnection</b>	479
Section 78.1: Delegate methods	479
Section 78.2: Synchronous Request	479
Section 78.3: Asynchronous request	480
<b>Chapter 79: NSURL</b>	481
Section 79.1: How to get last string component from NSURL String	481
Section 79.2: How to get last string component from URL (NSURL) in Swift	481
<b>Chapter 80: NSData</b>	482

第80.1节：将NSData转换为HEX字符串	482
第80.2节：创建NSData对象	482
第80.3节：将NSData转换为其他类型	482
<b>第81章：NSInvocation</b>	484
第81.1节：NSInvocation Objective-C	484
<b>第82章：NSUserActivity</b>	486
第82.1节：创建NSUserActivity	486
<b>第83章：NSPredicate</b>	487
第83.1节：使用NSPredicate进行表单验证	487
第83.2节：使用predicateWithBlock创建NSPredicate	488
第83.3节：使用predicateWithFormat创建NSPredicate	488
第83.4节：使用替换变量创建NSPredicate	489
第83.5节：使用带有`AND`、`OR`和`NOT`条件的NSPredicate	489
第83.6节：使用NSPredicate过滤数组	489
<b>第84章：NSBundle</b>	491
第84.1节：通过路径获取Bundle	491
第84.2节：获取 mainBundle	491
<b>第85章：CAAnimation</b>	492
第85.1节：将视图从一个位置动画到另一个位置	492
第85.2节：动画视图 - 投掷	492
第85.3节：推送视图动画	492
第85.4节：旋转视图	493
第85.5节：抖动视图	493
<b>第86章：并发</b>	494
第86.1节：调度组 - 等待其他线程完成	494
第86.2节：在主线程上执行	495
第86.3节：并发运行代码——在运行其他代码时运行代码	495
<b>第87章：CAGradientLayer</b>	496
第87.1节：创建CAGradientLayer	496
第87.2节：在CAGradientLayer中实现颜色变化动画	497
第87.3节：创建水平CAGradientLayer	498
第87.4节：创建多色水平CAGradientLayer	499
第87.5节：创建具有多种颜色的CGGradientLayer	500
<b>第88章：Safari服务</b>	502
第88.1节：使用SafariViewController打开URL	502
第88.2节：实现SFSafariViewControllerDelegate	502
第88.3节：向Safari阅读列表添加项目	503
<b>第89章：CALayer</b>	504
第89.1节：向CALayer添加变换（平移、旋转、缩放）	504
第89.2节：阴影	508
第89.3节：带自定义图像的发射器视图	509
第89.4节：圆角	510
第89.5节：使用CAEmitterLayer创建粒子	510
第89.6节：如何向CALayer添加UIImage	511
第89.7节：禁用动画	514
<b>第90章：iOS - 使用Robbie Hanson框架实现XMPP</b>	516
第90.1节：iOS XMPP Robbie Hanson示例与Openfire	516
<b>第91章：Swift与Objective-C的互操作性</b>	519
第91.1节：在Swift中使用Objective-C类	519

Section 80.1: Converting NSData to HEX string	482
Section 80.2: Creating NSData objects	482
Section 80.3: Converting NSData to other types	482
<b>Chapter 81: NSInvocation</b>	484
Section 81.1: NSInvocation Objective-C	484
<b>Chapter 82: NSUserActivity</b>	486
Section 82.1: Creating a NSUserActivity	486
<b>Chapter 83: NSPredicate</b>	487
Section 83.1: Form validation using NSPredicate	487
Section 83.2: Creating an NSPredicate Using predicateWithBlock	488
Section 83.3: Creating an NSPredicate Using predicateWithFormat	488
Section 83.4: Creating an NSPredicate with Substitution Variables	489
Section 83.5: NSPredicate with `AND`, `OR` and `NOT` condition	489
Section 83.6: Using NSPredicate to Filter an Array	489
<b>Chapter 84: NSBundle</b>	491
Section 84.1: Getting Bundle by Path	491
Section 84.2: Getting the Main Bundle	491
<b>Chapter 85: CAAnimation</b>	492
Section 85.1: Animate a view from one position to another	492
Section 85.2: Animate View - Toss	492
Section 85.3: Push View Animation	492
Section 85.4: Revolve View	493
Section 85.5: Shake View	493
<b>Chapter 86: Concurrency</b>	494
Section 86.1: Dispatch group - waiting for other threads completed	494
Section 86.2: Executing on the main thread	495
Section 86.3: Running code concurrently -- Running code while running other code	495
<b>Chapter 87: CAGradientLayer</b>	496
Section 87.1: Creating a CAGradientLayer	496
Section 87.2: Animating a color change in CAGradientLayer	497
Section 87.3: Creating a horizontal CAGradientLayer	498
Section 87.4: Creating a horizontal CAGradientLayer with multiple colors	499
Section 87.5: Creating a CGGradientLayer with multiple colors	500
<b>Chapter 88: Safari Services</b>	502
Section 88.1: Open a URL with SafariViewController	502
Section 88.2: Implement SFSafariViewControllerDelegate	502
Section 88.3: Add Items to Safari Reading List	503
<b>Chapter 89: CALayer</b>	504
Section 89.1: Adding Transforms to a CALayer (translate, rotate, scale)	504
Section 89.2: Shadows	508
Section 89.3: Emitter View with custom image	509
Section 89.4: Rounded corners	510
Section 89.5: Creating particles with CAEmitterLayer	510
Section 89.6: How to add a UIImage to a CALayer	511
Section 89.7: Disable Animations	514
<b>Chapter 90: iOS - Implementation of XMPP with Robbie Hanson framework</b>	516
Section 90.1: iOS XMPP Robbie Hanson Example with Openfire	516
<b>Chapter 91: Swift and Objective-C interoperability</b>	519
Section 91.1: Using Objective-C Classes in Swift	519

第91.2节：在Objective-C中使用Swift类	521
<b>第92章：自定义字体</b>	523
第92.1节：嵌入自定义字体	523
第92.2节：将自定义字体应用于Storyboard中的控件	523
第92.3节：Storyboard中的自定义字体	526
<b>第93章：AVSpeechSynthesizer</b>	529
第93.1节：创建基本的文本转语音	529
<b>第94章：本地化</b>	530
第94.1节：iOS中的本地化	530
<b>第95章：Alamofire</b>	531
第95.1节：手动验证	531
第95.2节：自动验证	531
第95.3节：链式响应处理器	531
第95.4节：发起请求	531
第95.5节：响应处理	531
第95.6节：响应处理器	532
<b>第96章：iBeacon</b>	533
第96.1节：iBeacon基础操作	533
第96.2节：测距iBeacon	533
第96.3节：扫描特定Beacon	534
<b>第97章：CLLocation</b>	535
第97.1节：使用距离过滤器	535
第97.2节：使用CLLocationManager获取用户位置	535
<b>第98章：检查iOS版本</b>	537
第98.1节：iOS 8及更高版本	537
第98.2节：Swift 2.0及更高版本	537
第98.3节：版本比较	537
第98.4节：设备iOS版本	537
<b>第99章：通用链接</b>	539
第99.1节：设置iOS应用程序（启用通用链接）	539
第99.2节：支持多个域名	542
第99.3节：为App-Site-Association文件签名	542
第99.4节：设置服务器	542
<b>第100章：iOS中的PDF创建</b>	544
第100.1节：创建PDF	544
第100.2节：显示PDF	545
第100.3节：多页PDF	546
第100.4节：从UIWebView中加载的任何Microsoft文档创建PDF	546
<b>第101章：应用内购买</b>	548
第101.1节：Swift 2中的单一应用内购买（IAP）	548
第101.2节：购买/订阅用户应用内购买的最基本步骤	550
第101.3节：在iTunesConnect中的设置	550
<b>第102章：CGContext参考</b>	552
第102.1节：绘制线条	552
第102.2节：绘制文本	552
<b>第103章：核心定位（Core Location）</b>	554
第103.1节：请求使用定位服务的权限	554
第103.2节：使用GPX文件添加自定义位置	557

Section 91.2: Using Swift Classes in Objective-C	521
<b>Chapter 92: Custom fonts</b>	523
Section 92.1: Embedding custom fonts	523
Section 92.2: Applying custom fonts to controls within a Storyboard	523
Section 92.3: Custom Fonts with Storyboard	526
<b>Chapter 93: AVSpeechSynthesizer</b>	529
Section 93.1: Creating a basic text to speech	529
<b>Chapter 94: Localization</b>	530
Section 94.1: Localization in iOS	530
<b>Chapter 95: Alamofire</b>	531
Section 95.1: Manual Validation	531
Section 95.2: Automatic Validation	531
Section 95.3: Chained Response Handlers	531
Section 95.4: Making a Request	531
Section 95.5: Response Handling	531
Section 95.6: Response Handler	532
<b>Chapter 96: iBeacon</b>	533
Section 96.1: iBeacon Basic Operation	533
Section 96.2: Ranging iBeacons	533
Section 96.3: Scanning specific Beacons	534
<b>Chapter 97: CLLocation</b>	535
Section 97.1: Distance Filter using	535
Section 97.2: Get User Location Using CLLocationManager	535
<b>Chapter 98: Checking iOS version</b>	537
Section 98.1: iOS 8 and later	537
Section 98.2: Swift 2.0 and later	537
Section 98.3: Compare versions	537
Section 98.4: Device iOS Version	537
<b>Chapter 99: Universal Links</b>	539
Section 99.1: Setup iOS Application (Enabling Universal Links)	539
Section 99.2: Supporting Multiple Domains	542
Section 99.3: Signing the App-Site-Association File	542
Section 99.4: Setup Server	542
<b>Chapter 100: PDF Creation in iOS</b>	544
Section 100.1: Create PDF	544
Section 100.2: Show PDF	545
Section 100.3: Multiple page PDF	546
Section 100.4: Create PDF from any Microsoft Document loaded in UIWebView	546
<b>Chapter 101: In-App Purchase</b>	548
Section 101.1: Single IAP in Swift 2	548
Section 101.2: Most basic steps for purchasing/subscribing a user to an IAP	550
Section 101.3: Set Up in iTunesConnect	550
<b>Chapter 102: CGContext Reference</b>	552
Section 102.1: Draw line	552
Section 102.2: Draw Text	552
<b>Chapter 103: Core Location</b>	554
Section 103.1: Request Permission to Use Location Services	554
Section 103.2: Add own custom location using GPX file	557

第103.3节：链接CoreLocation框架	557
第103.4节：后台定位服务	558
<b>第104章：FacebookSDK</b>	560
第104.1节：创建自定义“使用Facebook登录”按钮	560
第104.2节：FacebookSDK集成	560
第104.3节：获取Facebook用户数据	563
<b>第105章：AFNetworking</b>	564
第105.1节：在自定义线程上调度完成块	564
<b>第106章：CTCallCenter</b>	565
第106.1节：CallKit - iOS 10	565
第106.2节：即使在后台也能拦截来自应用的电话	565
<b>第107章：推送通知</b>	567
第107.1节：注册推送通知设备	567
第107.2节：注册用于推送通知的应用程序ID	570
第107.3节：测试推送通知	572
第107.4节：检查您的应用是否已注册推送通知	574
第107.5节：从.cer文件生成.pem证书，以传递给服务器开发人员	574
第107.6节：取消注册推送通知	574
第107.7节：设置应用图标徽章数字	575
第107.8节：注册（非交互式）推送通知	575
第107.9节：处理推送通知	576
<b>第108章：丰富推送通知扩展 - iOS 10</b>	578
第108.1节：通知内容扩展	578
第108.2节：实现	578
<b>第109章：丰富通知</b>	580
第109.1节：创建一个简单的UNNotificationContentExtension	580
<b>第110章：键值编码-键值观察</b>	585
第110.1节：观察NSObject子类的属性	585
第110.2节：KVO观察中上下文的使用	586
<b>第111章：初始化惯用法</b>	587
第111.1节：带块的工厂方法	587
第111.2节：使用元组以避免代码重复	587
第111.3节：使用位置常数初始化	587
第111.4节：在didSet中初始化属性	588
第111.5节：在自定义NSObject中分组outlets	588
第111.6节：使用then初始化	588
<b>第112章：Storyboard</b>	590
第112.1节：初始化	590
第112.2节：获取初始视图控制器	590
第112.3节：获取视图控制器	590
<b>第113章：后台模式和事件</b>	591
第113.1节：后台播放音频	591
<b>第114章：Fastlane</b>	593
第114.1节：fastlane工具	593
<b>第115章：CAShapeLayer</b>	594
第115.1节：绘制矩形	594
第115.2节：绘制圆形	594
第115.3节：CAShapeLayer动画	595

Section 103.3: Link CoreLocation Framework	557
Section 103.4: Location Services in the Background	558
<b>Chapter 104: FacebookSDK</b>	560
Section 104.1: Creating your own custom "Sign In With Facebook" button	560
Section 104.2: FacebookSDK Integration	560
Section 104.3: Fetching the facebook user data	563
<b>Chapter 105: AFNetworking</b>	564
Section 105.1: Dispatching completion block on a custom thread	564
<b>Chapter 106: CTCallCenter</b>	565
Section 106.1: CallKit - ios 10	565
Section 106.2: Intercepting calls from your app even from the background	565
<b>Chapter 107: Push Notifications</b>	567
Section 107.1: Registering device for Push Notifications	567
Section 107.2: Registering App ID for use with Push Notifications	570
Section 107.3: Testing push notifications	572
Section 107.4: Checking if your app is already registered for Push Notification	574
Section 107.5: Generating a .pem certificate from your .cer file, to pass on to the server developer	574
Section 107.6: Unregistering From Push Notifications	574
Section 107.7: Setting the application icon badge number	575
Section 107.8: Registering for (Non Interactive) Push Notification	575
Section 107.9: Handling Push Notification	576
<b>Chapter 108: Extension for rich Push Notification - iOS 10.</b>	578
Section 108.1: Notification Content Extension	578
Section 108.2: Implementation	578
<b>Chapter 109: Rich Notifications</b>	580
Section 109.1: Creating a simple UNNotificationContentExtension	580
<b>Chapter 110: Key Value Coding-Key Value Observation</b>	585
Section 110.1: Observing a property of a NSObject subclass	585
Section 110.2: Use of context for KVO Observation	586
<b>Chapter 111: Initialization idioms</b>	587
Section 111.1: Factory method with block	587
Section 111.2: Set to tuples to avoid code repetition	587
Section 111.3: Initialize with positional constants	587
Section 111.4: Initialize attributes in didSet	588
Section 111.5: Group outlets in a custom NSObject	588
Section 111.6: Initialize with then	588
<b>Chapter 112: Storyboard</b>	590
Section 112.1: Initialize	590
Section 112.2: Fetch Initial ViewController	590
Section 112.3: Fetch ViewController	590
<b>Chapter 113: Background Modes and Events</b>	591
Section 113.1: Play Audio in Background	591
<b>Chapter 114: Fastlane</b>	593
Section 114.1: fastlane tools	593
<b>Chapter 115: CAShapeLayer</b>	594
Section 115.1: Draw Rectangle	594
Section 115.2: Draw Circle	594
Section 115.3: CAShapeLayer Animation	595

<a href="#">第115.4节：基本的CAShapeLayer操作</a>	596
<b>第116章：WKWebView</b>	600
<a href="#">第116.1节：添加从应用程序包加载的自定义用户脚本</a>	600
<a href="#">第116.2节：从JavaScript发送消息并在本地端处理</a>	600
<a href="#">第116.3节：创建一个简单的网页浏览器</a>	601
<b>第117章：UUID（通用唯一标识符）</b>	609
<a href="#">第117.1节：苹果的IFA与IFV（广告标识符与供应商标识符）</a>	609
<a href="#">第117.2节：为iOS设备创建UUID字符串</a>	609
<a href="#">第117.3节：生成UUID</a>	609
<a href="#">第117.4节：供应商标识符</a>	610
<b>第118章：类别</b>	611
<a href="#">第118.1节：创建类别</a>	611
<b>第119章：处理URL方案</b>	614
<a href="#">第119.1节：使用内置URL方案打开邮件应用</a>	614
<a href="#">第119.2节：苹果URL方案</a>	614
<b>第120章：Realm</b>	617
<a href="#">第120.1节：带主键的RLMObject基类模型 - Objective-C</a>	617
<b>第121章：ARC（自动引用计数）</b>	618
<a href="#">第121.1节：启用/禁用文件的ARC</a>	618
<b>第122章：动态字体</b>	619
<a href="#">第122.1节：在WKWebView中匹配动态字体大小</a>	619
<a href="#">第122.2节：获取当前内容大小</a>	620
<a href="#">第122.3节：在iOS 10上无通知处理首选文本大小更改</a>	620
<a href="#">第122.4节：文本大小更改通知</a>	620
<b>第123章：SWRevealViewController</b>	621
<a href="#">第123.1节：使用SWRevealViewController设置基础应用</a>	621
<b>第124章：DispatchGroup</b>	624
<a href="#">第124.1节：介绍</a>	624
<b>第125章：GCD（Grand Central Dispatch）</b>	627
<a href="#">第125.1节：调度信号量</a>	627
<a href="#">第125.2节：调度组</a>	627
<a href="#">第125.3节：获取主线程</a>	628
<a href="#">第125.4节：创建调度队列</a>	629
<a href="#">第125.5节：串行与并发调度队列</a>	629
<b>第126章：尺寸类别与适应性</b>	631
<a href="#">第126.1节：特征集合</a>	631
<a href="#">第126.2节：使用特征集合更改更新自动布局</a>	631
<a href="#">第126.3节：支持iPad上的iOS多任务处理</a>	632
<b>第127章：IBOutlets</b>	634
<a href="#">第127.1节：在UI元素中使用IBOutlets</a>	634
<b>第128章：AWS SDK</b>	635
<a href="#">第128.1节：使用AWS SDK将图像或视频上传到S3</a>	635
<b>第129章：调试崩溃</b>	638
<a href="#">第129.1节：调试EXC_BAD_ACCESS</a>	638
<a href="#">第129.2节：查找崩溃信息</a>	639
<a href="#">第129.3节：调试SIGABRT和EXC_BAD_INSTRUCTION崩溃</a>	640
<b>第130章：CloudKit</b>	642
<a href="#">第130.1节：注册应用以使用CloudKit</a>	642

<a href="#">Section 115.4: Basic CAShapeLayer Operation</a>	596
<b>Chapter 116: WKWebView</b>	600
<a href="#">Section 116.1: Adding custom user script loaded from app bundle</a>	600
<a href="#">Section 116.2: Send messages from JavaScript and Handle them on the native side</a>	600
<a href="#">Section 116.3: Creating a Simple WebBrowser</a>	601
<b>Chapter 117: UUID (Universally Unique Identifier)</b>	609
<a href="#">Section 117.1: Apple's IFA vs. IFV (Apple Identifier for Advertisers vs. Identifier for Vendors)</a>	609
<a href="#">Section 117.2: Create UUID String for iOS devices</a>	609
<a href="#">Section 117.3: Generating UUID</a>	609
<a href="#">Section 117.4: Identifier for vendor</a>	610
<b>Chapter 118: Categories</b>	611
<a href="#">Section 118.1: Create a Category</a>	611
<b>Chapter 119: Handling URL Schemes</b>	614
<a href="#">Section 119.1: Using built-in URL scheme to open Mail app</a>	614
<a href="#">Section 119.2: Apple URL Schemes</a>	614
<b>Chapter 120: Realm</b>	617
<a href="#">Section 120.1: RLMObject Base Model Class with Primary Key - Objective-C</a>	617
<b>Chapter 121: ARC (Automatic Reference Counting)</b>	618
<a href="#">Section 121.1: Enable/Disable ARC on a file</a>	618
<b>Chapter 122: Dynamic Type</b>	619
<a href="#">Section 122.1: Matching Dynamic Type Font Size in WKWebView</a>	619
<a href="#">Section 122.2: Get the Current Content Size</a>	620
<a href="#">Section 122.3: Handling Preferred Text Size Change Without Notifications on iOS 10</a>	620
<a href="#">Section 122.4: Text Size Change Notification</a>	620
<b>Chapter 123: SWRevealViewController</b>	621
<a href="#">Section 123.1: Setting up a basic app with SWRevealViewController</a>	621
<b>Chapter 124: DispatchGroup</b>	624
<a href="#">Section 124.1: Introduction</a>	624
<b>Chapter 125: GCD (Grand Central Dispatch)</b>	627
<a href="#">Section 125.1: Dispatch Semaphore</a>	627
<a href="#">Section 125.2: Dispatch Group</a>	627
<a href="#">Section 125.3: Getting the Main Queue</a>	628
<a href="#">Section 125.4: Create a dispatch queue</a>	629
<a href="#">Section 125.5: Serial vs Concurrent Dispatch Queues</a>	629
<b>Chapter 126: Size Classes and Adaptivity</b>	631
<a href="#">Section 126.1: Trait Collections</a>	631
<a href="#">Section 126.2: Updating Auto Layout with Trait Collection Changes</a>	631
<a href="#">Section 126.3: Supporting iOS Multitasking on iPad</a>	632
<b>Chapter 127: IBOutlets</b>	634
<a href="#">Section 127.1: Using an IBOutlet in a UI Element</a>	634
<b>Chapter 128: AWS SDK</b>	635
<a href="#">Section 128.1: Upload an image or a video to S3 using AWS SDK</a>	635
<b>Chapter 129: Debugging Crashes</b>	638
<a href="#">Section 129.1: Debugging EXC_BAD_ACCESS</a>	638
<a href="#">Section 129.2: Finding information about a crash</a>	639
<a href="#">Section 129.3: Debugging SIGABRT and EXC_BAD_INSTRUCTION crashes</a>	640
<b>Chapter 130: CloudKit</b>	642
<a href="#">Section 130.1: Registering app for use with CloudKit</a>	642

第130.2节：使用CloudKit仪表板	642
第130.3节：将数据保存到CloudKit	643
<b>第131章：GameplayKit</b>	644
第131.1节：生成随机数	644
第131.2节：GKEntity和GKComponent	645
<b>第132章：从命令行构建和归档Xcode</b>	648
第132.1节：构建与归档	648
<b>第133章：XCTest框架 - 单元测试</b>	649
第133.1节：向Xcode项目添加测试文件	649
第133.2节：添加测试方法	650
第133.3节：编写测试类	651
第133.4节：将Storyboard和视图控制器作为实例添加到测试文件中	652
第133.5节：导入可测试的模块	652
第133.6节：触发视图加载和显示	653
第133.7节：开始测试	653
<b>第134章：AVPlayer和AVPlayerViewController</b>	654
第134.1节：使用AVPlayer和AVPlayerLayer播放媒体	654
第134.2节：使用AVPlayerViewController播放媒体	654
第134.3节：AVPlayer示例	654
<b>第135章：iOS中的深度链接</b>	656
第135.1节：向您自己的应用程序添加URL方案	656
第135.2节：基于URL方案打开应用程序	657
第135.3节：为您的应用程序设置深度链接	658
<b>第136章：核心图形</b>	660
第136.1节：创建核心图形上下文	660
第136.2节：向用户展示绘制的画布	660
<b>第137章：界面切换 (Segues)</b>	662
第137.1节：使用Segue在导航堆栈中向后导航	662
第137.2节：概述	662
第137.3节：在触发Segue之前准备视图控制器	663
第137.4节：决定是否执行调用的Segue	663
第137.5节：以编程方式触发Segue	663
<b>第138章：EventKit</b>	665
第138.1节：访问不同类型的日历	665
第138.2节：请求权限	665
第138.3节：添加事件	666
<b>第139章：SiriKit</b>	668
第139.1节：向应用添加Siri扩展	668
<b>第140章：联系人框架</b>	670
第140.1节：添加联系人	670
第140.2节：授权联系人访问	670
第140.3节：访问联系人	671
<b>第141章：iOS 10语音识别API</b>	673
第141.1节：语音转文本：识别捆绑包中包含的音频录音的语言	673
<b>第142章：StoreKit</b>	675
第142.1节：从App Store获取本地化的产品信息	675
<b>第143章：代码签名</b>	676
第143.1节：配置描述文件	676

Section 130.2: Using CloudKit Dashboard	642
Section 130.3: Saving data to CloudKit	643
<b>Chapter 131: GameplayKit</b>	644
Section 131.1: Generating random numbers	644
Section 131.2: GKEntity and GKComponent	645
<b>Chapter 132: Xcode Build &amp; Archive From Command Line</b>	648
Section 132.1: Build & Archive	648
<b>Chapter 133: XCTest framework - Unit Testing</b>	649
Section 133.1: Adding Test Files to Xcode Project	649
Section 133.2: Adding test methods	650
Section 133.3: Writing a test class	651
Section 133.4: Adding Storyboard and View Controller as instances to test file	652
Section 133.5: Import a module that it can be tested	652
Section 133.6: Trigger view loading and appearance	653
Section 133.7: Start Testing	653
<b>Chapter 134: AVPlayer and AVPlayerViewController</b>	654
Section 134.1: Playing Media using AVPlayer and AVPlayerLayer	654
Section 134.2: Playing Media Using AVPlayerViewController	654
Section 134.3: AVPlayer Example	654
<b>Chapter 135: Deep Linking in iOS</b>	656
Section 135.1: Adding a URL scheme to your own app	656
Section 135.2: Opening an app based on its URL scheme	657
Section 135.3: Setting up deeplink for your app	658
<b>Chapter 136: Core Graphics</b>	660
Section 136.1: Creating a Core Graphics Context	660
Section 136.2: Presenting the Drawn Canvas to User	660
<b>Chapter 137: Segues</b>	662
Section 137.1: Using Segues to navigate backwards in the navigation stack	662
Section 137.2: An Overview	662
Section 137.3: Preparing your view controller before triggering a Segue	663
Section 137.4: Deciding if an invoked Segue should be performed	663
Section 137.5: Trigger Segue Programmatically	663
<b>Chapter 138: EventKit</b>	665
Section 138.1: Accessing different types of calendars	665
Section 138.2: Requesting Permission	665
Section 138.3: Adding an event	666
<b>Chapter 139: SiriKit</b>	668
Section 139.1: Adding Siri Extension to App	668
<b>Chapter 140: Contacts Framework</b>	670
Section 140.1: Adding a Contact	670
Section 140.2: Authorizing Contact Access	670
Section 140.3: Accessing Contacts	671
<b>Chapter 141: iOS 10 Speech Recognition API</b>	673
Section 141.1: Speech to text: Recognize speech from a bundle contained audio recording	673
<b>Chapter 142: StoreKit</b>	675
Section 142.1: Get localized product information from the App Store	675
<b>Chapter 143: Code signing</b>	676
Section 143.1: Provisioning Profiles	676

<b>第144章：使用Application Loader创建上传到App Store的.ipa文件</b>	677
第144.1节：创建.ipa文件以通过Application Loader上传应用到App Store	677
<b>第145章：尺寸类别与自适应</b>	681
第145.1节：通过Storyboard实现尺寸类别与自适应	681
<b>第146章：MKDistanceFormatter</b>	687
第146.1节：从距离生成字符串	687
第146.2节：距离单位	687
第146.3节：单位样式	687
<b>第147章：3D触控</b>	689
第147.1节：使用Swift的3D Touch	689
第147.2节：3D Touch Objective-C示例	690
<b>第148章：GameCenter排行榜</b>	692
第148.1节：GameCenter排行榜	692
<b>第149章：钥匙串</b>	694
第149.1节：向钥匙串添加密码	694
第149.2节：钥匙串访问控制（带密码回退的TouchID）	695
第149.3节：在钥匙串中查找密码	696
第149.4节：在钥匙串中更新密码	697
第149.5节：从钥匙串中移除密码	697
第149.6节：使用一个文件进行钥匙串的添加、更新、移除和查找操作	698
<b>第150章：使用宏处理多环境</b>	701
第150.1节：使用多个目标和宏处理多环境	701
<b>第151章：设置视图背景</b>	716
第151.1节：填充UIView的背景图片	716
第151.2节：创建渐变背景视图	716
第151.3节：使用图片设置视图背景	716
第151.4节：设置视图背景	716
<b>第152章：Block</b>	718
第152.1节：自定义方法的自定义完成块	718
第152.2节：UIView动画	718
第152.3节：修改捕获的变量	718
<b>第153章：自动布局中的内容拥抱/内容压缩</b>	720
第153.1节：定义：内在内容大小	720
<b>第154章：iOS谷歌地点API</b>	721
第154.1节：从当前位置获取附近地点	721
<b>第155章：导航栏</b>	723
第155.1节：SWIFT示例	723
第155.2节：自定义默认导航栏外观	723
<b>第156章：应用程序范围操作</b>	724
第156.1节：获取最顶层的UIViewController	724
第156.2节：拦截系统事件	724
<b>第157章：CoreImage滤镜</b>	725
第157.1节：核心图像滤镜示例	725
<b>第158章：使用CoreImage/OpenCV进行人脸检测</b>	731
第158.1节：人脸和特征检测	731
<b>第159章：MPMediaPickerControllerDelegate</b>	734
第159.1节：使用MPMediaPickerControllerDelegate加载音乐并用AVAudioPlayer播放	734

<b>Chapter 144: Create .ipa File to upload on appstore with Applicationloader</b>	677
Section 144.1: create .ipa file to upload app to appstore with Application Loader	677
<b>Chapter 145: Size Classes and Adaptivity</b>	681
Section 145.1: Size Classes and Adaptivity through Storyboard	681
<b>Chapter 146: MKDistanceFormatter</b>	687
Section 146.1: String from distance	687
Section 146.2: Distance units	687
Section 146.3: Unit style	687
<b>Chapter 147: 3D Touch</b>	689
Section 147.1: 3D Touch with Swift	689
Section 147.2: 3 D Touch Objective-C Example	690
<b>Chapter 148: GameCenter Leaderboards</b>	692
Section 148.1: GameCenter Leaderboards	692
<b>Chapter 149: Keychain</b>	694
Section 149.1: Adding a Password to the Keychain	694
Section 149.2: Keychain Access Control (TouchID with password fallback)	695
Section 149.3: Finding a Password in the Keychain	696
Section 149.4: Updating a Password in the Keychain	697
Section 149.5: Removing a Password from the Keychain	697
Section 149.6: Keychain Add, Update, Remove and Find operations using one file	698
<b>Chapter 150: Handle Multiple Environment using Macro</b>	701
Section 150.1: Handle multiple environment using multiple target and macro	701
<b>Chapter 151: Set View Background</b>	716
Section 151.1: Fill background Image of a UIView	716
Section 151.2: Creating a gradient background view	716
Section 151.3: Set View backround with image	716
Section 151.4: Set View background	716
<b>Chapter 152: Block</b>	718
Section 152.1: Custom completion block for Custom Methods	718
Section 152.2: UIView Animations	718
Section 152.3: Modify captured variable	718
<b>Chapter 153: Content Hugging/Content Compression in Autolayout</b>	720
Section 153.1: Definition: Intrinsic Content Size	720
<b>Chapter 154: iOS Google Places API</b>	721
Section 154.1: Getting Nearby Places from Current Location	721
<b>Chapter 155: Navigation Bar</b>	723
Section 155.1: SWIFT Example	723
Section 155.2: Customize default navigation bar appearance	723
<b>Chapter 156: App wide operations</b>	724
Section 156.1: Get the top most UIViewController	724
Section 156.2: Intercept System Events	724
<b>Chapter 157: CoreImage Filters</b>	725
Section 157.1: Core Image Filter Example	725
<b>Chapter 158: Face Detection Using CoreImage/OpenCV</b>	731
Section 158.1: Face and Feature Detection	731
<b>Chapter 159: MPMediaPickerControllerDelegate</b>	734
Section 159.1: Load music with MPMediaPickerControllerDelegate and play it with AVAudioPlayer	734

<a href="#">第160章：图表（Coreplot）</a>	736
第160.1节：使用CorePlot制作图表	736
<a href="#">第161章：Swift中的FCM消息传递</a>	738
第161.1节：在Swift中初始化FCM	738
<a href="#">第162章：在iOS中创建自定义框架</a>	740
第162.1节：用Swift创建框架	740
<a href="#">第163章：自定义键盘</a>	741
第163.1节：自定义键盘示例	741
<a href="#">第164章：AirDrop</a>	748
第164.1节：AirDrop	748
<a href="#">第165章：SLComposeViewController</a>	749
第165.1节：用于Twitter、Facebook、新浪微博和腾讯微博的SLComposeViewController	749
<a href="#">第166章：iOS中的AirPrint教程</a>	751
第166.1节：AirPrint打印横幅文本	751
<a href="#">第167章：Carthage iOS设置</a>	753
第167.1节：Carthage安装Mac	753
<a href="#">第168章：Healthkit</a>	754
第168.1节：HealthKit	754
<a href="#">第169章：iOS中的Core_SpotLight</a>	757
第169.1节：Core-Spotlight	757
<a href="#">第170章：Core_Motion</a>	759
第170.1节：访问气压计以获取相对高度	759
<a href="#">第171章：二维码扫描器</a>	760
第171.1节：使用AVFoundation框架扫描二维码	760
第171.2节：UIViewController扫描二维码并显示视频输入	761
<a href="#">第172章：iOS的plist</a>	763
第172.1节：示例：	763
第172.2节：从Plist保存和编辑/删除数据	765
<a href="#">第173章：WCSessionDelegate</a>	767
第173.1节：手表套件控制器（WKInterfaceController）	767
<a href="#">第174章：应用代理（AppDelegate）</a>	768
第174.1节：通过AppDelegate方法的应用所有状态	768
第174.2节：AppDelegate的角色：	769
第174.3节：打开指定URL的资源	769
第174.4节：处理本地和远程通知	769
<a href="#">第175章：应用提交流程</a>	771
第175.1节：设置配置描述文件	771
第175.2节：归档代码	771
第175.3节：导出IPA文件	771
第175.4节：使用Application Loader上传IPA文件	772
<a href="#">第176章：文件句柄</a>	774
第176.1节：从文档目录分块读取文件	774
<a href="#">第177章：基本文本文件输入输出</a>	776
第177.1节：从文档文件夹读取和写入	776
<a href="#">第178章：iOS 语音合成（TTS）</a>	778
第178.1节：文本转语音	778
<a href="#">第179章：MPVolumeView</a>	779

<a href="#">Chapter 160: Graph (Coreplot)</a>	736
Section 160.1: Making graphs with CorePlot	736
<a href="#">Chapter 161: FCM Messaging in Swift</a>	738
Section 161.1: Initialize FCM in Swift	738
<a href="#">Chapter 162: Create a Custom framework in iOS</a>	740
Section 162.1: Create Framework in Swift	740
<a href="#">Chapter 163: Custom Keyboard</a>	741
Section 163.1: Custom KeyBoard Example	741
<a href="#">Chapter 164: AirDrop</a>	748
Section 164.1: AirDrop	748
<a href="#">Chapter 165: SLComposeViewController</a>	749
Section 165.1: SLComposeViewController for Twitter, facebook, SinaWeibo and TencentWeibo	749
<a href="#">Chapter 166: AirPrint tutorial in iOS</a>	751
Section 166.1: AirPrint printing Banner Text	751
<a href="#">Chapter 167: Carthage iOS Setup</a>	753
Section 167.1: Carthage Installation Mac	753
<a href="#">Chapter 168: Healthkit</a>	754
Section 168.1: HealthKit	754
<a href="#">Chapter 169: Core_SpotLight in iOS</a>	757
Section 169.1: Core-Spotlight	757
<a href="#">Chapter 170: Core Motion</a>	759
Section 170.1: Accessing Barometer to get relative altitude	759
<a href="#">Chapter 171: QR Code Scanner</a>	760
Section 171.1: Scanning QR code with AVFoudation framework	760
Section 171.2: UIViewController scanning for QR and displaying video input	761
<a href="#">Chapter 172: plist iOS</a>	763
Section 172.1: Example:	763
Section 172.2: Save and edit/delete data from Plist	765
<a href="#">Chapter 173: WCSessionDelegate</a>	767
Section 173.1: Watch kit controller (WKInterfaceController)	767
<a href="#">Chapter 174: AppDelegate</a>	768
Section 174.1: All States of Application through AppDelegate methods	768
Section 174.2: AppDelegate Roles:	769
Section 174.3: Opening a URL-Specified Resource	769
Section 174.4: Handling Local and Remote Notifications	769
<a href="#">Chapter 175: App Submission Process</a>	771
Section 175.1: Setup provisioning profiles	771
Section 175.2: Archive the code	771
Section 175.3: Export IPA file	771
Section 175.4: Upload IPA file using Application Loader	772
<a href="#">Chapter 176: FileHandle</a>	774
Section 176.1: Read file from document directory in chunks	774
<a href="#">Chapter 177: Basic text file I/O</a>	776
Section 177.1: Read and write from Documents folder	776
<a href="#">Chapter 178: iOS TTS</a>	778
Section 178.1: Text To Speech	778
<a href="#">Chapter 179: MPVolumeView</a>	779

<a href="#">第179.1节：添加 MPVolumeView</a>	779	<a href="#">Section 179.1: Adding a MPVolumeView</a>	779
<a href="#"><b>第180章：Objective-C 关联对象</b></a>	780	<a href="#"><b>Chapter 180: Objective-C Associated Objects</b></a>	780
<a href="#">第180.1节：基本关联对象示例</a>	780	<a href="#">Section 180.1: Basic Associated Object Example</a>	780
<a href="#"><b>第181章：在视图控制器之间传递数据（带消息框概念）</b></a>	781	<a href="#"><b>Chapter 181: Passing Data between View Controllers (with MessageBox-Concept)</b></a>	781
<a href="#">第181.1节：简单示例用法</a>	781	<a href="#">Section 181.1: Simple Example Usage</a>	781
<a href="#"><b>第182章：MVVM</b></a>	782	<a href="#"><b>Chapter 182: MVVM</b></a>	782
<a href="#">第182.1节：无响应式编程的MVVM</a>	782	<a href="#">Section 182.1: MVVM Without Reactive Programming</a>	782
<a href="#"><b>第183章：缓存在线图片</b></a>	785	<a href="#"><b>Chapter 183: Cache online images</b></a>	785
<a href="#">第183.1节：AlamofireImage</a>	785	<a href="#">Section 183.1: AlamofireImage</a>	785
<a href="#"><b>第184章：队列中的链式块（使用MKBlockQueue）</b></a>	786	<a href="#"><b>Chapter 184: Chain Blocks in a Queue (with MKBlockQueue)</b></a>	786
<a href="#">第184.1节：示例代码</a>	786	<a href="#">Section 184.1: Example Code</a>	786
<a href="#"><b>第185章：模拟器</b></a>	788	<a href="#"><b>Chapter 185: Simulator</b></a>	788
<a href="#">第185.1节：启动模拟器</a>	788	<a href="#">Section 185.1: Launching Simulator</a>	788
<a href="#">第185.2节：3D / 力触控模拟</a>	788	<a href="#">Section 185.2: 3D / Force Touch simulation</a>	788
<a href="#">第185.3节：更改设备型号</a>	789	<a href="#">Section 185.3: Change Device Model</a>	789
<a href="#">第185.4节：模拟器导航</a>	790	<a href="#">Section 185.4: Navigating Simulator</a>	790
<a href="#"><b>第186章：后台模式</b></a>	791	<a href="#"><b>Chapter 186: Background Modes</b></a>	791
<a href="#">第186.1节：开启后台模式功能</a>	791	<a href="#">Section 186.1: Turning on the Background Modes capability</a>	791
<a href="#">第186.2节：后台获取</a>	791	<a href="#">Section 186.2: Background Fetch</a>	791
<a href="#">第186.3节：测试后台获取</a>	792	<a href="#">Section 186.3: Testing background fetch</a>	792
<a href="#">第186.4节：后台音频</a>	794	<a href="#">Section 186.4: Background Audio</a>	794
<a href="#"><b>第187章：OpenGL</b></a>	795	<a href="#"><b>Chapter 187: OpenGL</b></a>	795
<a href="#">第187.1节：示例项目</a>	795	<a href="#">Section 187.1: Example Project</a>	795
<a href="#"><b>第188章：MVP架构</b></a>	796	<a href="#"><b>Chapter 188: MVP Architecture</b></a>	796
<a href="#">第188.1节：Dog.swift</a>	797	<a href="#">Section 188.1: Dog.swift</a>	797
<a href="#">第188.2节：DoggyService.swift</a>	797	<a href="#">Section 188.2: DoggyService.swift</a>	797
<a href="#">第188.3节：DoggyPresenter.swift</a>	797	<a href="#">Section 188.3: DoggyPresenter.swift</a>	797
<a href="#">第188.4节：DoggyView.swift</a>	798	<a href="#">Section 188.4: DoggyView.swift</a>	798
<a href="#">第188.5节：DoggyListViewController.swift</a>	798	<a href="#">Section 188.5: DoggyListViewController.swift</a>	798
<a href="#"><b>第189章：使用CoreBluetooth配置信标</b></a>	800	<a href="#"><b>Chapter 189: Configure Beacons with CoreBluetooth</b></a>	800
<a href="#">第189.1节：显示所有蓝牙低功耗（BLE）名称</a>	800	<a href="#">Section 189.1: Showing names of all Bluetooth Low Energy (BLE)</a>	800
<a href="#">第189.2节：连接并读取主值</a>	801	<a href="#">Section 189.2: Connect and read major value</a>	801
<a href="#">第189.3节：写入主值</a>	802	<a href="#">Section 189.3: Write major value</a>	802
<a href="#"><b>第190章：核心数据</b></a>	804	<a href="#"><b>Chapter 190: Core Data</b></a>	804
<a href="#">第190.1节：核心数据操作</a>	804	<a href="#">Section 190.1: Operations on core data</a>	804
<a href="#"><b>第191章：带仪器的配置文件</b></a>	805	<a href="#"><b>Chapter 191: Profile with Instruments</b></a>	805
<a href="#">第191.1节：时间分析器</a>	805	<a href="#">Section 191.1: Time Profiler</a>	805
<a href="#"><b>第192章：应用评分/审核请求</b></a>	814	<a href="#"><b>Chapter 192: Application rating/review request</b></a>	814
<a href="#">第192.1节：评分/审核iOS应用</a>	814	<a href="#">Section 192.1: Rate/Review iOS Application</a>	814
<a href="#"><b>第193章：MyLayout布局</b></a>	815	<a href="#"><b>Chapter 193: MyLayout</b></a>	815
<a href="#">第193.1节：使用MyLayout的简单演示</a>	815	<a href="#">Section 193.1: A Simple Demo to use MyLayout</a>	815
<a href="#"><b>第194章：模拟器构建</b></a>	817	<a href="#"><b>Chapter 194: Simulator Builds</b></a>	817
<a href="#">第194.1节：在模拟器上手动安装构建</a>	817	<a href="#">Section 194.1: Installing the build manually on simulator</a>	817
<a href="#"><b>第195章：使用GPX文件模拟iOS位置</b></a>	818	<a href="#"><b>Chapter 195: Simulating Location Using GPX files iOS</b></a>	818
<a href="#">第195.1节：您的.gpx文件：MPS_HQ.gpx</a>	818	<a href="#">Section 195.1: Your .gpx file: MPS_HQ.gpx</a>	818
<a href="#">第195.2节：设置此位置：</a>	818	<a href="#">Section 195.2: To set this location:</a>	818
<a href="#"><b>第196章：SqlCipher集成</b></a>	820	<a href="#"><b>Chapter 196: SqlCipher integration</b></a>	820

<a href="#">第196.1节：代码集成：</a>	820
<b>第197章：安全性</b>	821
<a href="#">第197.1节：保护iTunes备份中的数据</a>	821
<a href="#">第197.2节：使用SSL的传输安全</a>	822
<b>第198章：应用传输安全（ATS）</b>	824
<a href="#">第198.1节：加载所有HTTP内容</a>	824
<a href="#">第198.2节：选择性加载HTTP内容</a>	824
<a href="#">第198.3节：端点要求SSL</a>	825
<b>第199章：选择最佳iOS架构模式的指南</b>	826
<a href="#">第199.1节：MVP模式</a>	826
<a href="#">第199.2节：MVVM模式</a>	827
<a href="#">第199.3节：VIPER模式</a>	828
<a href="#">第199.4节：MVC模式</a>	829
<b>第200章：多播代理</b>	830
<a href="#">第200.1节：适用于任何控件的多播代理</a>	830
<b>第201章：使用图像资源</b>	834
<a href="#">第201.1节：使用图像资源的启动图像</a>	834
<a href="#">第201.2节：使用图像资源的应用图标</a>	837
<b>第202章：Objective-C中的运行时</b>	842
<a href="#">第202.1节：使用关联对象</a>	842
<b>第203章：模态展示样式</b>	844
<a href="#">第203.1节：使用界面构建器探索模态展示样式</a>	844
<b>第204章：CydiaSubstrate调整</b>	853
<a href="#">第204.1节：使用Theos创建新调整</a>	853
<b>第205章：从图像创建视频</b>	855
<a href="#">第205.1节：从UIImages创建视频</a>	855
<b>第206章：可编码（ Codable）</b>	857
<a href="#">第206.1节：Swift 4中使用 Codable 与 JSONEncoder 和 JSONDecoder</a>	857
<b>第207章：异步加载图像</b>	859
<a href="#">第207.1节：最简单的方法</a>	859
<a href="#">第207.2节：下载后检查单元格是否仍然可见</a>	859
<b>第208章：添加Swift桥接头文件</b>	860
<a href="#">第208.1节：如何手动创建Swift桥接头文件</a>	860
<a href="#">第208.2节：Xcode自动创建</a>	860
<b>第209章：创建应用程序ID</b>	862
<a href="#">第209.1节：创建应用内购买产品</a>	862
<a href="#">第209.2节：创建沙盒用户</a>	864
<b>第210章：Swift：在AppDelegate中更改rootViewController以呈现主界面或登录/引导流程</b>	865
<a href="#">第210.1节：选项1：替换根视图控制器（良好）</a>	865
<a href="#">第210.2节：以模态方式呈现替代流程（更佳）</a>	865
<b>鸣谢</b>	867
<b>你可能也喜欢</b>	874

<a href="#">Section 196.1: Integration of code:</a>	820
<b>Chapter 197: Security</b>	821
<a href="#">Section 197.1: Securing Data in iTunes Backups</a>	821
<a href="#">Section 197.2: Transport Security using SSL</a>	822
<b>Chapter 198: App Transport Security (ATS)</b>	824
<a href="#">Section 198.1: Load all HTTP content</a>	824
<a href="#">Section 198.2: Selectively load HTTP content</a>	824
<a href="#">Section 198.3: Endpoints require SSL</a>	825
<b>Chapter 199: Guideline to choose best iOS Architecture Patterns</b>	826
<a href="#">Section 199.1: MVP Patterns</a>	826
<a href="#">Section 199.2: MVVM Pattern</a>	827
<a href="#">Section 199.3: VIPER Pattern</a>	828
<a href="#">Section 199.4: MVC pattern</a>	829
<b>Chapter 200: Multicast Delegates</b>	830
<a href="#">Section 200.1: Multicast delegates for any controls</a>	830
<b>Chapter 201: Using Image Assets</b>	834
<a href="#">Section 201.1: LaunchImage using Image Assets</a>	834
<a href="#">Section 201.2: App Icon using Image Assets</a>	837
<b>Chapter 202: Runtime in Objective-C</b>	842
<a href="#">Section 202.1: Using Associated Objects</a>	842
<b>Chapter 203: ModelPresentationStyles</b>	844
<a href="#">Section 203.1: Exploring ModalPresentationStyle using Interface Builder</a>	844
<b>Chapter 204: CydiaSubstrate tweak</b>	853
<a href="#">Section 204.1: Create new tweak using Theos</a>	853
<b>Chapter 205: Create a video from images</b>	855
<a href="#">Section 205.1: Create Video from UIImages</a>	855
<b>Chapter 206: Codable</b>	857
<a href="#">Section 206.1: Use of Codable with JSONEncoder and JSONDecoder in Swift 4</a>	857
<b>Chapter 207: Load images async</b>	859
<a href="#">Section 207.1: Easiest way</a>	859
<a href="#">Section 207.2: Check that the cell is still visible after download</a>	859
<b>Chapter 208: Adding a Swift Bridging Header</b>	860
<a href="#">Section 208.1: How to create a Swift Bridging Header Manually</a>	860
<a href="#">Section 208.2: Xcode create automatically</a>	860
<b>Chapter 209: Creating an App ID</b>	862
<a href="#">Section 209.1: Creating In-App Purchase Products</a>	862
<a href="#">Section 209.2: Creating a Sandbox User</a>	864
<b>Chapter 210: Swift: Changing the rootViewController in AppDelegate to present main or login/onboarding flow</b>	865
<a href="#">Section 210.1: Option 1: Swap the Root View Controller (Good)</a>	865
<a href="#">Section 210.2: Option 2: Present Alternative Flow Modally (Better)</a>	865
<b>Credits</b>	867
<b>You may also like</b>	874

欢迎随意免费分享此PDF，  
本书的最新版本可从以下网址下载：  
<https://goalkicker.com/iOSBook>

本*iOS®开发者专业笔记*一书汇编自[Stack Overflow Documentation](#)，内容由Stack Overflow的优秀贡献者撰写。  
文本内容采用知识共享署名-相同方式共享许可协议发布，详见本书末尾对各章节贡献者的致谢。图片版权归各自所有者所有，除非另有说明。

本书为非官方免费书籍，旨在教育用途，与官方*iOS®开发者*团体或公司及Stack Overflow无关。所有商标及注册商标均为其各自公司所有。

本书所提供的信息不保证正确或准确，使用风险自负。

请将反馈和更正发送至[web@petercv.com](mailto:web@petercv.com)

Please feel free to share this PDF with anyone for free,  
latest version of this book can be downloaded from:  
<https://goalkicker.com/iOSBook>

This *iOS® Developer Notes for Professionals* book is compiled from [Stack Overflow Documentation](#), the content is written by the beautiful people at Stack Overflow. Text content is released under Creative Commons BY-SA, see credits at the end of this book whom contributed to the various chapters. Images may be copyright of their respective owners unless otherwise specified

This is an unofficial free book created for educational purposes and is not affiliated with official *iOS® Developer* group(s) or company(s) nor Stack Overflow. All trademarks and registered trademarks are the property of their respective company owners

The information presented in this book is not guaranteed to be correct nor accurate, use at your own risk

Please send feedback and corrections to [web@petercv.com](mailto:web@petercv.com)

# 第1章：iOS开发入门

版本	发布日期
<a href="#">iPhone 操作系统 2</a>	2008-07-11
<a href="#">iPhone 操作系统 3</a>	2009-06-17
<a href="#">iOS 4</a>	2010-06-08
<a href="#">iOS 5</a>	2011-10-12
<a href="#">iOS 6</a>	2012-09-19
<a href="#">iOS 7</a>	2013-09-18
<a href="#">iOS 8</a>	2014-09-17
<a href="#">iOS 8.1</a>	2014-10-20
<a href="#">iOS 8.2</a>	2015-03-09
<a href="#">iOS 8.3</a>	2015-04-08
<a href="#">iOS 8.4</a>	2015-06-30
<a href="#">iOS 9</a>	2015-09-16
<a href="#">iOS 9.1</a>	2015-10-22
<a href="#">iOS 9.2</a>	2015-12-08
<a href="#">iOS 9.3</a>	2016-03-21
<a href="#">iOS 10.0.1</a>	2016-09-13
<a href="#">iOS 10.1</a>	2016-10-24
<a href="#">iOS 10.2</a>	2016-12-12
<a href="#">iOS 10.2.1</a>	2017-01-23
<a href="#">iOS 10.3</a>	2017-03-27
<a href="#">iOS 10.3.3</a>	2017-07-19
<a href="#">iOS 11.0.0</a>	2017-09-19
<a href="#">iOS 11.2.0</a>	2017-12-02
<a href="#">iOS 11.3.0</a>	2018-03-29

## 第1.1节：创建默认的单视图应用程序

要开发iOS应用程序，您应该从一个名为Xcode的应用程序开始。您也可以使用其他替代工具，但Xcode是苹果官方的工具。请注意，它只能在macOS上运行。最新的官方版本是Xcode 8.3.3，Xcode 9（目前处于测试版）预计将在今年晚些时候发布。

- 启动您的Mac，如果尚未安装，请从App Store安装Xcode。

（如果您不想使用App Store或遇到问题，也可以从苹果开发者网站下载Xcode，但请确保选择最新的正式版本，不要选择测试版。）

## Chapter 1: Getting started with iOS Development

Version	Release Date
<a href="#">iPhone OS 2</a>	2008-07-11
<a href="#">iPhone OS 3</a>	2009-06-17
<a href="#">iOS 4</a>	2010-06-08
<a href="#">iOS 5</a>	2011-10-12
<a href="#">iOS 6</a>	2012-09-19
<a href="#">iOS 7</a>	2013-09-18
<a href="#">iOS 8</a>	2014-09-17
<a href="#">iOS 8.1</a>	2014-10-20
<a href="#">iOS 8.2</a>	2015-03-09
<a href="#">iOS 8.3</a>	2015-04-08
<a href="#">iOS 8.4</a>	2015-06-30
<a href="#">iOS 9</a>	2015-09-16
<a href="#">iOS 9.1</a>	2015-10-22
<a href="#">iOS 9.2</a>	2015-12-08
<a href="#">iOS 9.3</a>	2016-03-21
<a href="#">iOS 10.0.1</a>	2016-09-13
<a href="#">iOS 10.1</a>	2016-10-24
<a href="#">iOS 10.2</a>	2016-12-12
<a href="#">iOS 10.2.1</a>	2017-01-23
<a href="#">iOS 10.3</a>	2017-03-27
<a href="#">iOS 10.3.3</a>	2017-07-19
<a href="#">iOS 11.0.0</a>	2017-09-19
<a href="#">iOS 11.2.0</a>	2017-12-02
<a href="#">iOS 11.3.0</a>	2018-03-29

### Section 1.1: Creating a default Single View Application

To develop an application for iOS, you should start with an application called Xcode. There are other alternative tools you can use, but Xcode is Apple's official tool. Note, however, that it only runs on macOS. The latest official version is Xcode 8.3.3 with Xcode 9 (currently in beta) due to be released later this year.

- Boot up your Mac and install [Xcode from the App Store](#) if it's not already installed.

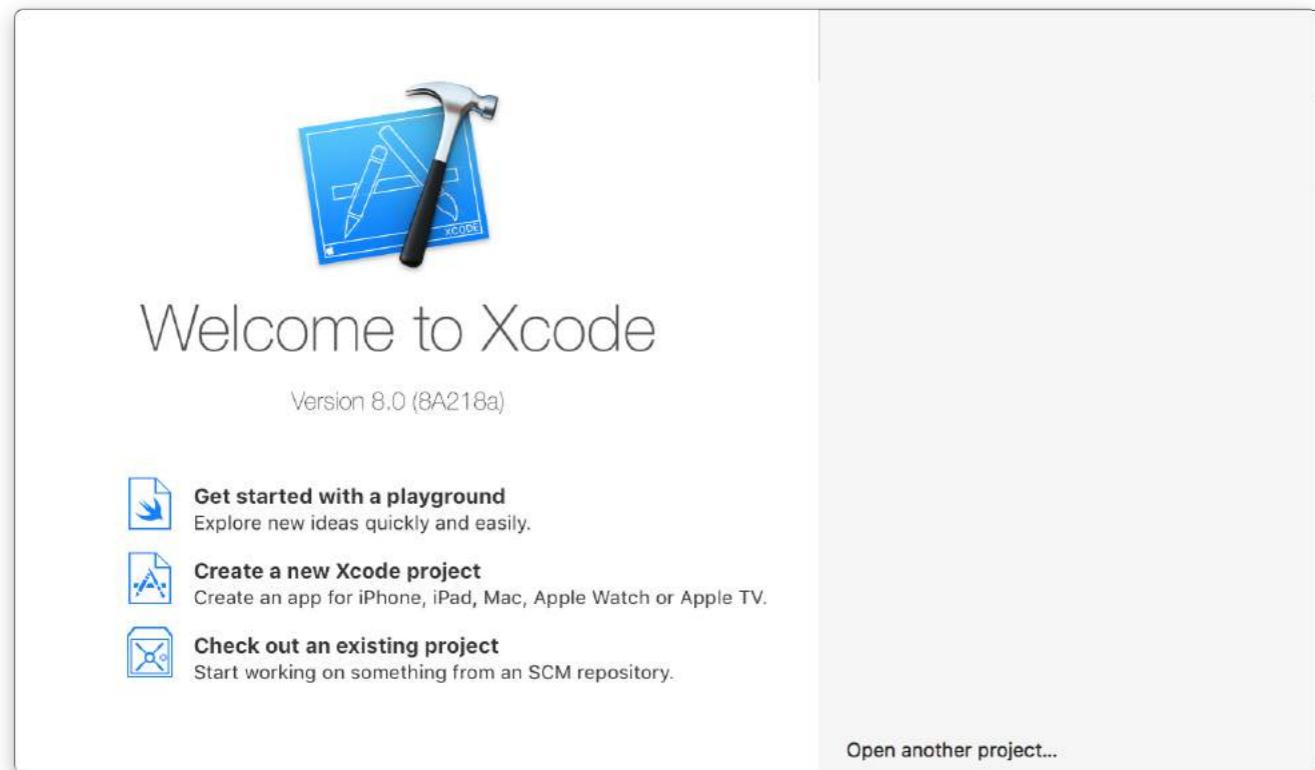
(If you prefer not to use the App Store or have problems, you can also [download Xcode from the Apple Developer website](#), but make sure that you select the latest release version and **not** a beta version.)



- 打开Xcode。将会打开以下窗口：



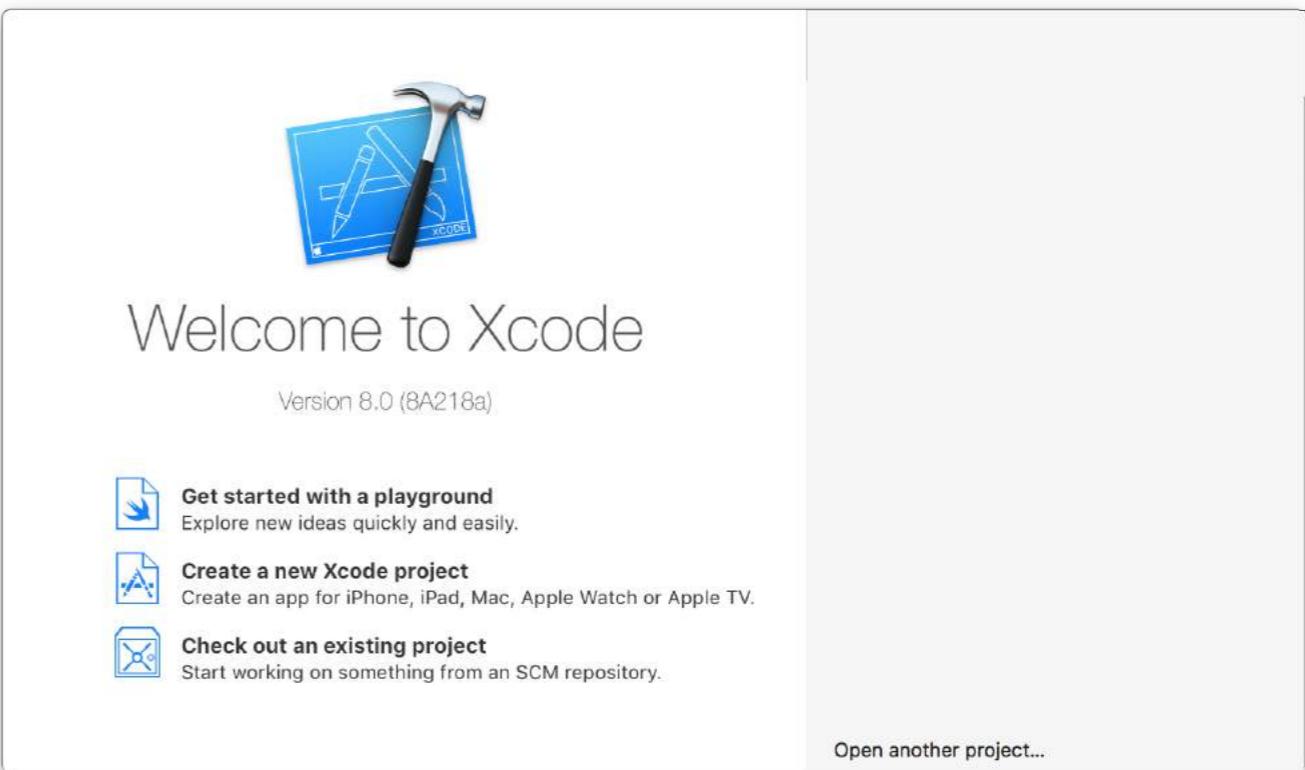
- Open Xcode. The following window will open:



该窗口为您提供以下选项：

- 开始使用游乐场：这是随Swift语言和Xcode 6引入的。它是一个交互式区域，可用于编写小段代码以检查运行时变化。这是Swift学习者了解新Swift特性的绝佳方式。
- 创建一个新的Xcode项目：选择此选项，将创建一个带有默认配置的新项目。
- 检现有项目：该选项用于从仓库位置检出项目，例如，从SVN检出项目。

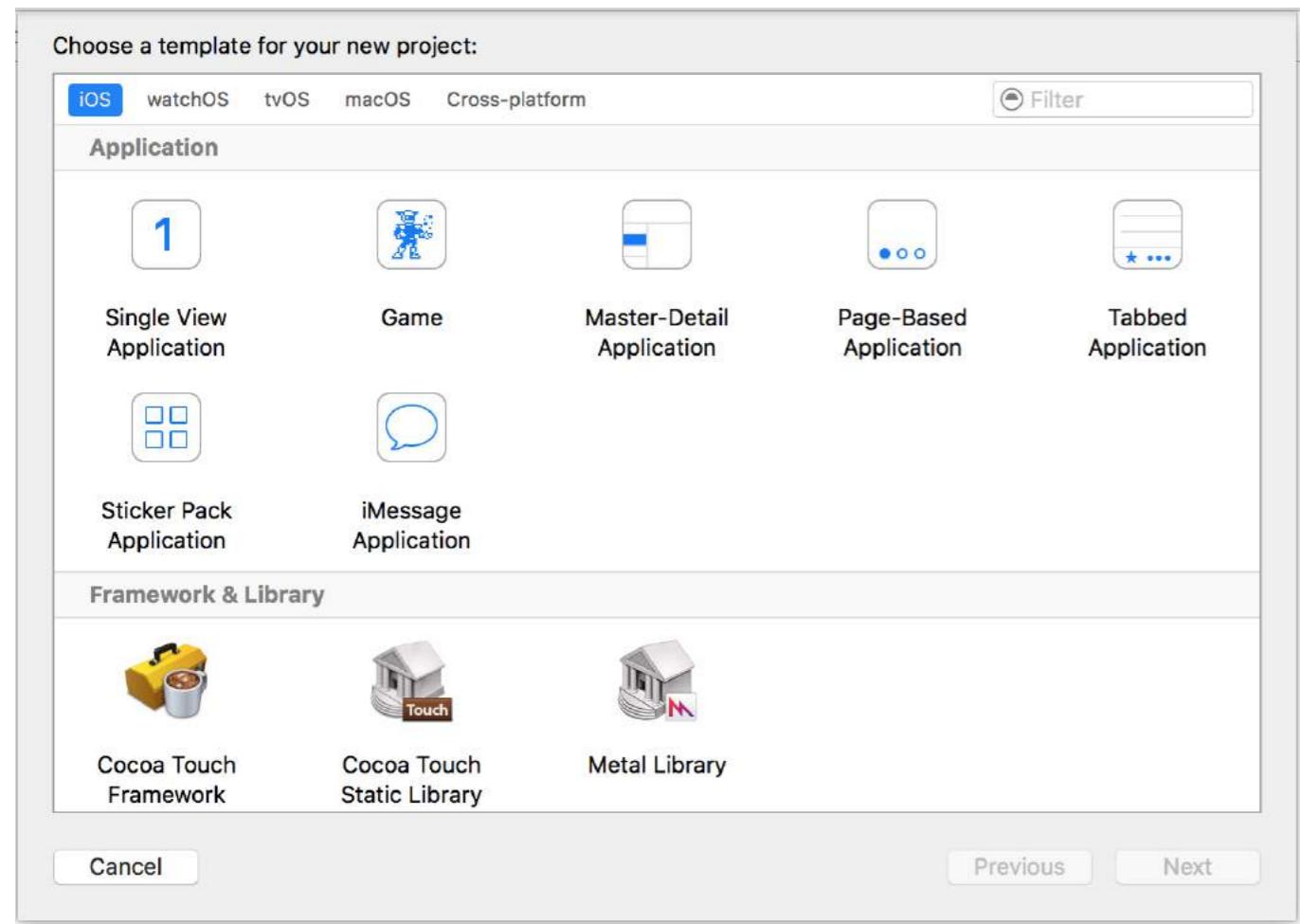
3. 选择第二个选项创建一个新的Xcode项目，Xcode将要求您进行一些初始项目设置：



The window presents you with the following options:

- **Getting started with a playground:** This was introduced with the Swift language and Xcode 6. It's an interactive area which can be used to write small pieces of code to check runtime changes. It's a great way for Swift learners to be introduced to new Swift features.
- **Create a new Xcode project:** Choose this option, which creates a new project with default configuration.
- **Check out an existing project:** This is used to check out a project from a repository location, for example, check out a project from SVN.

3. Select the second option **Create a new Xcode project** and Xcode will ask you to do some initial project setup:



此向导用于选择您的项目模板。有5个选项：

- iOS：用于创建iOS应用、库和框架
- watchOS：用于创建watchOS应用程序、库和框架
- tvOS：用于创建tvOS应用程序、库和框架
- macOS：用于创建macOS应用程序、库、框架、软件包、AppleScript等。
- 跨平台：用于创建跨平台应用程序、模板和应用内购买内容。

你可以看到有许多不同的应用程序模板。这些模板有助于提升你的开发效率；它们预先构建了一些基本的项目设置，如用户界面和类文件。

这里，我们将使用第一个选项，iOS。

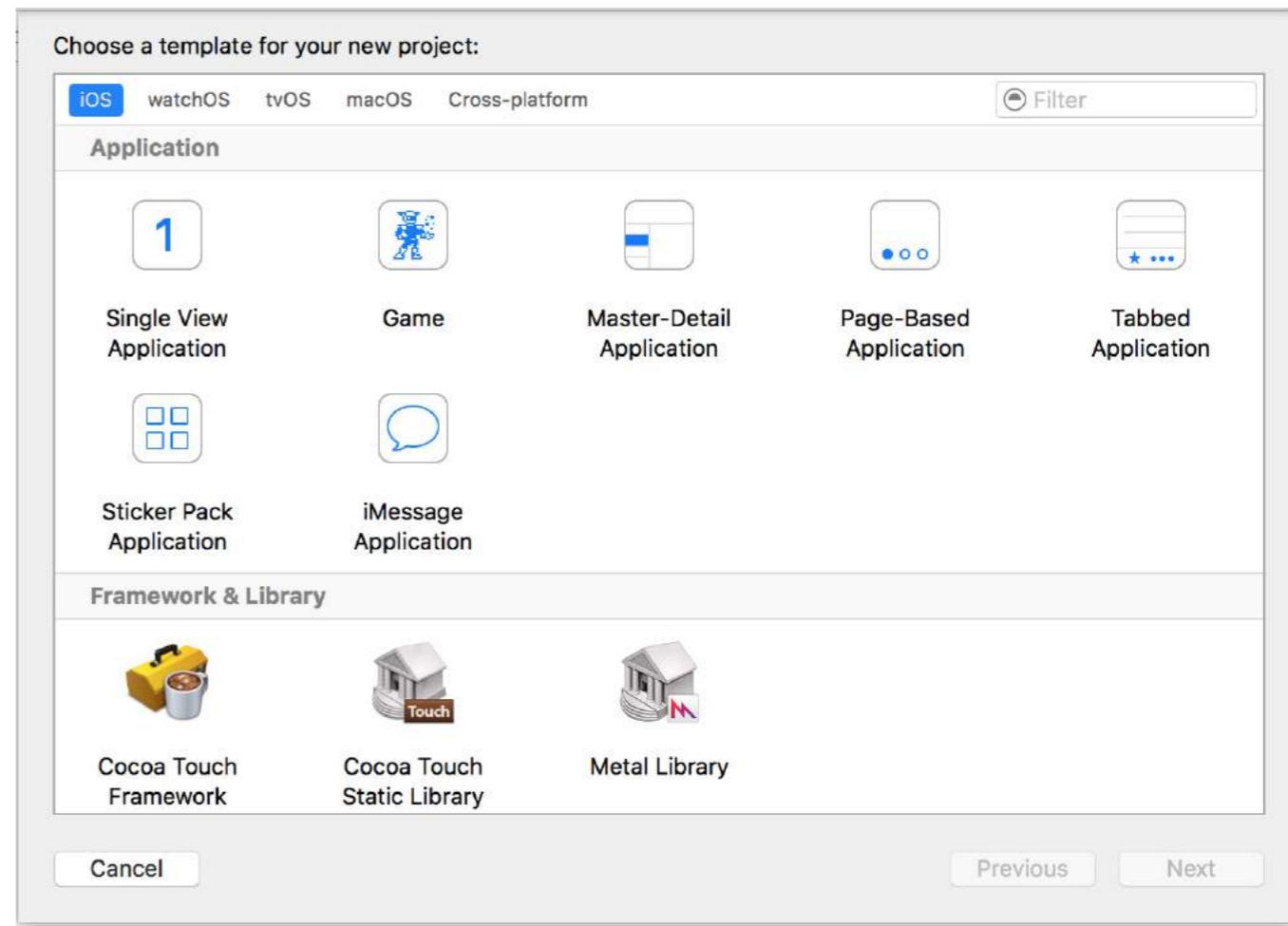
#### 1. 主从应用程序：

该模板包含一个组合的主界面和详细界面：主界面包含与详细界面相关的对象。选择主界面中的对象会改变详细界面。你可以在iPad上的“设置”、“备忘录”和“通讯录”应用中看到这种类型的界面。

#### 2. 基于页面的应用程序：

该模板用于创建基于页面的应用程序。页面是由一个容器持有的不同视图。

#### 3. 单视图应用程序：



This wizard is used to select your project template. There are 5 options:

- **iOS:** Used to create iOS apps, libraries and frameworks
- **watchOS:** Used to create watchOS apps, libraries and frameworks
- **tvOS:** Used to create tvOS apps, libraries and frameworks
- **macOS:** Used to create macOS apps, libraries, frameworks, packages, AppleScripts, etc.
- **Cross-platform:** Used to create cross-platform apps, templates and In-App Purchase Contents

You can see that there are many different templates for your application. These templates are helpful to boost your development; they are pre-built with some basic project setups like UI interfaces and class files.

Here, we'll use the first option, **iOS**.

#### 1. Master-Detail Application:

This template contains a combined master and detail interface: the master contains objects which are related to the detail interface. Selecting objects in the master will change the details interface. You can see this kind UI in the Settings, Notes and Contacts applications on the iPad.

#### 2. Page-Based Application:

This template is used to create the page-based application. Pages are different views held by one container.

#### 3. Single View Application:

这是一个普通应用程序开发模板。适合初学者学习应用程序流程。

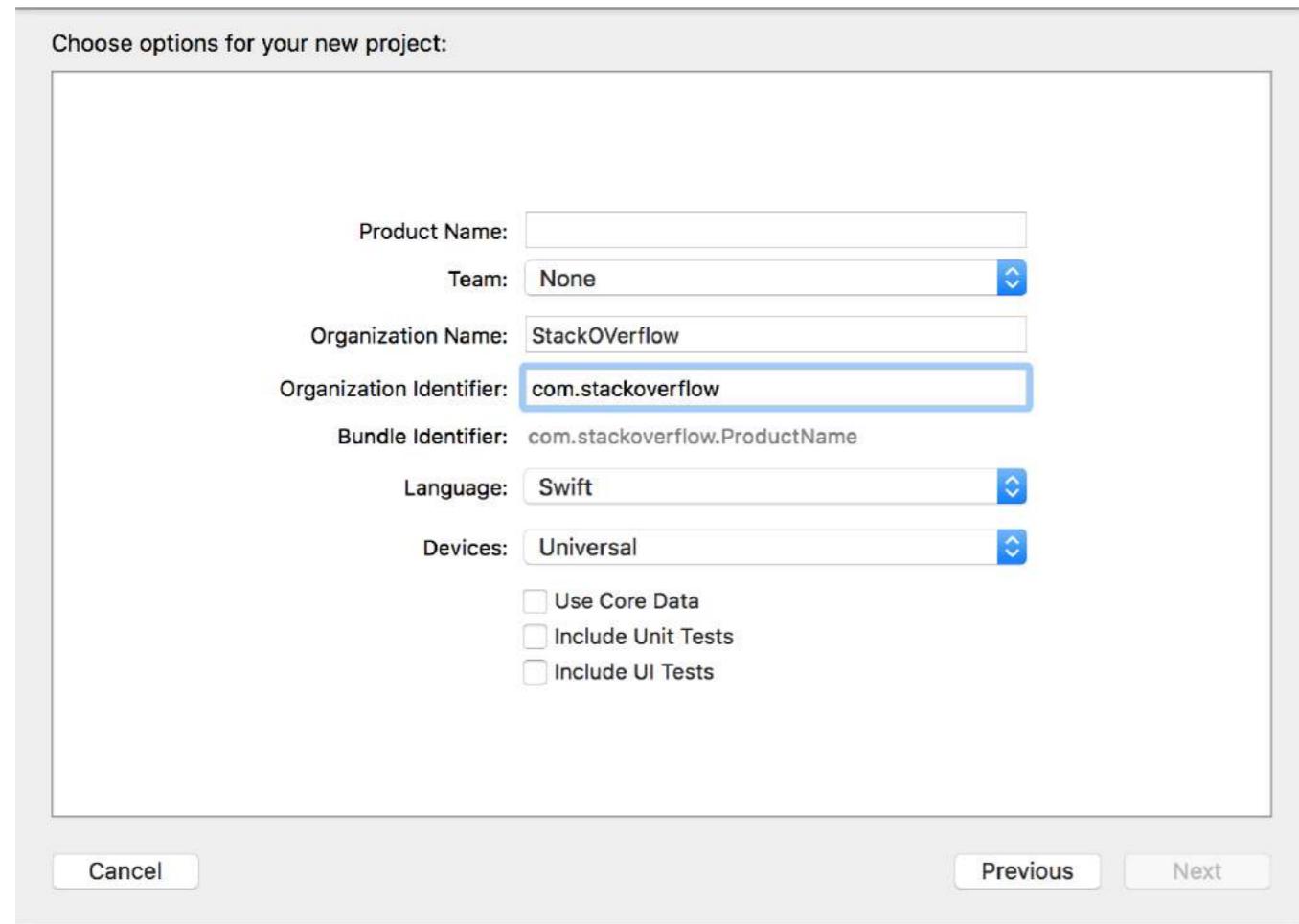
#### 4. 标签式应用程序：

该模板在应用程序底部创建标签。每个标签都有不同的用户界面和不同的导航流程。你可以在时钟、iTunes Store、iBooks 和 App Store 等应用中看到此模板的使用。

#### 5. 游戏：

这是游戏开发的起点。你可以进一步使用 SceneKit、SpriteKit、OpenGL ES 和 Metal 等游戏技术。

#### 4. 在本例中，我们将从单视图应用程序开始



向导将帮助你定义项目属性：

- 产品名称：项目/应用程序的名称
- 组织名称：你所在组织
- 的名称组织标识符：用于捆绑标识符的唯一组织标识符。建议遵循反向域名服务命名法。
- 捆绑标识符：此字段非常重要。它基于你的项目名称和组织标识符，请谨慎选择。捆绑标识符将用于将来在设备上安装应用程序以及上传应用到 iTunes Connect (这是我们上传应用以发布到 App Store 的地方)。它是识别你的应用程序的唯一键。

This is a normal application development template. This is good for beginners to learn application flow.

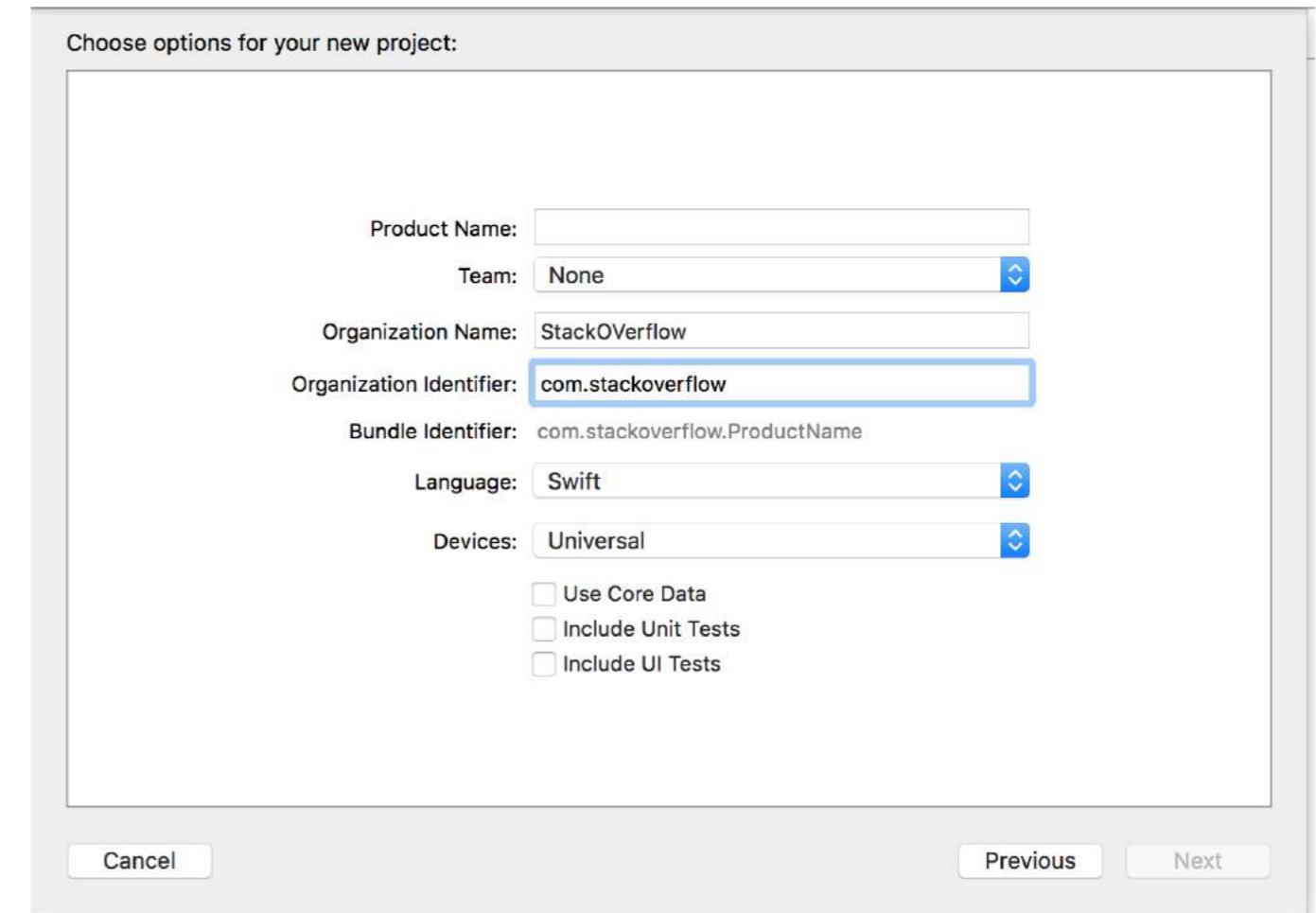
#### 4. Tabbed Application:

This template creates tabs at the bottom part of an application. Each tab has a different UI and a different navigation flow. You can see this template used in apps like Clock, iTunes Store, iBooks and App Store.

#### 5. Game:

This is a starting point for game development. You can go further with game technologies like SceneKit, SpriteKit, OpenGL ES and Metal.

#### 4. In this example, we will start with **Single View Application**



The wizard helps you to define project properties:

- **Product Name:** The name of the project / application
- **Organization Name:** The name of the organization in which you are involved
- **Organization Identifier:** The unique organization identifier which is used in the bundle identifier. It is recommended to follow reverse domain name service notation.
- **Bundle Identifier:** *This field is very important.* It is based on your project name and organization identifier, choose wisely. The bundle identifier will be used in the future to install the application on a device and upload the app to iTunes Connect (which is the place we upload apps to be published on the App Store). It's a unique key to identify your application.

- **语言**：您想使用的编程语言。在这里，如果未选择，可以将 Objective-C 更改为 Swift。
- **设备**：您的应用支持的设备，之后可以更改。显示有 iPhone、iPad 和通用。通用应用支持 iPhone 和 iPad 设备，建议在不需要仅在某一种设备上运行应用时选择此选项。
- **使用 Core Data**：如果您想在项目中使用 Core Data 模型，请勾选此项，它将为您创建一个 .xcdatamodel 文件。如果您事先不确定，也可以稍后添加此文件。
- **包含单元测试**：这将配置单元测试目标并创建用于单元测试的类
- **包含 UI 测试**：这将配置 UI 测试目标并创建用于 UI 测试的类

点击 下一步，系统会询问您想要创建项目目录的位置。

点击 创建，您将看到带有已定义项目设置的 Xcode 界面。您可以看到一些类和 Storyboard 文件。

这是一个单视图应用程序的基本模板。

在窗口左上角，确认已选择模拟器（例如这里显示的“iPhone 6”），然后按下三角形的运行按钮。



5. 一个新应用将打开一模拟器（首次运行可能需要一些时间，如果第一次出现错误，您可能需要尝试两次）。此应用为我们提供了已创建应用的设备模拟。它几乎看起来像真实设备！它包含一些类似真实设备的应用。您可以模拟方向、位置、摇晃手势、内存警告、通话状态栏、手指触摸、锁屏、重启、主屏等操作。

你将看到一个纯白色的应用程序，因为我们还没有对模板进行任何更改。

所以开始你自己的项目吧。这是一个漫长的旅程，等待你的有许多新的机会！

如果你不确定下一步该做什么，可以试试苹果的“Jump Right In”教程。你已经完成了最初的几步，所以已经领先一步了。

## 第1.2节：你好，世界

设置好Xcode后，启动你的第一个iOS应用并不难。

在下面的示例中，我们将：

- 启动一个新项目
- 添加一个标签
- 向控制台打印信息。
- 在模拟器中运行

### 启动一个新项目

当 Xcode 欢迎界面出现时，选择创建一个新的 Xcode 项目。或者，你也可以选择文件 >

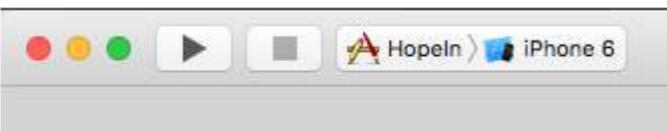
- **Language**: The programming language which you would like to use. Here you can change Objective-C to Swift if it's not selected.
- **Devices**: Supported devices for your application that can be changed later. It shows iPhone, iPad, and Universal. Universal applications support iPhone and iPad devices, and it's recommended to select this option when it's not necessary to run the app on only one kind of device.
- **Use Core Data**: If you would like to use Core Data Model in your project then mark it as selected, and it will create a file for the `.xcdatamodel`. You can also add this file later on if you don't know in advance.
- **Include Unit Tests**: This configures the unit test target and creates classes for unit testing
- **Include UI test**: This configures the UI test target and creates classes for UI testing

Click on **Next** and it will ask you for a location where you want to create project directory.

Click on **Create** and you will see the Xcode UI with an already defined project setup. You can see some classes and Storyboard files.

This is a basic template for a Single View Application.

At the top left of the window, check that a simulator is selected (e.g. "iPhone 6" as shown here) and then press the triangular RUN button.



5. A new application will open—Simulator (this may take some time the first time you run it and you may need to try twice if you see an error the first time). This application provides us with device simulation for created applications. It almost looks like a real device! It contains some applications like a real device. You can simulate orientations, location, shake gesture, memory warnings, In-Call Status bar, finger touch, lock, reboot, home etc.

You will see a plain white application because we have not made any changes to the template yet.

So start your own. it's a long run and there are lots of new opportunities waiting for you!

If you are not sure where to go next, try out Apple's '[Jump Right In](#)' tutorial. You have already performed the first few steps so are off to a head start.

## Section 1.2: Hello World

After setting up Xcode, it is not difficult to get your first iOS up and running. In the following example we will:

- Start a new project
- Add a label
- Printing message to console.
- Run in the simulator

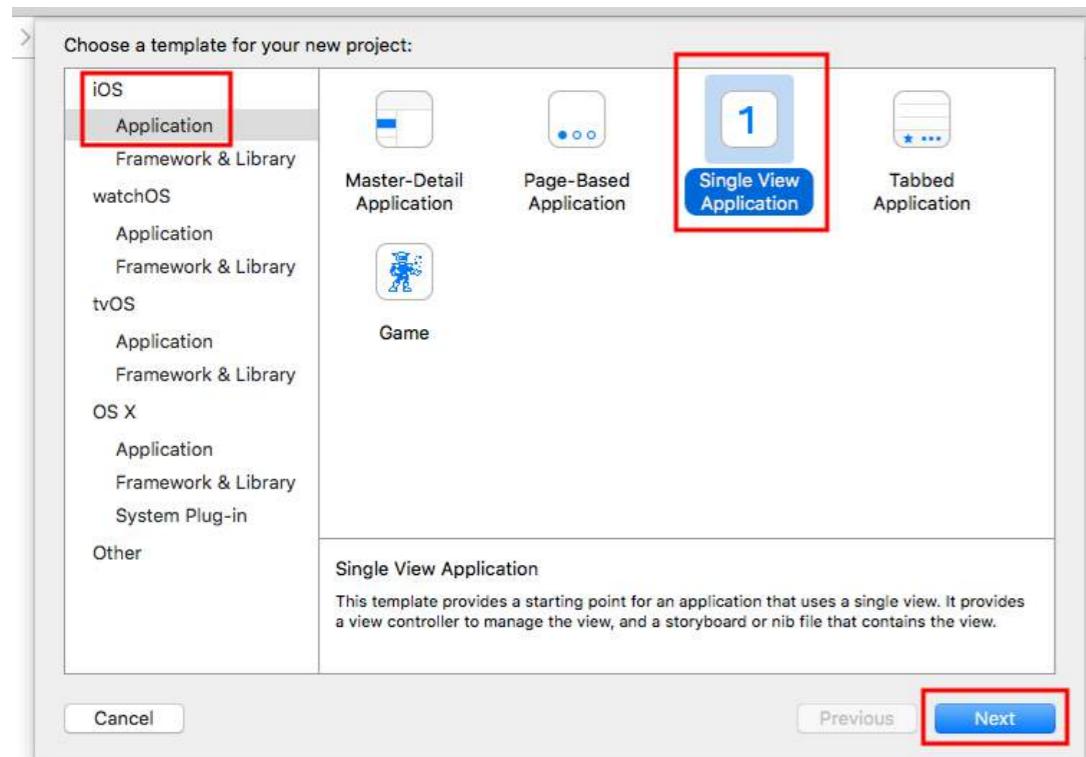
### Starting a new project

When the Xcode welcome screen comes up, choose **Create a new Xcode project**. Alternatively, you could do **File >**

如果你已经打开了Xcode菜单，选择“新建 > 项目...”。



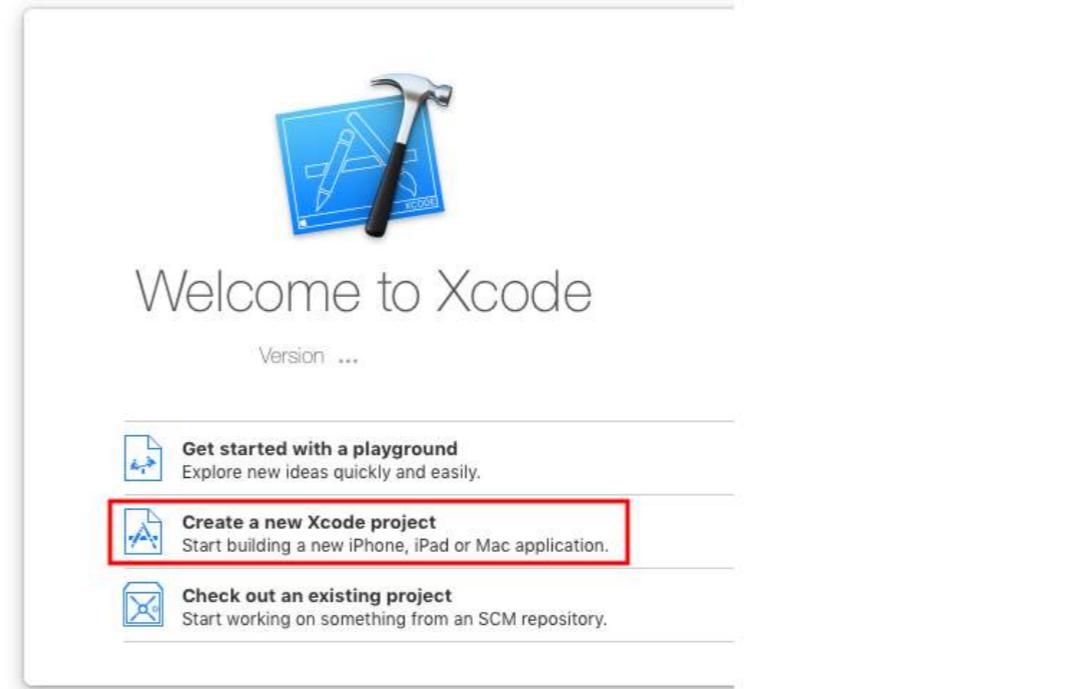
选择单视图应用程序，然后点击下一步。



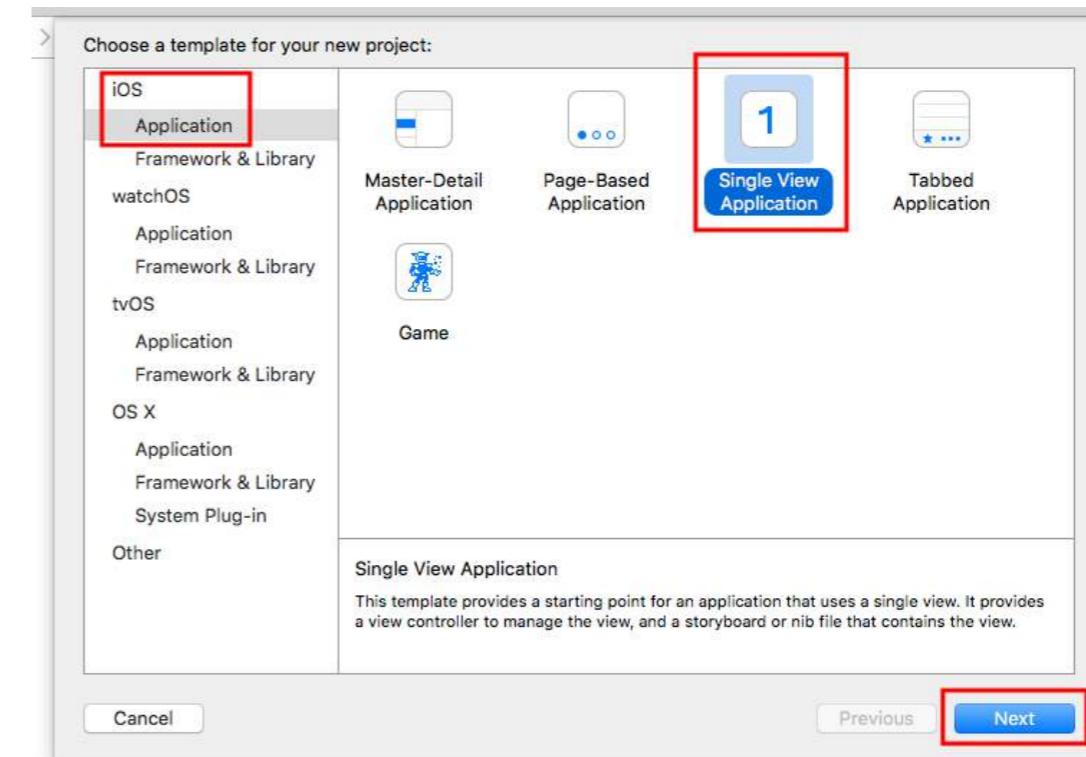
在产品名称中输入“HelloWorld”（或者你想要的名称），并确保在语言下选择了Swift。

- 通用意味着你的应用程序可以在iPhone和iPad上运行。
- 使用Core Data指的是持久化数据存储，我们的Hello World应用不需要这个功能。
- 在这个示例中我们不会进行单元测试或界面测试，但养成添加它们的习惯也无妨。

New > Project... from the Xcode menu if you already have it open.

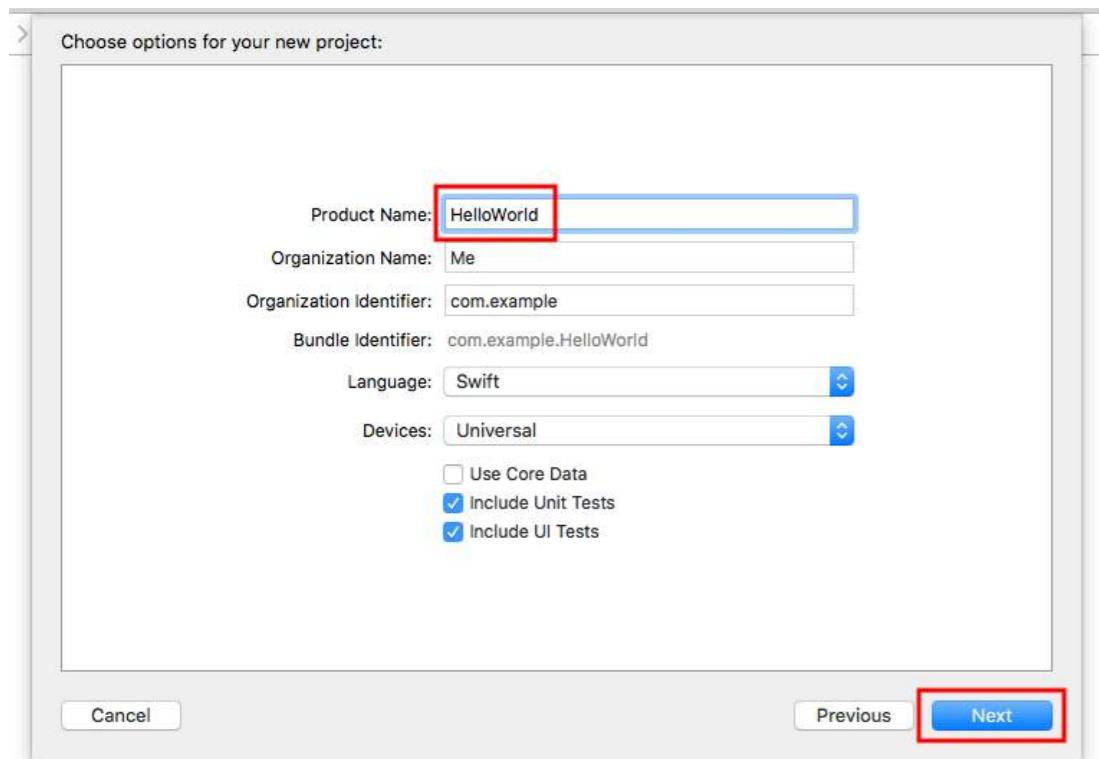


Choose a **Single View Application** and click **Next**.

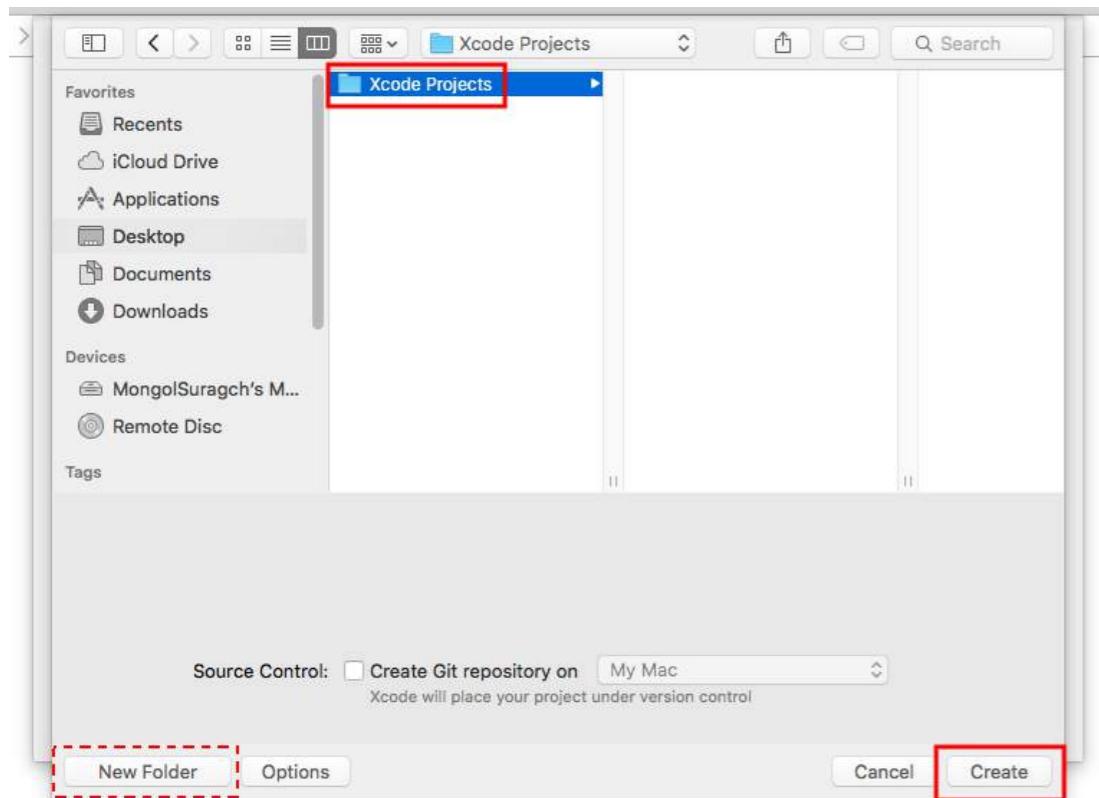


Write "HelloWorld" for the **Product Name** (or whatever you want really) and under **Language**, make sure **Swift** is selected.

- **Universal** means that your app will run on both the iPhone and iPad.
- **Use Core Data** refers to persistent data storage, which is not needed in our Hello World app.
- We will not be doing **Unit Tests** or **UI Tests** in this example, but it doesn't hurt to get into the habit of adding them.



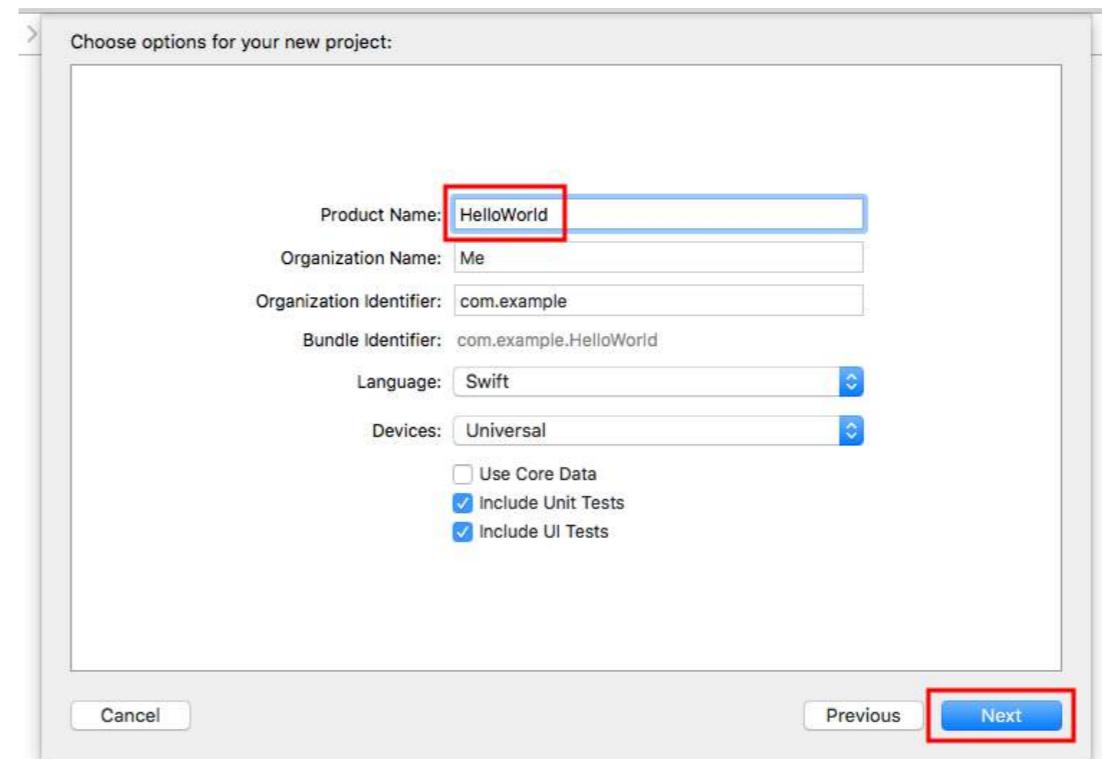
选择一个已有文件夹或创建一个新文件夹来保存你的Xcode项目。这个文件夹将作为未来的默认位置。我们这里创建了一个名为“Xcode Projects”的文件夹。然后点击创建。你可以选择源代码管理（用于同步到像GitHub这样的网站），但本例中我们不需要它。



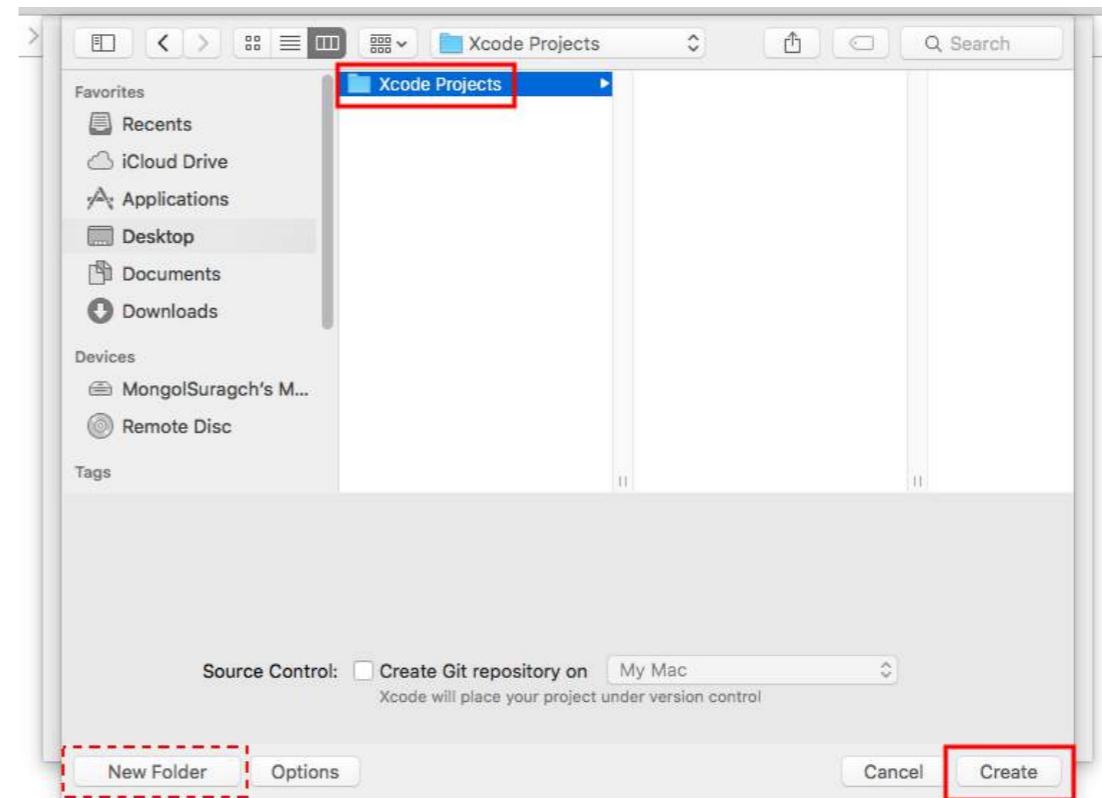
### 添加标签

这是一个Xcode项目的文件结构。

在项目导航器中选择Main.storyboard。



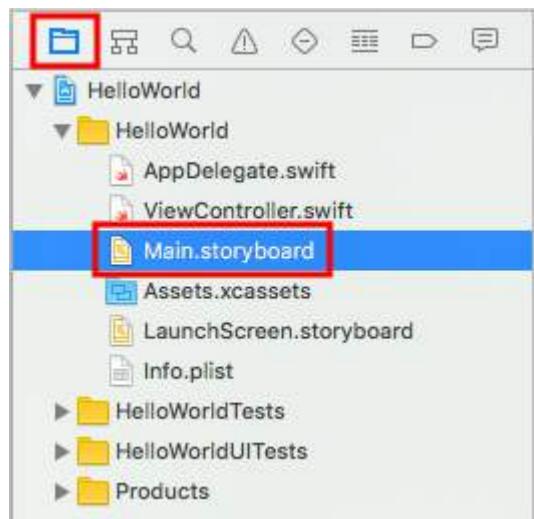
Choose an existing folder or create a new one where you will save your Xcode projects. This will be the default in the future. We created one here called "Xcode Projects". Then click **Create**. You can select Source Control if you like (used when syncing to sites like [GitHub](#)), but we won't be needing it in this example.



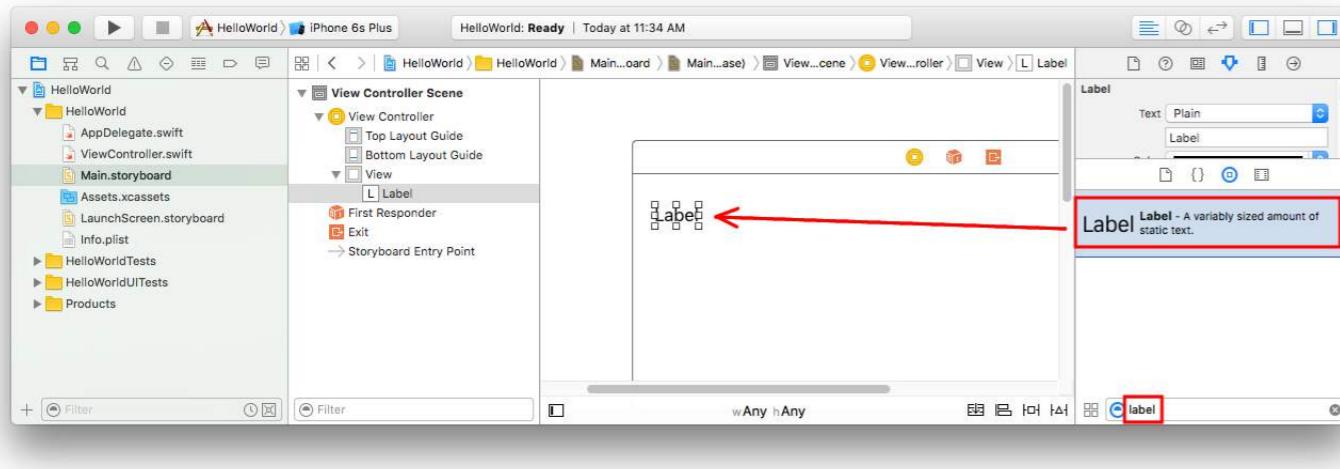
### Adding a label

This is the file structure of an Xcode project.

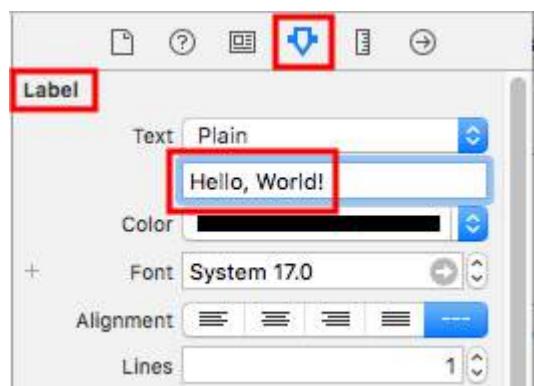
Select *Main.storyboard* in the Project Navigator.



在Xcode右下角的对象库搜索栏中输入“label”。然后将`UILabel`拖动到 storyboard视图控制器上。大致放置在左上角区域。



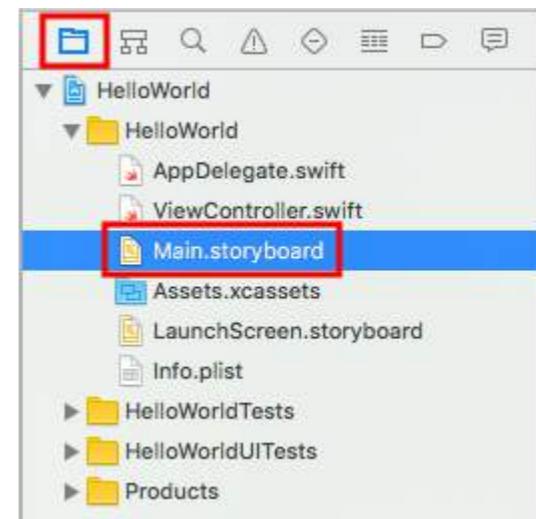
确保在storyboard上选中标签，然后在**属性检查器**中，将文本更改为“Hello, World!”。由于文本长度变长，您需要调整标签在storyboard上的大小和位置。



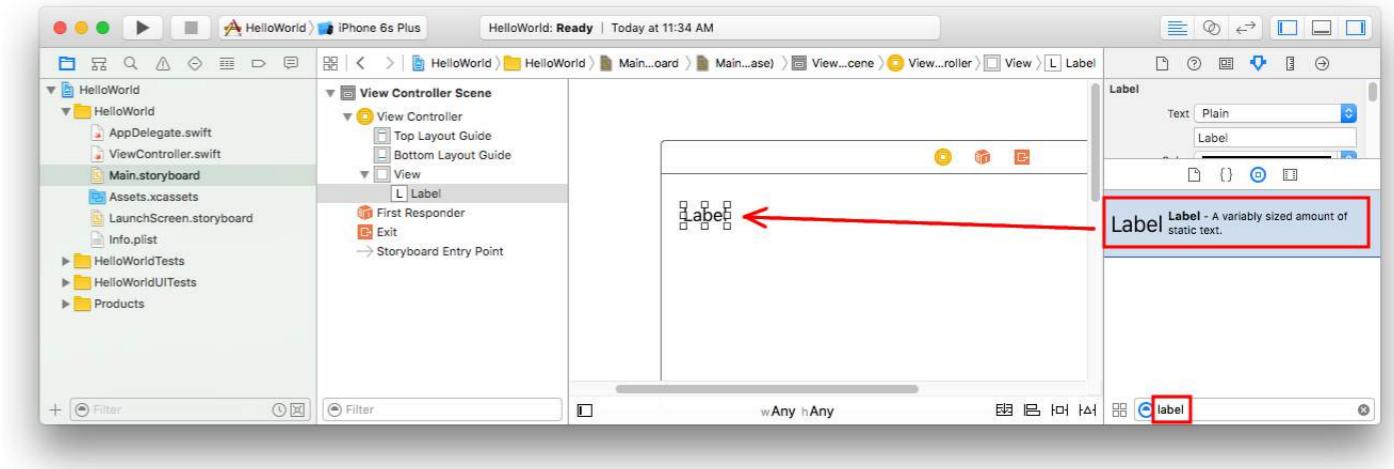
或者，双击storyboard上的标签，将其编辑为“Hello, World!”。无论哪种方式，storyboard 应类似如下所示：



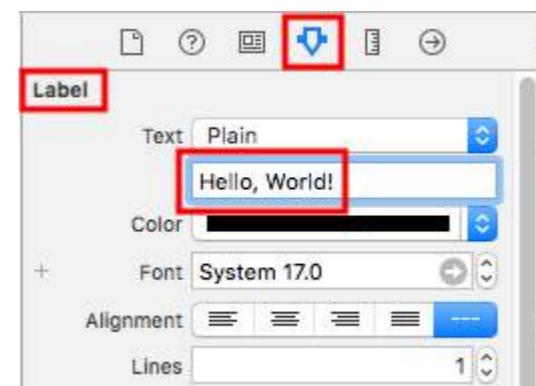
添加代码



Type "label" in the search field of the Object Library in the bottom right of Xcode. Then drag the `UILabel` onto the storyboard View Controller. Place it generally in the region of the top left corner.



Make sure the label is selected on the storyboard and then in the **Attributes Inspector**, change the text to "Hello, World!" You will then have to resize and reposition the label on the storyboard since the text length is longer now.

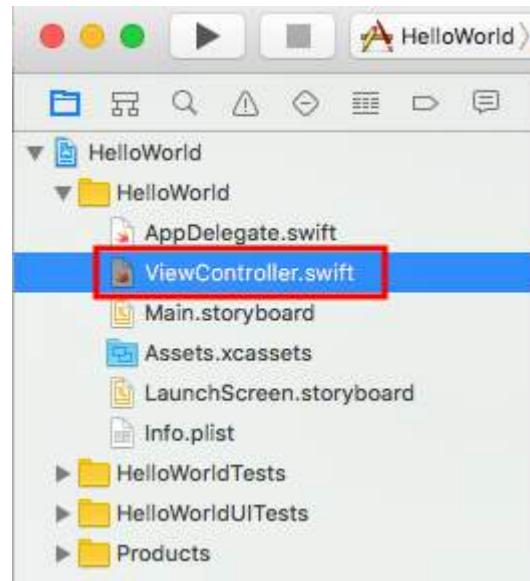


Alternatively, double-click the label on the storyboard to edit it to be "Hello, World!". At any rate, the storyboard should look something like this:



Adding Code

在项目导航器中选择ViewController.swift。



在viewDidLoad()方法中添加print("Successfully created my first iOS application.")。代码应类似如下。

```
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        // 当应用运行时打印到控制台
        print("成功创建了我的第一个iOS应用程序。")
    }

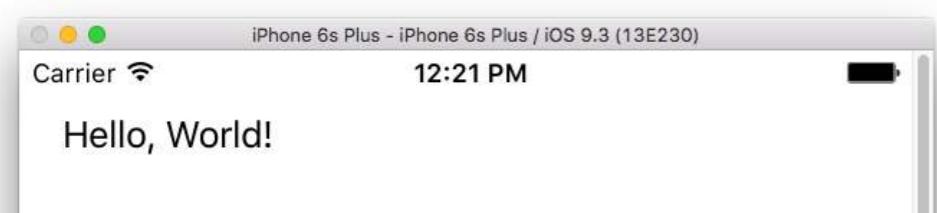
    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // 处理可以重新创建的资源。
    }
}
```

在模拟器中运行应用程序



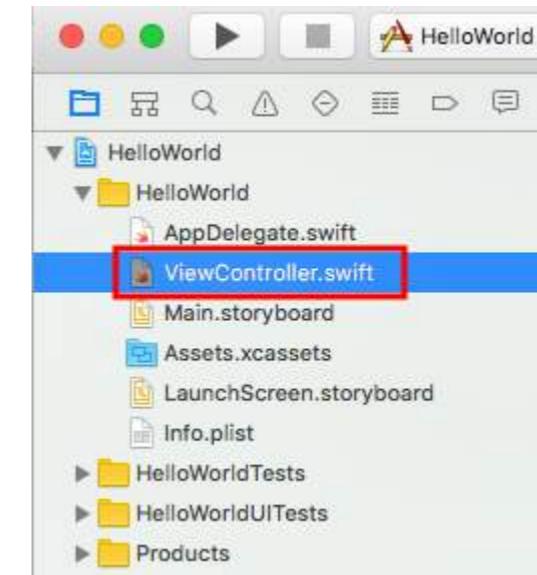
按下运行按钮以构建并运行应用程序。在此示例中，当前模拟器设备（称为“方案”）默认为iPhone 6s Plus。Xcode的较新版本将默认为较新的方案。您也可以通过点击名称选择其他方案。我们将保持默认方案。

模拟器首次运行时需要一些时间启动。启动后，界面应如下所示：



在模拟器菜单中，您可以选择窗口 > 缩放以缩小窗口，或按⌘cmd + 1/2/3/4/5以切换到100% /

Select ViewController.swift in the Project Navigator.



Add `print("Successfully created my first iOS application.")` to the viewDidLoad() method. It should look something like this.

```
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        // print to the console when app is run
        print("Successfully created my first iOS application.")
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
}
```

Running the app in the simulator



Press the Run button to build and run the app. In this example the current simulator device (referred to as a "scheme") defaulted to the iPhone 6s Plus. Newer versions of Xcode will default to newer schemes. You can also choose other schemes by clicking the name. We will just stick with the default.

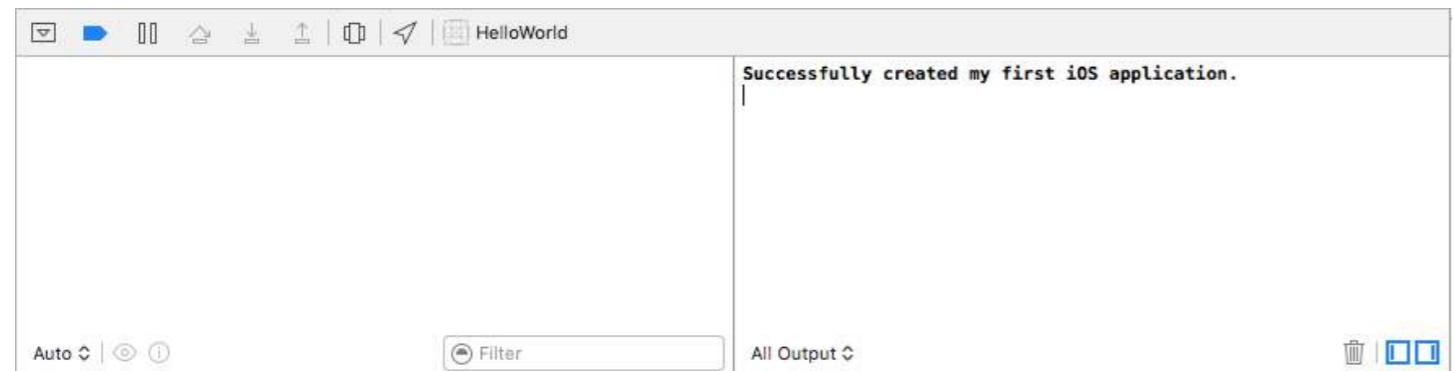
The simulator will take some time to start on the first run. Once running, it should look like this:



In the simulator menu, you can choose **Window > Scale** to make it smaller, or press ⌘cmd + 1/2/3/4/5 for 100% /

75% / 50% / 33% / 25% 比例分别为..

Xcode 调试区域（底部）也应该在控制台打印出“成功创建了我的第一个 iOS 应用程序。”这条信息。“成功创建了我的第一个 iOS 应用程序。”是你在添加代码部分程序化打印的字符串。

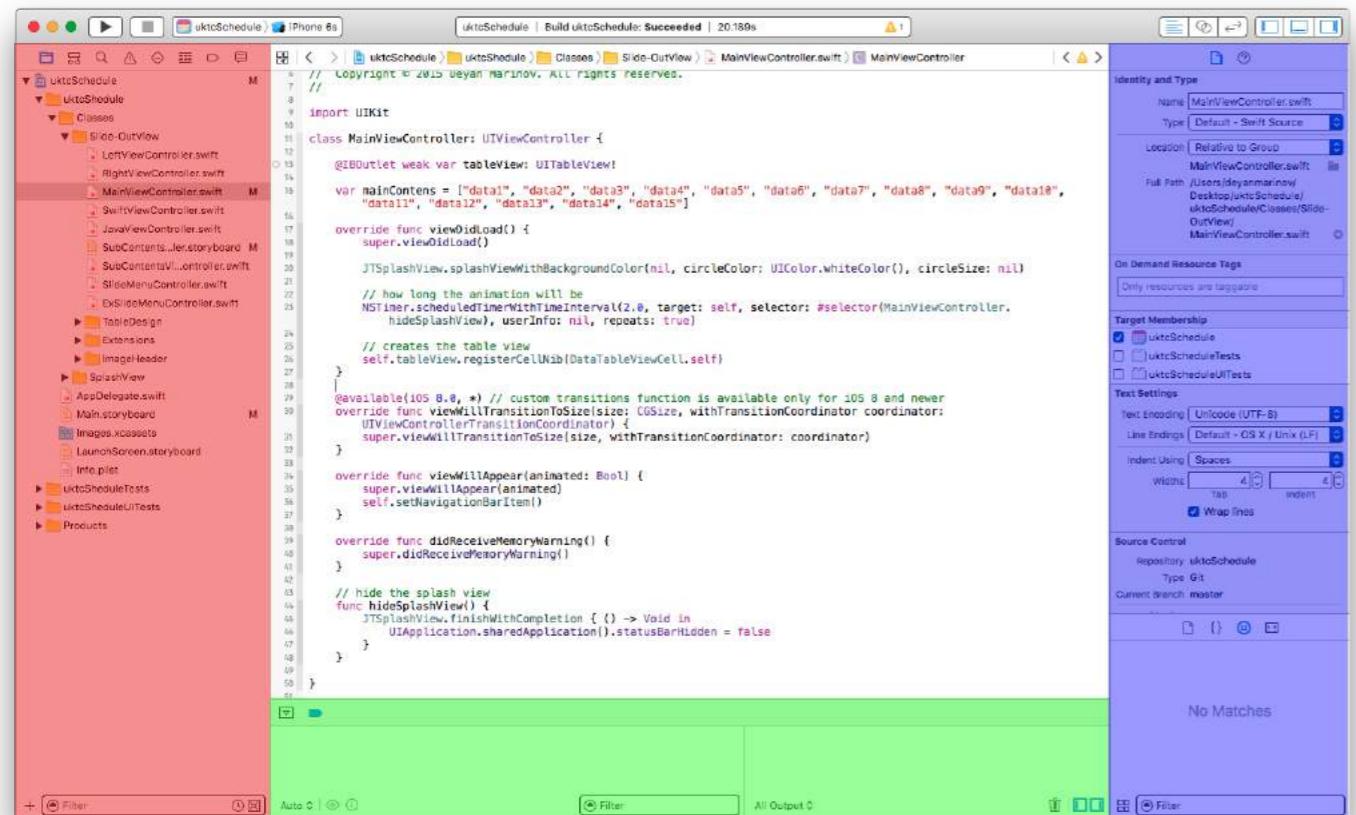


继续

接下来你应该学习自动布局约束。这些约束帮助你在故事板上定位控件，使其在任何设备尺寸和方向下都能显示良好。

## 第1.3节：Xcode 界面

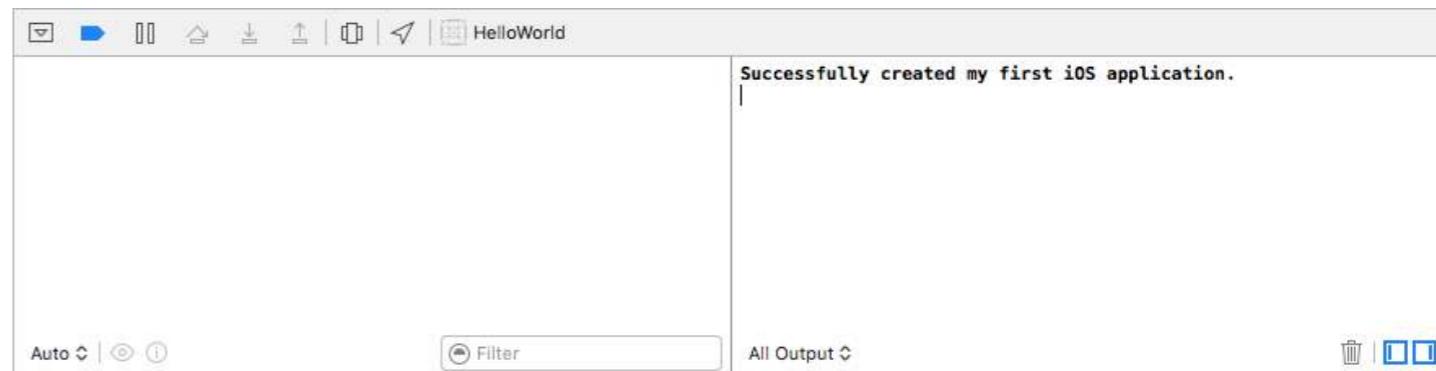
在 Xcode 中，你有三个独立的工作区域——导航器（红色）、调试区域（绿色）和实用工具（蓝色）。



工作区窗口始终包含编辑区域。当你在项目中选择一个文件时，其内容会显示在编辑区域，Xcode 会在合适的编辑器中打开该文件。例如，在上图中，编辑区域显示的是 MainViewController.swift，这是一个 Swift 代码文件，选中它是在工作区左侧的导航器区域。

75% / 50% / 33% / 25% scale respectively..

The Xcode debug area (at the bottom) should have also printed "Successfully created my first iOS application." to the console. "Successfully created my first iOS application." message is the string you printed programmatically in the **Add code** part.

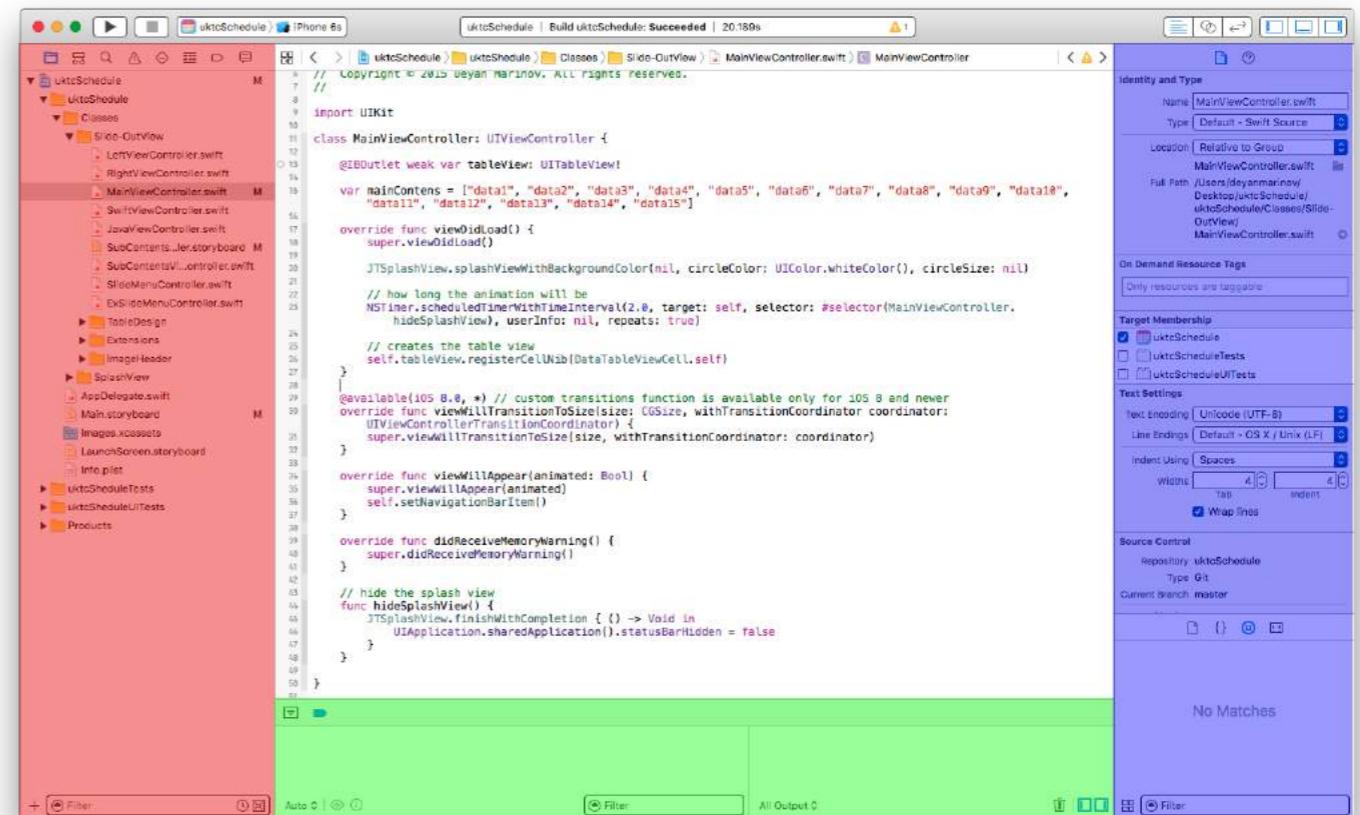


Going on

You should learn about Auto Layout constraints next. These help you to position your controls on the storyboard so that they look good on any device size and orientation.

## Section 1.3: Xcode Interface

In the Xcode, you have three separate areas of working - Navigators (in red), Debug area(in green) and Utilities(in blue).



The workspace window always includes the editor area. When you select a file in your project, its contents appear in the editor area, where Xcode opens the file in an appropriate editor. For example, in the image above, the editor area MainViewController.swift, a swift code file that is selected in the Navigator area on the left of the workspace

窗口。

## 导航器区域



导航器窗口包含以下八个选项：

- 项目导航器。添加、删除、分组以及管理项目中的文件，或选择文件以在编辑区查看或编辑其内容。
- 符号导航器。以列表或层级方式浏览项目中的符号。过滤栏左侧的按钮允许你将显示的符号限制为仅类和协议的组合、仅项目中的符号，或仅容器。
- 查找导航器 使用搜索选项和过滤器快速查找项目中的任意字符串。
- 问题导航器。查看打开、分析和构建项目时发现的诊断、警告和错误等问题。
- 测试导航器。创建、管理、运行和审查单元测试。
- 调试导航器。检查程序执行过程中指定时间点或位置的运行线程及相关堆栈信息。
- 断点导航器。通过指定触发条件等特性来微调断点。
- 报告导航器。查看构建、运行、调试、持续集成和源代码控制任务的历史记录。

## 编辑器

Xcode 中的大部分开发工作都在编辑区进行，编辑区是工作区窗口中始终可见的主要区域。你最常用的编辑器有：

- 源代码编辑器。编写和编辑源代码。

window.

## Navigator Area



The navigator window contains the following eight options:

- **Project navigator.** Add, delete, group, and otherwise manage files in your project, or choose a file to view or edit its contents in the editor area.
- **Symbol navigator.** Browse the symbols in your project as a list or hierarchy. Buttons on the left of the filter bar let you limit the shown symbols to a combination of only classes and protocols, only symbols in your project, or only containers.
- **Find navigator** Use search options and filters to quickly find any string within your project.
- **Issue navigator.** View issues such as diagnostics, warnings, and errors found when opening, analyzing, and building your project.
- **Test navigator.** Create, manage, run, and review unit tests.
- **Debug navigator.** Examine the running threads and associated stack information at a specified point or time during program execution.
- **Breakpoint navigator.** Fine-tune breakpoints by specifying characteristics such as triggering conditions.
- **Report navigator.** View the history of your build, run, debug, continuous integration, and source control tasks.

## The Editors

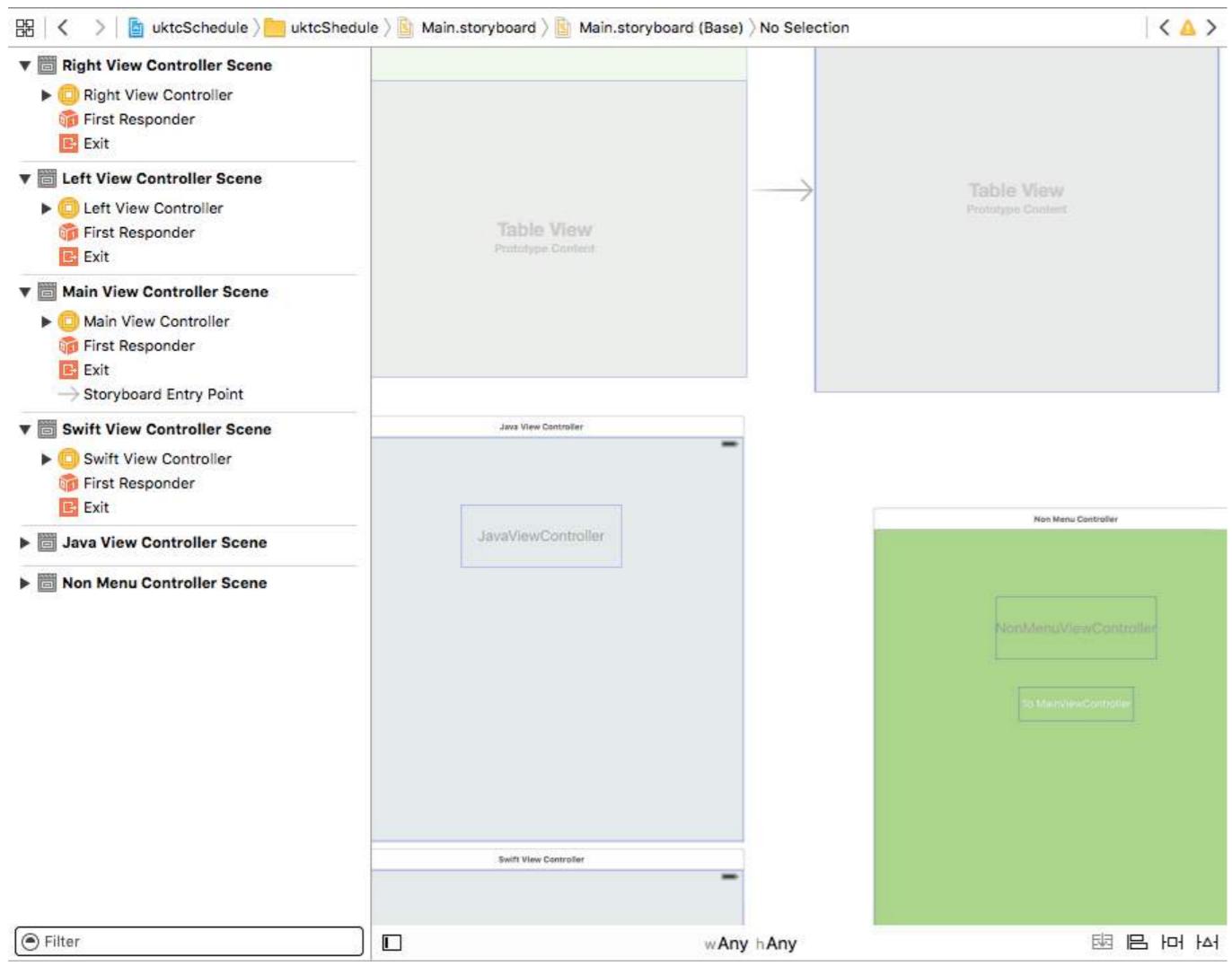
Most development work in Xcode occurs in the editor area, the main area that is always visible within the workspace window. The editors you use most often are:

- **Source editor.** Write and edit source code.

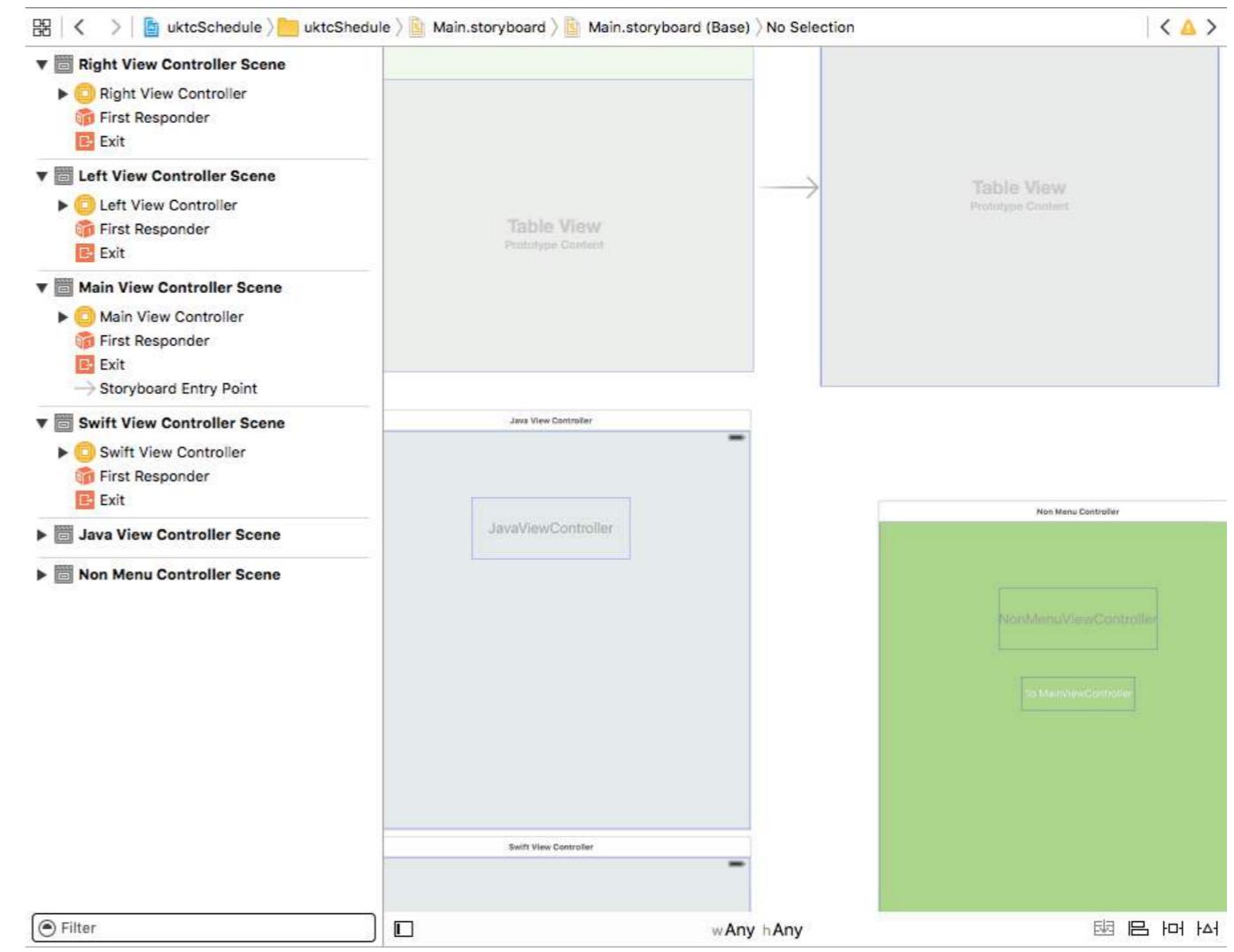
```
1 //  
2 // LeftViewController.swift  
3 // uktcSchedule  
4 //  
5 // Created by Deyan Marinov on 10/9/15.  
6 // Copyright © 2015 Deyan Marinov. All rights reserved.  
7 //  
8  
9 import UIKit  
10  
11 enum LeftMenu: Int {  
12     case Main = 0  
13     case Swift  
14     case Java  
15 }  
16  
17 protocol LeftMenuProtocol : class {  
18     func changeViewController(menu: LeftMenu)  
19 }  
20  
21 class LeftViewController : UIViewController, LeftMenuProtocol {  
22  
23     @IBOutlet weak var tableView: UITableView!  
24     var menus = ["Main", "Swift", "Java"]  
25     var mainViewController: UIViewController!  
26     var swiftViewController: UIViewController!  
27     var javaViewController: UIViewController!  
28     var goViewController: UIViewController!  
29     var nonMenuViewController: UIViewController!  
30     var imageHeaderView: ImageHeaderView!  
31  
32     required init?(coder aDecoder: NSCoder) {  
33         super.init(coder: aDecoder)  
34     }  
35  
36     override func viewDidLoad() {  
37         super.viewDidLoad()  
38         self.tableView.separatorColor = UIColor(red: 224/255, green: 224/255, blue: 224/255, alpha: 1.0)  
39  
40         let storyboard = UIStoryboard(name: "Main", bundle: nil)  
41         let swiftViewController = storyboard.instantiateViewControllerWithIdentifier("SwiftViewController") as!  
             SwiftViewController  
        self.swiftViewController = UINavigationController(rootViewController: swiftViewController)  
42  
43         let javaViewController = storyboard.instantiateViewControllerWithIdentifier("JavaViewController") as!  
             JavaViewController  
        self.javaViewController = UINavigationController(rootViewController: javaViewController)  
44  
45         self.tableView.reloadData()  
46     }  
47 }
```

- 界面构建器。 图形化创建和编辑用户界面文件。

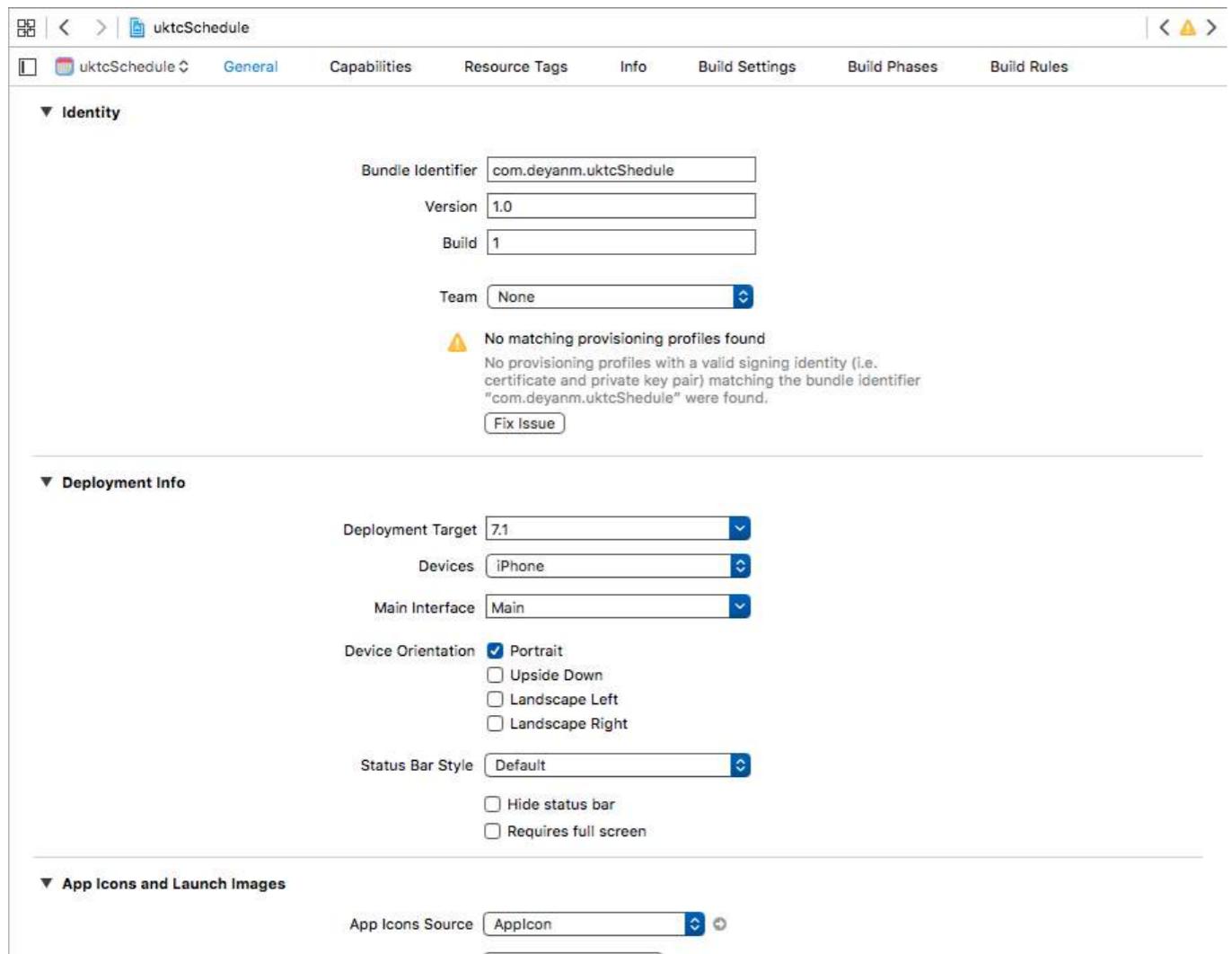
- **Interface Builder.** Graphically create and edit user interface files.



- **项目编辑器。** 查看和编辑应用程序的构建方式，例如通过指定构建选项、目标架构和应用权限。



- **Project editor.** View and edit how your apps should be built, such as by specifying build options, target architectures, and app entitlements.



使用工具栏右侧的编辑器配置按钮，为特定任务配置编辑区域：

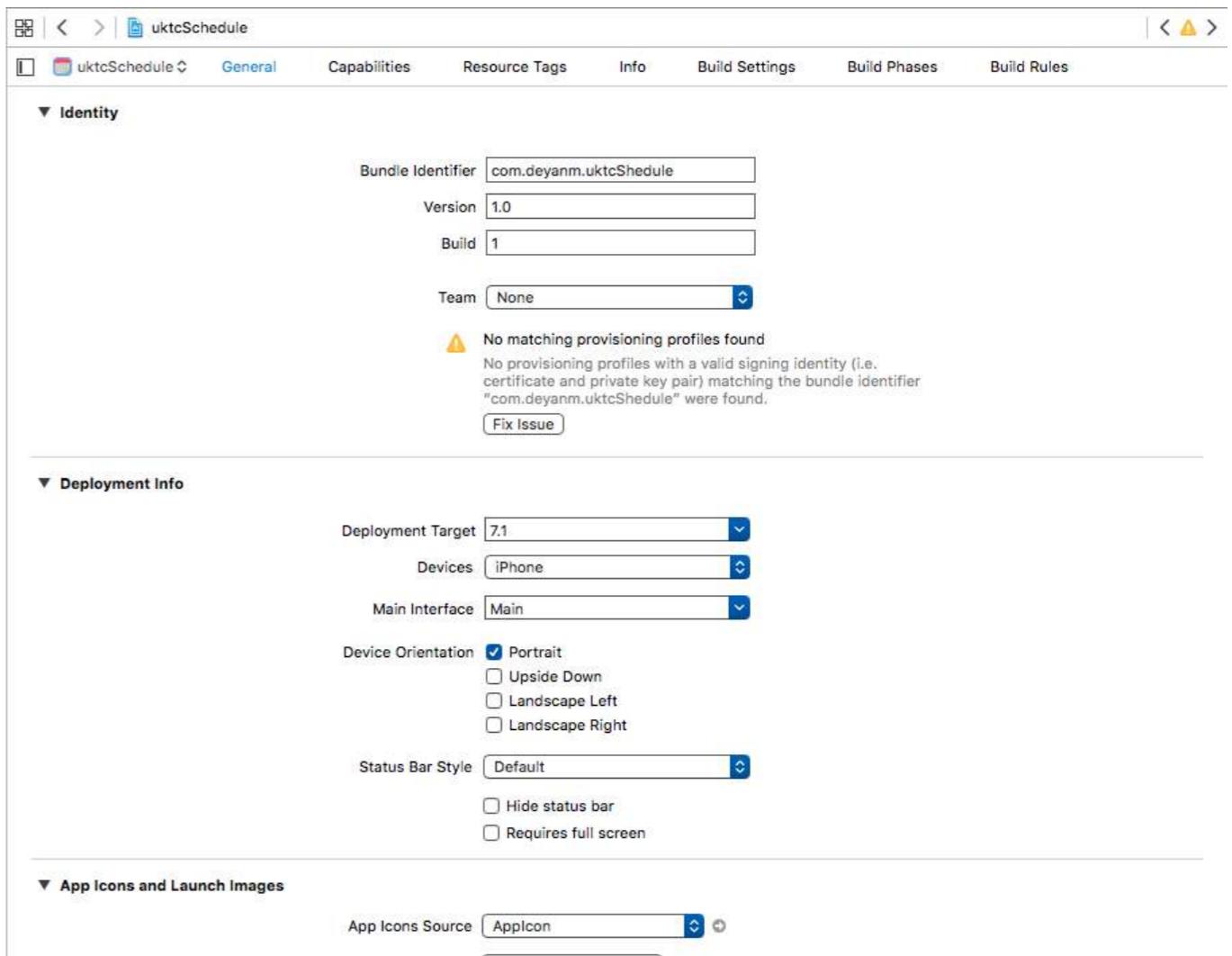


- **标准编辑器。** 用所选文件的内容填充编辑区域。
- **辅助编辑器。** 显示一个单独的编辑窗格，内容与标准编辑窗格中的内容逻辑相关。你也可以更改内容。
- **版本编辑器。** 显示所选文件在一个窗格中的内容与该同一文件的另一个版本在第二个窗格中的差异。此编辑器仅在项目处于源代码管理状态时可用。

## 公用事业领域的资源和要素

工作区窗口最右侧的实用工具区域让您快速访问以下资源：检查器，用于查看和修改编辑器中打开文件的属性；项目中可用的现成资源库

实用工具区域的顶部面板显示检查器。底部窗格则让您访问资源库。



Configure the editor area for a given task with the editor configuration buttons on the right side of the toolbar:

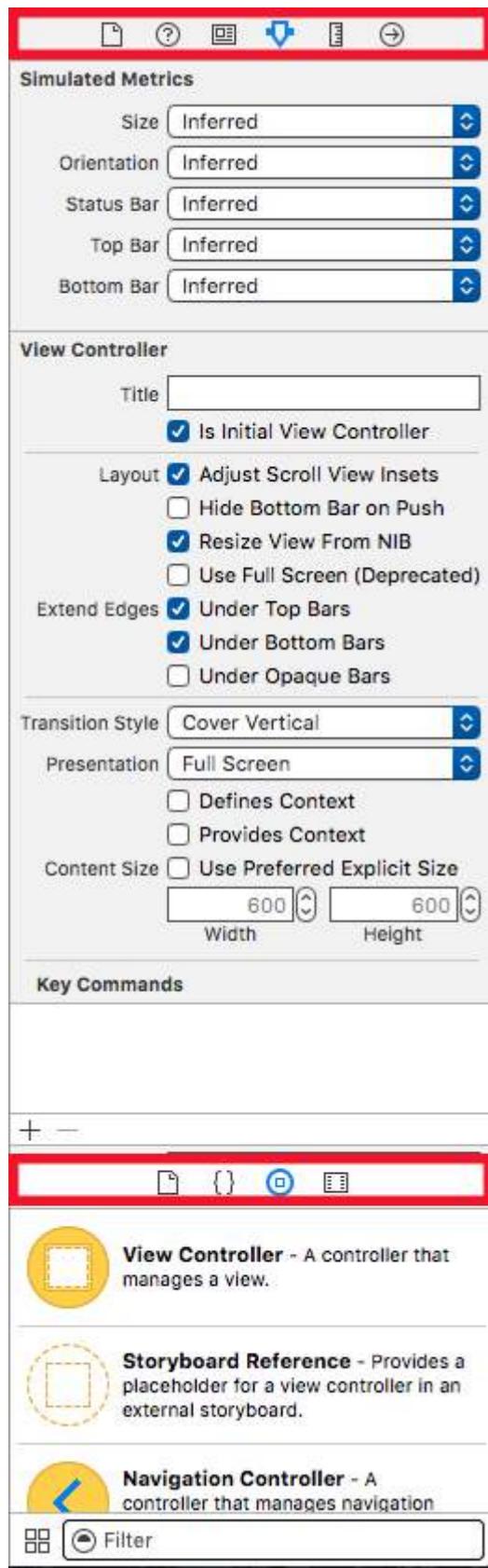


- **Standard Editor.** Fills the editor area with the contents of the selected file.
- **Assistant Editor.** Presents a separate editor pane with content logically related to content in the standard editor pane. You can also change the content.
- **Version Editor.** Shows the differences between the selected file in one pane and another version of that same file in a second pane. This editor works only when your project is under source control.

## Resources and Elements in Utilities Area

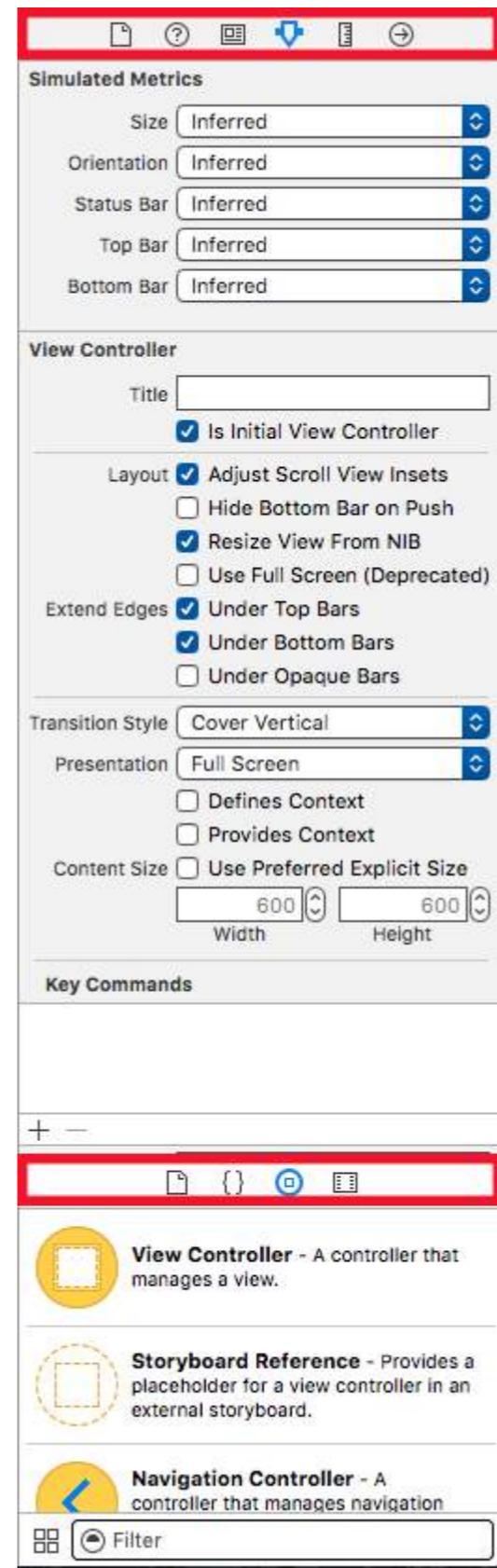
The utilities area on the far right of the workspace window gives you quick access to these resources: Inspectors, for viewing and modifying characteristics of the file open in an editor Libraries of ready-made resources for use in your project

**The top panel of the utilities area displays inspectors. The bottom pane gives you access to libraries.**



第一个面板（红色高亮）是检查器栏，使用它选择最适合当前任务的检查器。检查器栏中始终显示两个检查器（某些编辑器中还提供额外的检查器）：

- **文件检查器。**查看和管理所选文件的元数据。通常，您会本地化故事板和其他媒体文件，并更改用户界面文件的设置。
- **快速帮助。**查看文件中符号、界面元素或构建设置的详细信息。例如，快速帮助显示方法的简要描述、方法的声明位置和方式、作用域、参数以及其平台和架构的可用性。



The first panel (highlighted in red) is the **Inspector bar**, use it to choose the inspector best suited to your current task. Two inspectors are always visible in the inspector bar (additional inspectors are available in some editors):

- **File inspector.**View and manage metadata for the selected file. Typically you will localize storyboards and other media files and change settings for user interface files.
- **Quick Help.**View details about a symbol, an interface element, or a build setting in the file. For example, Quick Help displays a concise description of a method, where and how the method is declared, its scope, the parameters it takes, and its platform and architecture availability.

使用资源库栏（第二个红色高亮）访问项目中可直接使用的资源库：

- 文件模板。常见类型文件和代码结构的模板。
- 代码片段。用于软件中的简短源代码片段，如类声明、控制流程、代码块声明以及常用苹果技术的模板。
- 对象。应用程序用户界面的项目。
- 媒体。包含图形、图标、声音文件等的文件。

要使用库，请将其直接拖动到相应区域。例如，要使用代码片段，请将其从库中拖动到源代码编辑器；要从文件模板创建源文件，请将模板拖动到项目导航器。

要限制所选库中显示的项目，请在筛选栏（底部窗格）的文本字段中输入相关文本。例如，在文本字段中输入“按钮”以显示对象库中的所有按钮。

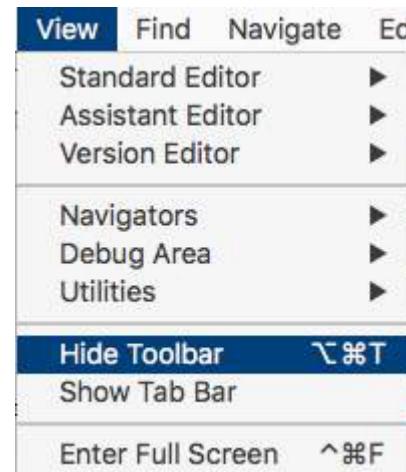
#### 使用工作区工具栏管理任务

工作区窗口顶部的工具栏提供对常用命令的快速访问。运行按钮构建并运行您的产品。停止按钮终止正在运行的代码。方案菜单允许您配置要构建和运行的产品。活动查看器通过显示状态消息、构建进度及有关项目的其他信息，展示当前执行任务的进度。

编辑器配置按钮（第一组三个按钮）允许您配置编辑器区域，工作区配置按钮（第二组三个按钮）用于隐藏或显示可选的导航器、调试和实用工具区域。



视图菜单包含隐藏或显示工具栏的命令。



## 第1.4节：用Swift 3创建您的第一个程序

这里介绍如何用Swift 3语言创建第一个基础程序。首先您需要具备任何基础编程语言知识，或者如果没有，也可以准备从头开始学习。

Use the **Library bar** (the second highlighted in red) to access ready-to-use libraries of resources for your project:

- **File templates.** Templates for common types of files and code constructs.
- **Code snippets.** Short pieces of source code for use in your software, such as class declarations, control flows, block declarations, and templates for commonly used Apple technologies.
- **Objects.** Items for your app's user interface.
- **Media.** Files containing graphics, icons, sound files, and the like.

To use a library, drag it directly to the appropriate area. For example, to use a code snippet, drag it from the library to the source editor; to create a source file from a file template, drag its template to the project navigator.

To restrict the items displayed in a selected library, type relevant text into the text field in the **Filter bar** (the bottom pane). For example, type “button” in the text field to show all the buttons in the Objects library.

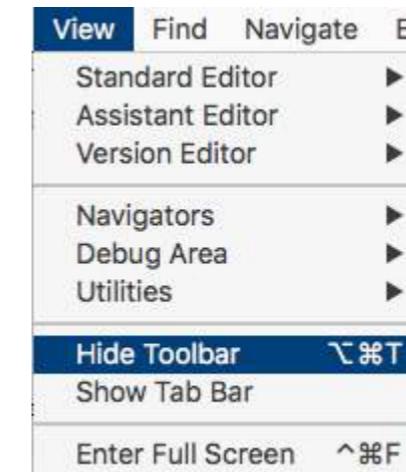
#### Manage Tasks with the Workspace Toolbar

The toolbar at the top of the workspace window provides quick access to frequently used commands. The **Run button** builds and runs your products. The **Stop button** terminates your running code. The **Scheme menu** lets you configure the products you want to build and run. The **activity viewer** shows the progress of tasks currently executing by displaying status messages, build progress, and other information about your project.

The **editor configuration buttons** (the first group of three buttons) let you configure the editor area, and the **workspace configuration buttons** (the second group of three buttons) hide or show the optional navigator, debug, and utilities areas.



The **View menu** includes commands to hide or show the toolbar.



## Section 1.4: Create your first program in Swift 3

Here I am presenting how to create first basic program in Swift 3 language. First you need to have any basic programming language knowledge or not having then be ready to learn it from beginning.

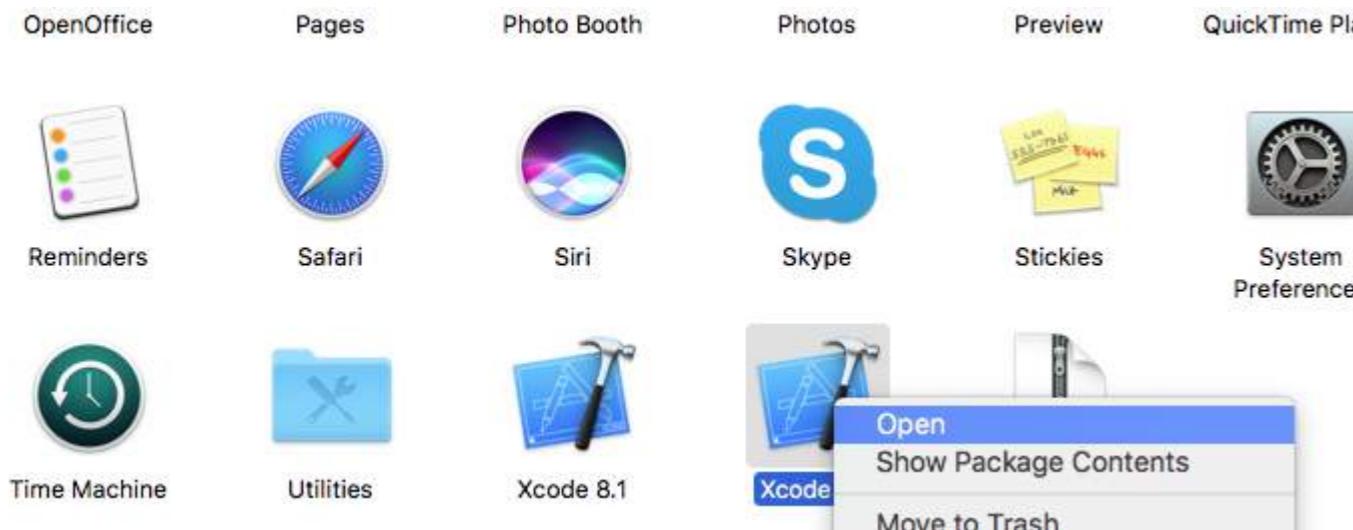
开发要求：

1. MAC OS - 版本 10.11.6 或更高，适用于新的 Xcode 8.2
2. Xcode - 版本 8.2 [Apple 关于 Xcode 介绍的文档。](#)

Xcode 8.2 具有新的 Swift 3 语言特性和兼容 iOS 10 的新 API。

## 创建你的第一个程序

首先进入应用程序并打开你的 Xcode 8.2。



之后你将看到该界面



然后选择创建新项目，之后你将看到下一个界面

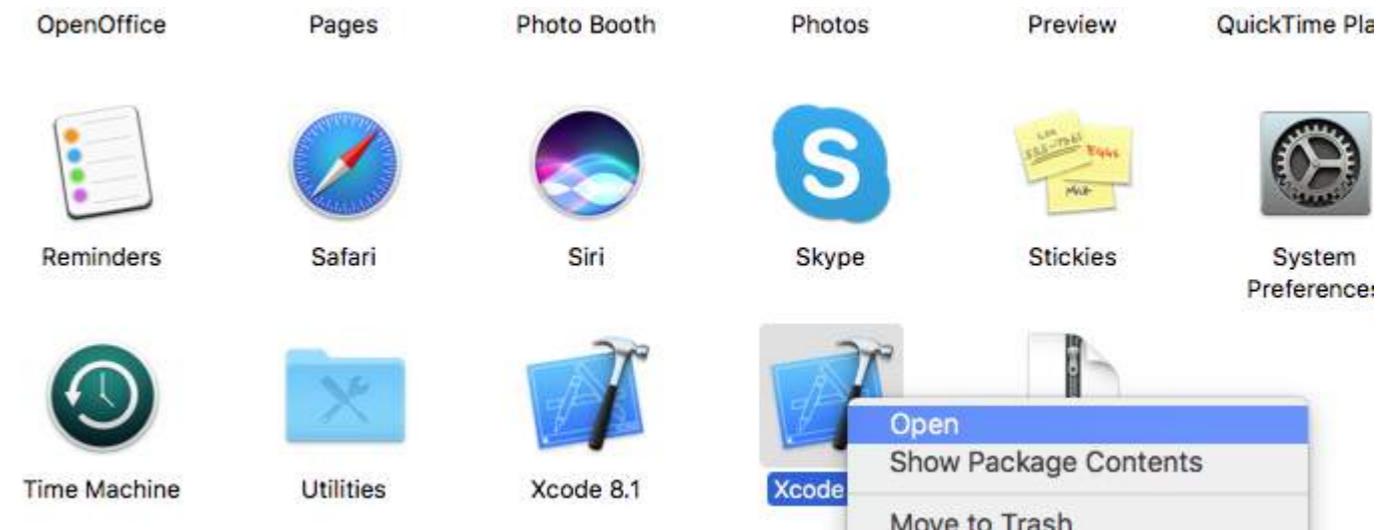
Requirements for developments:

1. MAC OS - Version 10.11.6 or later for new Xcode 8.2
2. Xcode - Version 8.2 [Apple Document for Xcode introduction.](#)

Xcode 8.2 has new Swift 3 language features with new iOS 10 compatible API's.

## Create your first program

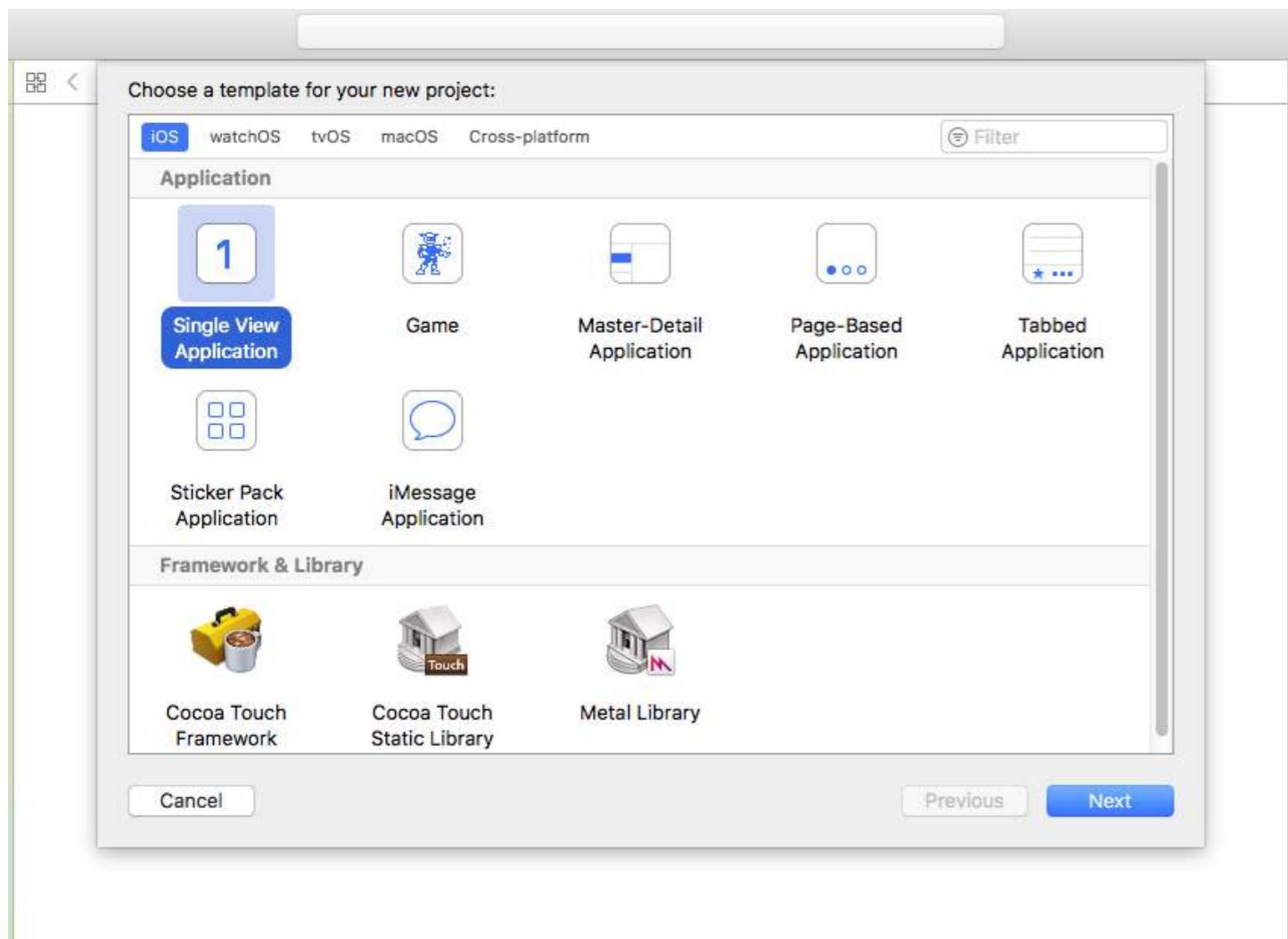
First go to Application and open your Xcode 8.2.



After that you will see the screen



Then choose Create new Project and after that you will see next screen



这也是 Xcode 中非常重要的部分，用于选择我们的项目类型。我们需要根据操作系统类型选择项目。  
顶部有五种类型的选项：

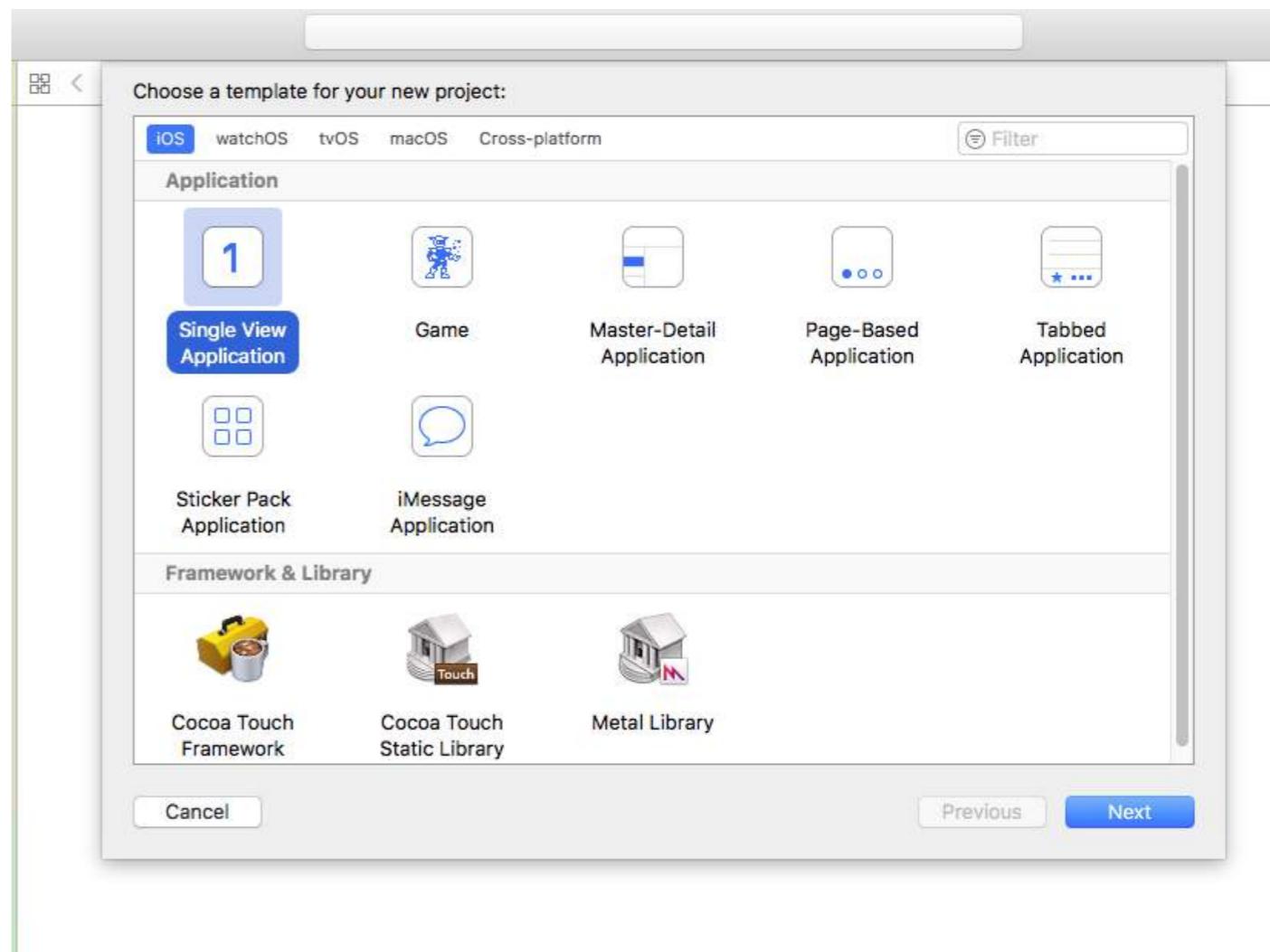
[1.iOS](#)

[2.watchOS](#)

[3.macOS](#)

[4.跨平台](#)

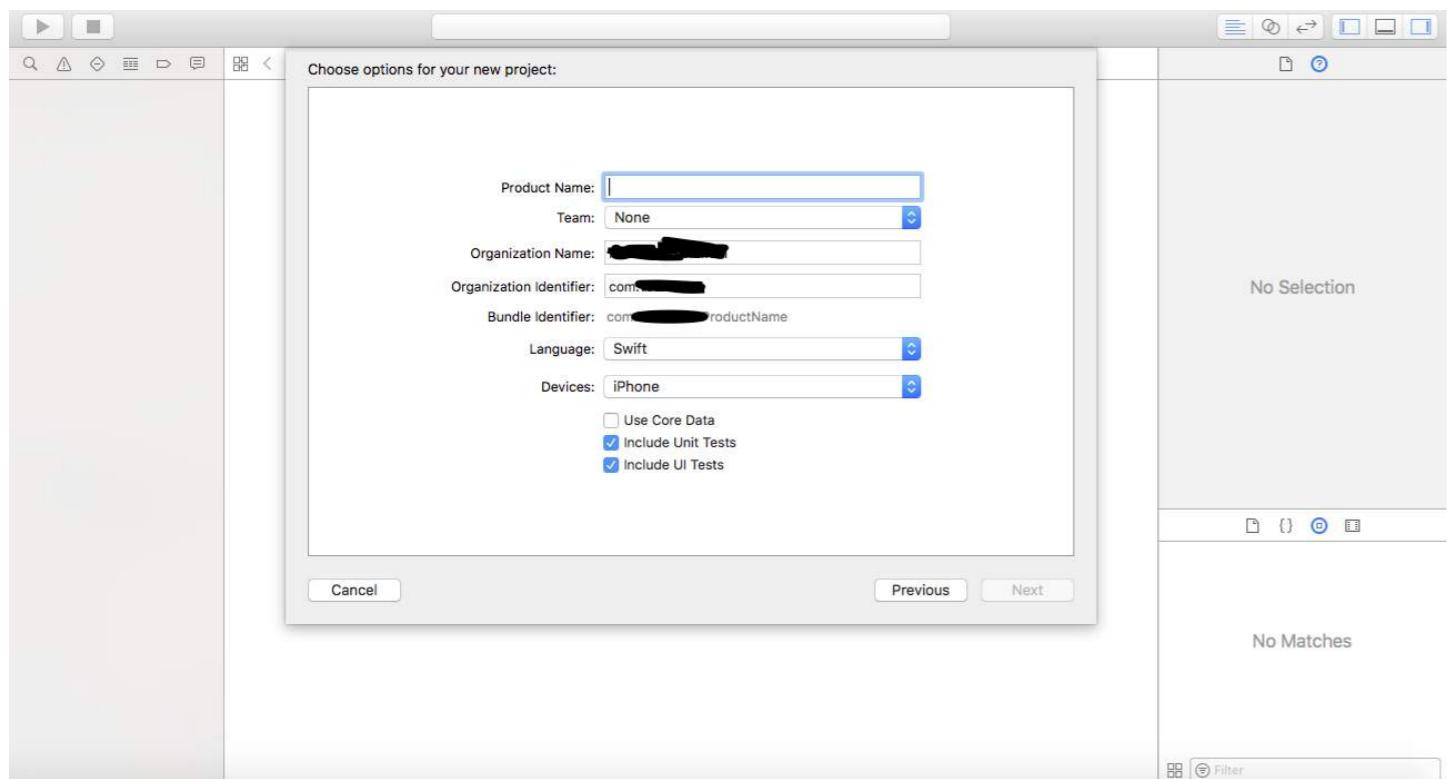
现在我们选择 iOS 平台进行开发，并创建一个非常基础的项目，选择单视图应用程序选项：



This is also very important part inside Xcode for selecting our project type. We need to choose our project according to types of OS. There are five types of options available on the top side:

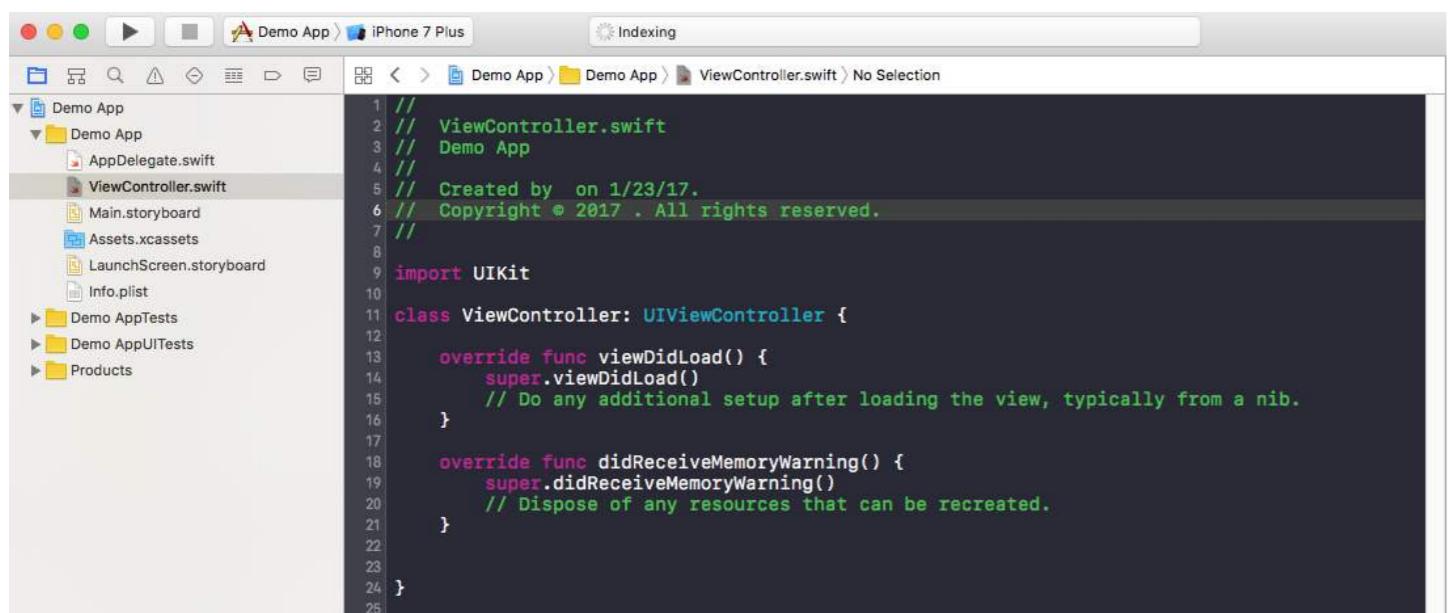
1. [iOS](#)
2. [watchOS](#)
3. [macOS](#)
4. Cross-platform

Now we are choosing iOS platform for development and creating very basic project with the single view application option:



然后我们需要填写产品名称，这将代表你的Bundle名称和应用程序名称。

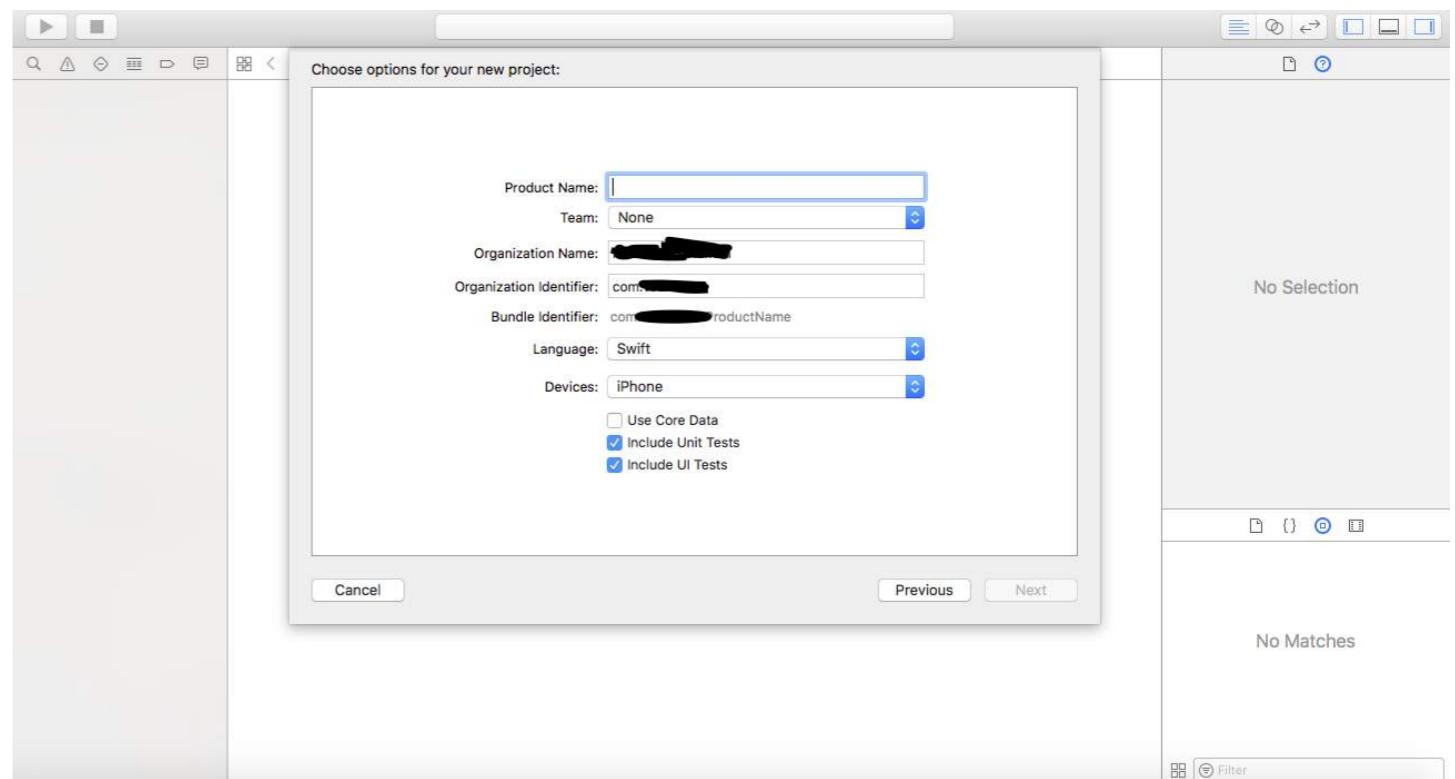
应用程序名称你可以根据需求以后更改。然后点击“创建”，之后你的界面将如下所示：



在这个类中你可以看到文件名是ViewController.swift，类名也是ViewController，继承自UIViewController父类，最后我们创建了第一个变量，名称为myString，类型为'String'。在'super.viewDidLoad()'下添加以下内容

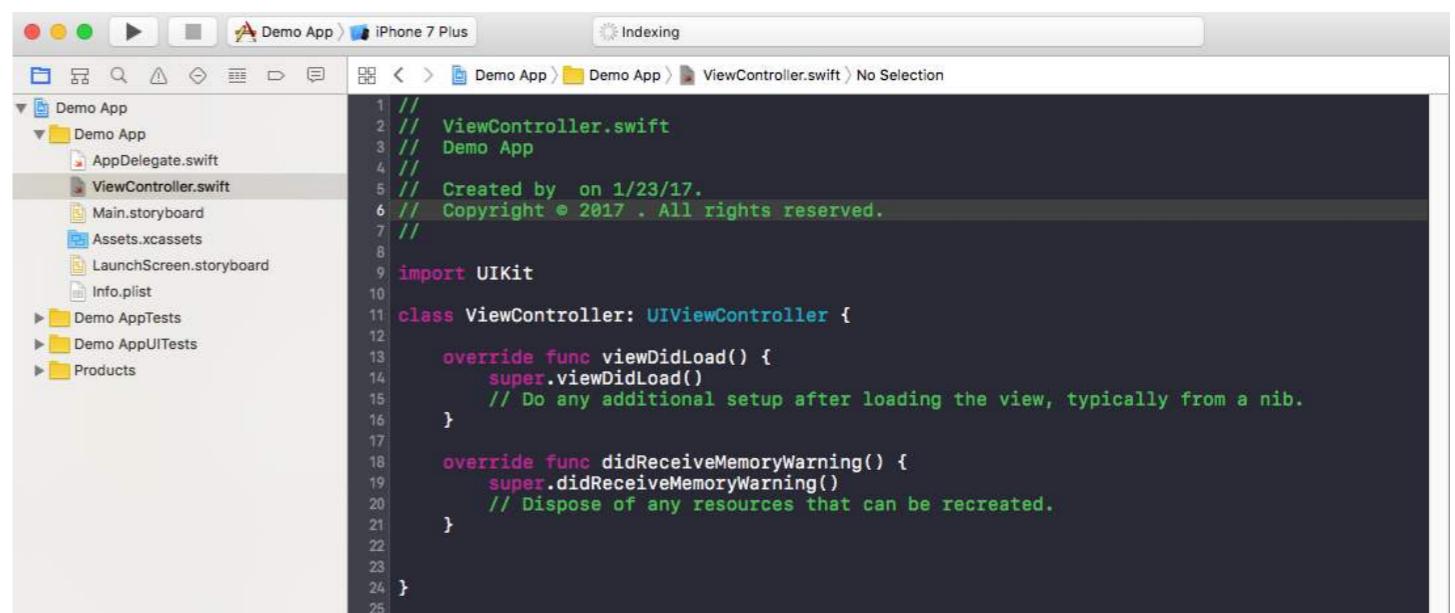
```
let myString = "Hello, World!"
```

我们将打印该变量的内容。首先，在屏幕左上角选择你的模拟器类型，然后点击“运行”按钮。



Then we need to give Product Name, this will represent your Bundle name and application name.

Application name you can change later as per your requirements. Then we need to click "Create" and after that your screen will look like this one below:



Inside this class you can see the file name is ViewController.swift and inside the class the name is also ViewController which is inheritance by the UIViewController super class and finally we're creating our first variable which name is **myString** of the type 'String'. Add the following under 'super.viewDidLoad()'

```
let myString = "Hello, World!"
```

We are going to print the content of this variable. First, select your simulator type at the top left hand side of the screen and then click on the "Run" button.

The screenshot shows the Xcode interface with the 'Demo App' project selected. The left sidebar displays the project structure: Demo App (target), Demo App (group), AppDelegate.swift, ViewController.swift, Main.storyboard, Assets.xcassets, LaunchScreen.storyboard, and Info.plist. The right pane shows the 'ViewController.swift' file with the following code:

```
5 // Created by on 1/23/17.  
6 // Copyright © 2017 . All rights reserved.  
7 //  
8 import UIKit  
9 class ViewController: UIViewController {  
10    override func viewDidLoad() {  
11        super.viewDidLoad()  
12        let myString = "Hello, World!"  
13        //Print here value of myString variable  
14        print(myString)  
15    }  
16}  
17  
18}
```

The bottom right corner of the Xcode window shows the output 'Hello, World!'.

之后你的输出将显示在右下角的终端中。恭喜，这是你在Xcode中的第一个Hello World程序。

The screenshot shows the Xcode interface with the 'Demo App' project selected. The left sidebar displays the project structure: Demo App (target), Demo App (group), AppDelegate.swift, ViewController.swift, Main.storyboard, Assets.xcassets, LaunchScreen.storyboard, and Info.plist. The right pane shows the 'ViewController.swift' file with the following code:

```
5 // Created by on 1/23/17.  
6 // Copyright © 2017 . All rights reserved.  
7 //  
8 import UIKit  
9 class ViewController: UIViewController {  
10    override func viewDidLoad() {  
11        super.viewDidLoad()  
12        let myString = "Hello, World!"  
13        //Print here value of myString variable  
14        print(myString)  
15    }  
16}  
17  
18}
```

The bottom right corner of the Xcode window shows the output 'Hello, World!'.

After that your output will be shown on terminal which is on bottom right hand side. Congratulations, This is your first Hello World program inside Xcode.

# 第二章：UILabel

UILabel 类实现了一个只读的文本视图。您可以使用此类绘制一行或多行静态文本，例如用于标识用户界面其他部分的文本。基础的 UILabel 类支持标签文本的简单和复杂样式。您还可以控制外观的各个方面，例如标签是否使用阴影或高亮绘制。如果需要，您可以通过子类化进一步自定义文本的外观。

## 第2.1节：创建 UILabel

### 使用框架

当您知道要为标签设置的确切尺寸时，可以使用 CGRect 框架初始化一个 UILabel。

#### Swift

```
let frame = CGRect(x: 0, y: 0, width: 200, height: 21)
let label = UILabel(frame: frame)
view.addSubview(label)
```

#### Objective-C

```
CGRect frame = CGRectMake(0, 0, 200, 21);
UILabel *label = [[UILabel alloc] initWithFrame:frame];
[view addSubview:label];
```

### 使用自动布局

当你希望 iOS 在运行时动态计算 UILabel 的框架时，可以为其添加约束。

#### Swift

```
let label = UILabel()
label.backgroundColor = .red
label.translatesAutoresizingMaskIntoConstraints = false
view.addSubview(label)

NSLayoutConstraint.activate([
    //将标签的顶部固定到其父视图的顶部：
    label.topAnchor.constraint(equalTo: view.topAnchor)

    //将标签的左侧固定到其父视图的左侧
    //如果字母是从左到右排列，或者固定到其
    //父视图的右侧，如果字母是从右到左排列：
    label.leadingAnchor.constraint(equalTo: view.leadingAnchor)

    //将标签的底部固定到其父视图的底部：
    label.bottomAnchor.constraint(equalTo: view.bottomAnchor)

    //标签的宽度应等于100点：
    label.widthAnchor.constraint(equalToConstant: 100)
])
```

#### Objective-C

```
UILabel *label = [[UILabel alloc] init];

使用 Objective-C + 可视化格式语言 (VFL)
UILabel *label = [UILabel new]; label.translatesAutoresizingMaskIntoConstraints = NO; [self.view addSubview:label];
// 添加水平约束，左右各有5的内边距，从leading和trailing开始 [self.view
addConstraints:[NSLayoutConstraint constraintsWithVisualFormat:@"V:|-5-[labelName]-5-|" options:0 metrics:nil
```

# Chapter 2: UILabel

The UILabel class implements a read-only text view. You can use this class to draw one or multiple lines of static text, such as those you might use to identify other parts of your user interface. The base UILabel class provides support for both simple and complex styling of the label text. You can also control over aspects of appearance, such as whether the label uses a shadow or draws with a highlight. If needed, you can customize the appearance of your text further by subclassing.

## Section 2.1: Create a UILabel

### With a Frame

When you know the exact dimensions you want to set for your label, you can initialize a UILabel with a CGRect frame.

#### Swift

```
let frame = CGRect(x: 0, y: 0, width: 200, height: 21)
let label = UILabel(frame: frame)
view.addSubview(label)
```

#### Objective-C

```
CGRect frame = CGRectMake(0, 0, 200, 21);
UILabel *label = [[UILabel alloc] initWithFrame:frame];
[view addSubview:label];
```

### With Auto Layout

You can add constraints on a UILabel when you want iOS to dynamically calculate its frame at runtime.

#### Swift

```
let label = UILabel()
label.backgroundColor = .red
label.translatesAutoresizingMaskIntoConstraints = false
view.addSubview(label)
```

```
NSLayoutConstraint.activate([
    //stick the top of the label to the top of its superview:
    label.topAnchor.constraint(equalTo: view.topAnchor)

    //stick the left of the label to the left of its superview
    //if the alphabet is left-to-right, or to the right of its
    //superview if the alphabet is right-to-left:
    label.leadingAnchor.constraint(equalTo: view.leadingAnchor)

    //stick the label's bottom to the bottom of its superview:
    label.bottomAnchor.constraint(equalTo: view.bottomAnchor)

    //the label's width should be equal to 100 points:
    label.widthAnchor.constraint(equalToConstant: 100)
])
```

#### Objective-C

```
UILabel *label = [[UILabel alloc] init];
```

### With Objective-C + Visual Format Language (VFL)

```
UILabel *label = [UILabel new]; label.translatesAutoresizingMaskIntoConstraints = NO; [self.view addSubview:label];
// add horizontal constraints with 5 left and right padding from the leading and trailing [self.view
addConstraints:[NSLayoutConstraint constraintsWithVisualFormat:@"V:|-5-[labelName]-5-|" options:0 metrics:nil
```

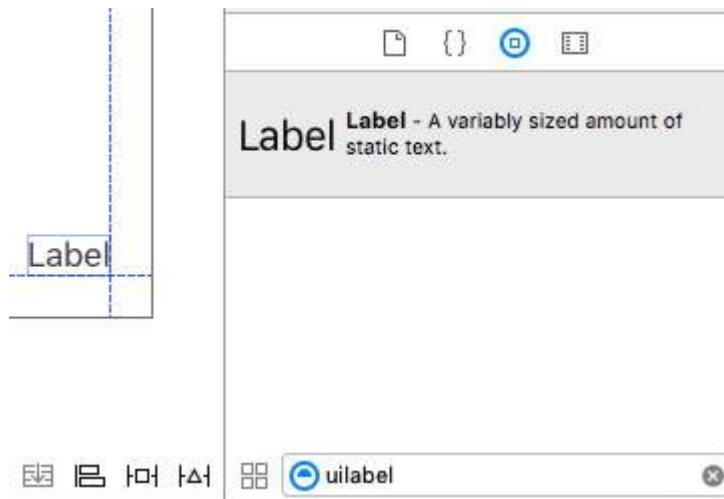
```
views:@[@"labelName":label]]; // 垂直约束，使用父视图的高度，顶部和底部无内边距 [self.view addConstraints:[  
NSLayoutConstraint constraintsWithVisualFormat:@"H:|[labelName]|"  
options:0 metrics:nil views:@[@"labelName":label]]]
```

VFL 文档可以在这里找到 [here](#)

标签创建后，务必通过自动布局设置尺寸。如果设置不当，Xcode 会显示错误。

## 使用界面构建器

你也可以使用界面构建器，通过从对象库面板拖拽一个标签（Label）到画布中的视图，向你的 Storyboard 或 .xib 文件添加一个 UILabel：

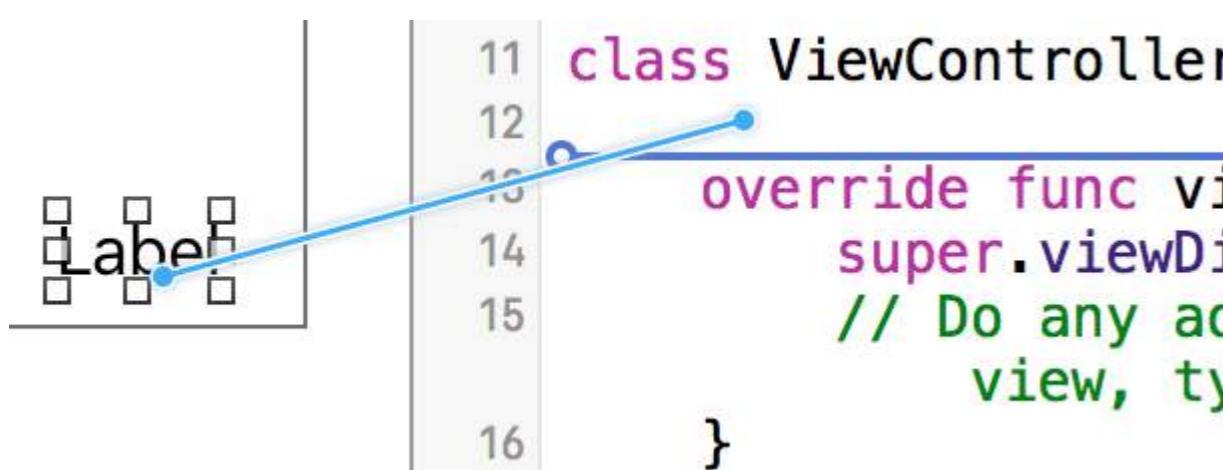


与其通过代码指定UILabel的位置和大小，不如使用Storyboard或.xib，通过自动布局（Auto Layout）为控件添加约束。

为了访问从storyboard或.xib创建的这个标签，需要为该标签创建一个IBOutlet。

## 界面构建器与视图控制器之间的连接

在Storyboard或.xib中添加UILabel后，可以按住Control键，然后将鼠标从UILabel拖动到ViewController，或者右键点击拖动到代码中，效果相同，从而将其与代码连接。



在属性对话框中，可以设置UILabel的名称，并将其设置为strong或weak。有关strong和weak的更多信息，请参见此处，

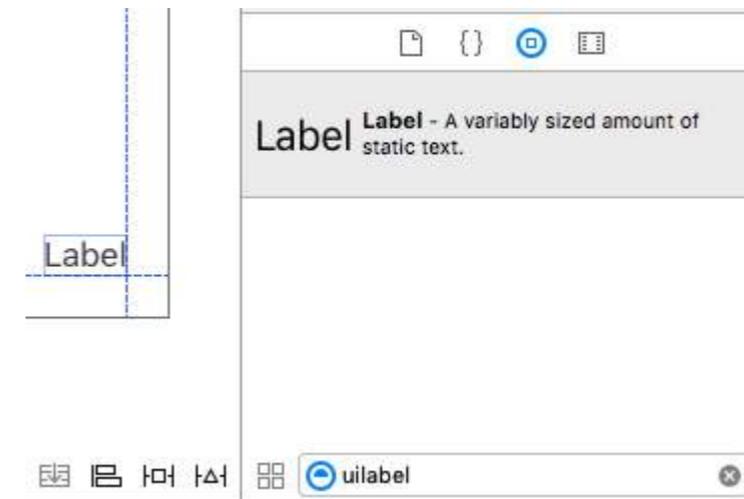
```
views:@[@"labelName":label]]; // vertical constraints that will use the height of the superView with no padding on  
top and bottom [self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat:@"H:|[labelName]|"  
options:0 metrics:nil views:@[@"labelName":label]]]
```

VFL documentation can be found [here](#)

After the label has been created, be sure to set the dimensions via Auto Layout. Xcode will display errors if it is done improperly.

## With Interface Builder

You also use Interface Builder to add a [UILabel](#) to your Storyboard or [.xib](#) file by dragging a Label from the Object Library panel and dropping it into a view in the canvas:

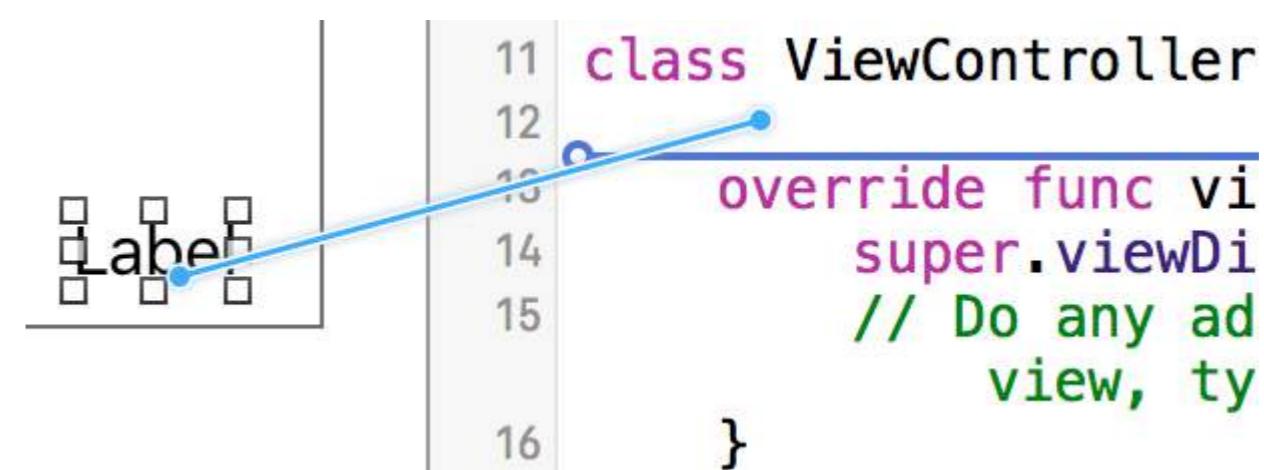


Instead of specifying a frame (position and size) for a [UILabel](#) programmatically, a Storyboard or a [.xib](#) lets you use Auto Layout to add constraints to the control.

In order to access this label created from storyboard or xib create an IBOutlet of this label.

## Linking Between Interface Builder and View Controller

Once you have added a [UILabel](#) to your Storyboard or [.xib](#) the file you can link it to your code by pressing Control + and then dragging the mouse between the [UILabel](#) to your [ViewController](#), or you could drag to the code while right clicking for it to have the same effect.



In the properties dialog, you can set the name of [UILabel](#), and set it as [strong](#) or [weak](#). For more information about [strong](#) and [weak](#), see this,

另一种方法是通过代码创建Outlet，如下所示：

#### Swift

```
@IBOutlet weak var nameLabel : UILabel!
```

#### Objective-C

```
@property (nonatomic, weak) IBOutlet UILabel *nameLabel;
```

## 第2.2节：行数

当你创建一个标签并设置的文本超过它能显示的单行时，文本会被截断，你只会看到以三个点 (...) 结尾的一行文本。这是因为一个名为numberOfLines的属性被设置为1，因此只显示一行。这是处理UILabel时的常见错误，许多人认为这是一个bug，或者他们可能使用多个标签来显示多行文本，但只需编辑这个属性，就可以让UILabel接受指定数量的行数。例如，如果将此属性设置为5，标签可以显示1、2、3、4或5行数据。

#### 以编程方式设置值

要设置此属性，只需为其赋一个新的整数值：

#### Swift

```
label.numberOfLines = 2
```

#### Objective-C

```
label.numberOfLines = 2;
```

#### 注意

可以将此属性设置为0。但这并不意味着它不会接受任何行，而是意味着标签可以有任意多的行（即“无限”）：

#### Swift

```
label.numberOfLines = 0
```

#### Objective-C

```
label.numberOfLines = 0;
```

#### 注意

如果标签有高度约束，则会遵守该约束。在这种情况下，`label.numberOfLines = 0`可能无法按预期工作。

#### 注意

对于更复杂的多行文本，UITextView 可能更合适。

#### 在界面构建器中设置值

你可以不用通过代码设置 `numberOfLines`，而是使用 Storyboard 或 .xib文件来设置 `numberOfLines` 属性。这样，我们就能达到与上述代码相同的效果。

The other way is to make the outlet programmatically as follows:

#### Swift

```
@IBOutlet weak var nameLabel : UILabel!
```

#### Objective-C

```
@property (nonatomic, weak) IBOutlet UILabel *nameLabel;
```

## Section 2.2: Number of Lines

When you make a label and set its text to be more than a single line that it can display, it will be truncated and you will see only one line of text ending with three dots (...). This is because a property called `numberOfLines` is set to 1, and therefore only one line will be displayed. It is a common mistake in handling `UILabels`, and many people think of it as a bug, or they may use more than one label to show more than a line of text, but just by editing this property, we can tell a `UILabel` to accept up to the specified number of lines. For example, if this property is set to 5, the label can show 1, 2, 3, 4 or 5 lines of data.

#### Setting the value programmatically

To set this property, simply assign a new integer to it:

#### Swift

```
label.numberOfLines = 2
```

#### Objective-C

```
label.numberOfLines = 2;
```

#### Note

It is possible to set this property to 0. However, this doesn't mean that it won't accept any lines, instead it means that the label can have as many lines as needed (aka "Infinity"):

#### Swift

```
label.numberOfLines = 0
```

#### Objective-C

```
label.numberOfLines = 0;
```

#### Note

If the label has a height constraint, the constraint will be respected. In this case, `label.numberOfLines = 0` may not work as expected.

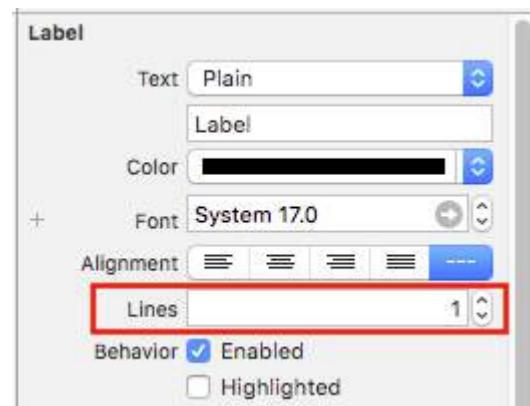
#### Note

For a more complex multi-line text, UITextView may be a better fit.\*

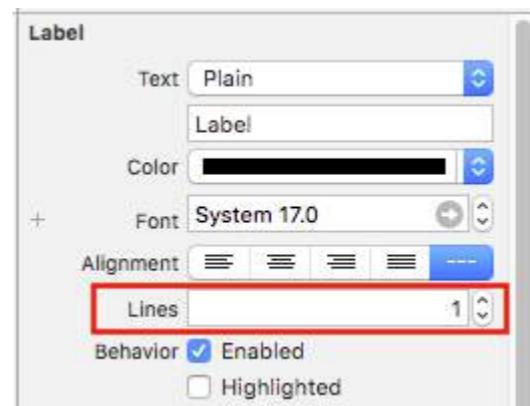
#### Setting the value in the Interface Builder

Instead of setting `numberOfLines` programmatically, you can use a Storyboard or a .xib and set the `numberOfLines` property. That way, we achieve the same results as the above code.

如下所示：



Like as below:



## 第2.3节：设置字体

### Swift

```
let label = UILabel()
```

### Objective-C

```
UILabel *label = [[UILabel alloc] init];  
或者  
UILabel *label = [UILabel new]; // 调用 alloc-init 的便捷方法
```

### 更改默认字体大小

#### Swift

```
label.font = UIFont.systemFontOfSize(17)
```

#### Swift 3

```
label.font = UIFont.systemFont(ofSize: 17)
```

### Objective-C

```
label.font = [UIFont systemFontOfSize:17];
```

### 使用特定字体粗细

版本 ≥ iOS 8.2

#### Swift

```
label.font = UIFont.systemFontOfSize(17, weight: UIFontWeightBold)
```

#### Swift3

```
label.font = UIFont.systemFont(ofSize: 17, weight: UIFontWeightBold)
```

### Objective-C

```
label.font = [UIFont systemFontOfSize:17 weight:UIFontWeightBold];
```

版本 < iOS 8.2

#### Swift

```
label.font = UIFont.boldSystemFontOfSize(17)
```

#### Swift3

```
label.font = UIFont.boldSystemFont(ofSize: 17)
```

## Section 2.3: Set Font

### Swift

```
let label = UILabel()
```

### Objective-C

```
UILabel *label = [[UILabel alloc] init];  
or  
UILabel *label = [UILabel new]; // convenience method for calling alloc-init
```

### Change the default font's size

#### Swift

```
label.font = UIFont.systemFontOfSize(17)
```

#### Swift 3

```
label.font = UIFont.systemFont(ofSize: 17)
```

### Objective-C

```
label.font = [UIFont systemFontOfSize:17];
```

### Use a specific font weight

Version ≥ iOS 8.2

#### Swift

```
label.font = UIFont.systemFontOfSize(17, weight: UIFontWeightBold)
```

#### Swift3

```
label.font = UIFont.systemFont(ofSize: 17, weight: UIFontWeightBold)
```

### Objective-C

```
label.font = [UIFont systemFontOfSize:17 weight:UIFontWeightBold];
```

Version < iOS 8.2

#### Swift

```
label.font = UIFont.boldSystemFontOfSize(17)
```

#### Swift3

```
label.font = UIFont.boldSystemFont(ofSize: 17)
```

## Objective-C

```
label.font = [UIFont boldSystemFontOfSize:17];
```

### 使用动态字体样式。

字体和字号将基于用户偏好的阅读大小。

#### Swift

```
label.font = UIFont.preferredFontForTextStyle(UIFontTextStyleBody)
```

#### Swift 3

```
label.font = UIFont.preferredFont(forTextStyle: .body)
```

## Objective-C

```
label.font = [UIFont preferredFontForTextStyle:UIFontTextStyleBody];
```

### 使用完全不同的字体

#### Swift

```
label.font = UIFont(name: "Avenir", size: 15)
```

## Objective-C

```
label.font = [UIFont fontWithName:@"Avenir" size:15];
```

### 覆盖字体大小

在不知道字体系列的情况下设置字体大小的一种方法是使用UILabel的font属性。

#### Swift

```
label.font = label.font.fontWithSize(15)
```

#### Swift 3

```
label.font = label.font.withSize(15)
```

## Objective-C

```
label.font = [label.font fontWithSize:15];
```

### 使用自定义字体 Swift

[请参考此链接](#)

## 第2.4节：文本颜色

您可以使用 label 的 textColor 属性为标签的全部文本应用文本颜色。

#### Swift

```
label.textColor = UIColor.redColor()  
label.textColor = UIColor(red: 64.0/255.0, green: 88.0/255.0, blue: 41.0/225.0, alpha: 1)
```

#### Swift 3

```
label.textColor = UIColor.red
```

## Objective-C

```
label.font = [UIFont boldSystemFontOfSize:17];
```

### Use a Dynamic Type text style.

The font and point size will be based on the user's preferred reading size.

#### Swift

```
label.font = UIFont.preferredFontForTextStyle(UIFontTextStyleBody)
```

#### Swift 3

```
label.font = UIFont.preferredFont(forTextStyle: .body)
```

## Objective-C

```
label.font = [UIFont preferredFontForTextStyle:UIFontTextStyleBody];
```

### Use a different font altogether

#### Swift

```
label.font = UIFont(name: "Avenir", size: 15)
```

## Objective-C

```
label.font = [UIFont fontWithName:@"Avenir" size:15];
```

### Override font size

A way to set the font size without knowing the font family is to use the **font** property of the **UILabel**.

#### Swift

```
label.font = label.font.fontWithSize(15)
```

#### Swift 3

```
label.font = label.font.withSize(15)
```

## Objective-C

```
label.font = [label.font fontWithSize:15];
```

### Use Custom Font Swift

[Refer to this link](#)

## Section 2.4: Text Color

You can use the label's textColor property to apply a text color to the entire text of the label.

#### Swift

```
label.textColor = UIColor.redColor()  
label.textColor = UIColor(red: 64.0/255.0, green: 88.0/255.0, blue: 41.0/225.0, alpha: 1)
```

#### Swift 3

```
label.textColor = UIColor.red
```

```
label.textColor = UIColor(red: 64.0/255.0, green: 88.0/255.0, blue: 41.0/225.0, alpha: 1)
```

#### Objective-C

```
label.textColor = [UIColor redColor];
label.textColor = [UIColor colorWithRed:64.0f/255.0f green:88.0f/255.0f blue:41.0f/255.0f
alpha:1.0f];
```

#### 为部分文本应用文本颜色

你也可以通过使用NSAttributedString来改变部分文本的颜色（或其他属性）：

#### Objective-C

```
attributedString = [[NSMutableAttributedString alloc] initWithString:@"草是绿色的；天空
是蓝色的。"];
[attributedString addAttribute: NSForegroundColorAttributeName value:[UIColor greenColor]
range:NSMakeRange(13, 5)];
[attributedString addAttribute: NSForegroundColorAttributeName value:[UIColor blueColor]
range:NSMakeRange(31, 4)];
label.attributedText = attributedString;
```

#### Swift

```
let attributedString = NSMutableAttributedString(string: "草是绿色的；天空是蓝色的。")
attributedString.addAttribute(NSForegroundColorAttributeName, value: UIColor.green(), range:
NSRange(location: 13, length: 5))
attributedString.addAttribute(NSForegroundColorAttributeName, value: UIColor.blue(), range:
NSRange(location: 31, length: 4))
label.attributedText = attributedString
```

## 第2.5节：背景颜色

#### Swift

```
label.backgroundColor = UIColor.redColor()
label.backgroundColor = .redColor()
```

#### Swift 3

```
label.backgroundColor = UIColor.red
```

#### Objective-C

```
label.backgroundColor = [UIColor redColor];
```

## 第2.6节：自适应大小

假设你在storyboard上有一个UILabel，并且你已经在  
ViewController.swift / ViewController.m中为它创建了一个IBOutlet，命名为labelOne。

为了让更改更容易被看到，在viewDidLoad

当你想根据标签中存储的内容自动调整标签大小时，使用函数sizeToFit。

```
label.textColor = UIColor(red: 64.0/255.0, green: 88.0/255.0, blue: 41.0/225.0, alpha: 1)
```

#### Objective-C

```
label.textColor = [UIColor redColor];
label.textColor = [UIColor colorWithRed:64.0f/255.0f green:88.0f/255.0f blue:41.0f/255.0f
alpha:1.0f];
```

#### Applying text color to a portion of the text

You can also vary the text color (or other attributes) of portions of the text by using NSAttributedString:

#### Objective-C

```
attributedString = [[NSMutableAttributedString alloc] initWithString:@"The grass is green; the sky
is blue."];
[attributedString addAttribute: NSForegroundColorAttributeName value:[UIColor greenColor]
range:NSMakeRange(13, 5)];
[attributedString addAttribute: NSForegroundColorAttributeName value:[UIColor blueColor]
range:NSMakeRange(31, 4)];
label.attributedText = attributedString;
```

#### Swift

```
let attributedString = NSMutableAttributedString(string: "The grass is green; the sky is blue.")
attributedString.addAttribute(NSForegroundColorAttributeName, value: UIColor.green(), range:
NSRange(location: 13, length: 5))
attributedString.addAttribute(NSForegroundColorAttributeName, value: UIColor.blue(), range:
NSRange(location: 31, length: 4))
label.attributedText = attributedString
```

## Section 2.5: Background Color

#### Swift

```
label.backgroundColor = UIColor.redColor()
label.backgroundColor = .redColor()
```

#### Swift 3

```
label.backgroundColor = UIColor.red
```

#### Objective-C

```
label.backgroundColor = [UIColor redColor];
```

## Section 2.6: Size to fit

Suppose you have a UILabel on your storyboard and you have created an IBOutlet for it in  
ViewController.swift / ViewController.m and named it labelOne.

To make the changes easily visible, change the backgroundColor and textColor of labelOne in the viewDidLoad  
method:

The function sizeToFit is used when you want to automatically resize a label based on the content stored within it.

## Swift

```
labelOne.backgroundColor = UIColor.blueColor()
labelOne.textColor = UIColor.whiteColor()
labelOne.text = "Hello, World!"
labelOne.sizeToFit()
```

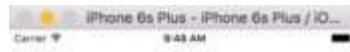
## Swift 3

```
labelOne.backgroundColor = UIColor.blue
labelOne.textColor = UIColor.white
labelOne.text = "Hello, World!"
labelOne.sizeToFit()
```

## Objective-C

```
labelOne.backgroundColor = [UIColor blueColor];
labelOne.textColor = [UIColor whiteColor];
labelOne.text = @"Hello, World!";
[labelOne sizeToFit];
```

上述代码的输出结果是：



如你所见，文本完全适合labelOne，因此没有变化。sizeToFit只会改变标签的框架。

让我们将文本改为稍长一些的内容：

```
labelOne.text = "Hello, World! I'm glad to be alive!"
```

现在，labelOne看起来像这样：

## Swift

```
labelOne.backgroundColor = UIColor.blueColor()
labelOne.textColor = UIColor.whiteColor()
labelOne.text = "Hello, World!"
labelOne.sizeToFit()
```

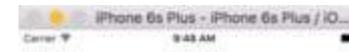
## Swift 3

```
labelOne.backgroundColor = UIColor.blue
labelOne.textColor = UIColor.white
labelOne.text = "Hello, World!"
labelOne.sizeToFit()
```

## Objective-C

```
labelOne.backgroundColor = [UIColor blueColor];
labelOne.textColor = [UIColor whiteColor];
labelOne.text = @"Hello, World!";
[labelOne sizeToFit];
```

The output for the above code is:



As you can see, there is no change as the text is perfectly fitting in labelOne. sizeToFit only changes the label's frame.

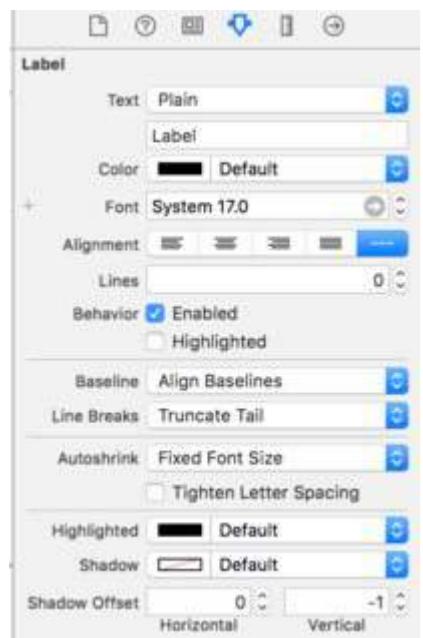
Let's change the text to a slightly longer one:

```
labelOne.text = "Hello, World! I'm glad to be alive!"
```

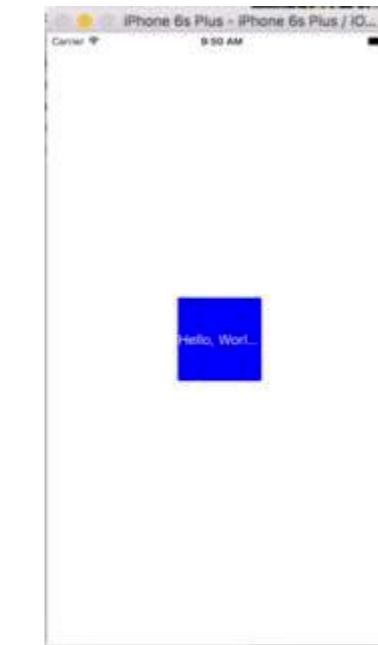
Now, labelOne looks like this:



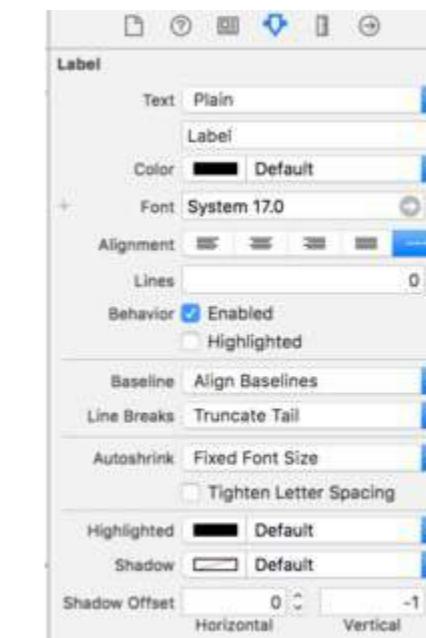
即使调用 `sizeToFit` 也不会改变任何东西。这是因为默认情况下，UILabel显示的numberOfLines被设置为1。让我们在Storyboard中将其改为零：



这次，当我们运行应用程序时，labelOne显示正确：



Even calling `sizeToFit` doesn't change anything. This is because by default, the `numberOfLines` shown by the `UILabel` is set to 1. Let's change it to zero on the storyboard:



This time, when we run the app, labelOne appears correctly:



numberOfLines 属性也可以在 ViewController 文件中更改：

```
// Objective-C  
labelOne.numberOfLines = 0;  
  
// Swift  
labelOne.numberOfLines = 0
```

## 第2.7节：文本对齐

### Swift

```
label.textAlignment = NSTextAlignment.left  
//或者更简短的写法  
label.textAlignment = .left
```

NSTextAlignment 枚举中的任何值都是有效的：.left, .center, .right, .justified, .natural

### Objective-C

```
label.textAlignment = NSTextAlignmentLeft;
```

枚举NSTextAlignment 中的任何值都是有效的：NSTextAlignmentLeft、NSTextAlignmentCenter、NSTextAlignmentRight、NSTextAlignmentJustified、NSTextAlignmentNatural

UILabel 中不支持垂直对齐：在UILabel 内将文本垂直对齐到顶部

## 第2.8节：计算内容边界（例如动态单元格高度）

计算标签所占框架的一个常见用例是为了适当调整表视图单元格的大小。  
推荐的做法是使用 NSString 方法

```
boundingRectWithSize:options:attributes:context:。
```

options 接受字符串绘制选项：

- NSStringDrawingUsesLineFragmentOrigin 应当用于多行标签



The numberOfLines property can also be changed in the ViewController file :

```
// Objective-C  
labelOne.numberOfLines = 0;  
  
// Swift  
labelOne.numberOfLines = 0
```

## Section 2.7: Text alignment

### Swift

```
label.textAlignment = NSTextAlignment.left  
//or the shorter  
label.textAlignment = .left
```

Any value in the [NSTextAlignment](#) enum is valid: .left, .center, .right, .justified, .natural

### Objective-C

```
label.textAlignment = NSTextAlignmentLeft;
```

Any value in the [NSTextAlignment](#) enum is valid: NSTextAlignmentLeft, NSTextAlignmentCenter, NSTextAlignmentRight, NSTextAlignmentJustified, NSTextAlignmentNatural

Vertical alignment in [UILabel](#) is not supported out of the box: [Vertically align text to top within a UILabel](#)

## Section 2.8: Calculate Content Bounds (for i.e. dynamic cell heights)

A common use case for wanting to calculate the frame a label will take up is for sizing table view cells appropriately.  
The recommended way of doing this is using the [NSString](#) method  
boundingRectWithSize:options:attributes:context:.

options takes String drawing options:

- NSStringDrawingUsesLineFragmentOrigin should be used for labels with multiple lines

- 如果有最大行数，应使用|操作符添加NSStringDrawingTruncatesLastVisibleLine

attributes是一个影响属性字符串的NSDictionary（完整列表见Apple Docs），但影响高度的因素包括：

- NSFontAttributeName**：非常重要，字号和字体系列是标签显示大小的关键部分。
- NSParagraphStyleAttributeName**：用于自定义文本的显示方式。这包括行间距、文本对齐方式、截断样式以及其他一些选项。如果你没有明确更改这些值，通常不需要太担心，但如果你在界面构建器（IB）中切换过某些值，这可能很重要。

context 应该为 `nil`，因为主要的 `NSStringDrawingContext` 用例是允许字体调整大小以适应指定的矩形，而在计算动态高度时通常不需要这样做。

## Objective C

```
- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath {
    UITableViewCell *cell = [tableView cellForRowAtIndexPath:indexPath];
    NSString *labelContent = cell.theLabel.text;
    // 如果你对字体做了最小的自定义，或者对硬编码几个值比较熟悉,
    // 你也可以选择直接从数据源获取内容
    // NSString *labelContent = [self.dataSource objectAtIndex:indexPath];
    // 如果从数据源获取，值可能是硬编码的
    NSFont *labelFont = [cell.theLabel font];
    // 即使你没有编写任何更改代码，NSParagraphStyle的这些值可能在界面构建器中被修改过
    NSMutableParagraphStyle *paragraphStyle = [NSMutableParagraphStyle new];
    paragraphStyle.lineBreakMode = NSLineBreakByWordWrapping;
    paragraphStyle.alignment = NSTextAlignmentCenter;
    NSDictionary *attributes = @{@"NSFontAttributeName": labelFont,
                                  NSParagraphStyleAttributeName: paragraphStyle};
    // 宽度对于高度也很重要
    CGFloat labelWidth = CGRectGetWidth(cell.theLabel.frame);
    // 如果你之前一直在硬编码，到这里你可以通过
    // 从 tableView.bounds.size.width 中减去左右的内边距来获得这个值
    // CGFloat labelWidth = CGRectGetWidth(tableView.frame) - 20.0f - 20.0f;
    CGRect bodyBounds = [labelContent boundingRectWithSize:CGSizeMake(width, CGFLOAT_MAX)
options:NSStringDrawingUsesLineFragmentOrigin attributes:attributes context:nil];
    return CGRectGetHeight(bodyBounds) + heightOfObjectsOnTopOfLabel + heightOfObjectBelowLabel;
}
```

## Swift 3

```
override func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
    var cell = tableView.cellForRow(atIndexPath: indexPath)!
    var labelContent = cell.theLabel.text
    var labelFont = cell.theLabel.font
    var paragraphStyle = NSMutableParagraphStyle()
    paragraphStyle.lineBreakMode = .byWordWrapping
}
```

- `NSStringDrawingTruncatesLastVisibleLine` should be added using the | operator if there are a maximum number of lines

attributes is an `NSDictionary` of attributes that effect attributed strings (full list: [Apple Docs](#)) but the factors that effect height include:

- NSFontAttributeName**: Very important, the size and font family is a critical part of the label's displayed size.
- NSParagraphStyleAttributeName**: For customizing how the text is displayed. This includes line spacing, text alignment, truncation style, and a few other options. If you did not explicitly change any of these values you should not have to worry about this much, but may be important if you toggled some values on IB.

context should be `nil` since the primary `NSStringDrawingContext` use case is for allowing font to resize to fit a specified rect, which shouldn't be the case if we're calculating a dynamic height.

## Objective C

```
- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath {
    UITableViewCell *cell = [tableView cellForRowAtIndexPath:indexPath];
    NSString *labelContent = cell.theLabel.text;
    // you may choose to get the content directly from the data source if you have done minimal
    // customizations to the font or are comfortable with hardcoding a few values
    // NSString *labelContent = [self.dataSource objectAtIndex:indexPath];
    // value may be hardcoded if retrieved from data source
    NSFont *labelFont = [cell.theLabel font];
    // The NSParagraphStyle, even if you did not code any changes these values may have been
    // altered in IB
    NSMutableParagraphStyle *paragraphStyle = [NSMutableParagraphStyle new];
    paragraphStyle.lineBreakMode = NSLineBreakByWordWrapping;
    paragraphStyle.alignment = NSTextAlignmentCenter;
    NSDictionary *attributes = @{@"NSFontAttributeName": labelFont,
                                  NSParagraphStyleAttributeName: paragraphStyle};
    // The width is also important to the height
    CGFloat labelWidth = CGRectGetWidth(cell.theLabel.frame);
    // If you have been hardcoding up to this point you will be able to get this value by
    // subtracting the padding on left and right from tableView.bounds.size.width
    // CGFloat labelWidth = CGRectGetWidth(tableView.frame) - 20.0f - 20.0f;
    CGRect bodyBounds = [labelContent boundingRectWithSize:CGSizeMake(width, CGFLOAT_MAX)
options:NSStringDrawingUsesLineFragmentOrigin attributes:attributes context:nil];
    return CGRectGetHeight(bodyBounds) + heightOfObjectsOnTopOfLabel + heightOfObjectBelowLabel;
}
```

## Swift 3

```
override func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
    var cell = tableView.cellForRow(atIndexPath: indexPath)!
    var labelContent = cell.theLabel.text
    var labelFont = cell.theLabel.font
    var paragraphStyle = NSMutableParagraphStyle()
    paragraphStyle.lineBreakMode = .byWordWrapping
}
```

```

paragraphStyle.alignment = .center

var attributes = [NSFontAttributeName: labelFont, NSParagraphStyleAttributeName: paragraphStyle]

var labelWidth: CGFloat = cell.theLabel.frame.width

var bodyBounds = labelContent.boundingRect(withSize: CGSize(width: width, height: CGFLOAT_MAX), options: .usesLineFragmentOrigin, attributes: attributes, context: nil)

return bodyBounds.height + heightOfObjectsOnTopOfLabel + heightOfObjectBelowLabel
}

```

相反, 如果你确实有一个设定的最大行数, 你首先需要计算单行的高度, 以确保我们不会得到超过允许高度的值:

```

// 我们通过省略 NSStringDrawingUsesLineFragmentOrigin 选项来计算单行高度,
// 该选项会假设标签宽度无限大
CGRect singleLineRect = [labelContent boundingRectWithSize:CGSizeMake(CGFloat(MAX, CGFloat(MAX)
options:NSStringDrawingTruncatesLastVisibleLine
context:nil];

CGFloat lineHeight = CGRectGetHeight(singleLineRect);
CGFloat maxHeight = lineHeight * cell.theLabel.numberOfLines;

// 现在你可以适当地调用该方法
CGRect bodyBounds = [labelContent boundingRectWithSize:CGSizeMake(width, maxHeight)
options:(NSStringDrawingUsesLineFragmentOrigin|NSStringDrawingTruncatesLastVisibleLine)
attributes:attributes context:nil];

return CGRectGetHeight(bodyBounds) + heightOfObjectsOnTopOfLabel + heightOfObjectBelowLabel;
}

```

## 第2.9节：标签富文本

### 01. 下划线文本 :- 单线/双线, 删除线 :- 单线/双线

#### 步骤1

选择标签并将标签类型从普通改为富文本

```

paragraphStyle.alignment = .center

var attributes = [NSFontAttributeName: labelFont, NSParagraphStyleAttributeName: paragraphStyle]

var labelWidth: CGFloat = cell.theLabel.frame.width

var bodyBounds = labelContent.boundingRect(withSize: CGSize(width: width, height: CGFLOAT_MAX), options: .usesLineFragmentOrigin, attributes: attributes, context: nil)

return bodyBounds.height + heightOfObjectsOnTopOfLabel + heightOfObjectBelowLabel
}

```

Conversely, if you do have a set maximum number of lines you will first need calculate the height of a single line to make sure we don't get a value taller than the allowed size:

```

// We calculate the height of a line by omitting the NSStringDrawingUsesLineFragmentOrigin
option, which will assume an infinitely wide label
CGRect singleLineRect = [labelContent boundingRectWithSize:CGSizeMake(CGFloat(MAX, CGFloat(MAX)
options:NSStringDrawingTruncatesLastVisibleLine
context:nil];

CGFloat lineHeight = CGRectGetHeight(singleLineRect);
CGFloat maxHeight = lineHeight * cell.theLabel.numberOfLines;

// Now you can call the method appropriately
CGRect bodyBounds = [labelContent boundingRectWithSize:CGSizeMake(width, maxHeight)
options:(NSStringDrawingUsesLineFragmentOrigin|NSStringDrawingTruncatesLastVisibleLine)
attributes:attributes context:nil];

return CGRectGetHeight(bodyBounds) + heightOfObjectsOnTopOfLabel + heightOfObjectBelowLabel;
}

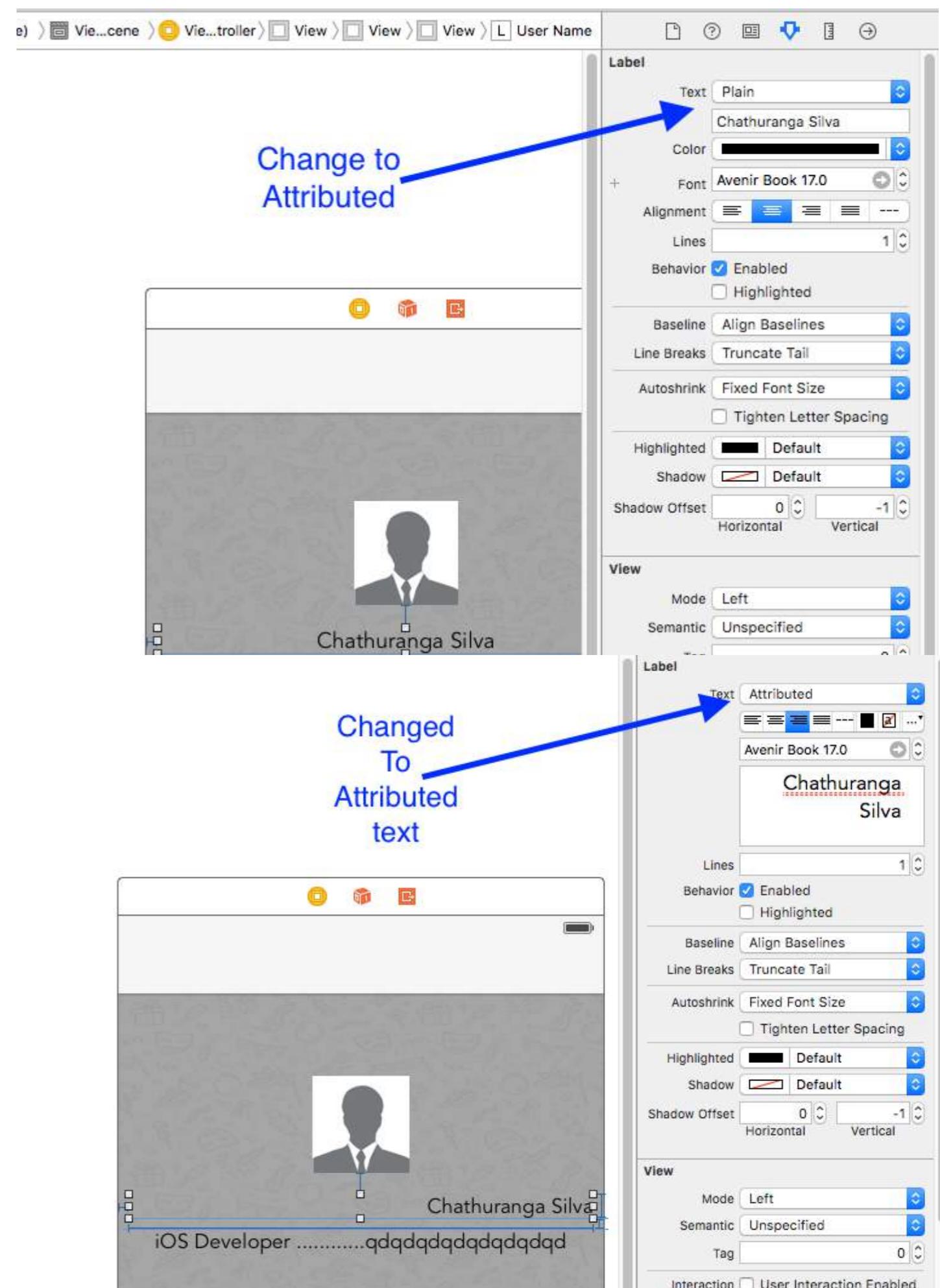
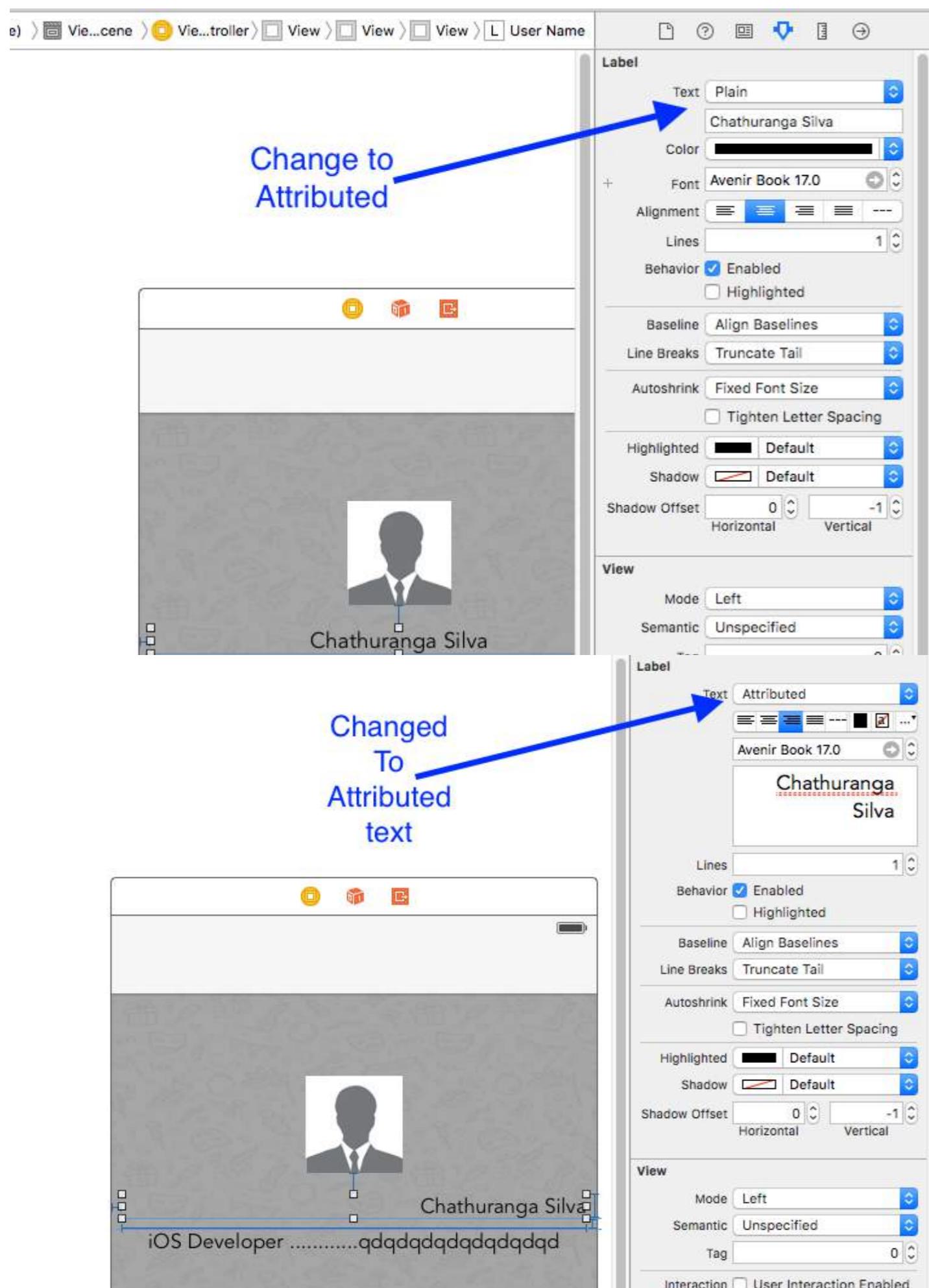
```

## Section 2.9: Label Attributed Text

### 01. Underline Text :- Single/Double Line , Strike Through :- Single/Double Line

#### Step 1

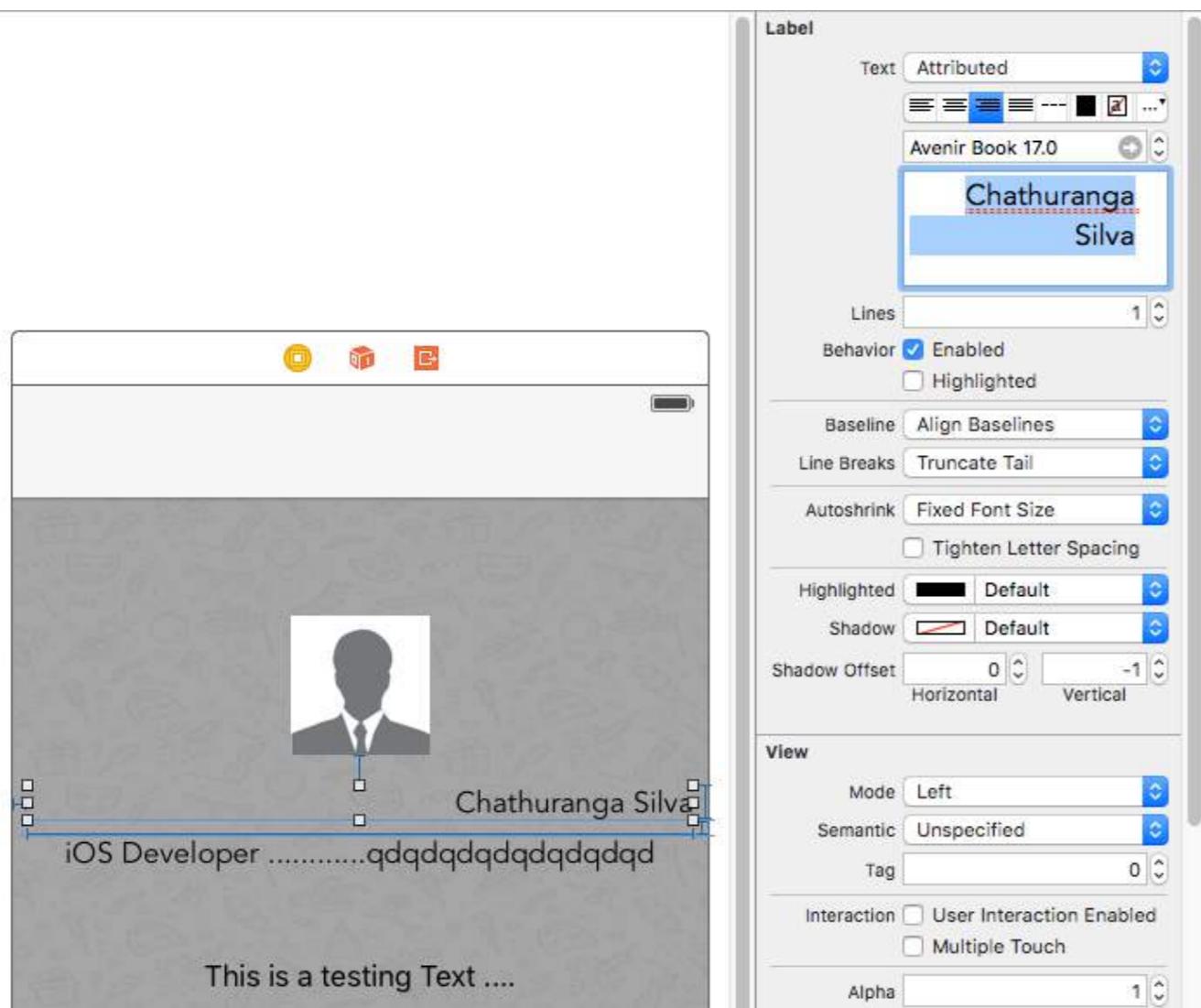
Select the Label and change the label type Plain to Attributed



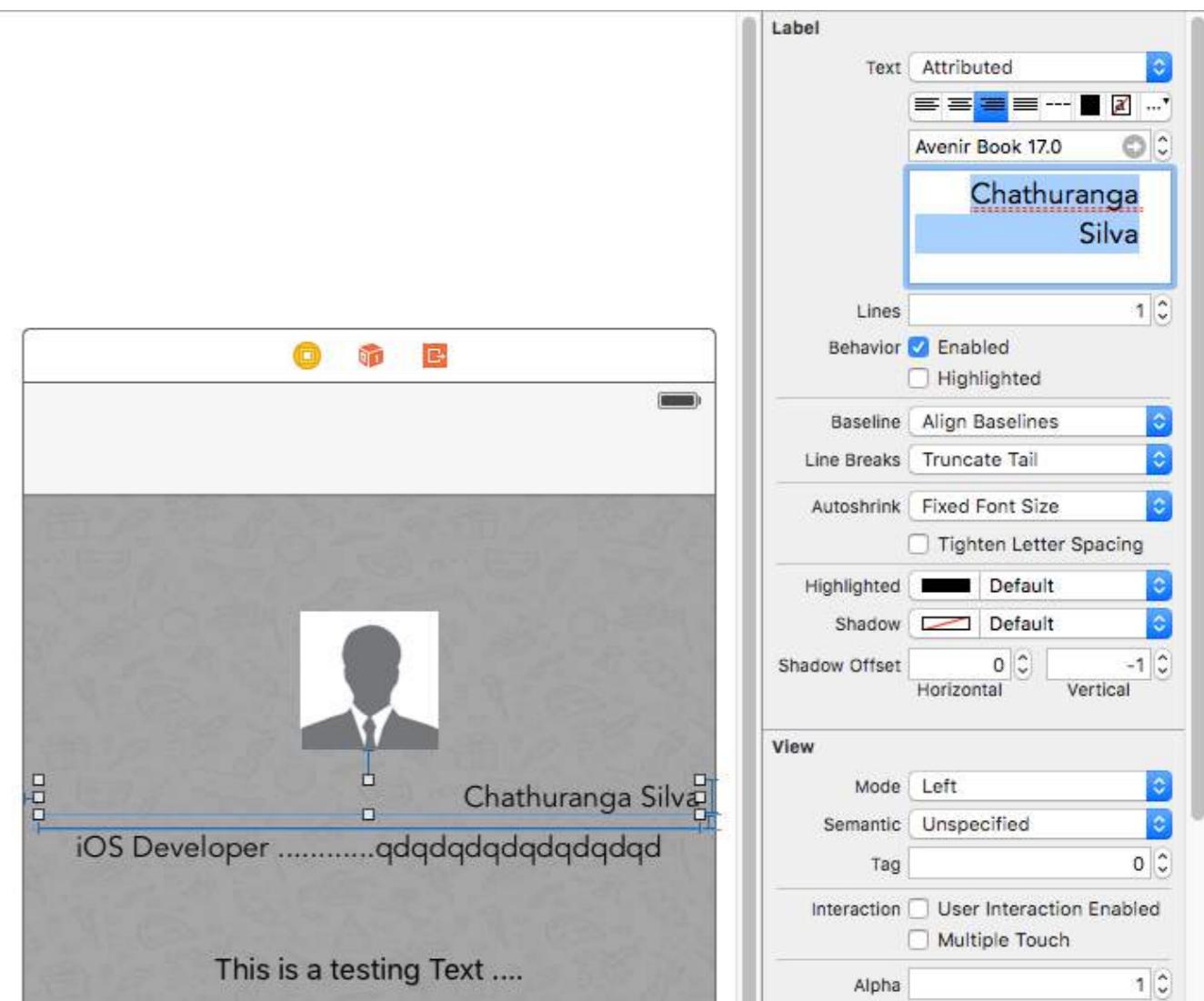
## 步骤 2

## Step 2

点击标签文本并右键点击



Click the label text and Right click

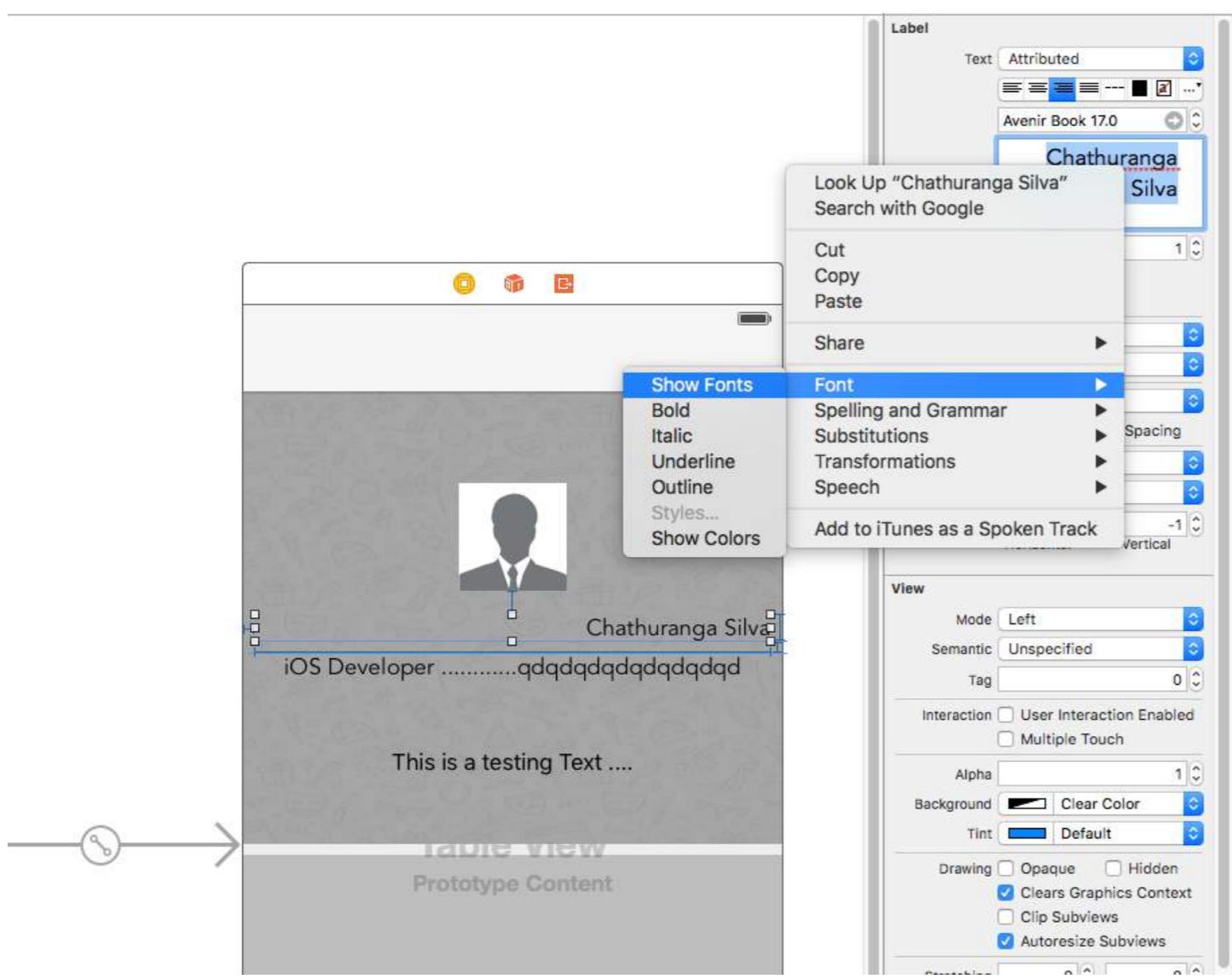


步骤 3

然后点击字体 -> 显示字体

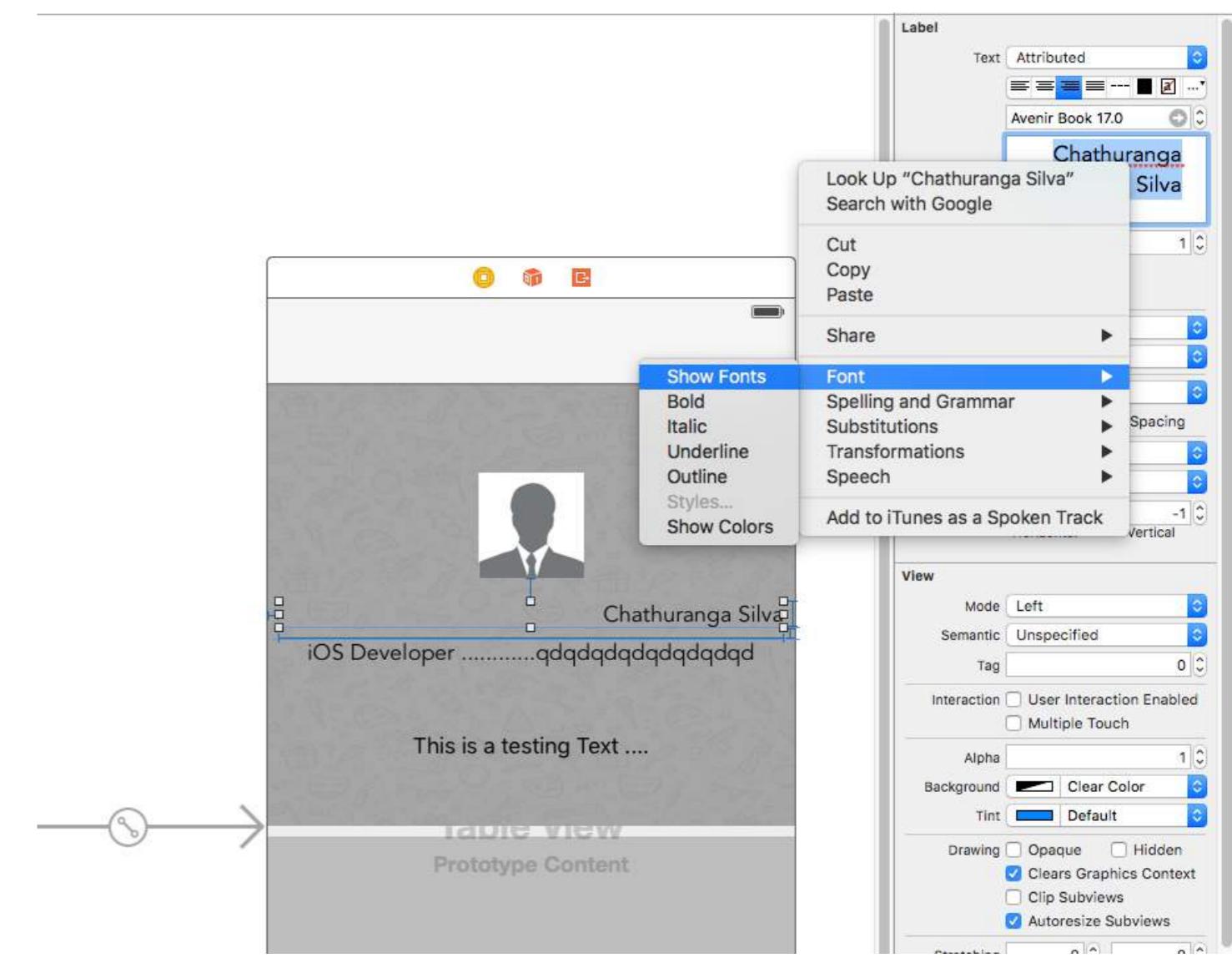
Step 3

Then click Font -> Show Fonts



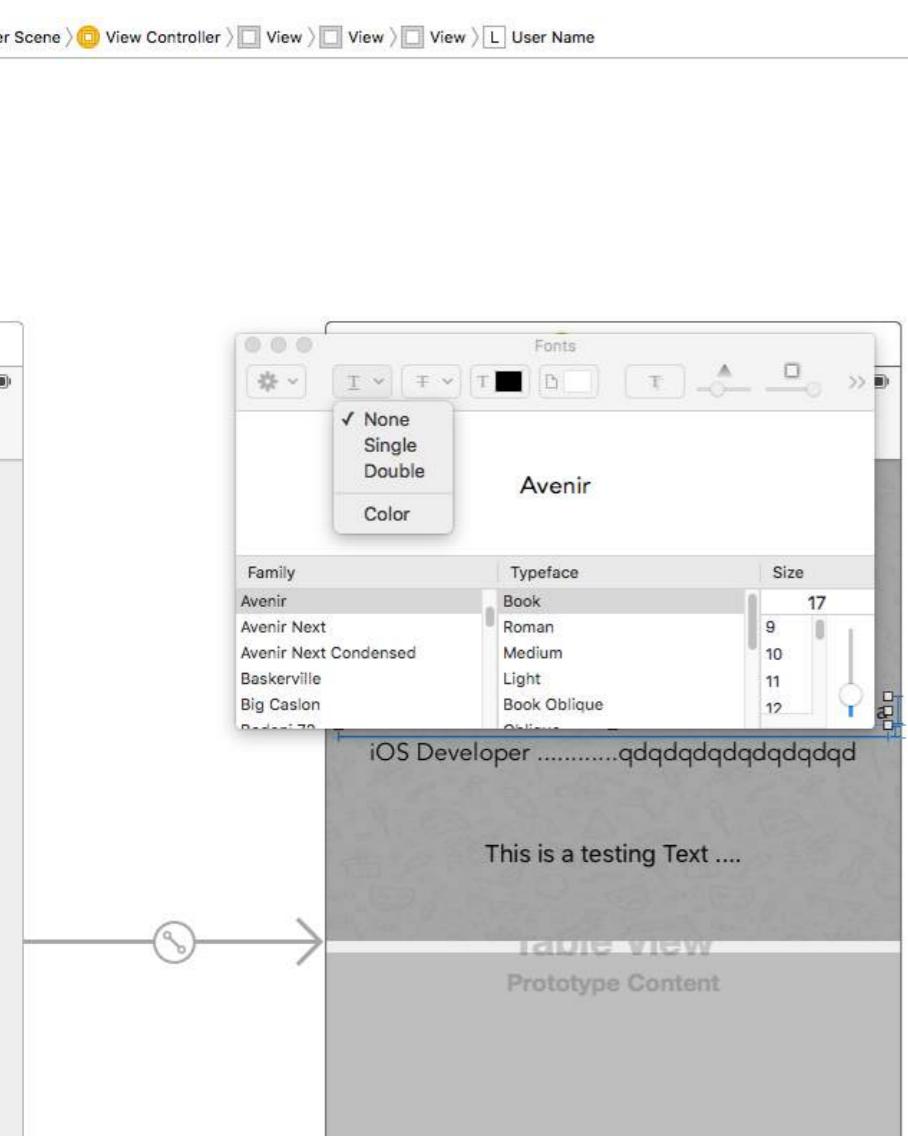
#### 步骤 4

然后会显示字体视图，点击下划线按钮使文本带下划线，或点击删除线按钮使文本带删除线。并选择单线或双线。

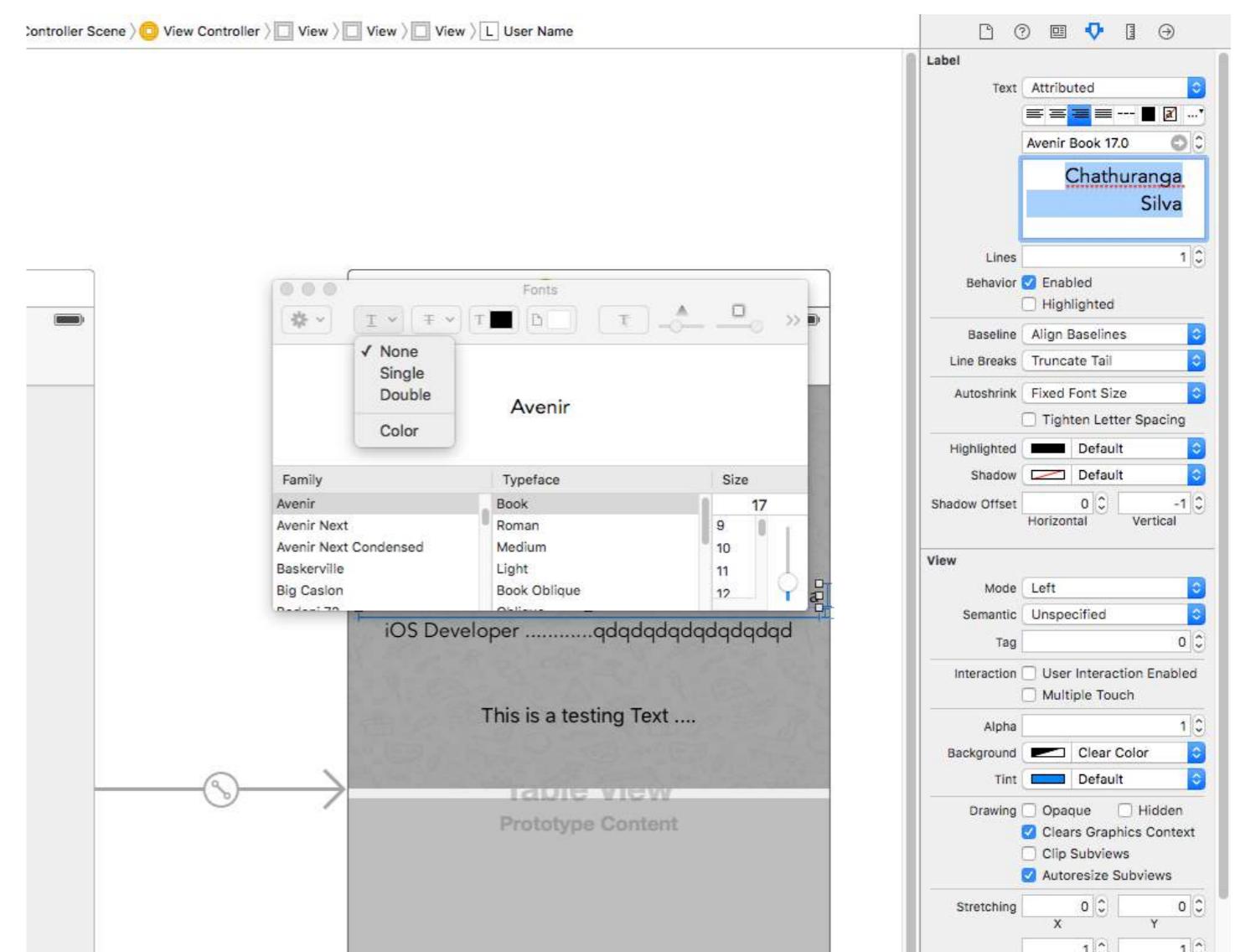


#### Step 4

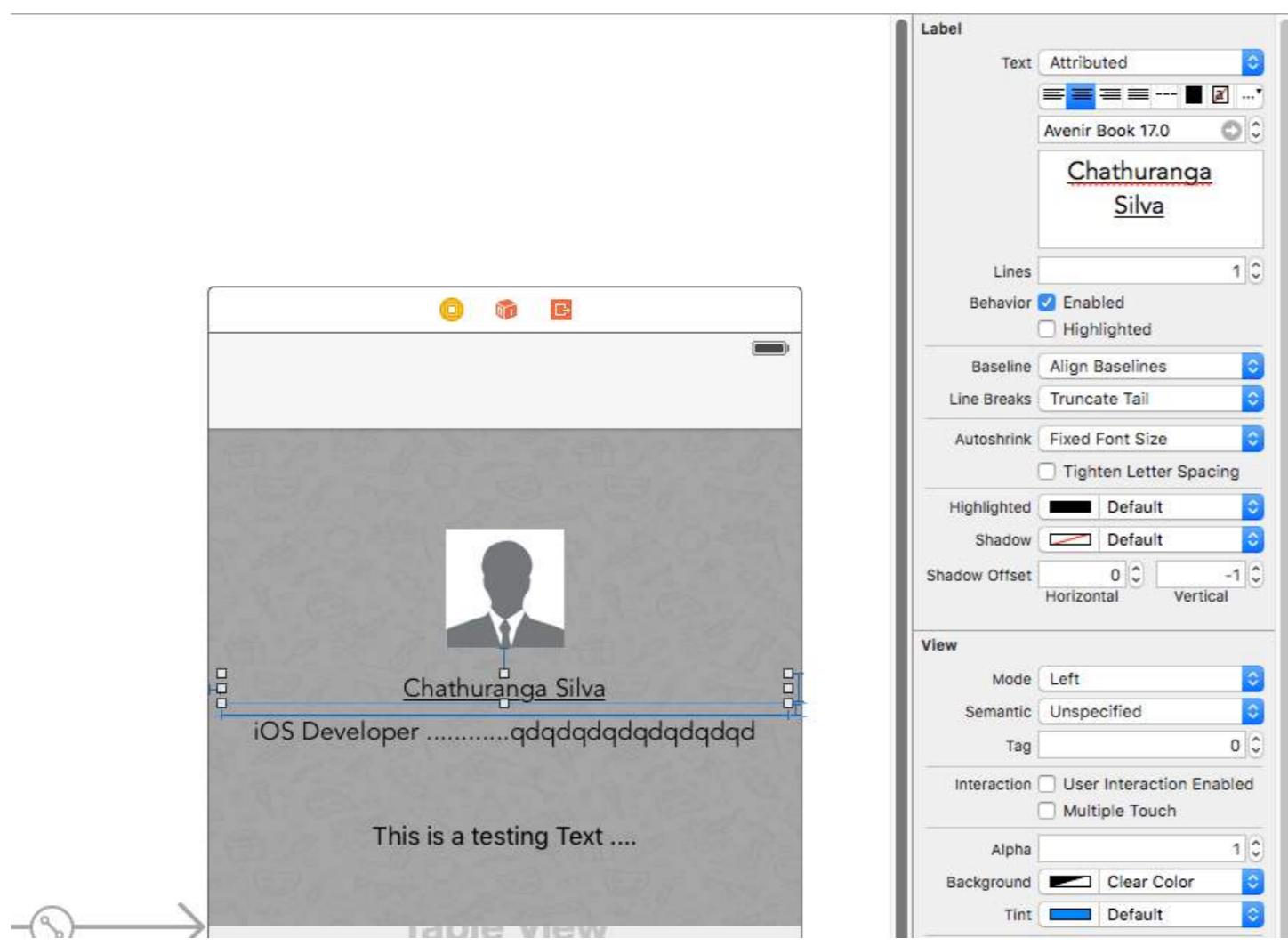
Then font view will show up and click underline button to make text underline or click strikethrough button to make the text strikethrough. And select single line or double line.



最后点击回车，标签将根据您的选择显示为下划线或删除线。

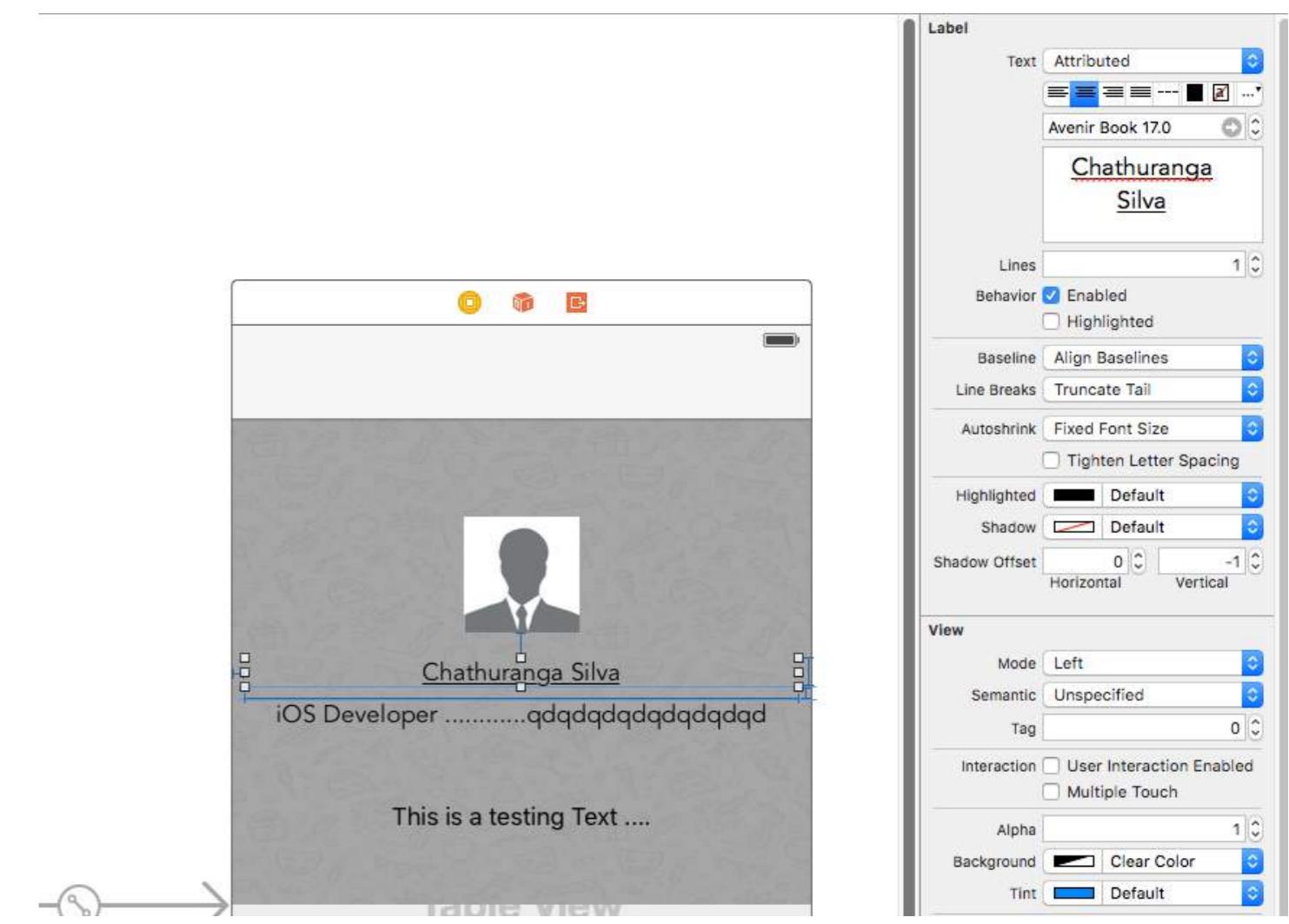


Finally click enter and label will be shown underline or strikethrough according to your selection.



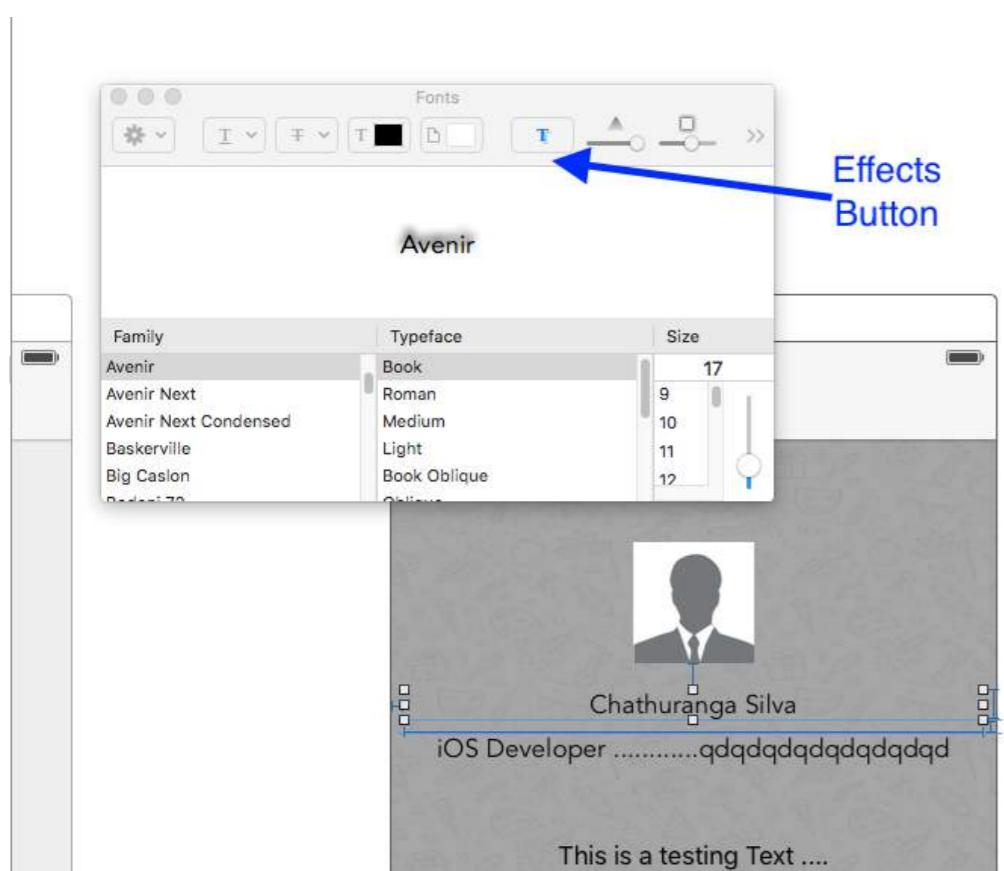
## 02. 添加文字阴影/背景模糊效果

按照上述描述获取字体视图并点击效果按钮。

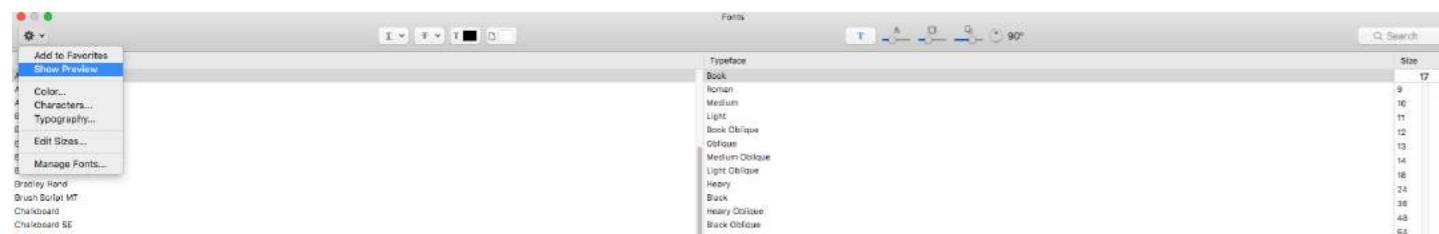
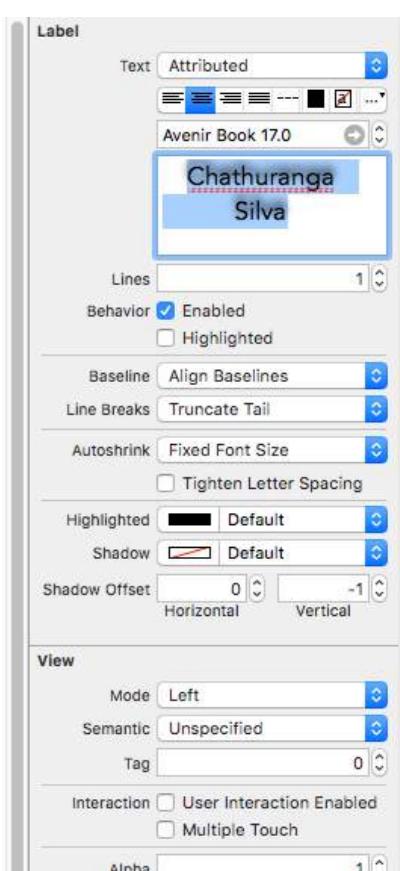
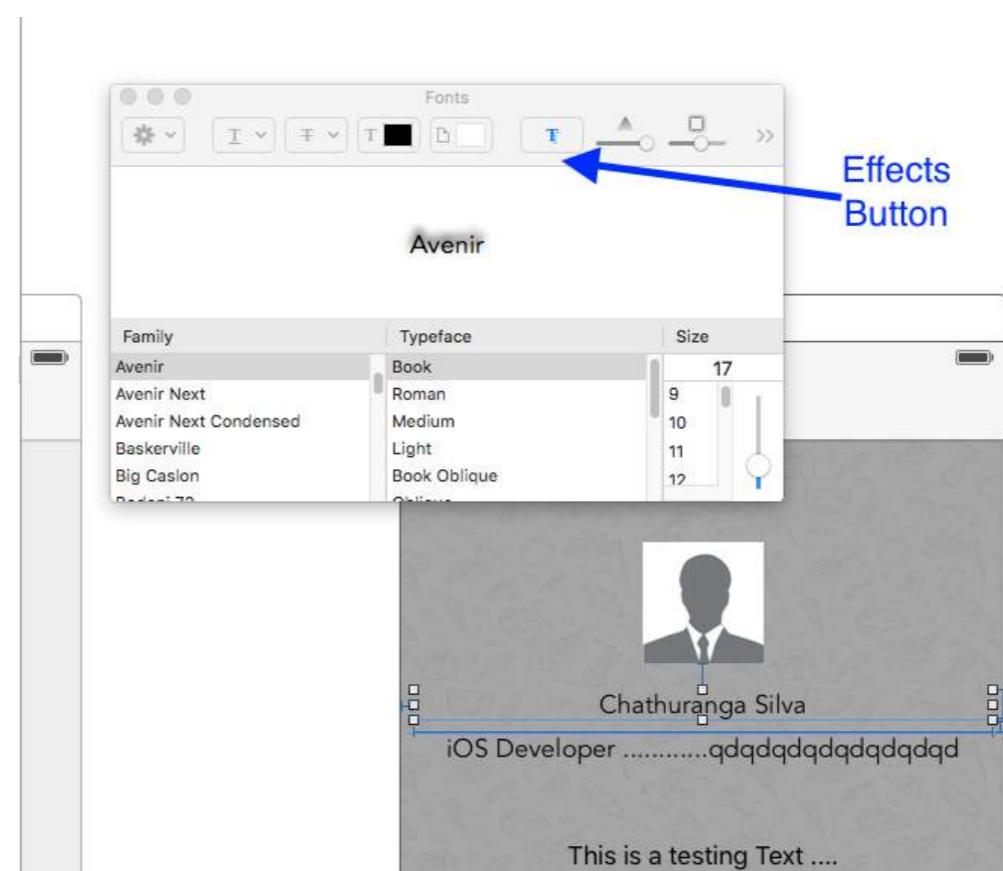
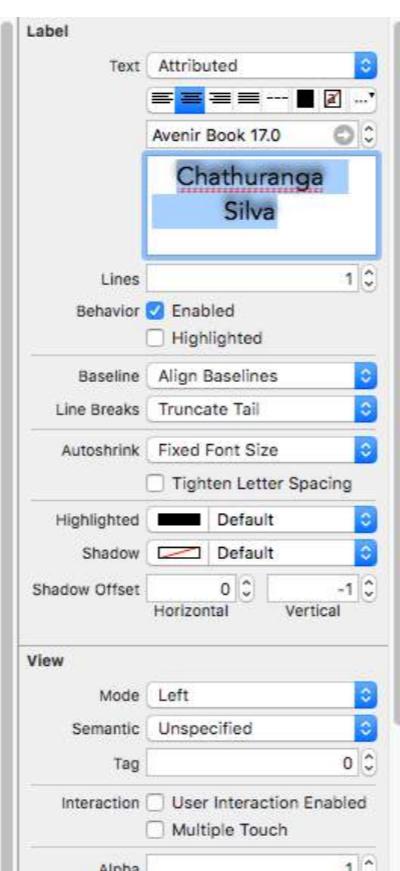


## 02. Add text shadow/background blur effects

Get the Font view as the above described and click the effects button.



如果看不到预览，请在设置中点击显示图像



最后根据你的喜好更改阴影和偏移。



Finally change shadow and offset according to your preferences.



## 第2.10节：可点击标签

**注意：**在大多数情况下，使用UIButton比制作一个可点击的UILabel更好。只有在你确定出于某种原因不想使用UIButton时，才使用本示例。

1. 创建标签
2. 启用用户交互
3. 添加UITapGestureRecognizer

创建可点击UILabel的关键是启用用户交互。

### Swift

```
let label = UILabel()
```

## Section 2.10: Clickable Label

**NOTE:** In most cases, it is better to use a `UIButton` instead of making a `UILabel` you can tap on. Only use this example, if you are sure, that you don't want to use a `UIButton` for some reason.

1. Create label
2. Enable user interaction
3. Add UITapGestureRecognizer

The key to create a clickable `UILabel` is to enable user interaction.

### Swift

```
let label = UILabel()
```

```

label.userInteractionEnabled = true

let gesture = UITapGestureRecognizer(target: self, action: #selector(labelClicked(_:)))
label.addGestureRecognizer(gesture)

```

#### Objective-C

```

UILabel *label = [[UILabel alloc] init];
[label setUserInteractionEnabled:YES];

UITapGestureRecognizer* gesture = [[UITapGestureRecognizer alloc] initWithTarget:self
action:@selector(labelClicked:)];
[label addGestureRecognizer:gesture];

```

#### 在Storyboard的属性检查器中设置“userInteractionEnabled”

你可以不用代码，直接在Storyboard中选择UILabel并勾选该选项：



```

label.userInteractionEnabled = true

```

```

let gesture = UITapGestureRecognizer(target: self, action: #selector(labelClicked(_:)))
label.addGestureRecognizer(gesture)

```

#### Objective-C

```

UILabel *label = [[UILabel alloc] init];
[label setUserInteractionEnabled:YES];

UITapGestureRecognizer* gesture = [[UITapGestureRecognizer alloc] initWithTarget:self
action:@selector(labelClicked:)];
[label addGestureRecognizer:gesture];

```

#### Setting "userInteractionEnabled" in storyboard's attributes inspector

Instead of using code, you can select the UILabel inside the storyboard and check the option:



## 第2.11节：使用约束实现可变高度

你可以使用自动布局制作一个动态高度的UILabel。

你需要将numberOfLines设置为零 (0) ，并通过设置一个关系类型为.GreaterThanOrEqualTo的高度约束来添加最小高度

版本 ≥ iOS 6

#### Swift

```

label.numberOfLines = 0

let heightConstraint = NSLayoutConstraint(
    item: label,
    attribute: .Height,
    relatedBy: .GreaterThanOrEqualTo,
    toItem: nil,
    attribute: .NotAnAttribute,
    multiplier: 0,
    constant: 20
)

label.addConstraint(heightConstraint)

```

版本 ≥ iOS 9

#### Swift

```

label.numberOfLines = 0
label.translatesAutoresizingMaskIntoConstraints = false
label.heightAnchor.constraintGreaterThanOrEqualToConstant(20).active = true

```

## Section 2.11: Variable height using constraints

You can make an UILabel with a dynamic height using auto layout.

You need to set the numberOfLines to zero (0), and add a minimal height by setting up a constraints with a relation of type .GreaterThanOrEqualTo on the .Height attribute

Version ≥ iOS 6

#### Swift

```

label.numberOfLines = 0

let heightConstraint = NSLayoutConstraint(
    item: label,
    attribute: .Height,
    relatedBy: .GreaterThanOrEqualTo,
    toItem: nil,
    attribute: .NotAnAttribute,
    multiplier: 0,
    constant: 20
)

```

label.addConstraint(heightConstraint)

Version ≥ iOS 9

#### Swift

```

label.numberOfLines = 0
label.translatesAutoresizingMaskIntoConstraints = false
label.heightAnchor.constraintGreaterThanOrEqualToConstant(20).active = true

```

## 第2.12节：换行模式

#### 使用代码

## Section 2.12: LineBreakMode

#### Using code

`UILabel.lineBreakMode: NSLineBreakMode`

## Swift

`label.lineBreakMode = .ByTruncatingTail`

- `.ByWordWrapping`
- `.ByCharWrapping`
- `.ByClipping`
- `.ByTruncatingHead`
- `.ByTruncatingTail`
- `.ByTruncatingMiddle`

## Swift 3

`label.lineBreakMode = .byTruncatingTail`

- `.byWordWrapping`
- `.byCharWrapping`
- `.byClipping`
- `.byTruncatingHead`
- `.byTruncatingTail`
- `.byTruncatingMiddle`

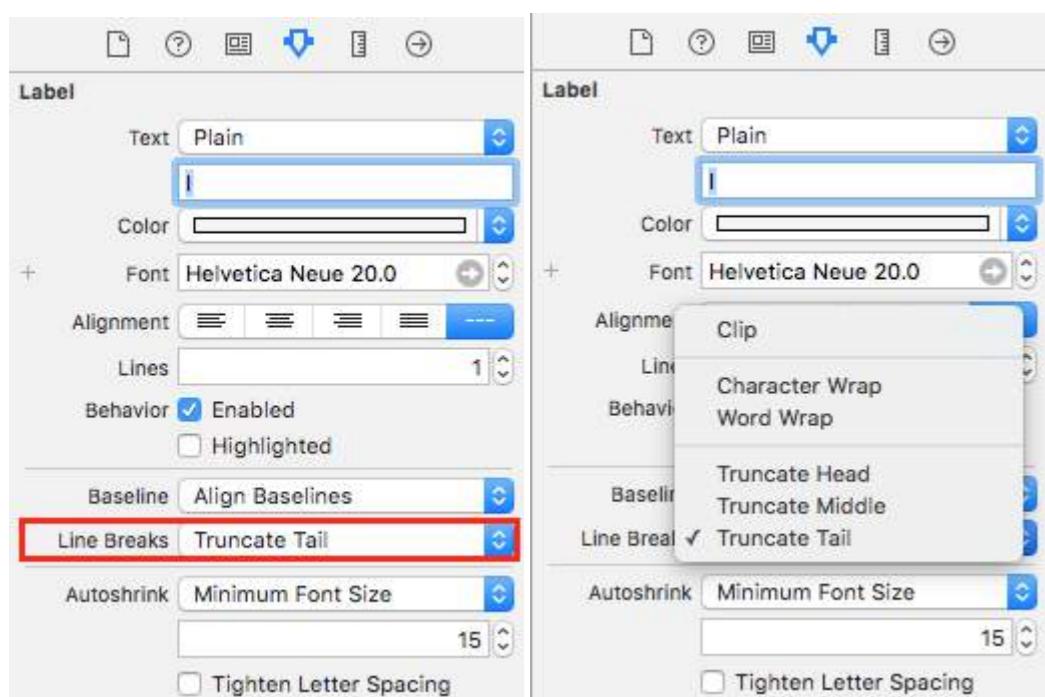
## Objective-C

`[label setLineBreakMode:NSUTFLineBreakByTruncatingTail];`

- `NSLineBreakByWordWrapping`
- `NSLineBreakByCharWrapping`
- `NSLineBreakByClipping`
- `NSLineBreakByTruncatingHead`
- `NSLineBreakByTruncatingTail`
- `NSLineBreakByTruncatingMiddle`

## 使用故事板

这也可以在UILabel的属性检查器中设置：



`UILabel.lineBreakMode: NSLineBreakMode`

## Swift

`label.lineBreakMode = .ByTruncatingTail`

- `.ByWordWrapping`
- `.ByCharWrapping`
- `.ByClipping`
- `.ByTruncatingHead`
- `.ByTruncatingTail`
- `.ByTruncatingMiddle`

## Swift 3

`label.lineBreakMode = .byTruncatingTail`

- `.byWordWrapping`
- `.byCharWrapping`
- `.byClipping`
- `.byTruncatingHead`
- `.byTruncatingTail`
- `.byTruncatingMiddle`

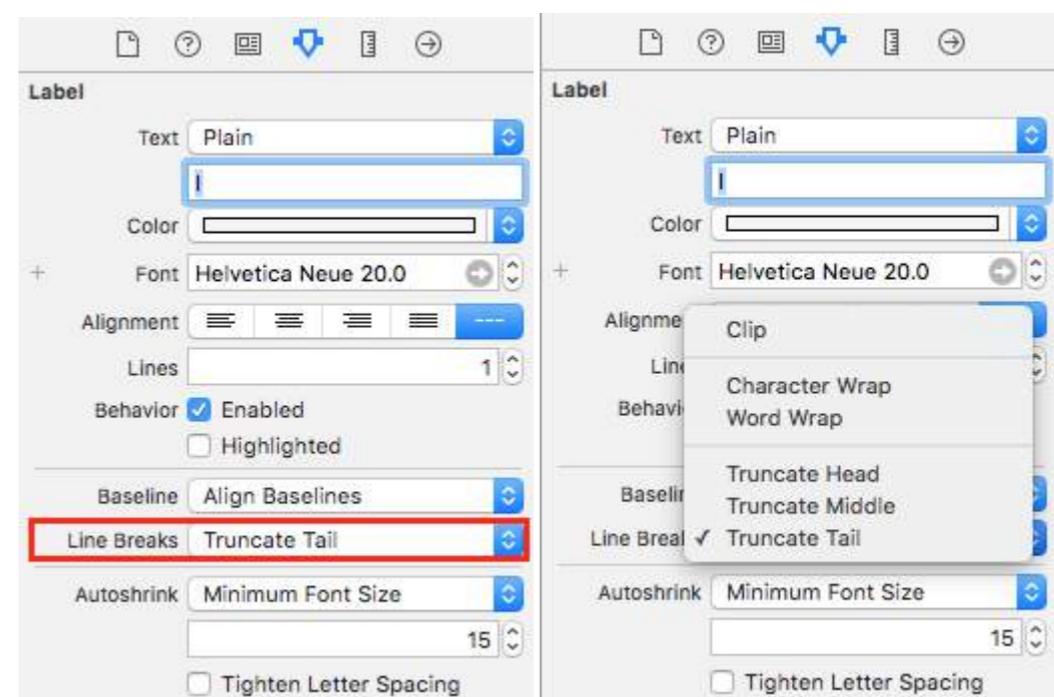
## Objective-C

`[label setLineBreakMode:NSUTFLineBreakByTruncatingTail];`

- `NSLineBreakByWordWrapping`
- `NSLineBreakByCharWrapping`
- `NSLineBreakByClipping`
- `NSLineBreakByTruncatingHead`
- `NSLineBreakByTruncatingTail`
- `NSLineBreakByTruncatingMiddle`

## Using storyboard

This can also be set in the attributes inspector of a UILabel:



## 常量

- 换行 - 换行发生在单词边界，除非单词本身无法放在一行内
- 字符换行 - 换行发生在第一个无法放下的字符之前
- 裁剪 - 线条不会超出文本容器的边缘绘制截断开头 - 该行显示为末尾适合
- 容器，且行首缺失的文本用省略号符号表示
- 截断尾部——该行显示为开头内容适合容器，行尾缺失的文本用省略号符号表示
- 中间截断——该行显示方式使开头和结尾内容适合容器，中间缺失的文本用省略号符号表示

## 第2.13节：为文本添加阴影

### Swift

```
label1.layer.shadowOffset = CGSize(width: 3, height: 3)
label1.layer.shadowOpacity = 0.7
label1.层.阴影半径 = 2
```

### Swift 3

```
label1.layer.shadowOffset = CGSize(width: 3, height: 3)
label1.layer.shadowOpacity = 0.7
label1.层.阴影半径 = 2
```

### Objective-C

```
label1.layer.shadowOffset = CGSizeMake(3, 3);
label1.layer.shadowOpacity = 0.7;
label1.层.阴影半径 = 2;
```

I Like My Cat

## 第2.14节：更改现有标签中的文本

更改现有UILabel的文本可以通过访问并修改UILabel的text属性来完成。这可以直接使用String字面量完成，也可以间接使用变量完成。

### 使用String字面量设置文本

#### Swift

```
label.text = "新的文本"
```

#### Objective-C

```
// 点符号表示法
label.text = @"新的文本";
```

## Constants

- Word Wrapping - wrapping occurs at word boundaries, unless the word itself doesn't fit on a single line
- Char Wrapping - wrapping occurs before the first character that doesn't fit
- Clipping - lines are simply not drawn past the edge of the text container
- Truncating Head - the line is displayed so that the end fits in the container and the missing text at the beginning of the line is indicated by an ellipsis glyph
- Truncating Tail - the line is displayed so that the beginning fits in the container and the missing text at the end of the line is indicated by an ellipsis glyph
- Truncating Middle - the line is displayed so that the beginning and end fit in the container and the missing text in the middle is indicated by an ellipsis glyph

## Section 2.13: Add shadows to text

### Swift

```
label1.layer.shadowOffset = CGSize(width: 3, height: 3)
label1.layer.shadowOpacity = 0.7
label1.layer.shadowRadius = 2
```

### Swift 3

```
label1.layer.shadowOffset = CGSize(width: 3, height: 3)
label1.layer.shadowOpacity = 0.7
label1.layer.shadowRadius = 2
```

### Objective-C

```
label1.layer.shadowOffset = CGSizeMake(3, 3);
label1.layer.shadowOpacity = 0.7;
label1.layer.shadowRadius = 2;
```

I Like My Cat

## Section 2.14: Changing Text in an Existing Label

Changing the text of an existing `UILabel` can be done by accessing and modifying the `text` property of the `UILabel`. This can be done directly using `String` literals or indirectly using variables.

### Setting the text with String literals

#### Swift

```
label.text = "the new text"
```

#### Objective-C

```
// Dot Notation
label.text = @"the new text";
```

```
// 消息模式  
[label setText:@"新的文本"];
```

#### 使用变量设置文本

##### Swift

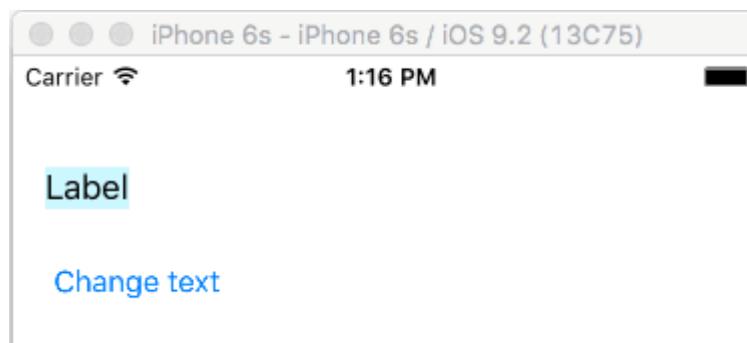
```
let stringVar = "基本字符串变量"  
label.text = stringVar
```

##### Objective-C

```
NSString * stringVar = @"基本字符串变量";  
  
// 点符号表示法  
label.text = stringVar;  
  
// 消息模式  
[label setText: stringVar];
```

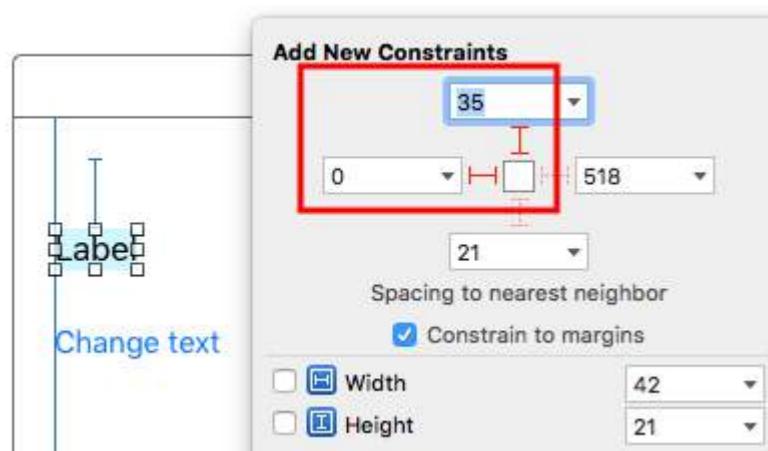
## 第2.15节：自动调整标签大小以适应文本

此示例展示了当文本内容变化时，标签的宽度如何自动调整。



#### 固定左边和顶部边缘

只需使用自动布局为标签的左侧和顶部添加约束。



之后它会自动调整大小。

#### 注意事项

- 此示例来自这个Stack Overflow回答。

```
// Message Pattern  
[label setText:@"the new text"];
```

#### Setting the text with a variable

##### Swift

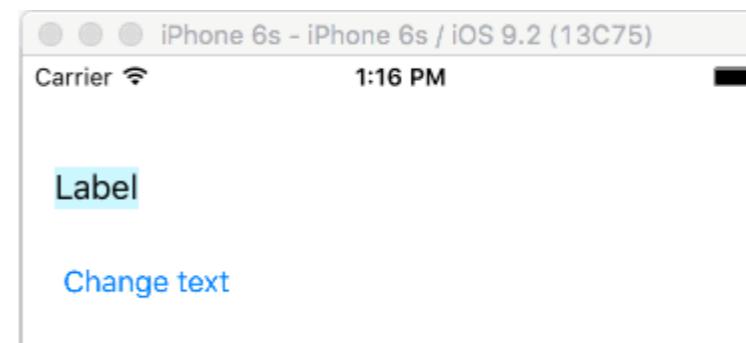
```
let stringVar = "basic String var"  
label.text = stringVar
```

##### Objective-C

```
NSString * stringVar = @"basic String var";  
  
// Dot Notation  
label.text = stringVar;  
  
// Message Pattern  
[label setText: stringVar];
```

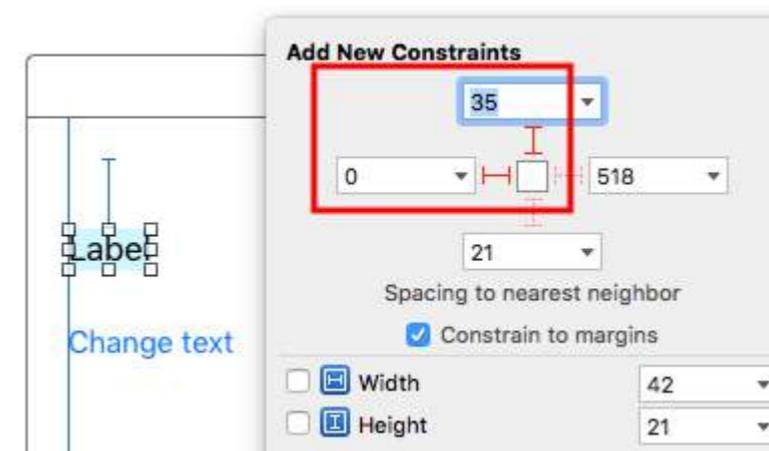
## Section 2.15: Auto-size label to fit text

This example shows how a label's width can automatically resize when the text content changes.



#### Pin the left and top edges

Just use auto layout to add constraints to pin the left and top sides of the label.



After that it will automatically resize.

#### Notes

- This example comes from [this Stack Overflow answer](#).

- 不要为宽度和高度添加约束。标签的大小是基于其文本内容的固有尺寸。
- 使用自动布局时无需设置sizeToFit。示例项目的完整代码如下：

```
import UIKit
class ViewController: UIViewController {

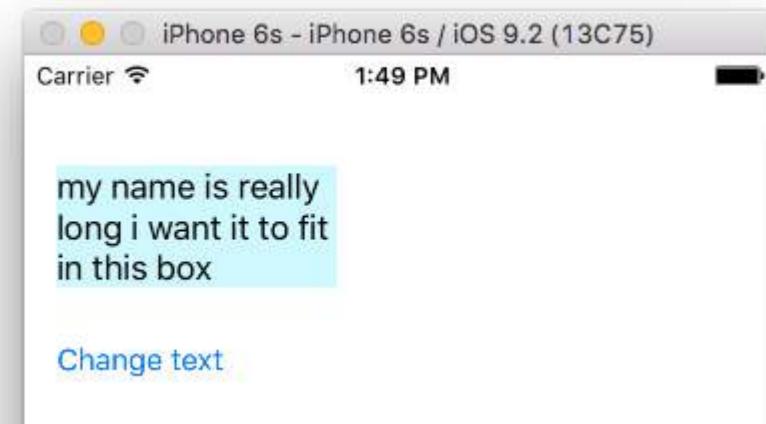
    @IBOutlet weak var myLabel: UILabel!

    @IBAction func changeTextButtonTapped(sender: UIButton) {
        myLabel.text = "我的名字非常长，我希望它能适应这个框"
    }
}
```

- 此方法也可用于像此示例中那样正确地水平排列多个标签。



- 如果你希望标签自动换行，则在IB中将行数设置为0，并添加 `myLabel.preferredMaxLayoutWidth = 150 // 或者代码中任意值。` (按钮也被固定在标签的底部，因此当标签高度增加时按钮会向下移动。)



## 第2.16节：严格根据文本和

字体获取UILabel的尺寸

`NSString` 提供了方法 `boundingRectWithSize`，可用于预测基于文本和字体的 `UILabel` 的最终 `CGSize`，无需创建 `UILabel`

### Objective-C

```
[[text boundingRectWithSize:maxSize options:(NSStringDrawingTruncatesLastVisibleLine |
NSStringDrawingUsesLineFragmentOrigin) attributes:@{NSFontAttributeName: fontName} context:nil]
size];
```

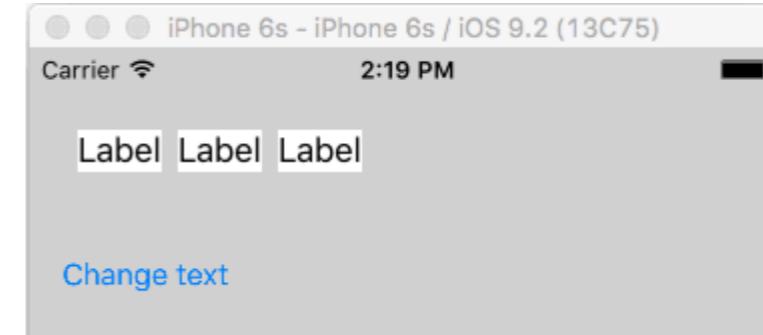
- Don't add constraints for the width and height. Labels have an *intrinsic* size based on their text content.
- No need to set `sizeToFit` when using auto layout. The complete code for the example project is here:

```
import UIKit
class ViewController: UIViewController {

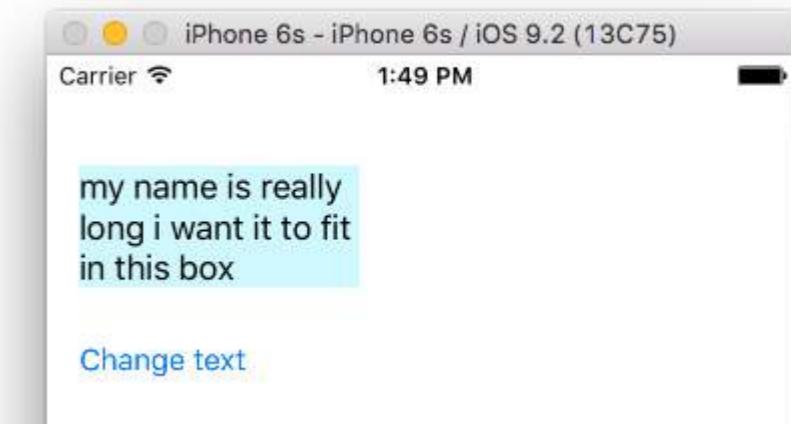
    @IBOutlet weak var myLabel: UILabel!

    @IBAction func changeTextButtonTapped(sender: UIButton) {
        myLabel.text = "my name is really long i want it to fit in this box"
    }
}
```

- This method can also be used to correctly space multiple labels horizontally as in [this example](#).



- If you want your label to line wrap then set the number of lines to 0 in IB and add `myLabel.preferredMaxLayoutWidth = 150 // or whatever` in code. (The button is also pinned to the bottom of the label so that it will move down when the label height increased.)



## Section 2.16: Get UILabel's size strictly based on its text and font

`NSString` provides method `boundingRectWithSize` which can be used to predict the resulting `CGSize` of a `UILabel` based on its text and font without the need of creating a `UILabel`

### Objective-C

```
[[text boundingRectWithSize:maxSize options:(NSStringDrawingTruncatesLastVisibleLine |
NSStringDrawingUsesLineFragmentOrigin) attributes:@{NSFontAttributeName: fontName} context:nil]
size];
```

## Swift

```
let nsText = text as NSString?
nsText?.boundingRectWithSize(maxSize, options: [.TruncatesLastVisibleLine,
.UsesLineFragmentOrigin], attributes: [NSFontAttributeName: fontName], context: nil).size
```

## Swift

创建标签和标签高度约束的IBOutlet。在你将文本赋值给标签的地方添加以下代码。

```
@IBOutlet var lblDescriptionHeightConstration: NSLayoutConstraint!
@IBOutlet weak var lblDescription: UILabel!

let maxWidth = UIScreen.mainScreen().bounds.size.width - 40
let sizeOfLabel = self.lblDesc.sizeThatFits(CGSize(width: maxWidth, height: CGFloat.max))
self.lblDescriptionHeightConstration.constant = sizeOfLabel.height
```

注意：“40”是屏幕左右两侧的间距。

## 第2.17节：高亮及高亮文本颜色

### Objective-C

```
UILabel *label = [[UILabel alloc] init];
label.highlighted = YES;
label.highlightedTextColor = [UIColor redColor];
```

## Swift

```
let label = UILabel()
label.highlighted = true
label.highlightedTextColor = UIColor.redColor()
```

## Swift 3

```
let label = UILabel()
label.isHighlighted = true
label.highlightedTextColor = UIColor.red
```

## 第2.18节：两端对齐文本

## Swift

```
let sampleText = "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in
voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
```

```
// 创建标签
let label = UILabel(frame: CGRectMake(0, 0, view.frame.size.width, 400))
label.numberOfLines = 0
label.lineBreakMode = NSLineBreakMode.ByWordWrapping
```

```
// 通过段落样式实现文本两端对齐
let paragraphStyle = NSMutableParagraphStyle()
paragraphStyle.alignment = NSTextAlignment.Justified
```

## Swift

```
let nsText = text as NSString?
nsText?.boundingRectWithSize(maxSize, options: [.TruncatesLastVisibleLine,
.UsesLineFragmentOrigin], attributes: [NSFontAttributeName: fontName], context: nil).size
```

## Swift

Create Label and label Height constraint outlet. Add below code where you will assign text to label.

```
@IBOutlet var lblDescriptionHeightConstration: NSLayoutConstraint!
@IBOutlet weak var lblDescription: UILabel!

let maxWidth = UIScreen.mainScreen().bounds.size.width - 40
let sizeOfLabel = self.lblDesc.sizeThatFits(CGSize(width: maxWidth, height: CGFloat.max))
self.lblDescriptionHeightConstration.constant = sizeOfLabel.height
```

Note: "40" is the space of left and right side of screen.

## Section 2.17: Highlighted and Highlighted Text Color

### Objective-C

```
UILabel *label = [[UILabel alloc] init];
label.highlighted = YES;
label.highlightedTextColor = [UIColor redColor];
```

## Swift

```
let label = UILabel()
label.highlighted = true
label.highlightedTextColor = UIColor.redColor()
```

## Swift 3

```
let label = UILabel()
label.isHighlighted = true
label.highlightedTextColor = UIColor.red
```

## Section 2.18: Justify Text

## Swift

```
let sampleText = "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in
voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
```

```
// Create label
let label = UILabel(frame: CGRectMake(0, 0, view.frame.size.width, 400))
label.numberOfLines = 0
label.lineBreakMode = NSLineBreakMode.ByWordWrapping
```

```
// Justify text through paragraph style
let paragraphStyle = NSMutableParagraphStyle()
paragraphStyle.alignment = NSTextAlignment.Justified
```

```

let attributes = [NSParagraphStyleAttributeName: paragraphStyle, NSBaselineOffsetAttributeName: NSNumber(float: 0)]
let attributedString = NSAttributedString(string: sampleText, attributes: attributes)
label.attributedText = attributedString
view.addSubview(label)

```

#### Objective-C

```

NSString *sampleText = @"Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.";

// 创建标签
UILabel *label = [[UILabel alloc] initWithFrame:CGRectMake(0, 0, self.view.frame.size.width, 400)];
label.numberOfLines = 0;
label.lineBreakMode = NSLineBreakByWordWrapping;

// 通过段落样式对齐文本
NSMutableParagraphStyle *paragraphStyle = [[NSMutableParagraphStyle alloc] init];
paragraphStyle.alignment = NSTextAlignmentJustified;
NSAttributedString *attributedString = [[NSAttributedString alloc] initWithString:sampleText
attributes:@{
    NSParagraphStyleAttributeName : paragraphStyle,
    NSBaselineOffsetAttributeName : [NSNumber numberWithFloat:0]
}];
label.attributedText = attributedString;
[self.view addSubview:label];

```

## 第2.19节：根据未知文本长度动态调整标签框架

有时我们需要根据动态内容调整UILabel的大小，而文本长度未知。在本例中，UILabel的宽度固定为280点，高度无限制，假设为9999。根据文本样式和maximumLabelSize估算框架。

#### Objective-C

```

UILabel * label = [[UILabel alloc] init];

NSString *message = @"Some dynamic text for label";

//设置文本和样式（如果有）。
label.text = message;

label.numberOfLines = 0;

CGSize maximumLabelSize = CGSizeMake(280, 9999); //280：标签最大宽度，9999：标签最大高度。

// 使用UILabel的字体信息计算大小
CGSize expectedLabelSize = [label sizeThatFits:maximumLabelSize];

// iOS 7.0中已弃用
//CGSize expectedLabelSize = [message sizeWithFont:label.font constrainedToSize:maximumLabelSize
//lineBreakMode:NSLineBreakByWordWrapping];

// 创建一个填充了 UILabel 框架数据的框架

```

```

let attributes = [NSParagraphStyleAttributeName: paragraphStyle, NSBaselineOffsetAttributeName: NSNumber(float: 0)]
let attributedString = NSAttributedString(string: sampleText, attributes: attributes)
label.attributedText = attributedString
view.addSubview(label)

```

#### Objective-C

```

NSString *sampleText = @"Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.";

// Create label
UILabel *label = [[UILabel alloc] initWithFrame:CGRectMake(0, 0, self.view.frame.size.width, 400)];
label.numberOfLines = 0;
label.lineBreakMode = NSLineBreakByWordWrapping;

// Justify text through paragraph style
NSMutableParagraphStyle *paragraphStyle = [[NSMutableParagraphStyle alloc] init];
paragraphStyle.alignment = NSTextAlignmentJustified;
NSAttributedString *attributedString = [[NSAttributedString alloc] initWithString:sampleText
attributes:@{
    NSParagraphStyleAttributeName : paragraphStyle,
    NSBaselineOffsetAttributeName : [NSNumber numberWithFloat:0]
}];
label.attributedText = attributedString;
[self.view addSubview:label];

```

## Section 2.19: Dynamic label frame from unknown text length

Sometimes we have to resize a UILabel based on dynamic content where the text length is unknown. In this example, width of the UILabel is fixed at 280 points and the height is infinite, lets say 9999. Estimating the frame with respect to the text style and maximumLabelSize.

#### Objective-C

```

UILabel * label = [[UILabel alloc] init];

NSString *message = @"Some dynamic text for label";

//set the text and style if any.
label.text = message;

label.numberOfLines = 0;

CGSize maximumLabelSize = CGSizeMake(280, 9999); //280:max width of label and 9999-max height of label.

// use font information from the UILabel to calculate the size
CGSize expectedLabelSize = [label sizeThatFits:maximumLabelSize];

//Deprecated in iOS 7.0
//CGSize expectedLabelSize = [message sizeWithFont:label.font constrainedToSize:maximumLabelSize
//lineBreakMode:NSLineBreakByWordWrapping];

// create a frame that is filled with the UILabel frame data

```

```
CGRect newFrame = label.frame;  
  
// 调整框架大小为计算后的尺寸  
newFrame.size.height = expectedLabelSize.height;  
  
// 将计算后的框架赋值给 UILabel 框架  
label.frame = newFrame;
```

### Swift

```
var message: String = "Some dynamic text for label"  
// 设置文本及样式 (如有)  
label.text = message  
label.numberOfLines = 0  
var maximumLabelSize: CGSize = CGSize(width: 280, height: 9999)  
var expectedLabelSize: CGSize = label.sizeThatFits(maximumLabelSize)  
// 创建一个填充了 UILabel 框架数据的框架  
var newFrame: CGRect = label.frame  
// 调整框架大小为计算后的尺寸  
newFrame.size.height = expectedLabelSize.height  
// 将计算后的框架赋值给 UILabel 框架  
label.frame = newFrame
```

```
CGRect newFrame = label.frame;  
  
// resizing the frame to calculated size  
newFrame.size.height = expectedLabelSize.height;  
  
// put calculated frame into UILabel frame  
label.frame = newFrame;
```

### Swift

```
var message: String = "Some dynamic text for label"  
//set the text and style if any.  
label.text = message  
label.numberOfLines = 0  
var maximumLabelSize: CGSize = CGSize(width: 280, height: 9999)  
var expectedLabelSize: CGSize = label.sizeThatFits(maximumLabelSize)  
// create a frame that is filled with the UILabel frame data  
var newFrame: CGRect = label.frame  
// resizing the frame to calculated size  
newFrame.size.height = expectedLabelSize.height  
// put calculated frame into UILabel frame  
label.frame = newFrame
```

# 第三章：UILabel 文本下划线

## 3.1 节：使用 Objective C 为 UILabel 文本添加下划线

```
UILabel *label=[[UILabel alloc]initWithFrame:CGRectMake(0, 0, 320, 480)];  
label.backgroundColor=[UIColor lightGrayColor];  
NSMutableAttributedString *attributedString;  
attributedString = [[NSMutableAttributedString alloc] initWithString:@"应用下划线"];  
[attributedString addAttribute:NSUnderlineStyleAttributeName value:@1 range:NSMakeRange(0,  
[attributedString length])];  
[label setAttributedText:attributedString];
```

## 第3.2节：使用Swift在UILabel中添加下划线文本

```
let label = UILabel.init(frame: CGRect(x: 0, y:0, width: 100, height: 40))  
label.backgroundColor = .lightGray  
let attributedString = NSMutableAttributedString.init(string: "应用下划线")  
attributedString.addAttribute(NSUnderlineStyleAttributeName, value: 1, range:  
NSRange.init(location: 0, length: attributedString.length))  
label.attributedText = attributedString
```

# Chapter 3: UILabel text underlining

## Section 3.1: Underlining a text in a UILabel using Objective C

```
UILabel *label=[[UILabel alloc]initWithFrame:CGRectMake(0, 0, 320, 480)];  
label.backgroundColor=[UIColor lightGrayColor];  
NSMutableAttributedString *attributedString;  
attributedString = [[NSMutableAttributedString alloc] initWithString:@"Apply Underlining"];  
[attributedString addAttribute:NSUnderlineStyleAttributeName value:@1 range:NSMakeRange(0,  
[attributedString length])];  
[label setAttributedText:attributedString];
```

## Section 3.2: Underlining a text in UILabel using Swift

```
let label = UILabel.init(frame: CGRect(x: 0, y:0, width: 100, height: 40))  
label.backgroundColor = .lightGray  
let attributedString = NSMutableAttributedString.init(string: "Apply Underlining")  
attributedString.addAttribute(NSUnderlineStyleAttributeName, value: 1, range:  
NSRange.init(location: 0, length: attributedString.length))  
label.attributedText = attributedString
```

# 第4章：UILabel中的attributedText

标签当前显示的样式化文本。

您可以使用attributedText属性在UILabel中添加HTML文本，或自定义单个UILabel文本的不同属性

## 第4.1节：UILabel中的HTML文本

```
NSString * htmlString = @"><html><body> <b> HTML中的示例加粗文本 </b> </body></html>";
NSAttributedString * attrStr = [[NSAttributedString alloc] initWithData:[htmlString
dataUsingEncoding:NSUTFStringEncoding] options:@{ NSDocumentTypeDocumentAttribute:
NSHTMLTextDocumentType } documentAttributes:nil error:nil];

UILabel * yourLabel = [[UILabel alloc] init];
yourLabel.attributedText = attrStr;
```

## 第4.2节：在单个UILabel中为文本设置不同属性

第一步需要执行的是创建一个NSMutableAttributedString对象。我们创建NSMutableAttributedString而不是NSAttributedString的原因是它允许我们向其中追加字符串。

```
NSString *fullStr = @"Hello World!";
NSMutableAttributedString *attString =[[NSMutableAttributedString alloc]initWithString:fullStr];

// 查找文本范围。
NSRange rangeHello = [fullStr rangeOfString:@"Hello"];
NSRange rangeWorld = [fullStr rangeOfString:@"World!"];

// 为Hello添加字体样式
[attString addAttribute: NSFontAttributeName
    value: [UIFont fontWithName:@"Copperplate" size:14]
    range: rangeHello];
// 为Hello添加文本颜色
[attString addAttribute: NSForegroundColorAttributeName
    value: [UIColor blueColor]
    range: rangeHello];

// 为 World! 添加字体样式
[attString addAttribute: NSFontAttributeName
    value: [UIFont fontWithName:@"Chalkduster" size:20]
    range: rangeWorld];
// 为 World! 添加文本颜色
[attString addAttribute: NSForegroundColorAttributeName
    value: [UIColor colorWithRed:(66.0/255.0) green:(244.0/255.0) blue:(197.0/255.0)
alpha:1]
    range: rangeWorld];

// 设置为 UILabel 的 attributedText
UILabel * yourLabel = [[UILabel alloc] initWithFrame:CGRectMake(10, 150, 200, 100)];
yourLabel.attributedText = attString;
[self.view addSubview:yourLabel];
```

输出：

# Chapter 4: attributedText in UILabel

The current styled text that is displayed by the label.

You can add HTML text in `UILabel` using `attributedText` property or customized single `UILabel` text with different property

## Section 4.1: HTML text in UILabel

```
NSString * htmlString = @"><html><body> <b> Example bold text in HTML </b> </body></html>";
NSAttributedString * attrStr = [[NSAttributedString alloc] initWithData:[htmlString
dataUsingEncoding:NSUTFStringEncoding] options:@{ NSDocumentTypeDocumentAttribute:
NSHTMLTextDocumentType } documentAttributes:nil error:nil];

UILabel * yourLabel = [[UILabel alloc] init];
yourLabel.attributedText = attrStr;
```

## Section 4.2: Set different property to text in single UILabel

The first step you need to perform is to create a `NSMutableAttributedString` object. The reason we create a `NSMutableAttributedString` instead of `NSAttributedString` is because it enables us to append string to it.

```
NSString *fullStr = @"Hello World!";
NSMutableAttributedString *attString =[[NSMutableAttributedString alloc]initWithString:fullStr];

// Finding the range of text.
NSRange rangeHello = [fullStr rangeOfString:@"Hello"];
NSRange rangeWorld = [fullStr rangeOfString:@"World!"];

// Add font style for Hello
[attString addAttribute: NSFontAttributeName
    value: [UIFont fontWithName:@"Copperplate" size:14]
    range: rangeHello];
// Add text color for Hello
[attString addAttribute: NSForegroundColorAttributeName
    value: [UIColor blueColor]
    range: rangeHello];

// Add font style for World!
[attString addAttribute: NSFontAttributeName
    value: [UIFont fontWithName:@"Chalkduster" size:20]
    range: rangeWorld];
// Add text color for World!
[attString addAttribute: NSForegroundColorAttributeName
    value: [UIColor colorWithRed:(66.0/255.0) green:(244.0/255.0) blue:(197.0/255.0)
alpha:1]
    range: rangeWorld];

// Set it to UILabel as attributedText
UILabel * yourLabel = [[UILabel alloc] initWithFrame:CGRectMake(10, 150, 200, 100)];
yourLabel.attributedText = attString;
[self.view addSubview:yourLabel];
```

Output :

HELLO World!

HELLO World!

# 第5章：UIButton

[UIButton : UIControl](#) 拦截触摸事件并在被点击时向目标对象发送动作消息。你可以设置按钮的标题、图片及其他外观属性。此外，你可以为每个按钮状态指定不同的外观。

## 第5.1节：创建UIButton

UIButton可以通过指定框架进行初始化：

### Swift

```
let button = UIButton(frame: CGRect(x: x, y: y, width: width, height: height))
```

### Objective C

```
UIButton *button = [[UIButton alloc] initWithFrame:CGRectMake(x, y, width, height)];
```

可以这样创建特定类型的UIButton：

### Swift

```
let button = UIButton(type: .Custom)
```

### Objective C

```
UIButton *button = [UIButton buttonWithType:UIButtonTypeCustom];
```

其中 type 是一个 UIButtonType：

```
enum UIButtonType : Int {  
    case Custom  
    case System  
    case DetailDisclosure  
    case InfoLight  
    case InfoDark  
    case ContactAdd  
    static var RoundedRect: UIButtonType { get }  
}
```

## 第5.2节：将方法附加到按钮

要向按钮添加方法，首先创建一个动作方法：

### Objective-C

```
-(void) someButtonAction{  
    NSLog(@"按钮被点击");  
}
```

### Swift

```
func someButtonAction() {
```

# Chapter 5: UIButton

[UIButton : UIControl](#) intercepts touch events and sends an action message to a target object when it's tapped. You can set the title, image, and other appearance properties of a button. In addition, you can specify a different appearance for each button state.

## Section 5.1: Creating a UIButton

UIButtons can be initialized in a frame:

### Swift

```
let button = UIButton(frame: CGRect(x: x, y: y, width: width, height: height))
```

### Objective C

```
UIButton *button = [[UIButton alloc] initWithFrame:CGRectMake(x, y, width, height)];
```

A specific type of UIButton can be created like this:

### Swift

```
let button = UIButton(type: .Custom)
```

### Objective C

```
UIButton *button = [UIButton buttonWithType:UIButtonTypeCustom];
```

where type is a UIButtonType:

```
enum UIButtonType : Int {  
    case Custom  
    case System  
    case DetailDisclosure  
    case InfoLight  
    case InfoDark  
    case ContactAdd  
    static var RoundedRect: UIButtonType { get }  
}
```

## Section 5.2: Attaching a Method to a Button

To add a method to a button, first create an action method:

### Objective-C

```
-(void) someButtonAction{  
    NSLog(@"Button is tapped");  
}
```

### Swift

```
func someButtonAction() {
```

```
    print("按钮被点击")
}
```

现在要将此动作方法添加到您的按钮，您需要编写以下代码行：

#### Objective C

```
[yourButtonInstance addTarget:self action:@selector(someButtonAction)
forControlEvents:UIControlEventTouchUpInside];
```

#### Swift

```
yourButtonInstance.addTarget(self, action: #selector(someButtonAction), forControlEvents:
.touchesUpInside)
```

对于 ControlEvents，所有 ENUM UIControlEvents 的成员均有效。

## 第5.3节：设置字体

#### Swift

```
myButton.titleLabel?.font = UIFont(name: "YourFontName", size: 20)
```

#### Objective C

```
myButton.titleLabel.font = [UIFont fontWithName:@"YourFontName" size:20];
```

## 第5.4节：设置图片

#### Swift

```
button.setImage(UIImage(named:"test-image"), forState: .normal)
```

#### Objective C

```
[self.button setImage:[UIImage imageNamed:@"test-image"] forState:UIControlStateNormal];
```

#### 多重控制状态

你也可以为多个UIControlStates设置同一张图片，例如为Selected和Highlighted状态设置相同的图片：

#### Swift

```
button.setImage(UIImage(named:"test-image"), forState:[.selected, .highlighted])
```

#### Objective C

```
[self.button setImage:[UIImage imageNamed:@"test-image"]
forState:UIControlStateSelected|UIControlStateHighlighted];
```

## 第5.5节：严格根据文本和

字体获取UIButton的大小

要根据字体获取UIButton文本的准确大小，请使用函数intrinsicContentSize。

```
    print("Button is tapped")
}
```

Now to add this action method to your button, you have to write following line of code:

#### Objective C

```
[yourButtonInstance addTarget:self action:@selector(someButtonAction)
forControlEvents:UIControlEventTouchUpInside];
```

#### Swift

```
yourButtonInstance.addTarget(self, action: #selector(someButtonAction), forControlEvents:
.touchesUpInside)
```

For ControlEvents, all members of ENUM UIControlEvents are valid.

## Section 5.3: Setting Font

#### Swift

```
myButton.titleLabel?.font = UIFont(name: "YourFontName", size: 20)
```

#### Objective C

```
myButton.titleLabel.font = [UIFont fontWithName:@"YourFontName" size:20];
```

## Section 5.4: Set Image

#### Swift

```
button.setImage(UIImage(named:"test-image"), forState: .normal)
```

#### Objective C

```
[self.button setImage:[UIImage imageNamed:@"test-image"] forState:UIControlStateNormal];
```

#### Multiple Control States

You can also set an image for multiple UIControlStates, for example to set the same image for the Selected and Highlighted state:

#### Swift

```
button.setImage(UIImage(named:"test-image"), forState:[.selected, .highlighted])
```

#### Objective C

```
[self.button setImage:[UIImage imageNamed:@"test-image"]
forState:UIControlStateSelected|UIControlStateHighlighted];
```

## Section 5.5: Get UIButton's size strictly based on its text and font

To get the the exact size of a UIButton's text based on its font, use the function intrinsicContentSize.

## Swift

```
button.intrinsicContentSize.width
```

## Objective-C

```
button.intrinsicContentSize.width;
```

## 第5.6节：禁用UIButton

可以通过以下方式禁用按钮

## Swift

```
myButton.isEnabled = false
```

## Objective-C :

```
myButton.enabled = NO;
```

按钮将变为灰色：

Button

如果不希望按钮禁用时外观发生变化，请将adjustsImageWhenDisabled设置为false / NO

## 第5.7节：设置标题

## Swift

```
button.setTitle(titleString, forState: controlState)
```

## Objective C

```
[button setTitle:(NSString *) forState:(UIControlState)];
```

将默认标题设置为“Hello, World!”

## Swift

```
button.setTitle("Hello, World!", forState: .normal)
```

## Objective C

```
[button setTitle:@"Hello, World!" forControlState:UIControlStateNormal];
```

## 第5.8节：设置标题颜色

```
//Swift
```

## Swift

```
button.intrinsicContentSize.width
```

## Objective-C

```
button.intrinsicContentSize.width;
```

## Section 5.6: Disabling a UIButton

A button can be disabled by

## Swift

```
myButton.isEnabled = false
```

## Objective-C:

```
myButton.enabled = NO;
```

The button will become gray:

Button

If you don't want the button appearance to change when disabled set adjustsImageWhenDisabled to `false` / `NO`

## Section 5.7: Set title

## Swift

```
button.setTitle(titleString, forState: controlState)
```

## Objective C

```
[button setTitle:(NSString *) forState:(UIControlState)];
```

To set the default title to "Hello, World!"

## Swift

```
button.setTitle("Hello, World!", forState: .normal)
```

## Objective C

```
[button setTitle:@"Hello, World!" forControlState:UIControlStateNormal];
```

## Section 5.8: Set title color

```
//Swift
```

```
button.setTitleColor(color, forControlState: controlState)  
//Objective-C  
[button setTitleColor:(nullable UIColor *) forState:(UIControlState)];
```

将标题颜色设置为蓝色

```
//Swift  
button.setTitleColor(.blue, for: .normal)  
  
//Objective-C  
[button setTitleColor:[UIColor blueColor] forState:UIControlStateNormal]
```

## 第5.9节：水平对齐内容

### Swift

```
//将内容对齐到框架左侧  
button.contentHorizontalAlignment = .left  
  
//将内容对齐到框架右侧  
button.contentHorizontalAlignment = .right  
  
//将内容对齐到框架中心  
button.contentHorizontalAlignment = .center  
  
//使内容填充框架  
button.contentHorizontalAlignment = .fill
```

### Objective C

```
//将内容对齐到左侧  
button.contentHorizontalAlignment = UIControlContentHorizontalAlignmentLeft;  
  
//内容左对齐  
button.contentHorizontalAlignment = UIControlContentHorizontalAlignmentRight;  
  
//内容右对齐  
button.contentHorizontalAlignment = UIControlContentHorizontalAlignmentCenter;  
  
//内容居中对齐  
button.contentHorizontalAlignment = UIControlContentHorizontalAlignmentFill;
```

## 第5.10节：获取标题标签

如果存在底层标题标签，可以使用以下方法获取

### Swift

```
var label: UILabel? = button.titleLabel
```

### Objective C

```
UILabel *label = button.titleLabel;
```

这可以用来设置标题标签的字体，例如

```
button.setTitleColor(color, forControlState: controlState)
```

```
//Objective-C  
[button setTitleColor:(nullable UIColor *) forState:(UIControlState)];
```

To set the title color to blue

```
//Swift  
button.setTitleColor(.blue, for: .normal)  
  
//Objective-C  
[button setTitleColor:[UIColor blueColor] forState:UIControlStateNormal]
```

## Section 5.9: Horizontally aligning contents

### Swift

```
//Align contents to the left of the frame  
button.contentHorizontalAlignment = .left  
  
//Align contents to the right of the frame  
button.contentHorizontalAlignment = .right  
  
//Align contents to the center of the frame  
button.contentHorizontalAlignment = .center  
  
//Make contents fill the frame  
button.contentHorizontalAlignment = .fill
```

### Objective C

```
//Align contents to the left  
button.contentHorizontalAlignment = UIControlContentHorizontalAlignmentLeft;  
  
//Align contents to the right  
button.contentHorizontalAlignment = UIControlContentHorizontalAlignmentRight;  
  
//Align contents to the center  
button.contentHorizontalAlignment = UIControlContentHorizontalAlignmentCenter;  
  
//Align contents to fill the frame  
button.contentHorizontalAlignment = UIControlContentHorizontalAlignmentFill;
```

## Section 5.10: Getting the title label

The underlying title label, if one exists, can be fetched using

### Swift

```
var label: UILabel? = button.titleLabel
```

### Objective C

```
UILabel *label = button.titleLabel;
```

This can be used to set the font of the title label, for example

## Swift

```
button.titleLabel?.font = UIFont.boldSystemFontOfSize(12)
```

## Objective C

```
button.titleLabel.font = [UIFont boldSystemFontOfSize:12];
```

# 第5.11节：通过代码（编程方式）向UIButton添加动作

要向按钮添加方法，首先创建一个动作方法：

## Objective-C

```
-(void)someButtonAction:(id)sender {  
    // sender是被点击的对象，在本例中是按钮。  
    NSLog(@"按钮被点击");  
}
```

## Swift

```
func someButtonAction() {  
    print("按钮被点击")  
}
```

现在要将此动作方法添加到您的按钮，您需要编写以下代码行：

## Objective C

```
[yourButtonInstance addTarget:self action:@selector(someButtonAction)  
forControlEvents:UIControlEventTouchUpInside];
```

## Swift

```
yourButtonInstance.addTarget(self, action: #selector(someButtonAction), forControlEvents:  
.TouchUpInside)
```

对于ControlEvents参数， ENUM UIControlEvents的所有成员均有效。

## Swift

```
button.titleLabel?.font = UIFont.boldSystemFontOfSize(12)
```

## Objective C

```
button.titleLabel.font = [UIFont boldSystemFontOfSize:12];
```

# Section 5.11: Adding an action to an UIButton via Code (programmatically)

To add a method to a button, first create an action method:

## Objective-C

```
-(void)someButtonAction:(id)sender {  
    // sender is the object that was tapped, in this case its the button.  
    NSLog(@"Button is tapped");  
}
```

## Swift

```
func someButtonAction() {  
    print("Button is tapped")  
}
```

Now to add this action method to your button, you have to write following line of code:

## Objective C

```
[yourButtonInstance addTarget:self action:@selector(someButtonAction)  
forControlEvents:UIControlEventTouchUpInside];
```

## Swift

```
yourButtonInstance.addTarget(self, action: #selector(someButtonAction), forControlEvents:  
.TouchUpInside)
```

For ControlEvents parameter, all members of ENUM [UIControlEvents](#) are valid.

# 第6章：UIDatePicker

## 第6.1节：创建日期选择器

### Swift

```
let datePicker = UIDatePicker(frame: CGRect(x: 0, y: 0, width: 320, height: 200))
```

### Objective-C

```
UIDatePicker *datePicker = [[UIDatePicker alloc] initWithFrame:CGRectMake(0, 0, 320, 200)];
```

## 第6.2节：设置最小-最大日期

您可以设置UIDatePicker可以显示的最小和最大日期。

### 最小日期

```
[datePicker setMinimumDate:[NSDate date]];
```

### 最大日期

```
[datePicker setMaximumDate:[NSDate date]];
```

## 第6.3节：模式

UIDatePicker有多种选择器模式。

```
enum UIDatePickerMode : Int {  
    case Time  
    case Date  
    case DateAndTime  
    case CountDownTimer  
}
```

- 时间 - 日期选择器显示小时、分钟和（可选的）上午/下午标识。
- 日期 - 日期选择器显示月份、日期和年份。
- 日期和时间 - 日期选择器显示日期（作为统一的星期几、月份和日期值）加上小时、分钟和（可选的）上午/下午标识。
- 倒计时 - 日期选择器显示小时和分钟值，例如 [ 1 | 53 ]。应用程序必须设置一个定时器以在适当的间隔触发，并在秒数倒计时时设置日期选择器。

设置属性 datePickerMode

```
let datePicker = UIDatePicker(frame: CGRect(x: 0, y: 0, width: 320, height: 200))  
datePicker.datePickerMode = .Date
```

## 第6.4节：设置分钟间隔

您可以更改属性minuteInterval以设置分钟滚轮显示的间隔。默认值为1，最大值为30。

```
let datePicker = UIDatePicker(frame: CGRect(x: 0, y: 0, width: 320, height: 200))  
datePicker.minuteInterval = 15
```

# Chapter 6: UIDatePicker

## Section 6.1: Create a Date Picker

### Swift

```
let datePicker = UIDatePicker(frame: CGRect(x: 0, y: 0, width: 320, height: 200))
```

### Objective-C

```
UIDatePicker *datePicker = [[UIDatePicker alloc] initWithFrame:CGRectMake(0, 0, 320, 200)];
```

## Section 6.2: Setting Minimum-Maximum Date

You can set the minimum and the maximum date that UIDatePicker can show.

### Minimum date

```
[datePicker setMinimumDate:[NSDate date]];
```

### Maximum date

```
[datePicker setMaximumDate:[NSDate date]];
```

## Section 6.3: Modes

UIDatePicker has various picker modes.

```
enum UIDatePickerMode : Int {  
    case Time  
    case Date  
    case DateAndTime  
    case CountDownTimer  
}
```

- Time - The date picker displays hours, minutes, and (optionally) an AM/PM designation.
- Date - The date picker displays months, days of the month, and years.
- DateAndTime - The date picker displays dates (as unified day of the week, month, and day of the month values) plus hours, minutes, and (optionally) an AM/PM designation.
- CountDownTimer - The date picker displays hour and minute values, for example [ 1 | 53 ]. The application must set a timer to fire at the proper interval and set the date picker as the seconds tick down.

Setting property datePickerMode

```
let datePicker = UIDatePicker(frame: CGRect(x: 0, y: 0, width: 320, height: 200))  
datePicker.datePickerMode = .Date
```

## Section 6.4: Setting minute interval

You can change property minuteInterval to set the interval displayed by the minutes wheel. The default value is 1, the maximum value is 30.

```
let datePicker = UIDatePicker(frame: CGRect(x: 0, y: 0, width: 320, height: 200))  
datePicker.minuteInterval = 15
```

## 第6.5节：倒计时持续时间

该属性的NSTimeInterval值表示倒计时模式下日期选择器开始倒计时的秒数。如果日期选择器的模式不是CountDown Timer，则忽略此值。最大值为86,399秒（23:59）

```
let datePicker = UIDatePicker(frame: CGRect(x: 0, y: 0, width: 320, height: 200))
datePicker.countDownDuration = 60 * 60
```

## Section 6.5: Count Down Duration

The NSTimeInterval value of this property indicates the seconds from which the date picker in countdown-timer mode counts down. If the mode of the date picker is not CountDownTimer, this value is ignored. Maximum value is 86,399 seconds (23:59)

```
let datePicker = UIDatePicker(frame: CGRect(x: 0, y: 0, width: 320, height: 200))
datePicker.countDownDuration = 60 * 60
```

# 第7章：UILocalNotification

本地通知允许您的应用在不需要服务器的情况下通知用户相关内容。

与由服务器触发的远程通知不同，本地通知是在应用内调度和触发的。通知的总体目的是增加用户与应用的互动，邀请或吸引用户打开并使用应用。

UILocalNotification在iOS 10中已被弃用。请改用UserNotifications框架。

## 第7.1节：调度本地通知

请确保您已查看“注册本地通知”，以使其正常工作：

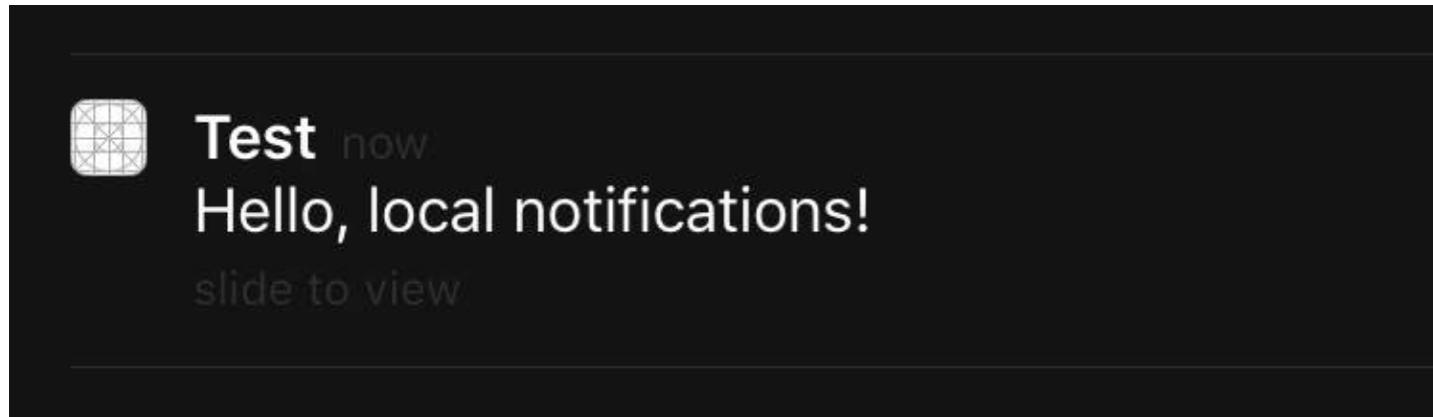
### Swift

```
let notification = UILocalNotification()
notification.alertBody = "Hello, local notifications!"
notification.fireDate = NSDate().dateByAddingTimeInterval(10) // 当前时间10秒后
UIApplication.sharedApplication().scheduleLocalNotification(notification)
```

### Objective-C

```
UILocalNotification *notification = [[UILocalNotification alloc] init];
notification.alertBody = @"Hello, local notifications!";
notification.fireDate = [NSDate dateWithTimeIntervalSinceNow:10]; // 现在起10秒后
[[UIApplication sharedApplication] scheduleLocalNotification:notification];
```

要在iOS模拟器中查看通知，按<sup>⌘</sup>H (control-command-H) 返回主屏幕，然后按<sup>⌘</sup>L (command-L) 锁定设备。等待几秒钟，通知应该会出现（通知的显示方式会根据“注册本地通知”中讨论的通知类型有所不同）：



在通知上滑动以返回应用（注意，如果你在第一个视图控制器的viewDidLoad、viewWillAppear、viewDidAppear等方法中调用此操作，通知将会被重新调度）。

## 第7.2节：立即显示本地通知

如果你想立即显示本地通知，应调用：

### Swift 3

```
UIApplication.shared.presentLocalNotificationNow(notification)
```

# Chapter 7: UILocalNotification

Local notifications allow your app to notify the user about content which does not require the use of a server.

Unlike remote notifications which are triggered from a server, local notifications are scheduled and triggered within an app. Notifications in general are targeted to increase user interaction with the app, inviting or tempting the user to open and interact with it.

UILocalNotification was deprecated in iOS 10. Use the UserNotifications framework instead.

## Section 7.1: Scheduling a local notification

Make sure you see Registering for local notifications in order for this to work:

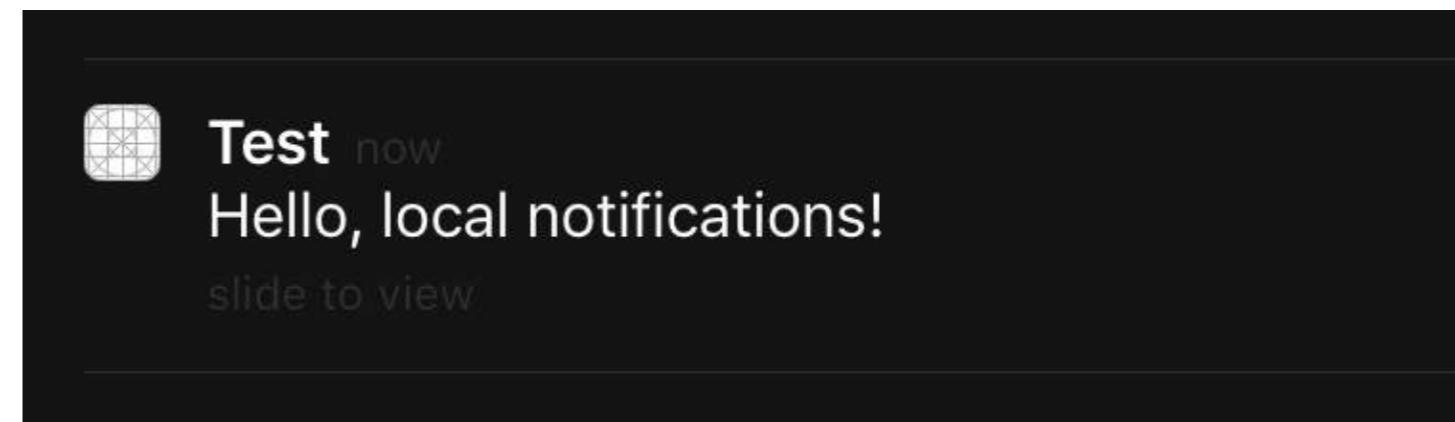
### Swift

```
let notification = UILocalNotification()
notification.alertBody = "Hello, local notifications!"
notification.fireDate = NSDate().dateByAddingTimeInterval(10) // 10 seconds after now
UIApplication.sharedApplication().scheduleLocalNotification(notification)
```

### Objective-C

```
UILocalNotification *notification = [[UILocalNotification alloc] init];
notification.alertBody = @"Hello, local notifications!";
notification.fireDate = [NSDate dateWithTimeIntervalSinceNow:10]; // 10 seconds after now
[[UIApplication sharedApplication] scheduleLocalNotification:notification];
```

To see the notification in the iOS Simulator, type <sup>⌘</sup>H (control-command-H) to go home and then type <sup>⌘</sup>L (command-L) to lock the device. Wait a few seconds, and the notification should appear (this appearance will vary depending on notification type discussed in "Registering for local notifications"):



Swipe on the notification to get back to the app (note that if you called this in the first view controller's viewDidLoad, viewWillAppear, viewDidAppear, etc., the notification will be scheduled again).

## Section 7.2: Presenting a local notification immediately

If you want to show local notification immediately, you should call:

### Swift 3

```
UIApplication.shared.presentLocalNotificationNow(notification)
```

## Swift 2

```
UIApplication.sharedApplication().presentLocalNotificationNow(notification)
```

### Objective-C

```
[[UIApplication sharedApplication] presentLocalNotificationNow:notification];
```

使用此方法的一个优点是你不必设置

UILocalNotification对象的fireDate和timeZone属性。

## 第7.3节：使用UUID管理本地通知

通常情况下，您需要能够管理通知，能够跟踪并取消它们。

### 跟踪通知

您可以为通知分配一个UUID（通用唯一标识符），以便跟踪它：

### Swift

```
let notification = UILocalNotification()
let uuid = NSUUID().uuidString
notification.userInfo = ["UUID": uuid]
UIApplication.shared.scheduleLocalNotification(notification)
```

### Objective-C

```
UILocalNotification *notification = [[UILocalNotification alloc] init];
NSString *uuid = [[NSUUID UUID] UUIDString];
notification.userInfo = @{@"UUID": uuid};
[[UIApplication sharedApplication] scheduleLocalNotification:notification];
```

### 取消通知

要取消通知，首先获取所有通知的列表，然后找到UUID匹配的通知。最后，取消该通知。

### Swift

```
let scheduledNotifications = UIApplication.shared.scheduledLocalNotifications

guard let scheduledNotifications = scheduledNotifications else {
    return
}

对于通知在scheduledNotifications中当"\(notification.userInfo!["UUID"]!"=="  
UUID_TO_CANCEL {  
    UIApplication.sharedApplication().cancelLocalNotification(notification)  
}
```

### Objective-C

```
NSArray *scheduledNotifications = [[UIApplication sharedApplication] scheduledLocalNotifications];
```

## Swift 2

```
UIApplication.sharedApplication().presentLocalNotificationNow(notification)
```

### Objective-C

```
[[UIApplication sharedApplication] presentLocalNotificationNow:notification];
```

An advantage of using this is so you won't have to set the fireDate and timeZone properties of your UILocalNotification object.

## Section 7.3: Managing local notifications using UUID

Often times you will need to be able to manage your notifications, by being able to keep track of them and cancel them.

### Track a notification

You can assign a UUID (universally unique identifier) to a notification, so you can track it:

### Swift

```
let notification = UILocalNotification()
let uuid = NSUUID().uuidString
notification.userInfo = ["UUID": uuid]
UIApplication.shared.scheduleLocalNotification(notification)
```

### Objective-C

```
UILocalNotification *notification = [[UILocalNotification alloc] init];
NSString *uuid = [[NSUUID UUID] UUIDString];
notification.userInfo = @{@"UUID": uuid};
[[UIApplication sharedApplication] scheduleLocalNotification:notification];
```

### Cancel a notification

To cancel a notification, we first get a list of all the notifications and then find the one with a matching UUID. Finally, we cancel it.

### Swift

```
let scheduledNotifications = UIApplication.shared.scheduledLocalNotifications

guard let scheduledNotifications = scheduledNotifications else {
    return
}

for notification in scheduledNotifications where "\(notification.userInfo!["UUID"]!" ==  
UUID_TO_CANCEL {  
    UIApplication.sharedApplication().cancelLocalNotification(notification)  
}
```

### Objective-C

```
NSArray *scheduledNotifications = [[UIApplication sharedApplication] scheduledLocalNotifications];
```

```

对于 (UILocalNotification *notification in scheduledNotifications) {
    如果 ([[notification.userInfo objectForKey:@"UUID"] compare: UUID_TO_CANCEL]) {
        [[UIApplication sharedApplication] cancelLocalNotification:notification];
        跳出;
    }
}

```

你可能想将所有这些UUID存储在Core Data或Realm中。

## 第7.4节：注册本地通知

版本 ≥ iOS 8

为了向用户展示本地通知，你必须在设备上注册你的应用：

### Swift

```

let settings = UIUserNotificationSettings(forTypes: [.Badge, .Sound, .Alert], categories: nil)
UIApplication.sharedApplication().registerUserNotificationSettings(settings)

```

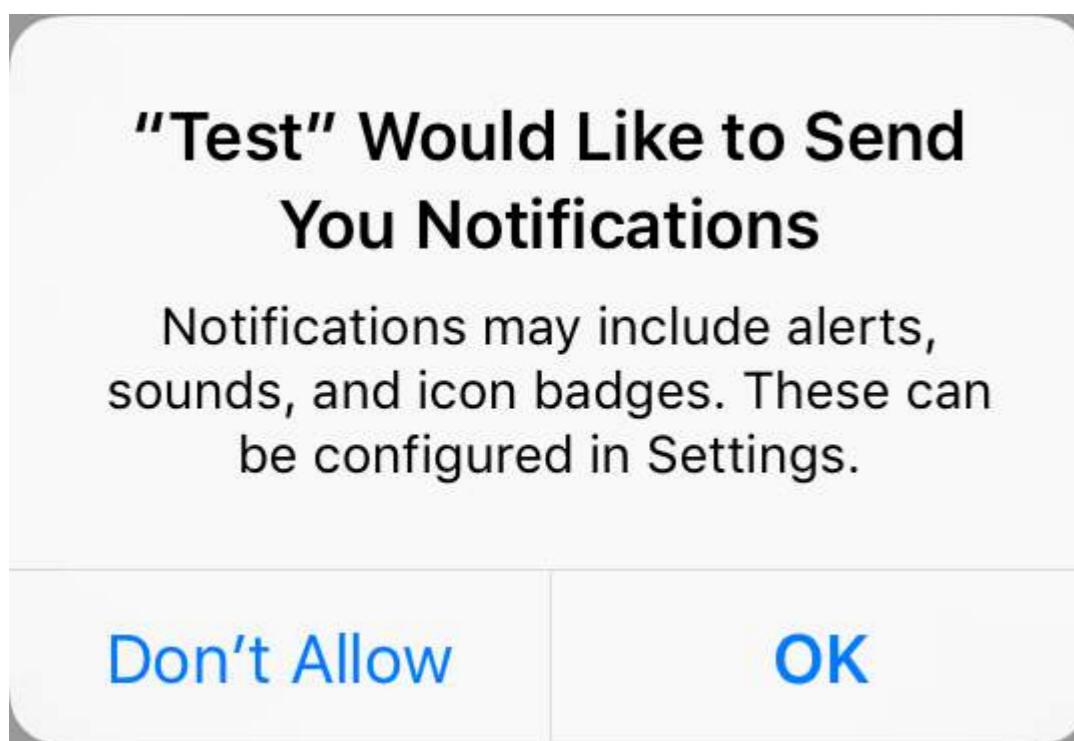
### Objective-C

```

UIUserNotificationSettings *settings = [UIUserNotificationSettings
settingsForTypes:(UIUserNotificationTypeBadge | UIUserNotificationTypeSound |
UIUserNotificationTypeAlert) categories:nil];
[[UIApplication sharedApplication] registerUserNotificationSettings:settings];

```

这将在首次调用时弹出一个警告：



无论用户选择什么，警告都不会再次显示，且更改必须由用户在设置中手动进行。

```

for (UILocalNotification *notification in scheduledNotifications) {
    if ([[notification.userInfo objectForKey:@"UUID"] compare: UUID_TO_CANCEL]) {
        [[UIApplication sharedApplication] cancelLocalNotification:notification];
        break;
    }
}

```

You would probably want to store all of these UUID's in Core Data or Realm.

## Section 7.4: Registering for local notifications

Version ≥ iOS 8

In order to present local notifications to the user, you have to register your app with the device:

### Swift

```

let settings = UIUserNotificationSettings(forTypes: [.Badge, .Sound, .Alert], categories: nil)
UIApplication.sharedApplication().registerUserNotificationSettings(settings)

```

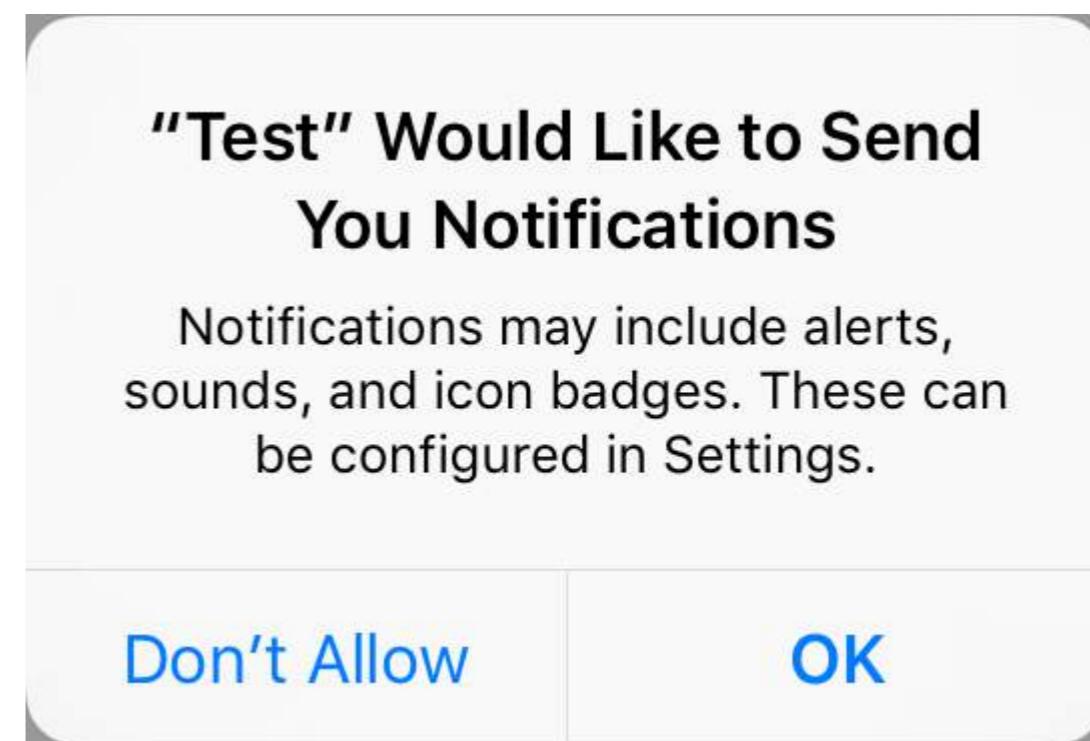
### Objective-C

```

UIUserNotificationSettings *settings = [UIUserNotificationSettings
settingsForTypes:(UIUserNotificationTypeBadge | UIUserNotificationTypeSound |
UIUserNotificationTypeAlert) categories:nil];
[[UIApplication sharedApplication] registerUserNotificationSettings:settings];

```

This will present an alert the first time it is called:



Regardless of what the user chooses, the alert will not show up again and changes will have to be initiated by the user in Settings.

## 第7.5节：iOS10中UILocalNotification的新特性

你可以使用UILocalNotification，旧的API在iOS10中仍然可用，但我们最好使用User Notifications框架中的API。此外，还有一些只能在iOS10 User Notifications框架中使用的新功能。

这同样适用于远程通知，更多信息请见：[Here](#)。

新特性：

1. 现在即使应用在前台，也可以展示警告、播放声音或增加角标，适用于iOS 10.2. 现在当用户点击（或滑动）操作按钮时，即使应用已被杀死，也能在一个地方处理所有事件。
3. 支持3D Touch替代滑动手势。
4. 现在你只需一行代码就能移除特定的本地通知。
5. 支持带自定义界面的丰富通知。

将UILocalNotification API转换为iOS10的用户通知框架API对我们来说非常简单，它们非常相似。

我写了一个演示，展示如何同时使用新旧API：[iOS10AdaptationTips](#)。

例如，

使用Swift实现：

1. 导入UserNotifications

```
/// 通知独立于UIKit
import UserNotifications
```

2. 请求本地通知授权

```
let center = UNUserNotificationCenter.current()
center.requestAuthorization(options: [.alert, .sound]) { (granted, error) in
    // 根据授权启用或禁用功能。
}
```

3. 安排本地通知

4. 更新应用图标徽章数字

```
@IBAction func triggerNotification(){
    let content = UNMutableNotificationContent()
    content.title = NSLocalizedString(forKey: "Elon said:", arguments: nil)
    content.body = NSLocalizedString(forKey: "Hello Tom! Get up, let's play with Jerry!", arguments: nil)
    content.sound = UNNotificationSound.default()
    content.badge = UIApplication.shared().applicationIconBadgeNumber + 1;
    content.categoryIdentifier = "com.elonchan.localNotification"
    // Deliver the notification in five seconds.
    let trigger = UNTimeIntervalNotificationTrigger.init(timeInterval: 60.0, repeats: true)
    let request = UNNotificationRequest.init(identifier: "FiveSecond", content: content,
    trigger: trigger)
```

## Section 7.5: what's new in UILocalNotification with iOS10

You can use [UILocalNotification](#), old APIs also works fine with iOS10, but we had better use the APIs in the User Notifications framework instead. There are also some new features, you can only use with iOS10 User Notifications framework.

This also happens to Remote Notification, for more information: [Here](#).

New Features:

1. Now you can either present alert, sound or increase badge while the app is in foreground too with iOS 10
2. Now you can handle all event in one place when user tapped (or slided) the action button, even while the app has already been killed.
3. Support 3D touch instead of sliding gesture.
4. Now you can remove specifical local notification just by one row code.
5. Support Rich Notification with custom UI.

It is really easy for us to convert [UILocalNotification](#) APIs to iOS10 User Notifications framework APIs, they are really similar.

I write a Demo here to show how to use new and old APIs at the same time: [iOS10AdaptationTips](#).

For example,

With Swift implementation:

1. import UserNotifications

```
/// Notification become independent from UIKit
import UserNotifications
```

2. request authorization for localNotification

```
let center = UNUserNotificationCenter.current()
center.requestAuthorization(options: [.alert, .sound]) { (granted, error) in
    // Enable or disable features based on authorization.
}
```

3. schedule localNotification

4. update application icon badge number

```
@IBAction func triggerNotification(){
    let content = UNMutableNotificationContent()
    content.title = NSLocalizedString(forKey: "Elon said:", arguments: nil)
    content.body = NSLocalizedString(forKey: "Hello Tom! Get up, let's play with Jerry!", arguments: nil)
    content.sound = UNNotificationSound.default()
    content.badge = UIApplication.shared().applicationIconBadgeNumber + 1;
    content.categoryIdentifier = "com.elonchan.localNotification"
    // Deliver the notification in five seconds.
    let trigger = UNTimeIntervalNotificationTrigger.init(timeInterval: 60.0, repeats: true)
    let request = UNNotificationRequest.init(identifier: "FiveSecond", content: content,
    trigger: trigger)
```

```

// Schedule the notification.
let center = UNUserNotificationCenter.current()
center.add(request)
}

@IBAction func stopNotification(_ sender: AnyObject) {
    let center = UNUserNotificationCenter.current()
    center.removeAllPendingNotificationRequests()
    // 或者你也可以移除指定的通知：
    // center.removePendingNotificationRequests(withIdentifier: ["FiveSecond"])
}

```

Objective-C 实现：

### 1. 导入UserNotifications

```
// 通知独立于 UIKit
#import <UserNotifications/UserNotifications.h>
```

### 2. 请求本地通知授权

```

UNUserNotificationCenter *center = [UNUserNotificationCenter currentNotificationCenter];
[center requestAuthorizationWithOptions:(UNAuthorizationOptionBadge |
UNAuthorizationOptionSound | UNAuthorizationOptionAlert)
completionHandler:^(BOOL granted, NSError * _Nullable error) {
    if (!error) {
        NSLog(@"请求授权成功！");
        [self showAlert];
    }
}];

```

### 3. 安排本地通知

### 4. 更新应用图标徽章数字

```

UNMutableNotificationContent *content = [[UNMutableNotificationContent alloc] init];
content.title = [NSString localizedUserNotificationStringForKey:@"埃隆说：" arguments:nil];
content.body = [NSString localizedUserNotificationStringForKey:@"你好，汤姆！起床了，我们去和杰瑞玩吧！" arguments:nil];
参数：nil];
内容。声音 = [UNNotificationSound defaultSound];

// 4. 更新应用图标角标数字
内容。角标 = [NSNumber numberWithInteger:([UIApplication sharedApplication].applicationIconBadgeNumber + 1)];
// Deliver the notification in five seconds.
UNTTimeIntervalNotificationTrigger *trigger = [UNTTimeIntervalNotificationTrigger triggerWithTimeInterval:5.f repeats:NO];
UNNotificationRequest *request = [UNNotificationRequest requestWithIdentifier:@"FiveSecond" content:content];
触发器:trigger];
/// 3. 安排本地通知
UNUserNotificationCenter *center = [UNUserNotificationCenter currentNotificationCenter];
[center addNotificationRequest:request completionHandler:^(NSError * _Nullable error) {
    如果 (!error) {
        NSLog(@"添加通知请求成功！");
    }
}]

```

```

// Schedule the notification.
let center = UNUserNotificationCenter.current()
center.add(request)
}

@IBAction func stopNotification(_ sender: AnyObject) {
    let center = UNUserNotificationCenter.current()
    center.removeAllPendingNotificationRequests()
    // or you can remove specifical notification:
    // center.removePendingNotificationRequests(withIdentifier: ["FiveSecond"])
}

```

Objective-C implementation:

### 1. import UserNotifications

```
// Notifications are independent from UIKit
#import <UserNotifications/UserNotifications.h>
```

### 2. request authorization for localNotification

```

UNUserNotificationCenter *center = [UNUserNotificationCenter currentNotificationCenter];
[center requestAuthorizationWithOptions:(UNAuthorizationOptionBadge |
UNAuthorizationOptionSound | UNAuthorizationOptionAlert)
completionHandler:^(BOOL granted, NSError * _Nullable error) {
    if (!error) {
        NSLog(@"request authorization succeeded!");
        [self showAlert];
    }
}];

```

### 3. schedule localNotification

### 4. update application icon badge number

```

UNMutableNotificationContent *content = [[UNMutableNotificationContent alloc] init];
content.title = [NSString localizedUserNotificationStringForKey:@"Elon said:" arguments:nil];
content.body = [NSString localizedUserNotificationStringForKey:@"Hello Tom! Get up, let's play with Jerry!" arguments:nil];
content.sound = [UNNotificationSound defaultSound];

// 4. update application icon badge number
content.badge = [NSNumber numberWithInteger:([UIApplication sharedApplication].applicationIconBadgeNumber + 1)];
// Deliver the notification in five seconds.
UNTTimeIntervalNotificationTrigger *trigger = [UNTTimeIntervalNotificationTrigger triggerWithTimeInterval:5.f repeats:NO];
UNNotificationRequest *request = [UNNotificationRequest requestWithIdentifier:@"FiveSecond" content:content trigger:trigger];
/// 3. schedule localNotification
UNUserNotificationCenter *center = [UNUserNotificationCenter currentNotificationCenter];
[center addNotificationRequest:request completionHandler:^(NSError * _Nullable error) {
    if (!error) {
        NSLog(@"add NotificationRequest succeeded!");
    }
}]

```

```
});
```

更多信息请访问：[iOS10AdaptationTips](#)。

#updated

由于未捕获的异常 'NSInternalInconsistencyException' 导致应用程序终止，原因：'时间间隔如果重复，必须至少为60'

```
let trigger = UNTimeIntervalNotificationTrigger.init(timeInterval: 60, repeats: true)
```

## 第7.6节：响应接收到的本地通知

**重要：**此代理方法仅在前台调用。

**Swift**

```
func application(application: UIApplication, didReceiveLocalNotification notification: UILocalNotification) {  
}
```

**Objective-C**

```
- (void)application:(UIApplication *)application didReceiveLocalNotification:(UILocalNotification *)notification {  
}
```

此方法通常在遵循UIApplicationDelegate协议的AppDelegate中重写。

## 第7.7节：在Swift 3.0 (iOS 10) 中注册和安排本地通知

在AppDeleg

ate中注册

```
import UserNotifications
```

在 `didFinishLaunchingWithOptions` 方法中，

```
UNUserNotificationCenter.current().requestAuthorization(options: [.alert,.sound,.badge]) {  
    (已授权, 错误) in  
  
    // 这里你可以检查请求是否被授权。  
}  
}
```

创建并安排通知。

```
let content = UNMutableNotificationContent()  
content.title = "10秒通知演示"
```

```
});
```

Go to here for more information: [iOS10AdaptationTips](#).

#updated

Terminating app due to uncaught exception 'NSInternalInconsistencyException', reason: 'time interval must be at least 60 if repeating'

```
let trigger = UNTimeIntervalNotificationTrigger.init(timeInterval: 60, repeats: true)
```

## Section 7.6: Responding to received local notification

**IMPORTANT:** This delegate method is only called in the foreground.

**Swift**

```
func application(application: UIApplication, didReceiveLocalNotification notification: UILocalNotification) {  
}
```

**Objective-C**

```
- (void)application:(UIApplication *)application didReceiveLocalNotification:(UILocalNotification *)notification {  
}
```

This method is generally overridden in the AppDelegate, which conforms to the UIApplicationDelegate protocol.

## Section 7.7: Register and Schedule Local Notification in Swift 3.0 (iOS 10)

**Registration**

in **AppDelegate**

```
import UserNotifications
```

in `didFinishLaunchingWithOptions` method,

```
UNUserNotificationCenter.current().requestAuthorization(options: [.alert,.sound,.badge]) {  
    (granted, error) in  
  
    // Here you can check Request is Granted or not.  
}  
}
```

Create and Schedule notification.

```
let content = UNMutableNotificationContent()  
content.title = "10 Second Notification Demo"
```

```
content.subtitle = "来自金刚狼"
content.body = "10秒后的通知 - 你的披萨准备好了！！"
content.categoryIdentifier = "myNotificationCategory"

let trigger = UNTimeIntervalNotificationTrigger(
    timeInterval: 10.0,
    repeats: false)

let request = UNNotificationRequest(
    identifier: "10.second.message",
    content: content,
trigger: trigger
)
UNUserNotificationCenter.current().add(request, completionHandler: nil)
```

无论代码的哪个部分被触发，如果您已允许通知权限，您将收到一条通知。

要正确测试，请确保您的应用处于后台模式。

```
content.subtitle = "From Wolverine"
content.body = "Notification after 10 seconds - Your pizza is Ready!!"
content.categoryIdentifier = "myNotificationCategory"

let trigger = UNTimeIntervalNotificationTrigger(
    timeInterval: 10.0,
    repeats: false)

let request = UNNotificationRequest(
    identifier: "10.second.message",
    content: content,
trigger: trigger
)
UNUserNotificationCenter.current().add(request, completionHandler: nil)
```

Where ever this part of code is triggered, If you have allowed Notification Permission, you will receive an notification.

To test it properly, Make sure your application in Background mode.

# 第8章：UIImage

## 第8.1节：创建UIImage

### 使用本地图片

#### Swift

```
let image = UIImage(named: "imageFromBundleOrAsset")
```

#### Objective-C

```
UIImage *image = [UIImage imageNamed:@"imageFromBundleOrAsset"];
```

#### 注意

方法 `imageNamed` 会将图片内容缓存到内存中。以这种方式加载许多大图片可能会导致内存不足警告，进而导致应用被终止。可以通过使用 `UIImage` 的 `imageWithContentsOfFile` 方法来解决，该方法不使用缓存。

### 使用NSData

#### Swift

```
let imageData = Data(base64Encoded: imageString, options: Data.Base64DecodingOptions.ignoreUnknownCharacters)
```

```
let image = UIImage(data: imageData!)
```

### 使用 UIColor

#### Swift

```
let color = UIColor.red
```

```
let size = CGSize(width: 200, height: 200)
```

```
UIGraphicsBeginImageContextWithOptions(size, false, 0.0)
UIGraphicsGetCurrentContext()!.setFillColor(color.cgColor)
UIGraphicsGetCurrentContext()!.fill(CGRect(origin: .zero, size: size))
let colorImage = UIGraphicsGetImageFromCurrentImageContext()
UIGraphicsEndImageContext()
```

#### Objective-C

```
UIColor *color=[UIColor redColor];
CGRect frame = CGRectMake(0, 0, 80, 100);
UIGraphicsBeginImageContext(frame.size);
CGContextRef context = UIGraphicsGetCurrentContext();
CGContextSetFillColorWithColor(context, [color CGColor]);
CGContextFillRect(context, frame);
UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
UIGraphicsEndImageContext();
```

### 使用文件内容

#### Objective-C

#### 示例：

```
UIImage *image = [UIImage imageWithContentsOfFile:[[NSBundle mainBundle] pathForResource:[cellCountry objectForKey:@"Country_Flag"] ofType:nil]];
```

#### 使用数组：

# Chapter 8: UIImage

## Section 8.1: Creating UIImage

### With local image

#### Swift

```
let image = UIImage(named: "imageFromBundleOrAsset")
```

#### Objective-C

```
UIImage *image = [UIImage imageNamed:@"imageFromBundleOrAsset"];
```

#### Note

The method `imageNamed` caches the image's contents to memory. Loading many large images that way can cause low memory warnings which can lead the app to be terminated. This can be fixed by utilising the method `imageWithContentsOfFile` of `UIImage`, which doesn't use caching.

### With NSData

#### Swift

```
let imageData = Data(base64Encoded: imageString, options: Data.Base64DecodingOptions.ignoreUnknownCharacters)
```

```
let image = UIImage(data: imageData!)
```

### With UIColor

#### Swift

```
let color = UIColor.red
let size = CGSize(width: 200, height: 200)
```

```
UIGraphicsBeginImageContextWithOptions(size, false, 0.0)
UIGraphicsGetCurrentContext()!.setFillColor(color.cgColor)
UIGraphicsGetCurrentContext()!.fill(CGRect(origin: .zero, size: size))
let colorImage = UIGraphicsGetImageFromCurrentImageContext()
UIGraphicsEndImageContext()
```

#### Objective-C

```
UIColor *color=[UIColor redColor];
CGRect frame = CGRectMake(0, 0, 80, 100);
UIGraphicsBeginImageContext(frame.size);
CGContextRef context = UIGraphicsGetCurrentContext();
CGContextSetFillColorWithColor(context, [color CGColor]);
CGContextFillRect(context, frame);
UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
UIGraphicsEndImageContext();
```

### With file content

#### Objective-C

#### Example:

```
UIImage *image = [UIImage imageWithContentsOfFile:[[NSBundle mainBundle] pathForResource:[cellCountry objectForKey:@"Country_Flag"] ofType:nil]];
```

#### Using Array:

示例：

```
NSMutableArray *imageArray = [[NSMutableArray alloc] init];

for (int imageNumber = 1; self.myPhoto != nil; imageNumber++) {
    NSString *fileName = [NSString stringWithFormat:@"%@.jpg", self.myPhoto];

    // 检查文件是否存在
    if ([UIImage imageNamed:fileName]) {
        // 如果存在，则将其添加到数组中
        [imageArray addObject:[UIImage imageWithContentsOfFile:[[NSBundle mainBundle] pathForResource:[NSString stringWithFormat:@"%@", fileName] ofType:@""]]];
    } else {
        break;
    }
}

// 在这里使用图像数组进行动画：
self.myImageView.animationImages = imageArray;
```

## 第8.2节：比较图像

`isEqual:` 方法是判断两个图像是否包含相同图像数据的唯一可靠方式。即使你用相同的缓存图像数据初始化图像对象，它们也可能彼此不同。确定它们是否相等的唯一方法是使用 `isEqual:` 方法，该方法比较实际的图像数据。清单1展示了比较图像的正确和错误方法。

来源：[苹果文档](#)

### Swift

```
// 加载同一张图片两次。
let image1 = UIImage(named: "MyImage")
let image2 = UIImage(named: "MyImage")

// 图片对象可能不同，但内容仍然相等
if let image1 = image1, image1.isEqual(image2) {
    // 正确。此方法正确比较了图像数据。
}

if image1 == image2 {
    // 错误！直接比较对象可能不起作用。
}
```

### Objective-C

```
// 加载同一张图片两次。
UIImage* image1 = [UIImage imageNamed:@"MyImage"];
UIImage* image2 = [UIImage imageNamed:@"MyImage"];

// 图片对象可能不同，但内容仍然相等
if ([image1 isEqual:image2]) {
    // 正确。此方法正确比较了图像数据。
}
```

Example:

```
NSMutableArray *imageArray = [[NSMutableArray alloc] init];

for (int imageNumber = 1; self.myPhoto != nil; imageNumber++) {
    NSString *fileName = [NSString stringWithFormat:@"%@.jpg", self.myPhoto];

    // check if a file exists
    if ([UIImage imageNamed:fileName]) {
        // if it exists, add it to the array
        [imageArray addObject:[UIImage imageWithContentsOfFile:[[NSBundle mainBundle] pathForResource:[NSString stringWithFormat:@"%@", fileName] ofType:@""]]];
    } else {
        break;
    }
}

//Using image array for animations here:
self.myImageView.animationImages = imageArray;
```

## Section 8.2: Comparing Images

The `isEqual:` method is the only reliable way to determine whether two images contain the same image data. The image objects you create may be different from each other, even when you initialize them with the same cached image data. The only way to determine their equality is to use the `isEqual:` method, which compares the actual image data. Listing 1 illustrates the correct and incorrect ways to compare images.

Source: [Apple Documentation](#)

### Swift

```
// Load the same image twice.
let image1 = UIImage(named: "MyImage")
let image2 = UIImage(named: "MyImage")

// The image objects may be different, but the contents are still equal
if let image1 = image1, image1.isEqual(image2) {
    // Correct. This technique compares the image data correctly.
}

if image1 == image2 {
    // Incorrect! Direct object comparisons may not work.
}
```

### Objective-C

```
// Load the same image twice.
UIImage* image1 = [UIImage imageNamed:@"MyImage"];
UIImage* image2 = [UIImage imageNamed:@"MyImage"];

// The image objects may be different, but the contents are still equal
if ([image1 isEqual:image2]) {
    // Correct. This technique compares the image data correctly.
}

if (image1 == image2) {
```

```
// 错误！直接比较对象可能不起作用。
```

```
}
```

## 第8.3节：带颜色的渐变图像

在CGRect中使用颜色创建渐变UIImage

Swift:

```
extension UIImage {
    static func gradientImageWithBounds(bounds: CGRect, colors: [CGColor]) -> UIImage {
        let gradientLayer = CAGradientLayer()
        gradientLayer.frame = bounds
        gradientLayer.colors = colors

        UIGraphicsBeginImageContext(gradientLayer.bounds.size)
        gradientLayer.render(in: UIGraphicsGetCurrentContext()!)
        let image = UIGraphicsGetImageFromCurrentImageContext()
        UIGraphicsEndImageContext()
        return image!
    }
}
```

用法：

```
let image = UIImage.gradientImageWithBounds(CGRect(x: 0, y: 0, width: 200, height: 200), colors: [UIColor.yellowColor().CGColor, UIColor.blueColor().CGColor])
```

Objective-C：

```
+ (UIImage *)gradientImageWithBounds:(CGRect)bounds colors:(NSArray *)colors {
    CAGradientLayer *gradientLayer = [CAGradientLayer layer];
    gradientLayer.frame = bounds;
    gradientLayer.colors = colors;

    UIGraphicsBeginImageContext(gradientLayer.bounds.size);
    [gradientLayer renderInContext:UIGraphicsGetCurrentContext()];
    UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    return image;
}
```

## 第8.4节：UIImage与base64编码的转换

编码

```
//先将图像转换为NSData
let imageData: NSData = UIImagePNGRepresentation(image)!
// 将NSData转换为base64编码
let strBase64:String = imageData.base64EncodedStringWithOptions(.Encoding64CharacterLineLength)
```

解码

```
let dataDecoded: NSData = NSData(base64EncodedString: strBase64, options: NSDataBase64DecodingOptions(rawValue: 0)!)
let decodedimage:UIImage = UIImage(data: dataDecoded)!
```

```
// Incorrect! Direct object comparisons may not work.
```

```
}
```

## Section 8.3: Gradient Image with Colors

Creating Gradient UIImage with colors in CGRect

Swift:

```
extension UIImage {
    static func gradientImageWithBounds(bounds: CGRect, colors: [CGColor]) -> UIImage {
        let gradientLayer = CAGradientLayer()
        gradientLayer.frame = bounds
        gradientLayer.colors = colors

        UIGraphicsBeginImageContext(gradientLayer.bounds.size)
        gradientLayer.render(in: UIGraphicsGetCurrentContext()!)
        let image = UIGraphicsGetImageFromCurrentImageContext()
        UIGraphicsEndImageContext()
        return image!
    }
}
```

Usage:

```
let image = UIImage.gradientImageWithBounds(CGRect(x: 0, y: 0, width: 200, height: 200), colors: [UIColor.yellowColor().CGColor, UIColor.blueColor().CGColor])
```

Objective-C:

```
+ (UIImage *)gradientImageWithBounds:(CGRect)bounds colors:(NSArray *)colors {
    CAGradientLayer *gradientLayer = [CAGradientLayer layer];
    gradientLayer.frame = bounds;
    gradientLayer.colors = colors;

    UIGraphicsBeginImageContext(gradientLayer.bounds.size);
    [gradientLayer renderInContext:UIGraphicsGetCurrentContext()];
    UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    return image;
}
```

## Section 8.4: Convert UIImage to/from base64 encoding

Encoding

```
//convert the image to NSData first
let imageData: NSData = UIImagePNGRepresentation(image)!
// convert the NSData to base64 encoding
let strBase64:String = imageData.base64EncodedStringWithOptions(.Encoding64CharacterLineLength)
```

Decoding

```
let dataDecoded: NSData = NSData(base64EncodedString: strBase64, options: NSDataBase64DecodingOptions(rawValue: 0)!)
let decodedimage:UIImage = UIImage(data: dataDecoded)!
```

## 第8.5节：截取UIView快照

```
//这里self.webView是我需要截屏的视图//截屏以jpg格式保存在应用程序目录中，以  
避免在视网膜显示设备上出现任何质量损失，即所有运行iOS 10的当前设备  
  
UIGraphicsBeginImageContextWithOptions(self.webView.bounds.size, NO, [UIScreen mainScreen].scale);  
[self.webView.layer renderInContext:UIGraphicsGetCurrentContext()];  
UIImage *image = UIGraphicsGetImageFromCurrentImageContext();  
UIGraphicsEndImageContext();  
NSString *jpgPath = [NSHomeDirectory() stringByAppendingPathComponent:@"Documents/Test.jpg"];  
[UIImageJPEGRepresentation(image, 1.0) writeToFile:jpgPath atomically:YES];  
UIImage *pop=[[UIImage alloc]initWithContentsOfFile:jpgPath];  
//pop 是最终的 jpg 格式高质量图像，分辨率与您选择的视图的像素完全一致，而不仅仅是点数
```

## 第 8.6 节：更改 UIImage 颜色

Swift 为 UIImage 添加此扩展：

```
extension UIImage {  
    func maskWithColor(color: UIColor) -> UIImage? {  
  
        let maskImage = self.CGImage  
        let width = self.size.width  
        let height = self.size.height  
        let bounds = CGRectMake(0, 0, width, height)  
  
        let colorSpace = CGColorSpaceCreateDeviceRGB()  
        let bitmapInfo = CGBitmapInfo(rawValue: CGImageAlphaInfo.PremultipliedLast.rawValue)  
        let bitmapContext = CGBitmapContextCreate(nil, Int(width), Int(height), 8, 0, colorSpace,  
        bitmapInfo.rawValue) //需要 bitmapInfo 的 rawValue  
  
        CGContextClipToMask(bitmapContext, bounds, maskImage)  
        CGContextSetFillColorWithColor(bitmapContext, color.CGColor)  
        CGContextFillRect(bitmapContext, bounds)  
  
        // 是空吗？  
        if let cImage = CGBitmapContextCreateImage(bitmapContext) {  
            let coloredImage = UIImage(CGImage: cImage)  
  
            return coloredImage  
  
        } else {  
            return nil  
        }  
    }  
}
```

然后，改变你的 UIImage 的颜色

```
my_image.maskWithColor(UIColor.blueColor())
```

在此链接中找到 [\\_](#)

## 第8.7节：将 UIColor 应用于 UIImage

通过如下方式，仅对 UIImage 实例应用 UIColor，即可在多个主题基础的应用中使用相同的 UIImage。

## Section 8.5: Take a Snapshot of a UIView

```
//Here self.webView is the view whose screenshot I need to take  
//The screenshot is saved in jpg format in the application directory to avoid any loss of quality  
in retina display devices i.e. all current devices running iOS 10  
UIGraphicsBeginImageContextWithOptions(self.webView.bounds.size, NO, [UIScreen mainScreen].scale);  
[self.webView.layer renderInContext:UIGraphicsGetCurrentContext()];  
UIImage *image = UIGraphicsGetImageFromCurrentImageContext();  
UIGraphicsEndImageContext();  
NSString *jpgPath = [NSHomeDirectory() stringByAppendingPathComponent:@"Documents/Test.jpg"];  
[UIImageJPEGRepresentation(image, 1.0) writeToFile:jpgPath atomically:YES];  
UIImage *pop=[[UIImage alloc] initWithContentsOfFile:jpgPath];  
//pop is the final image in jpg format and high quality with the exact resolution of the view you  
selected in pixels and not just points
```

## Section 8.6: Change UIImage Color

**Swift** Add this extension to UIImage :

```
extension UIImage {  
    func maskWithColor(color: UIColor) -> UIImage? {  
  
        let maskImage = self.CGImage  
        let width = self.size.width  
        let height = self.size.height  
        let bounds = CGRectMake(0, 0, width, height)  
  
        let colorSpace = CGColorSpaceCreateDeviceRGB()  
        let bitmapInfo = CGBitmapInfo(rawValue: CGImageAlphaInfo.PremultipliedLast.rawValue)  
        let bitmapContext = CGBitmapContextCreate(nil, Int(width), Int(height), 8, 0, colorSpace,  
        bitmapInfo.rawValue) //needs rawValue of bitmapInfo  
  
        CGContextClipToMask(bitmapContext, bounds, maskImage)  
        CGContextSetFillColorWithColor(bitmapContext, color.CGColor)  
        CGContextFillRect(bitmapContext, bounds)  
  
        //is it nil?  
        if let cImage = CGBitmapContextCreateImage(bitmapContext) {  
            let coloredImage = UIImage(CGImage: cImage)  
  
            return coloredImage  
  
        } else {  
            return nil  
        }  
    }  
}
```

Then, to change the color of your UIImage

```
my_image.maskWithColor(UIColor.blueColor())
```

Found [at this link](#)

## Section 8.7: Apply UIColor to UIImage

Use same UIImage with multiple theme base app by just applying UIColor to UIImage instance as following.

```
// *** 创建一个渲染模式为 AlwaysTemplate 的 UIImage 实例 ***
UIImage *imgMenu = [[UIImage imageNamed:@"iconMenu"]
imageWithRenderingMode:UIImageRenderingModeAlwaysTemplate];

// *** 现在将 `tintColor` 应用于 UIImageView 或 UIButton 的 UIImageView，并将图像转换为指定颜色 ***
[btn setImage:imgMenu forState:UIControlStateNormal]; // 在UIButton中设置UIImage.

[button.imageView setTintColor:[UIColor blueColor]]; // 将UIButton的图片颜色更改为蓝色
颜色
```

现在假设你想对UIImageView做同样的操作，可以使用以下代码

```
[imageView setImage:imgMenu]; // 给UIImageView赋值UIImage
[imageView setTintColor:[UIColor greenColor]]; // 将UIImageView的图片颜色更改为绿色。
[imageView setTintColor:[UIColor redColor]]; // 将UIImageView的图片颜色更改为红色。
```

## 第8.8节：使用文件内容创建和初始化图像对象

通过从指定路径的文件加载图像数据来创建并返回图像对象。

示例：

```
UIImage *image = [UIImage imageWithContentsOfFile:[[NSBundle mainBundle]
pathForResource:[cellCountry objectForKey:@"Country_Flag"] ofType:nil]];
```

使用数组：

示例

```
NSMutableArray *imageArray = [[NSMutableArray alloc] init];

for (int imageNumber = 1; self.myPhoto != nil; imageNumber++) {
    NSString *fileName = [NSString stringWithFormat:@"%@.jpg", self.myPhoto];

    // 检查文件是否存在
    if ([UIImage imageNamed:fileName]) {
        // 如果存在，则将其添加到数组中
        [imageArray addObject:[UIImage imageWithContentsOfFile:[[NSBundle
mainBundle]pathForResource:[NSString stringWithFormat:@"%@", fileName] ofType:@""]]];
    } else {
        break;
    }
}

// 使用图像数组进行动画
self.myImageView.animationImages = imageArray;
```

## 第8.9节：带边缘可调整大小的图像

在下面示例的消息气泡中：图像的角应保持不变，这由UIEdgeInsets指定，但图像的边框和中心应扩展以覆盖新的大小。



```
// *** Create an UIImage instance with RenderingMode AlwaysTemplate ***
UIImage *imgMenu = [[UIImage imageNamed:@"iconMenu"]
imageWithRenderingMode:UIImageRenderingModeAlwaysTemplate];

// *** Now Apply `tintColor` to `UIImageView` of UIImageView or UIButton and convert image in given
color ***
[btn setImage:imgMenu forState:UIControlStateNormal]; // Set UIImage in UIButton.

[button.imageView setTintColor:[UIColor blueColor]]; // It changes image color of UIButton to blue
color
```

Now lets say you want to do same with UIImageView then use following code

```
[imageView setImage:imgMenu]; // Assign UIImage to UIImageView
[imageView setTintColor:[UIColor greenColor]]; // Change imageview image color to green.
[imageView setTintColor:[UIColor redColor]]; // Change imageview image color to red.
```

## Section 8.8: Creating and Initializing Image Objects with file contents

Creating and returning an image object by loading the image data from the file at the specified path.

Example:

```
UIImage *image = [UIImage imageWithContentsOfFile:[[NSBundle mainBundle]
pathForResource:[cellCountry objectForKey:@"Country_Flag"] ofType:nil]];
```

Using Array:

Example

```
NSMutableArray *imageArray = [[NSMutableArray alloc] init];

for (int imageNumber = 1; self.myPhoto != nil; imageNumber++) {
    NSString *fileName = [NSString stringWithFormat:@"%@.jpg", self.myPhoto];

    // check if a file exists
    if ([UIImage imageNamed:fileName]) {
        // if it exists, add it to the array
        [imageArray addObject:[UIImage imageWithContentsOfFile:[[NSBundle
mainBundle]pathForResource:[NSString stringWithFormat:@"%@", fileName] ofType:@""]]];
    } else {
        break;
    }
}

// Using image array for animations here
self.myImageView.animationImages = imageArray;
```

## Section 8.9: Resizable image with caps

In the example of a message bubble illustrated below: the corners of the image should remain unchanged which is specified by UIEdgeInsets, but the borders and center of the image should expand to cover the new size.





```
let insets = UIEdgeInsetsMake(12.0, 20.0, 22.0, 12.0)
let image = UIImage(named: "test")
image?.resizableImageWithCapInsets(insets, resizingMode: .Stretch)
```

## 第8.10节：用于边界的渐变背景层

```
+ (CALayer *)gradientBGLayerForBounds:(CGRect)bounds colors:(NSArray *)colors
{
    CAGradientLayer * gradientBG = [CAGradientLayer layer];
    gradientBG.frame = bounds;
    gradientBG.colors = colors;
    return gradientBG;
}
```



```
let insets = UIEdgeInsetsMake(12.0, 20.0, 22.0, 12.0)
let image = UIImage(named: "test")
image?.resizableImageWithCapInsets(insets, resizingMode: .Stretch)
```

## Section 8.10: Gradient Background Layer for Bounds

```
+ (CALayer *)gradientBGLayerForBounds:(CGRect)bounds colors:(NSArray *)colors
{
    CAGradientLayer * gradientBG = [CAGradientLayer layer];
    gradientBG.frame = bounds;
    gradientBG.colors = colors;
    return gradientBG;
}
```

# 第9章：将NSAttributedString转换为UIImage

## 第9.1节：NSAttributedString 转 UIImage

### Objective-C

```
NSMutableAttributedString *str = [[NSMutableAttributedString alloc] initWithString:@"Hello. That  
is a test attributed string."];  
[str addAttribute:NSBackgroundColorAttributeName value:[UIColor yellowColor]  
range:NSMakeRange(3, 5)];  
[str addAttribute:NSForegroundColorAttributeName value:[UIColor greenColor]  
range:NSMakeRange(10, 7)];  
[str addAttribute:NSFontAttributeName value:[UIFont fontWithName:@"HelveticaNeue-Bold" size:20.0]  
range:NSMakeRange(20, 10)];  
UIImage *customImage = [self imageFromAttributedText:str];
```

函数 `imageFromAttributedText` 定义如下：

```
- (UIImage *)imageFromAttributedText:(NSMutableAttributedString *)text  
{  
    UIGraphicsBeginImageContextWithOptions(text.size, NO, 0.0);  
  
    // 在上下文中绘制  
    [text drawAtPoint:CGPointMake(0.0, 0.0)];  
  
    // 传输图像  
    UIImage *image = [UIGraphicsGetImageFromCurrentImageContext()  
imageWithRenderingMode:UIImageRenderingModeAlwaysOriginal];  
    UIGraphicsEndImageContext();  
  
    return image;  
}
```

# Chapter 9: Convert NSAttributedString to UIImage

## Section 9.1: NSAttributedString to UIImage Conversion

### Objective-C

```
NSMutableAttributedString *str = [[NSMutableAttributedString alloc] initWithString:@"Hello. That  
is a test attributed string."];  
[str addAttribute:NSBackgroundColorAttributeName value:[UIColor yellowColor]  
range:NSMakeRange(3, 5)];  
[str addAttribute:NSForegroundColorAttributeName value:[UIColor greenColor]  
range:NSMakeRange(10, 7)];  
[str addAttribute:NSFontAttributeName value:[UIFont fontWithName:@"HelveticaNeue-Bold" size:20.0]  
range:NSMakeRange(20, 10)];  
UIImage *customImage = [self imageFromAttributedText:str];
```

The function `imageFromAttributedText` is as defined below:

```
- (UIImage *)imageFromAttributedText:(NSMutableAttributedString *)text  
{  
    UIGraphicsBeginImageContextWithOptions(text.size, NO, 0.0);  
  
    // draw in context  
    [text drawAtPoint:CGPointMake(0.0, 0.0)];  
  
    // transfer image  
    UIImage *image = [UIGraphicsGetImageFromCurrentImageContext()  
imageWithRenderingMode:UIImageRenderingModeAlwaysOriginal];  
    UIGraphicsEndImageContext();  
  
    return image;  
}
```

# 第10章：UIImagePickerController

UIImagePickerController 提供了一个几乎开箱即用的解决方案，允许用户从他们的设备中选择一张图片，或使用相机拍照，然后展示该图片。通过遵循 UIImagePickerControllerDelegate，您可以在应用中创建逻辑，指定如何展示图片以及如何处理它（使用 didFinishPickingMediaWithInfo），同时也可以处理用户拒绝选择图片或拍照的情况（使用 imagePickerControllerDidCancel）。

## 第10.1节：UIImagePickerController 的通用用法

步骤1：创建控制器，设置代理，并遵循协议

```
//Swift
class ImageUploadViewController: UIViewController, UIImagePickerControllerDelegate,
UINavigationControllerDelegate {

    let imagePickerController = UIImagePickerController()

    override func viewDidLoad() {
        super.viewDidLoad()
        imagePickerController.delegate = self
    }
}

//Objective-C
@interface ImageUploadViewController : UIViewController
<UIImagePickerControllerDelegate, UINavigationControllerDelegate> {

    UIImagePickerController *imagePickerController;
}

@end

@implementation ImageUploadViewController
- (void)viewDidLoad {
    [super viewDidLoad];
    imagePickerController.delegate = self;
}

```

注意：我们实际上不会实现UINavigationControllerDelegate中定义的任何内容，但 UIImagePickerController继承自UINavigationController并改变了UINavigationController的行为。因此，我们仍然需要声明我们的视图控制器遵循该协议UINavigationControllerDelegate。

步骤2：每当你需要显示UIImagePickerController时：

```
//Swift
self.imagePickerController.sourceType = .Camera // 选项：.Camera, .PhotoLibrary,
.SavedPhotosAlbum
self.presentViewController(self.imagePickerController, animated: true, completion: nil)
```

# Chapter 10: UIImagePickerController

UIImagePickerController provides an almost out of the box solution to allow the user to select an image from their device or take a picture with the camera and then present that image. By conforming to the UIImagePickerControllerDelegate, you can create logic that specifies in your app how to present the image and what to do with it (using didFinishPickingMediaWithInfo) and also what to do if the user declines to select an image or take a picture (using imagePickerControllerDidCancel).

## Section 10.1: Generic usage of UIImagePickerController

Step 1: Create the controller, set the delegate, and conform to the protocol

```
//Swift
class ImageUploadViewController: UIViewController, UIImagePickerControllerDelegate,
UINavigationControllerDelegate {

    let imagePickerController = UIImagePickerController()

    override func viewDidLoad() {
        super.viewDidLoad()
        imagePickerController.delegate = self
    }
}

//Objective-C
@interface ImageUploadViewController : UIViewController
<UIImagePickerControllerDelegate, UINavigationControllerDelegate> {

    UIImagePickerController *imagePickerController;
}

@end

@implementation ImageUploadViewController
- (void)viewDidLoad {
    [super viewDidLoad];
    imagePickerController.delegate = self;
}

```

note: We actually will not implement anything defined in UINavigationControllerDelegate, but UIImagePickerController inherits from UINavigationController and changes the behavior of UINavigationController. Therefore, we still need to say our view controller conforms to UINavigationControllerDelegate.

Step 2: Whenever you need to show UIImagePickerController:

```
//Swift
self.imagePickerController.sourceType = .Camera // options: .Camera , .PhotoLibrary ,
.SavedPhotosAlbum
self.presentViewController(self.imagePickerController, animated: true, completion: nil)
```

```
//Objective-C
imagePickerController.sourceType = UIImagePickerControllerSourceTypeCamera; // 选项：
UIImagePickerControllerSourceTypeCamera, UIImagePickerControllerSourceTypePhotoLibrary,
UIImagePickerControllerSourceTypeSavedPhotosAlbum
[self presentViewController:imagePickerController animated:YES completion:nil];
```

步骤3：实现代理方法：

```
//Swift
func imagePickerController(picker: UIImagePickerController, didFinishPickingMediaWithInfo info:
[字符串: 任意对象]) {
    if let pickedImage = info[UIImagePickerControllerOriginalImage] as? UIImage {
        // 你现在有了pickedImage，在这里处理你的逻辑
    }
    self.dismissViewControllerAnimated(true, completion: nil)
}

func imagePickerControllerDidCancel(picker: UIImagePickerController) {
    self.dismissViewControllerAnimated(true, completion: nil)
}

//Objective-C
- (void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingMediaWithInfo:(NSDictionary *)info {
    UIImage *pickedImage = info[UIImagePickerControllerOriginalImage];
    if (pickedImage) {
        // 你现在有了pickedImage，在这里处理你的逻辑
    }
    [self dismissViewControllerAnimated:YES completion:nil];
}

- (void)imagePickerControllerDidCancel:(UIImagePickerController *)picker {
    [self dismissViewControllerAnimated:YES completion:nil];
}
```

```
//Objective-C
imagePickerController.sourceType = UIImagePickerControllerSourceTypeCamera; // options:
UIImagePickerControllerSourceTypeCamera, UIImagePickerControllerSourceTypePhotoLibrary,
UIImagePickerControllerSourceTypeSavedPhotosAlbum
[self presentViewController:imagePickerController animated:YES completion:nil];
```

Step 3: Implement the delegate methods:

```
//Swift
func imagePickerController(picker: UIImagePickerController, didFinishPickingMediaWithInfo info:
[String : AnyObject]) {
    if let pickedImage = info[UIImagePickerControllerOriginalImage] as? UIImage {
        // Your have pickedImage now, do your logic here
    }
    self.dismissViewControllerAnimated(true, completion: nil)
}

func imagePickerControllerDidCancel(picker: UIImagePickerController) {
    self.dismissViewControllerAnimated(true, completion: nil)
}

//Objective-C
- (void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingMediaWithInfo:(NSDictionary *)info {
    UIImage *pickedImage = info[UIImagePickerControllerOriginalImage];
    if (pickedImage) {
        //You have pickedImage now, do your logic here
    }
    [self dismissViewControllerAnimated:YES completion:nil];
}

- (void)imagePickerControllerDidCancel:(UIImagePickerController *)picker {
    [self dismissViewControllerAnimated:YES completion:nil];
}
```

# 第11章：UIImageView

## 第11.1节：使用标签遮罩UIImageView

这会使图像被遮罩成标签字母的形状：

### Objective-C

```
self.maskImage.layer.mask = self.maskLabel.layer;
self.maskImage.layer.masksToBounds = YES;
```

### Swift 3

```
maskImageView.mask = maskLabel
maskImageView.masksToBounds = true
```



## 第11.2节：将图像制作成圆形或圆角

本例展示了如何将UIView或UIImageView制作成带有一定半径的圆角，如下所示：

# Chapter 11: UIImageView

## Section 11.1: UIImageView masked with Label

This makes image masked to the shape of the letters of the label:

### Objective-C

```
self.maskImage.layer.mask = self.maskLabel.layer;
self.maskImage.layer.masksToBounds = YES;
```

### Swift 3

```
maskImageView.mask = maskLabel
maskImageView.masksToBounds = true
```

Here is the result:



## Section 11.2: Making an image into a circle or rounded

This example shows, how to make a `UIView` or `UIImageView`, rounded with some radius like this:



#### Objective-C

```
someImageView.layer.cornerRadius = CGRectGetHeight(someImageView.frame) / 2;
someImageView.clipsToBounds = YES;
```

#### Swift

```
someImageView.layer.cornerRadius = someImageView.frame.height/2
// 这应该能缓解添加透明度可能带来的性能影响 - 详见
http://stackoverflow.com/a/6254531/189804
// 如果采用此方法, 请务必使用Instruments检查滚动性能。
someImageView.layer.shouldRasterize = true
someImageView.clipsToBounds = true // 图像边界 (frame) 之外的所有部分都会被裁剪 (使圆角可见)
```

建议如果使用自动布局, 将 `someImageView.layer.cornerRadius` 代码放在 `viewDidLayoutSubviews` 中。这样当图像大小改变时, 图像的 `cornerRadius` 会更新。

```
override func viewDidLayoutSubviews() {
    super.viewDidLayoutSubviews()
    someImageView.layer.cornerRadius = someImageView.frame.size.width/2
    someImageView.layer.masksToBounds = true
}
```

## 第11.3节 : Mode属性如何影响图像

视图的 `content mode` 属性决定其内容的布局方式。在界面构建器中, 可以在属性检查器中选择各种模式。

#### Objective-C

```
someImageView.layer.cornerRadius = CGRectGetHeight(someImageView.frame) / 2;
someImageView.clipsToBounds = YES;
```

#### Swift

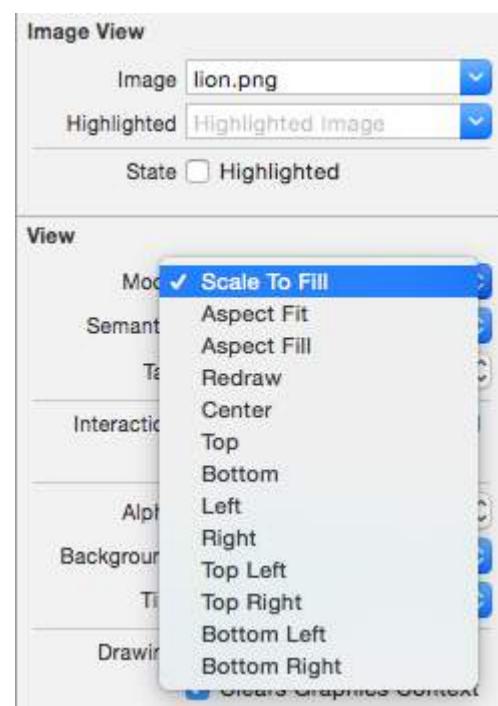
```
someImageView.layer.cornerRadius = someImageView.frame.height/2
// this should alleviate the performance hit that adding transparency may cause - see
http://stackoverflow.com/a/6254531/189804
// Be sure to check scrolling performance with Instruments if you take this approach.
someImageView.layer.shouldRasterize = true
someImageView.clipsToBounds = true // All parts of the image that are outside its bounds (the
frame) are cut out (makes the rounded corners visible)
```

It is suggested that if you use autolayout that you put the `someImageView.layer.cornerRadius` code in `viewDidLayoutSubviews`. This will allow the image's `cornerRadius` to update if the image changes size.

```
override func viewDidLayoutSubviews() {
    super.viewDidLayoutSubviews()
    someImageView.layer.cornerRadius = someImageView.frame.size.width/2
    someImageView.layer.masksToBounds = true
}
```

## Section 11.3: How the Mode property affects an image

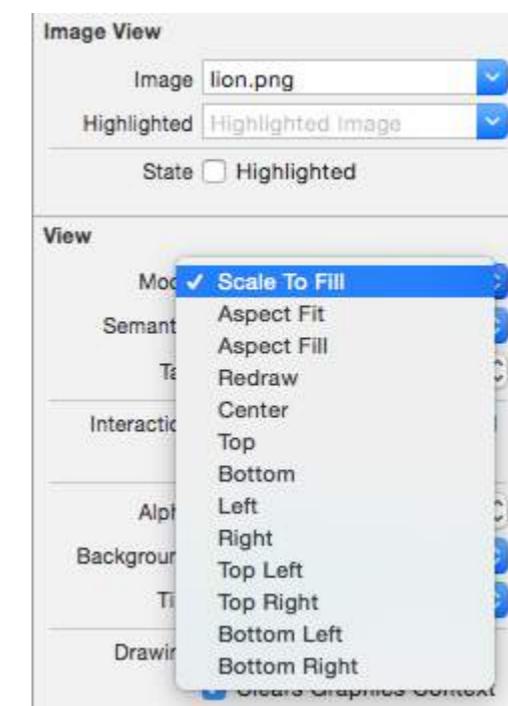
The `content mode` of a view tells how its content should be laid out. In the Interface Builder, the various modes can be selected in the Attributes Inspector.



让我们使用两个图像视图来看看各种模式的效果。



**缩放以填充**



Let's use two image views to see how the various modes work.



**Scale to Fill**



图像的高度和宽度被拉伸以匹配UIImageView的大小。

#### 等比例适应



图像的最长边（高度或宽度）被拉伸以匹配视图。这使得图像尽可能大，同时仍显示整个图像且不扭曲高度或宽度。（我将UIImageView的背景设置为蓝色，以便清楚其大小。）

#### 等比例填充



图像的最短边（高度或宽度）被拉伸以匹配视图。与“等比例适应”类似，图像的比例不会偏离其原始宽高比。

#### 重绘



重绘仅适用于需要自行缩放和调整大小的自定义视图。我们没有使用自定义视图，所以

The image heights and widths are stretched to match the size of the UIImageView.

#### Aspect Fit



The longest side (either height or width) of the image is stretched to match the view. This makes the image as big as possible while still showing the entire image and not distorting the height or width. (I set the UIImageView background to blue so that its size is clear.)

#### Aspect Fill



The shortest side (either height or width) of the image is stretched to match the view. Like "Aspect Fit", the proportions of the image are not distorted from their original aspect ratio.

#### Redraw



Redraw is only for custom views that need to do their own scaling and resizing. We aren't using a custom view, so

我们不应该使用重绘。注意这里UIImageView给出的结果与“缩放填充”相同，但它在后台做了更多工作。

关于重绘，Apple 文档中说：

内容模式适合重用视图的内容，但当您特别希望自定义视图在缩放和调整大小操作期间重新绘制自身时，也可以将内容模式设置为UIViewContentModeRedraw值。将视图的内容模式设置为此值会强制系统在几何形状变化时调用视图的drawRect:方法。通常，应尽可能避免使用此值，且绝对不应在标准系统视图中使用它。

we shouldn't use Redraw. Notice that here [UIImageView](#) just gives us the same result as Scale to Fill, but it is doing more work behind the scenes.

About Redraw, the [Apple documentation](#) says:

Content modes are good for recycling the contents of your view, but you can also set the content mode to the [UIViewContentModeRedraw](#) value when you specifically want your custom views to redraw themselves during scaling and resizing operations. Setting your view's content mode to this value forces the system to call your view's drawRect: method in response to geometry changes. In general, you should avoid using this value whenever possible, and you should certainly not use it with the standard system views.

## 居中



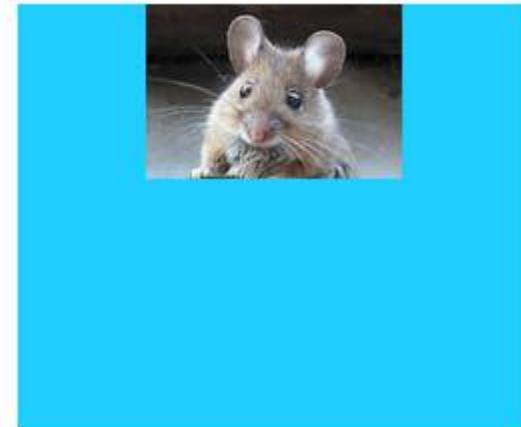
图像在视图中居中，但图像的长度和宽度未被拉伸。

## Center



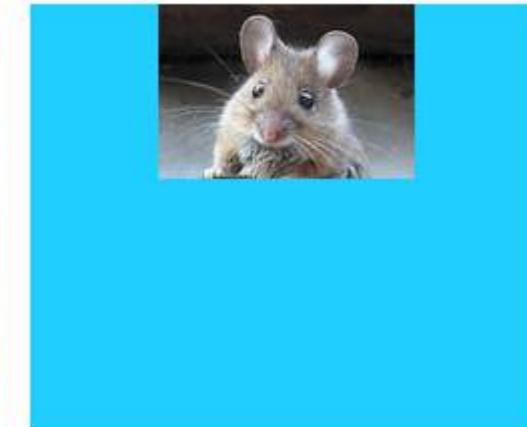
The image is centered in the view, but the length and width of the image are not stretched.

## 顶部



图像的顶部边缘在视图顶部水平居中，图像的长度和宽度未被拉伸。

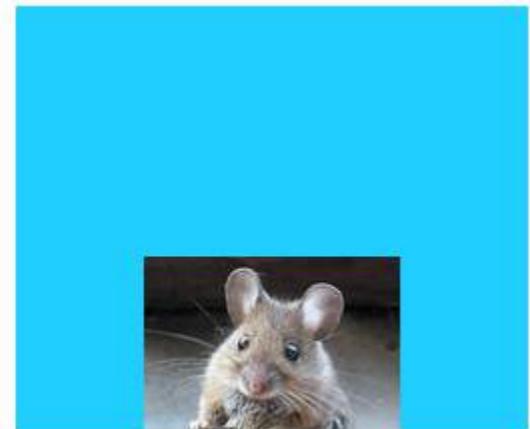
## Top



The top edge of the image is centered horizontally at the top of the view, and the length and width of the image are not stretched.

## 底部

## Bottom



图像的底部边缘在视图底部水平居中，图像的长度和宽度未被拉伸。

#### 左侧



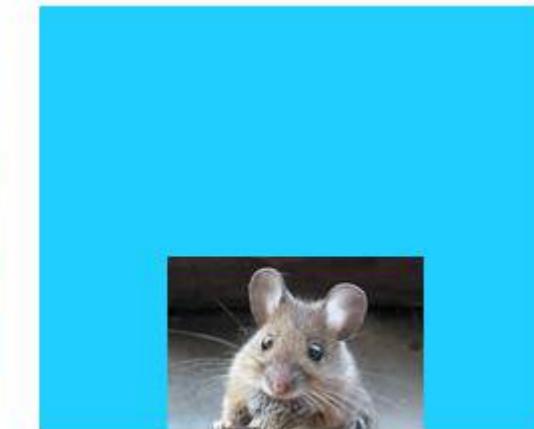
图像的左边缘在视图左侧垂直居中，图像的长度和宽度未被拉伸。

#### 右侧



图像的右边缘垂直居中于视图的右侧，图像的长度和宽度不被拉伸。

#### 左上角



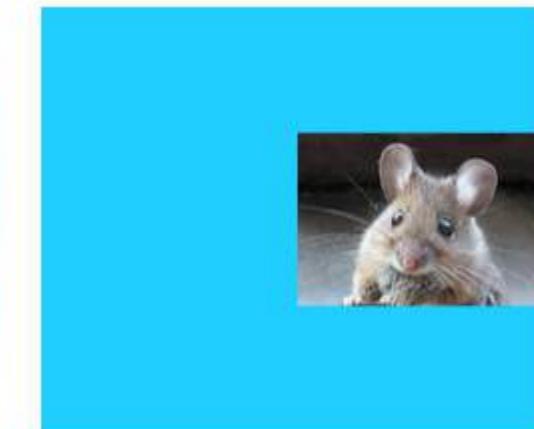
The bottom edge of the image is centered horizontally at the bottom of the view, and the length and width of the image are not stretched.

#### Left



The left edge of the image is centered vertically at the left of the view, and the length and width of the image are not stretched.

#### Right



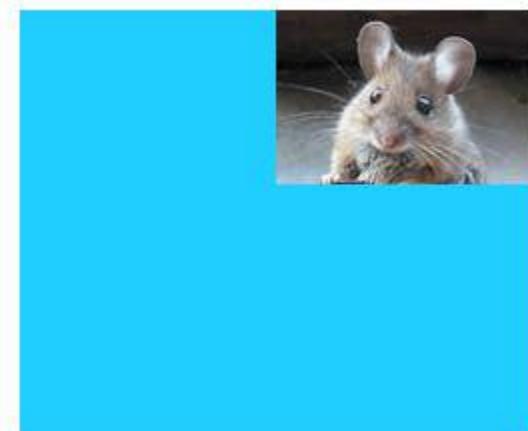
The right edge of the image is centered vertically at the right of the view, and the length and width of the image are not stretched.

#### Top Left



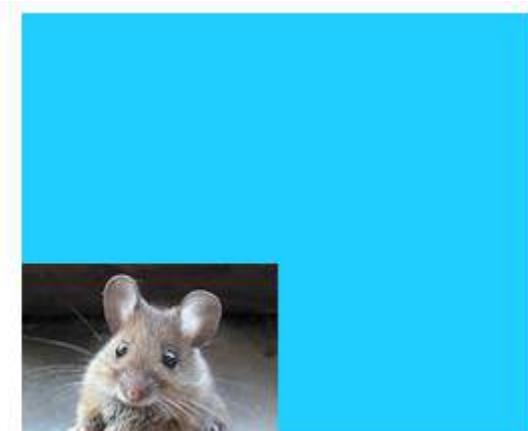
图像的左上角放置在视图的左上角。图像的长度和宽度不被拉伸。

#### 右上角



图像的右上角放置在视图的右上角。图像的长度和宽度不被拉伸。

#### 左下角



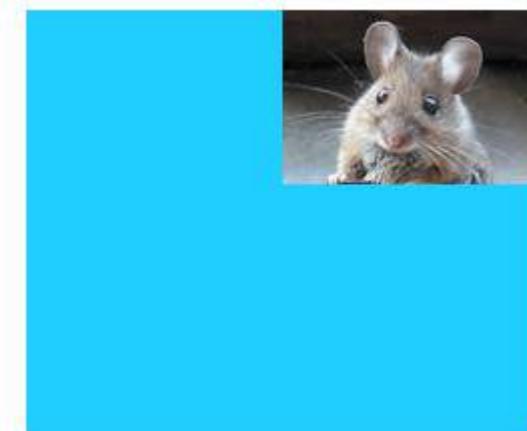
图像的左下角放置在视图的左下角。图像的长度和宽度不被拉伸。

#### 右下角



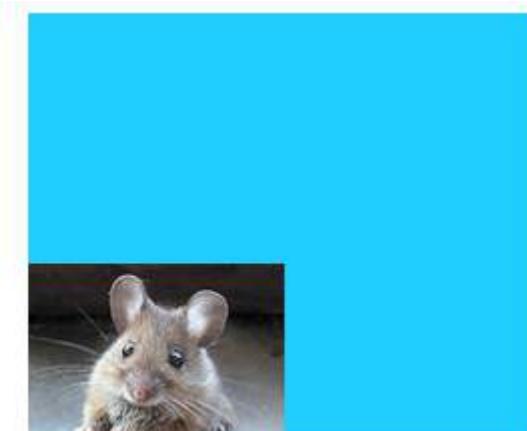
The top left corner of the image is placed at the top left corner of the view. The length and width of the image are not stretched.

#### Top Right



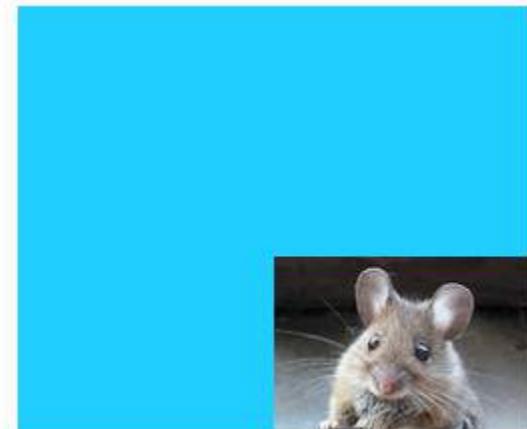
The top right corner of the image is placed at the top right corner of the view. The length and width of the image are not stretched.

#### Bottom Left



The bottom left corner of the image is placed at the bottom left corner of the view. The length and width of the image are not stretched.

#### Bottom Right

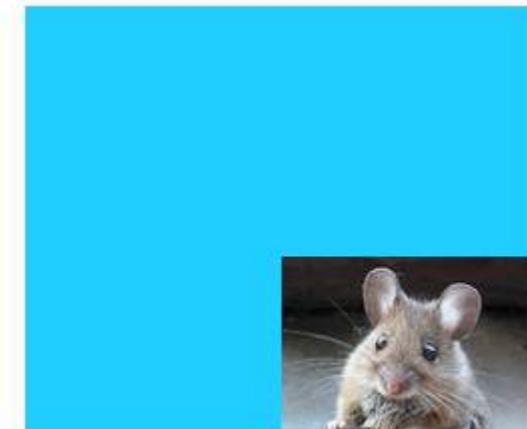


图像的右下角放置在视图的右下角。图像的长度和宽度不被拉伸。

#### 注意事项

- 此示例最初来自 [here](#).
- 如果内容（在我们的例子中是图像）与视图（在我们的例子中是 `UIImageView`）的大小相同，那么更改内容模式将不会有明显的区别。
- 请参见[this](#)和[this](#)问题，了解关于除 `UIImageView` 之外的视图内容模式的讨论。
- 在Swift中，要以编程方式设置内容模式，您可以执行以下操作：

```
imageView.contentMode = UIViewContentMode.scaleToFill  
imageView.contentMode = UIViewContentMode.scaleAspectFit  
imageView.contentMode = UIViewContentMode.scaleAspectFill  
imageView.contentMode = UIViewContentMode.redraw  
imageView.contentMode = UIViewContentMode.center  
imageView.contentMode = UIViewContentMode.top  
imageView.contentMode = UIViewContentMode.bottom  
imageView.contentMode = UIViewContentMode.left  
imageView.contentMode = UIViewContentMode.right  
imageView.contentMode = UIViewContentMode.topLeft  
imageView.contentMode = UIViewContentMode.topRight  
imageView.contentMode = UIViewContentMode.bottomLeft  
imageView.contentMode = UIViewContentMode.bottomRight
```



The bottom right corner of the image is placed at the bottom right corner of the view. The length and width of the image are not stretched.

#### Notes

- This example comes originally from [here](#).
- If the content (in our case the image) is the same size as the view (in our case the `UIImageView`), then changing the content mode will make no noticeable difference.
- See [this](#) and [this](#) question for a discussion about content modes for views other than `UIImageView`.
- In Swift, to set to set the content mode programmatically you do the following:

```
imageView.contentMode = UIViewContentMode.scaleToFill  
imageView.contentMode = UIViewContentMode.scaleAspectFit  
imageView.contentMode = UIViewContentMode.scaleAspectFill  
imageView.contentMode = UIViewContentMode.redraw  
imageView.contentMode = UIViewContentMode.center  
imageView.contentMode = UIViewContentMode.top  
imageView.contentMode = UIViewContentMode.bottom  
imageView.contentMode = UIViewContentMode.left  
imageView.contentMode = UIViewContentMode.right  
imageView.contentMode = UIViewContentMode.topLeft  
imageView.contentMode = UIViewContentMode.topRight  
imageView.contentMode = UIViewContentMode.bottomLeft  
imageView.contentMode = UIViewContentMode.bottomRight
```

## 第11.4节：UIImageView动画

您可以通过快速依次显示图像来为 `UIImageView` 制作动画，使用 `UIImageView` 的动画属性：

```
imageView.animationImages = [UIImage(named: "image1")!,  
                             UIImage(named: "image2")!,  
                             UIImage(named: "image3")!,  
                             UIImage(named: "image4")!,  
                             UIImage(named: "image5")!,  
                             UIImage(named: "image6")!,  
                             UIImage(named: "image7")!,  
                             UIImage(named: "image8")!]  
  
imageView.animationDuration = 0.3  
imageView.animationRepeatCount = 1
```

## Section 11.4: Animating a UIImageView

You can animate a `UIImageView` by quickly displaying images on it in a sequence using the `UIImageView`'s animation properties:

```
imageView.animationImages = [UIImage(named: "image1")!,  
                             UIImage(named: "image2")!,  
                             UIImage(named: "image3")!,  
                             UIImage(named: "image4")!,  
                             UIImage(named: "image5")!,  
                             UIImage(named: "image6")!,  
                             UIImage(named: "image7")!,  
                             UIImage(named: "image8")!]  
  
imageView.animationDuration = 0.3  
imageView.animationRepeatCount = 1
```

animationImages属性是一个UIImageView数组，当动画被触发时，会从上到下依次播放。

animationDuration属性是一个Double类型，表示动画运行的秒数。

animationRepeatCount 属性是一个 Int 类型，表示动画将运行的次数。

要开始和停止动画，可以调用相应的方法：

```
imageView.startAnimating()  
imageView.stopAnimating()
```

有一个方法 isAnimating()，它返回一个 Boolean 值，表示动画当前是否正在运行。

请注意，这不是创建动画的高效方式：它相当慢且资源消耗大。

考虑使用图层（Layers）或精灵（Sprites）以获得更好的效果

## 第11.5节：创建 UIImageView

要以编程方式创建 UIImageView，只需创建一个 UIImageView 实例：

```
//Swift  
let imageView = UIImageView()  
  
//Objective-C  
UIImageView *imageView = [[UIImageView alloc] init];
```

您可以使用CGRect设置UIImageView的大小和位置：

```
//Swift  
imageView.frame = CGRect(x: 0, y: 0, width: 200, height: 200)  
  
//Objective-C  
imageView.frame = CGRectMake(0,0,200,200);
```

或者你可以在初始化时设置大小：

```
//Swift  
UIImageView(frame: CGRect(x: 0, y: 0, width: 200, height: 200))  
  
//Objective-C  
UIImageView *imageView = [[UIImageView alloc] initWithFrame:CGRectMake(0,0,200,200);  
  
//定义 UIImageView 帧的另一种方式  
UIImageView *imageView = [[UIImageView alloc] init];  
CGRect imageViewFrame = imageView.frame;  
imageViewFrame.size.width = 200;  
imageViewFrame.size.height = 200;  
imageViewFrame.origin.x = 0;  
imageViewFrame.origin.y = 0;  
imageView.frame = imageViewFrame;
```

注意：使用 UIKit 时必须导入 UIImageView。

The animationImages property is an [Array](#) of UIImageViews that is run through from top to bottom when the animation is triggered.

The animationDuration property is a [Double](#) saying how many seconds the animation will run for.

The animationRepeatCount property is an [Int](#) that says how many times the animation will run.

To start and stop the animation, you can call the appropriate methods to do so:

```
imageView.startAnimating()  
imageView.stopAnimating()
```

There is method [isAnimating\(\)](#) which returns a Boolean value indicating whether the animation is running at a moment or not.

Please note that this's not a very efficient way to create animations: it's quite slow and resource-consuming.  
Consider using Layers or Sprites for better results

## Section 11.5: Create a UIImageView

To create a UIImageView programmatically, all you need to do is create an instance of [UIImageView](#):

```
//Swift  
let imageView = UIImageView()  
  
//Objective-C  
UIImageView *imageView = [[UIImageView alloc] init];
```

You can set the size and position of the UIImageView with a [CGRect](#):

```
//Swift  
imageView.frame = CGRect(x: 0, y: 0, width: 200, height: 200)  
  
//Objective-C  
imageView.frame = CGRectMake(0,0,200,200);
```

Or you can set the size during initialization:

```
//Swift  
UIImageView(frame: CGRect(x: 0, y: 0, width: 200, height: 200))  
  
//Objective-C  
UIImageView *imageView = [[UIImageView alloc] initWithFrame:CGRectMake(0,0,200,200);  
  
//Alternative way of defining frame for UIImageView  
UIImageView *imageView = [[UIImageView alloc] init];  
CGRect imageViewFrame = imageView.frame;  
imageViewFrame.size.width = 200;  
imageViewFrame.size.height = 200;  
imageViewFrame.origin.x = 0;  
imageViewFrame.origin.y = 0;  
imageView.frame = imageViewFrame;
```

Note: You must import UIKit to use a UIImageView.

## 第11.6节：更改图像颜色

```
//Swift  
imageView.tintColor = UIColor.redColor()  
imageView.image = imageView.image?.imageWithRenderingMode(.AlwaysTemplate)
```

```
//Swift 3  
imageView.tintColor = UIColor.red  
imageView.image = imageView.image?.withRenderingMode(.alwaysTemplate)
```

```
//Objective-C  
imageView.tintColor = [UIColor redColor];  
imageView.image = [imageView.image imageWithRenderingMode:UIImageRenderingModeAlwaysTemplate]
```

## 第11.7节：给UIImageView分配图像

您可以在初始化时，或稍后使用image属性给UIImageView分配图像：

```
//Swift  
UIImageView(image: UIImage(named: "image1"))  
  
UIImageView(image: UIImage(named: "image1"), highlightedImage: UIImage(named: "image2"))  
  
imageView.image = UIImage(named: "image1")  
  
//Objective-C  
[[UIImageView alloc] initWithImage:[UIImage imageNamed:@"image1"]];  
  
[[UIImageView alloc] initWithImage:[UIImage imageNamed:@"image1"] highlightedImage:[UIImage imageNamed:@"image2"]];  
  
imageView.image = [UIImage imageNamed:@"image1"];
```

## Section 11.6: Change color of an image

```
//Swift  
imageView.tintColor = UIColor.redColor()  
imageView.image = imageView.image?.imageWithRenderingMode(.AlwaysTemplate)
```

```
//Swift 3  
imageView.tintColor = UIColor.red  
imageView.image = imageView.image?.withRenderingMode(.alwaysTemplate)
```

```
//Objective-C  
imageView.tintColor = [UIColor redColor];  
imageView.image = [imageView.image imageWithRenderingMode:UIImageRenderingModeAlwaysTemplate]
```

## Section 11.7: Assigning an image to a UIImageView

You can assign an image to a `UIImageView` during initialization, or later using the `image` property:

```
//Swift  
UIImageView(image: UIImage(named: "image1"))  
  
UIImageView(image: UIImage(named: "image1"), highlightedImage: UIImage(named: "image2"))  
  
imageView.image = UIImage(named: "image1")  
  
//Objective-C  
[[UIImageView alloc] initWithImage:[UIImage imageNamed:@"image1"]];  
  
[[UIImageView alloc] initWithImage:[UIImage imageNamed:@"image1"] highlightedImage:[UIImage imageNamed:@"image2"]];  
  
imageView.image = [UIImage imageNamed:@"image1"];
```

# 第12章：调整UIImageView大小

## CG插值质量

用于渲染图像的插值质量级别。

插值质量是一个图形状态参数 `typedef enum CGInterpolationQuality CGInterpolationQuality;`

## 第12.1节：按尺寸和质量调整任何图像大小

```
- (UIImage *)drawImageBySize:(CGSize)size quality:(CGInterpolationQuality)quality
{
    UIGraphicsBeginImageContextWithOptions(size, NO, 0.0);
    CGContextRef context = UIGraphicsGetCurrentContext();
    CGContextSetInterpolationQuality(context, quality);
    [self drawInRect: CGRectMake(0, 0, size.width, size.height)];
    UIImage *resizedImage = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    return resizedImage;
}
```

# Chapter 12: Resizing UIImage

## CGInterpolationQuality

Levels of interpolation quality for rendering an image.

Interpolation quality is a graphics state parameter `typedef enum CGInterpolationQuality CGInterpolationQuality;`

## Section 12.1: Resize any image by size & quality

```
- (UIImage *)drawImageBySize:(CGSize)size quality:(CGInterpolationQuality)quality
{
    UIGraphicsBeginImageContextWithOptions(size, NO, 0.0);
    CGContextRef context = UIGraphicsGetCurrentContext();
    CGContextSetInterpolationQuality(context, quality);
    [self drawInRect: CGRectMake(0, 0, size.width, size.height)];
    UIImage *resizedImage = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    return resizedImage;
}
```

# 第13章：将UIImage裁剪成圆形

## 第13.1节：将图像裁剪成圆形 - Objective C

```
import #include <math.h>
```

viewDidLoad或loadView中的代码应类似如下

```
- (void)loadView
{
[super loadView];
UIImageView *imageView=[[UIImageView alloc]initWithFrame:CGRectMake(0, 50, 320, 320)];
[self.view addSubview:imageView];
UIImage *image=[UIImage imageNamed:@"Dubai-Photos-Images-Travel-Tourist-Images-
Pictures-800x600.jpg"];
imageView.image=[self circularScaleAndCropImage:[UIImage imageNamed:@"Dubai-Photos-Images-Travel-
Tourist-Images-Pictures-800x600.jpg"] frame:CGRectMake(0, 0, 320, 320)];
}
```

最后，执行主要操作的函数circularScaleAndCropImage定义如下

```
- (UIImage*)circularScaleAndCropImage:(UIImage*)image frame:(CGRect)frame {
    // 该函数返回一个基于image的新图像，已完成以下操作：
    // - 缩放以适应(CGRect) rect
    // - 并在半径为rectWidth/2的圆内裁剪

    // 创建位图图形上下文
    UIGraphicsBeginImageContextWithOptions(CGSizeMake(frame.size.width, frame.size.height), NO,
    0.0);
    CGContextRef context = UIGraphicsGetCurrentContext();

    // 获取宽度和高度
    CGFloat imageWidth = image.size.width;
    CGFloat imageHeight = image.size.height;
    CGFloat rectWidth = frame.size.width;
    CGFloat rectHeight = frame.size.height;

    // 计算缩放因子
    CGFloat scaleFactorX = rectWidth/imageWidth;
    CGFloat scaleFactorY = rectHeight/imageHeight;

    // 计算圆心
    CGFloat imageCentreX = rectWidth/2;
    CGFloat imageCentreY = rectHeight/2;

    // 创建并裁剪到一个圆形路径
    // (如果想要不同形状的裁剪，可以替换为任意闭合路径)
    CGFloat radius = rectWidth/2;
    CGContextBeginPath (context);
    CGContextAddArc (context, imageCentreX, imageCentreY, radius, 0, 2*M_PI, 0);
    CGContextClosePath (context);
    CGContextClip (context);

    // 设置图形上下文的缩放因子
    // 所有后续绘制调用都将按此因子缩放
    CGContextScaleCTM (context, scaleFactorX, scaleFactorY);

    // 绘制图像
    CGRect myRect = CGRectMake(0, 0, imageWidth, imageHeight);
```

# Chapter 13: Cut a UIImage into a circle

## Section 13.1: Cut a image into a circle - Objective C

```
import #include <math.h>
```

The code in the viewDidLoad or loadView should look something like this

```
- (void)loadView
{
[super loadView];
UIImageView *imageView=[[UIImageView alloc]initWithFrame:CGRectMake(0, 50, 320, 320)];
[self.view addSubview:imageView];
UIImage *image=[UIImage imageNamed:@"Dubai-Photos-Images-Travel-Tourist-Images-
Pictures-800x600.jpg"];
imageView.image=[self circularScaleAndCropImage:[UIImage imageNamed:@"Dubai-Photos-Images-Travel-
Tourist-Images-Pictures-800x600.jpg"] frame:CGRectMake(0, 0, 320, 320)];
}
```

Finally the function which does the heavy lifting circularScaleAndCropImage is as defined below

```
- (UIImage*)circularScaleAndCropImage:(UIImage*)image frame:(CGRect)frame {
    // This function returns a newImage, based on image, that has been:
    // - scaled to fit in (CGRect) rect
    // - and cropped within a circle of radius: rectWidth/2

    // Create the bitmap graphics context
    UIGraphicsBeginImageContextWithOptions(CGSizeMake(frame.size.width, frame.size.height), NO,
    0.0);
    CGContextRef context = UIGraphicsGetCurrentContext();

    // Get the width and heights
    CGFloat imageWidth = image.size.width;
    CGFloat imageHeight = image.size.height;
    CGFloat rectWidth = frame.size.width;
    CGFloat rectHeight = frame.size.height;

    // Calculate the scale factor
    CGFloat scaleFactorX = rectWidth/imageWidth;
    CGFloat scaleFactorY = rectHeight/imageHeight;

    // Calculate the centre of the circle
    CGFloat imageCentreX = rectWidth/2;
    CGFloat imageCentreY = rectHeight/2;

    // Create and CLIP to a CIRCULAR Path
    // (This could be replaced with any closed path if you want a different shaped clip)
    CGFloat radius = rectWidth/2;
    CGContextBeginPath (context);
    CGContextAddArc (context, imageCentreX, imageCentreY, radius, 0, 2*M_PI, 0);
    CGContextClosePath (context);
    CGContextClip (context);

    // Set the SCALE factor for the graphics context
    // All future draw calls will be scaled by this factor
    CGContextScaleCTM (context, scaleFactorX, scaleFactorY);

    // Draw the IMAGE
    CGRect myRect = CGRectMake(0, 0, imageWidth, imageHeight);
```

```

[image drawInRect:myRect];

UIImage *newImage = UIGraphicsGetImageFromCurrentImageContext();
UIGraphicsEndImageContext();

return newImage;
}

```

## 第13.2节：SWIFT 3示例

```

override func viewDidLoad() {
    super.viewDidLoad()
    // 在视图加载后执行任何额外的设置，通常来自nib文件。
    let imageView = UIImageView(frame: CGRect(x: CGFloat(0), y: CGFloat(50), width:
    CGFloat(320), height: CGFloat(320)))
    view.addSubview(imageView)
    let image = UIImage(named: "Dubai-Photos-Images-Travel-Tourist-Images-
    Pictures-800x600.jpg")
    imageView.image = circularScaleAndCropImage(UIImage(named: "Dubai-Photos-Images-Travel-
    Tourist-Images-Pictures-800x600.jpg")!, frame: CGRect(x: CGFloat(0), y: CGFloat(0), width:
    CGFloat(100), height: CGFloat(100)))
}

```

最后，执行主要操作的函数 `circularScaleAndCropImage` 定义如下

```

func circularScaleAndCropImage(_ image: UIImage, frame: CGRect) -> UIImage{
    // 该函数返回一个基于 image 的新图像，已被：
    // - 缩放以适应(CGRect) rect
    // - 并在半径为rectWidth/2的圆内裁剪
    // 创建位图图形上下文
    UIGraphicsBeginImageContextWithOptions(CGSize(width: CGFloat(frame.size.width), height:
    CGFloat(frame.size.height)), false, 0.0)
    let context: CGContext? = UIGraphicsGetCurrentContext()
    // 获取宽度和高度
    let imageWidth: CGFloat = image.size.width
    let imageHeight: CGFloat = image.size.height
    let rectWidth: CGFloat = frame.size.width
    let rectHeight: CGFloat = frame.size.height
    // 计算缩放因子
    let scaleFactorX: CGFloat = rectWidth / imageWidth
    let scaleFactorY: CGFloat = rectHeight / imageHeight
    // 计算圆心
    let imageCentreX: CGFloat = rectWidth / 2
    let imageCentreY: CGFloat = rectHeight / 2
    // 创建并裁剪到一个圆形路径
    // (如果你想要不同形状的裁剪路径，可以替换为任何闭合路径)
    let radius: CGFloat = rectWidth / 2
    context?.beginPath()
    context?.addArc(center: CGPoint(x: imageCentreX, y: imageCentreY), radius: radius,
    startAngle: CGFloat(0), endAngle: CGFloat(2 * Float.pi), clockwise: false)
    context?.closePath()
    context?.clip()
    // 设置图形上下文的缩放因子
    // 所有后续绘制调用都将按此因子缩放
    context?.scaleBy(x: scaleFactorX, y: scaleFactorY)
    // 绘制图像
    let myRect = CGRect(x: CGFloat(0), y: CGFloat(0), width: imageWidth, height: imageHeight)
    image.draw(in: myRect)
    let newImage: UIImage? = UIGraphicsGetImageFromCurrentImageContext()
    UIGraphicsEndImageContext()
}

```

```

[image drawInRect:myRect];

UIImage *newImage = UIGraphicsGetImageFromCurrentImageContext();
UIGraphicsEndImageContext();

return newImage;
}

```

## Section 13.2: SWIFT 3 Example

```

override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view, typically from a nib.
    let imageView = UIImageView(frame: CGRect(x: CGFloat(0), y: CGFloat(50), width:
    CGFloat(320), height: CGFloat(320)))
    view.addSubview(imageView)
    let image = UIImage(named: "Dubai-Photos-Images-Travel-Tourist-Images-
    Pictures-800x600.jpg")
    imageView.image = circularScaleAndCropImage(UIImage(named: "Dubai-Photos-Images-Travel-
    Tourist-Images-Pictures-800x600.jpg")!, frame: CGRect(x: CGFloat(0), y: CGFloat(0), width:
    CGFloat(100), height: CGFloat(100)))
}

```

Finally the function which does the heavy lifting `circularScaleAndCropImage` is as defined below

```

func circularScaleAndCropImage(_ image: UIImage, frame: CGRect) -> UIImage{
    // This function returns a newImage, based on image, that has been:
    // - scaled to fit in (CGRect) rect
    // - and cropped within a circle of radius: rectWidth/2
    // Create the bitmap graphics context
    UIGraphicsBeginImageContextWithOptions(CGSize(width: CGFloat(frame.size.width), height:
    CGFloat(frame.size.height)), false, 0.0)
    let context: CGContext? = UIGraphicsGetCurrentContext()
    // Get the width and heights
    let imageWidth: CGFloat = image.size.width
    let imageHeight: CGFloat = image.size.height
    let rectWidth: CGFloat = frame.size.width
    let rectHeight: CGFloat = frame.size.height
    // Calculate the scale factor
    let scaleFactorX: CGFloat = rectWidth / imageWidth
    let scaleFactorY: CGFloat = rectHeight / imageHeight
    // Calculate the centre of the circle
    let imageCentreX: CGFloat = rectWidth / 2
    let imageCentreY: CGFloat = rectHeight / 2
    // Create and CLIP to a CIRCULAR Path
    // (This could be replaced with any closed path if you want a different shaped clip)
    let radius: CGFloat = rectWidth / 2
    context?.beginPath()
    context?.addArc(center: CGPoint(x: imageCentreX, y: imageCentreY), radius: radius,
    startAngle: CGFloat(0), endAngle: CGFloat(2 * Float.pi), clockwise: false)
    context?.closePath()
    context?.clip()
    // Set the SCALE factor for the graphics context
    // All future draw calls will be scaled by this factor
    context?.scaleBy(x: scaleFactorX, y: scaleFactorY)
    // Draw the IMAGE
    let myRect = CGRect(x: CGFloat(0), y: CGFloat(0), width: imageWidth, height: imageHeight)
    image.draw(in: myRect)
    let newImage: UIImage? = UIGraphicsGetImageFromCurrentImageContext()
    UIGraphicsEndImageContext()
}

```

```
    return newImage!  
}
```

```
    return newImage!  
}
```

# 第14章：UITableView

一种简单、广泛使用且非常强大的视图，可以使用行和单列的形式呈现数据列表。用户可以垂直滚动浏览表视图中的项目，并可选择性地操作和选择内容。

## 第14.1节：自适应单元格

在iOS 8中，苹果引入了自适应单元格。使用自动布局明确布局你的UITableCells，UITableView会为你处理其余部分。行高会自动计算，默认的rowHeight值是UITableViewAutomaticDimension。

UITableView属性estimatedRowHeight在自适应单元格计算时使用。

当你创建自适应表视图单元格时，需要设置此属性并使用约束来定义单元格的大小。

-- 苹果, UITableView文档

```
self.tableView.estimatedRowHeight = 44.0
```

请注意，如果你希望所有单元格高度动态变化，则不需要实现tableView的代理方法heightForRowAtIndexpath。只需在必要时并在重新加载或加载表视图之前设置上述属性即可。但是，你可以通过以下函数为特定单元格设置固定高度，同时让其他单元格保持动态：

### Swift

```
override func tableView(tableView: UITableView, heightForRowAtIndexPath indexPath: NSIndexPath) -> CGFloat {
    switch indexPath.section {
    case 1:
        return 60
    default:
        return UITableViewAutomaticDimension
    }
}
```

### Objective-C

```
- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath {
    switch (indexPath.section) {
        case 1:
            return 60;
        default:
            return UITableViewAutomaticDimension;
    }
}
```

## 第14.2节：自定义单元格

自定义UITableViewCell可以实现非常强大、动态且响应迅速的界面。通过广泛的自定义并结合其他技术，你可以实现如下功能：在属性或界面元素变化时更新它们，在单元格中进行动画或绘制，用户滚动时高效加载视频，甚至在网络下载图片时显示它们。这里的可能性几乎是无限的。下面是一个简单的示例

# Chapter 14: UITableView

A simple, widely-used, yet very powerful view that can present data in a list form using rows and a single column. Users may scroll vertically through the items in a table view, and optionally manipulate and select content.

## Section 14.1: Self-Sizing Cells

In iOS 8 Apple introduced the self sizing cell. Layout your UITableViewCells with Autolayout explicitly and UITableView takes care of the rest for you. Row height is calculated automatically, by default rowHeight value is UITableViewAutomaticDimension.

UITableView property estimatedRowHeight is used when self-sizing cell is calculating.

When you create a self-sizing table view cell, you need to set this property and use constraints to define the cell's size.

-- Apple, UITableView Documentation

```
self.tableView.estimatedRowHeight = 44.0
```

Note that the tableView's delegate's heightForRowAtIndexPath is *unnecessary* if you want to have a dynamic height for all cells. Simply set the above property when necessary and before reloading or loading the table view. However, you can set specific cells' height while having others dynamic via the following function:

### Swift

```
override func tableView(tableView: UITableView, heightForRowAtIndexPath indexPath: NSIndexPath) -> CGFloat {
    switch indexPath.section {
    case 1:
        return 60
    default:
        return UITableViewAutomaticDimension
    }
}
```

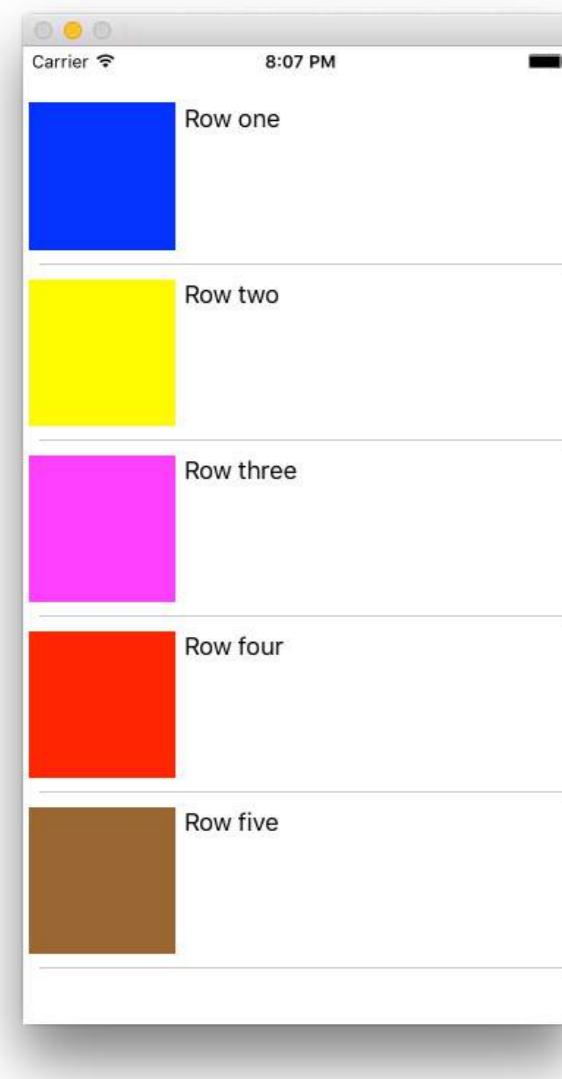
### Objective-C

```
- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath {
    switch (indexPath.section) {
        case 1:
            return 60;
        default:
            return UITableViewAutomaticDimension;
    }
}
```

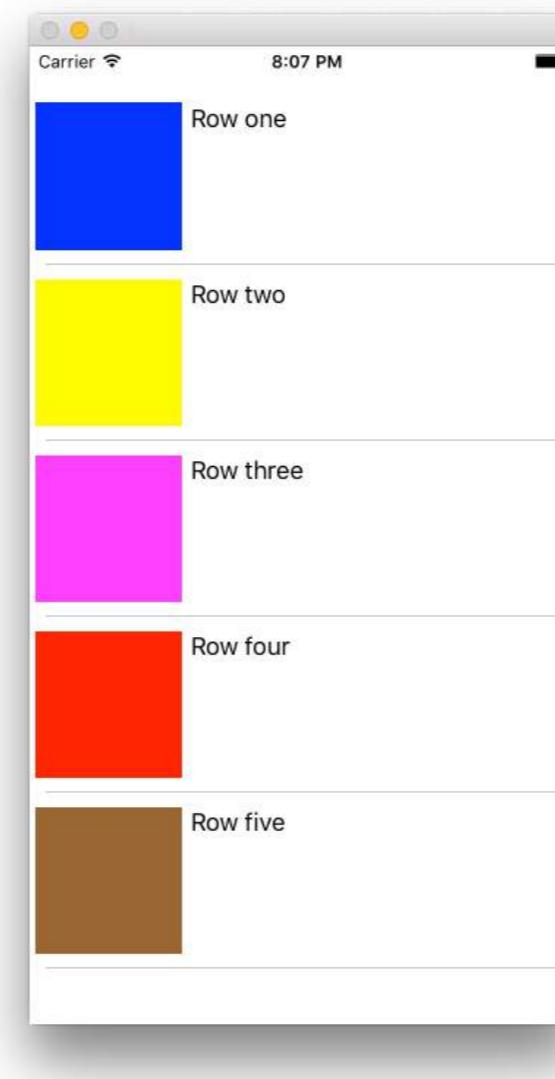
## Section 14.2: Custom Cells

Customizing a UITableViewCell can allow for very powerful, dynamic, and responsive interfaces. With extensive customization and in combination with other techniques you can do things like: update specific properties or interface elements as they change, animate or draw things in the cell, efficiently load video as the user scrolls, or even display pictures as they download from a network. The possibilities here are nearly endless. Below is a simple

自定义单元格的示例。



example of what a custom cell may look like.



本节涵盖基础内容，希望将来能扩展为详细介绍上述更复杂的流程。

## 创建自定义单元格

首先，创建一个UITableViewCell的子类（在Xcode中创建一个新的Cocoa Touch类，并将UITableView Cell设为父类）。下面是子类化后代码的示例。

### Swift

```
class CustomTableViewCell: UITableViewCell {
    static var identifier: String {
        return NSStringFromClass(self)
    }

    var customLabel: UILabel!

    override func awakeFromNib() {
        super.awakeFromNib()
        // 初始化代码
        customLabel = UILabel(frame: CGRect(x: 0, y: 0, width: contentView.frame.width, height: contentView.frame.height))
        customLabel.textAlignment = .center
    }
}
```

This section covers the basics, and hopefully will be expanded to detail more complex processes like those described above.

### Creating Your Custom Cell

First, create a new subclass of `UITableViewCell` (create a new Cocoa Touch Class in Xcode and set `UITableViewCell` as the superclass). Below is what your code may look like after subclassing.

### Swift

```
class CustomTableViewCell: UITableViewCell {
    static var identifier: String {
        return NSStringFromClass(self)
    }

    var customLabel: UILabel!

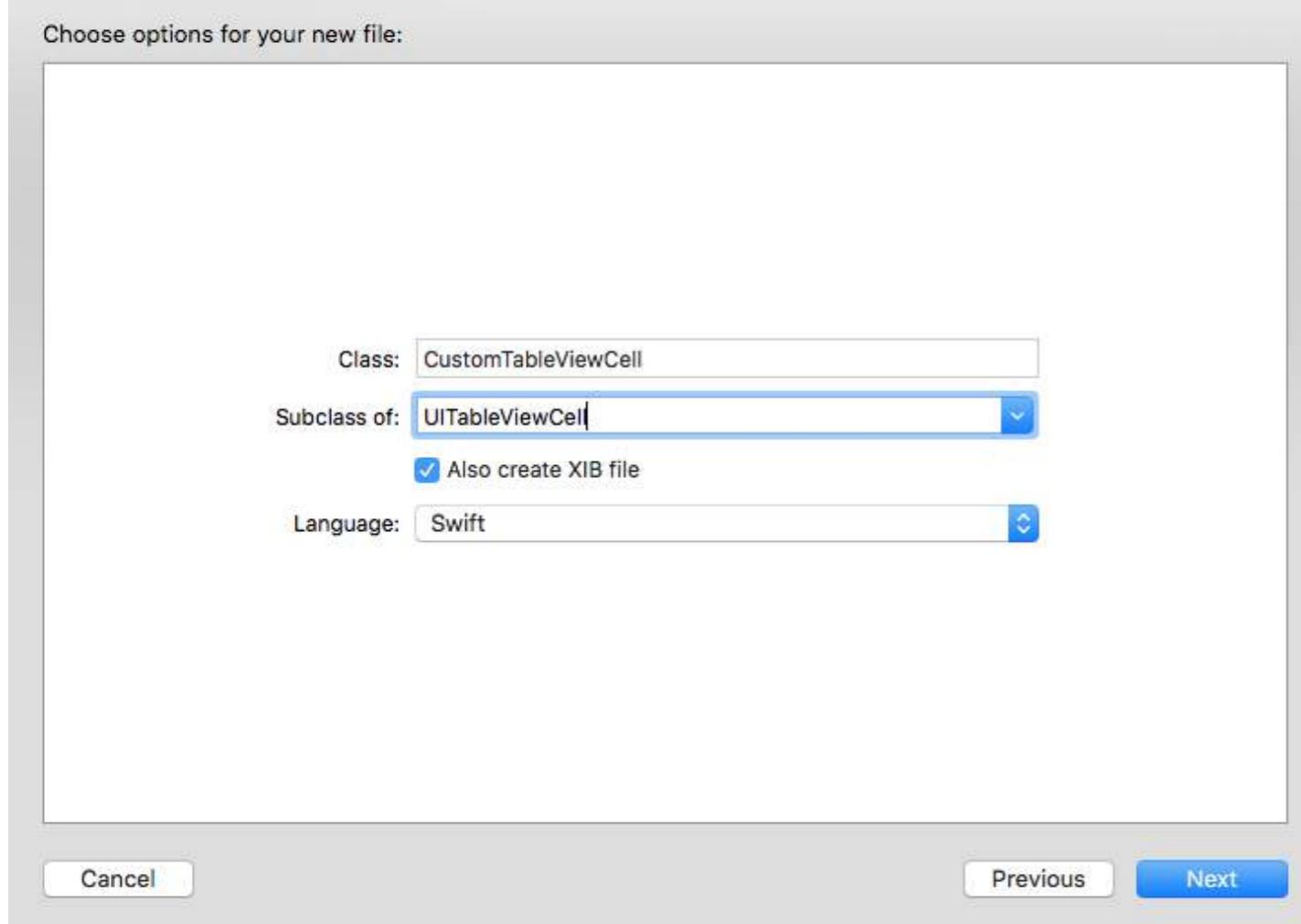
    override func awakeFromNib() {
        super.awakeFromNib()
        // Initialization code
        customLabel = UILabel(frame: CGRect(x: 0, y: 0, width: contentView.frame.width, height: contentView.frame.height))
        customLabel.textAlignment = .center
    }
}
```

```

contentView.addSubview(customLabel)
}
}

```

创建新文件时，可选择勾选“Also create a XIB file”以便使用界面构建器进行自定义。如果选择了此项，请将customLabel连接为@IBOutlet



在包含tableView的UIViewController中，注册新的自定义单元格类（见下文）。注意，只有当你没有在表视图的界面中使用Storyboard设计单元格时，才需要这样做。

### Swift

```

override func viewDidLoad() {
    super.viewDidLoad()

    // 注册单元格类
    tableView.register(CustomTableViewCell.self, forCellReuseIdentifier: CustomTableViewCell.identifier)
}

```

如果选择使用XIB文件，则改用registerNib：

### Swift

```

// 注册Nib
tableView.register(UINib(nibName: CustomTableViewCell.identifier, bundle: nil),
forCellReuseIdentifier: CustomTableViewCell.identifier)

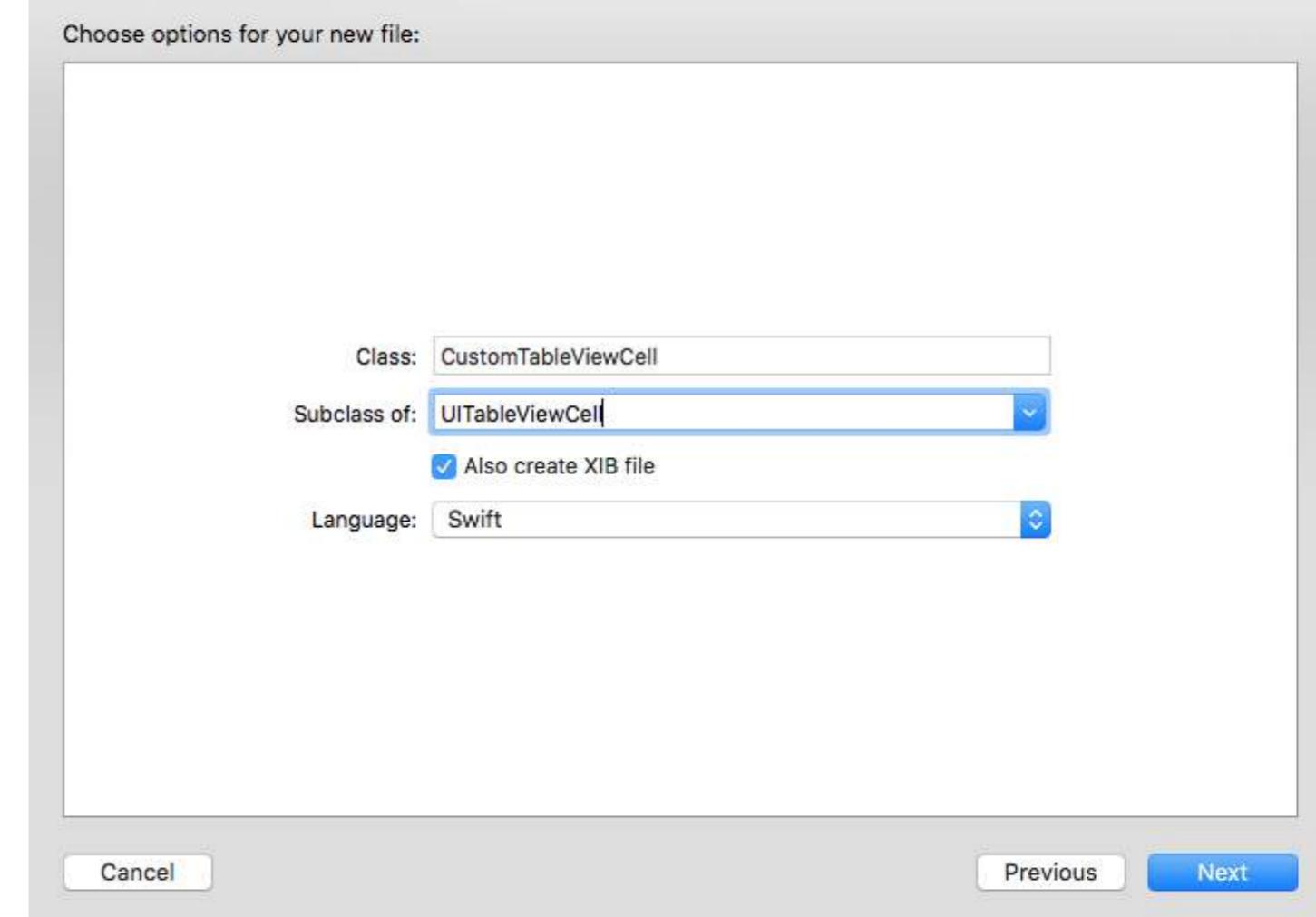
```

```

contentView.addSubview(customLabel)
}
}

```

Optionally, check 'Also create a XIB file' when creating your new file to customize using Interface Builder. In the case that you do, connect customLabel as an @IBOutlet



In a UIViewController containing the tableView, register the new custom cell's class (see below). Note, this is only necessary if you do not design the cell with a Storyboard in your table view's interface.

### Swift

```

override func viewDidLoad() {
    super.viewDidLoad()

    // Register Cell Class
    tableView.register(CustomTableViewCell.self, forCellReuseIdentifier: CustomTableViewCell.identifier)
}

```

If you chose to use a XIB file, registerNib instead:

### Swift

```

// Register Nib
tableView.register(UINib(nibName: CustomTableViewCell.identifier, bundle: nil),
forCellReuseIdentifier: CustomTableViewCell.identifier)

```

现在您的表视图(tableView)已经知道您的自定义单元格，您可以在cellForRowAtIndexPath中将其列出：

## Swift

```
func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
    // 加载 CustomTableViewCell。确保这里提供的标识符与您的单元格中的标识符匹配

    let cell: CustomTableViewCell =
        tableView.dequeueReusableCellWithIdentifier(CustomTableViewCell.identifier) as! CustomTableViewCell

    // 这里是关键所在——为您自己的单元格设置自定义属性
    cell.customLabel.text = "我的自定义单元格"

    return cell
}
```

## 第14.3节：分隔线

### 编辑分隔线的宽度

您可以通过更改单元格的  
layoutMargins: 属性，使表视图的分隔线在表格中延伸到不同的宽度。这可以通过多种方式实现。

### 更改特定单元格的分隔线

在您的表视图数据源的cellForRowAtIndexPath:方法或willDisplayCell:方法中，将单元格的layoutMargins:属性设置为UIEdgeInsetsZero（扩展到表格的全宽），或者设置为您想要的任何值。

### Objective-C

```
[cell setLayoutMargins:UIEdgeInsetsZero];

// 也可以使用 separatorInset
[cell setSeparatorInset:UIEdgeInsetsZero];
```

## Swift

```
func tableView(tableView: UITableView, willDisplayCell cell: UITableViewCell, forRowAtIndexPath indexPath: NSIndexPath) {
    cell.separatorInset = UIEdgeInsetsZero
    cell.layoutMargins = UIEdgeInsetsZero
}

func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
{
    cell.separatorInset = UIEdgeInsetsZero
    cell.layoutMargins = UIEdgeInsetsZero
}
```

### 移除所有分隔线

每个单元格之间的细灰线可能不是您想要的外观。隐藏它们相当简单。

在你包含的UIViewController的viewDidLoad:方法中添加以下代码。你也可以设置这个

Now that your tableView knows about your custom cell, you can dequeue it in cellForRowAtIndexPath:

## Swift

```
func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
    // Load the CustomTableViewCell. Make sure the identifier supplied here matches the one from
    // your cell
    let cell: CustomTableViewCell =
        tableView.dequeueReusableCellWithIdentifier(CustomTableViewCell.identifier) as! CustomTableViewCell

    // This is where the magic happens - setting a custom property on your very own cell
    cell.customLabel.text = "My Custom Cell"

    return cell
}
```

## Section 14.3: Separator Lines

### Editing the width of Separator Lines

You can set make your table view's separator lines extend the to various widths across the table by changing the layoutMargins: property on your cell(s). This can be achieved in a number of ways.

### Changing the Separator Lines for specific cells

In either your table view data source's cellForRowAtIndexPath: method or the willDisplayCell: method, set the cell's layoutMargins: property to UIEdgeInsetsZero (extends to full width of the table), or to whatever you may desire here.

### Objective-C

```
[cell setLayoutMargins:UIEdgeInsetsZero];

// May also use separatorInset
[cell setSeparatorInset:UIEdgeInsetsZero];
```

## Swift

```
func tableView(tableView: UITableView, willDisplayCell cell: UITableViewCell, forRowAtIndexPath indexPath: NSIndexPath) {
    cell.separatorInset = UIEdgeInsetsZero
    cell.layoutMargins = UIEdgeInsetsZero
}

func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
{
    cell.separatorInset = UIEdgeInsetsZero
    cell.layoutMargins = UIEdgeInsetsZero
}
```

### Remove all Separator Lines

The thin gray lines between each cell may not be exactly the look you're going for. It's fairly straightforward to hide them from view.

In your encompassing UIViewController's viewDidLoad: method add the following code. You may also set this

在加载或重新加载表视图之前的任何时间设置该属性（不一定非要在 viewDidLoad:方法中）。

#### Swift :

```
tableView.separatorStyle = .None
```

#### Objective-C :

```
tableView.separatorStyle = UITableViewCellStyleNone;
```

或者，可以在Storyboard或XIB中通过选择你的tableView并设置 separator（在属性检查器下）为None来更改该属性。

#### 隐藏多余的分隔线

你可以通过在UITableView底部设置一个空的页脚视图来隐藏空单元格的UITableViewCell分隔线：

#### Swift

```
tableView.tableFooterView = UIView()
```

#### Objective-C

```
tableView.tableFooterView = [[UIView alloc] initWithFrame:CGRectZero];
```

property at any time before loading or reloading the table view (does not necessarily need to be in the viewDidLoad: method).

#### Swift:

```
tableView.separatorStyle = .None
```

#### Objective-C:

```
tableView.separatorStyle = UITableViewCellStyleNone;
```

Alternatively, the property can be changed in your Storyboard or XIB by selecting your tableView and setting separator (under the attributes inspector) to None.

#### Hide excess Separator Lines

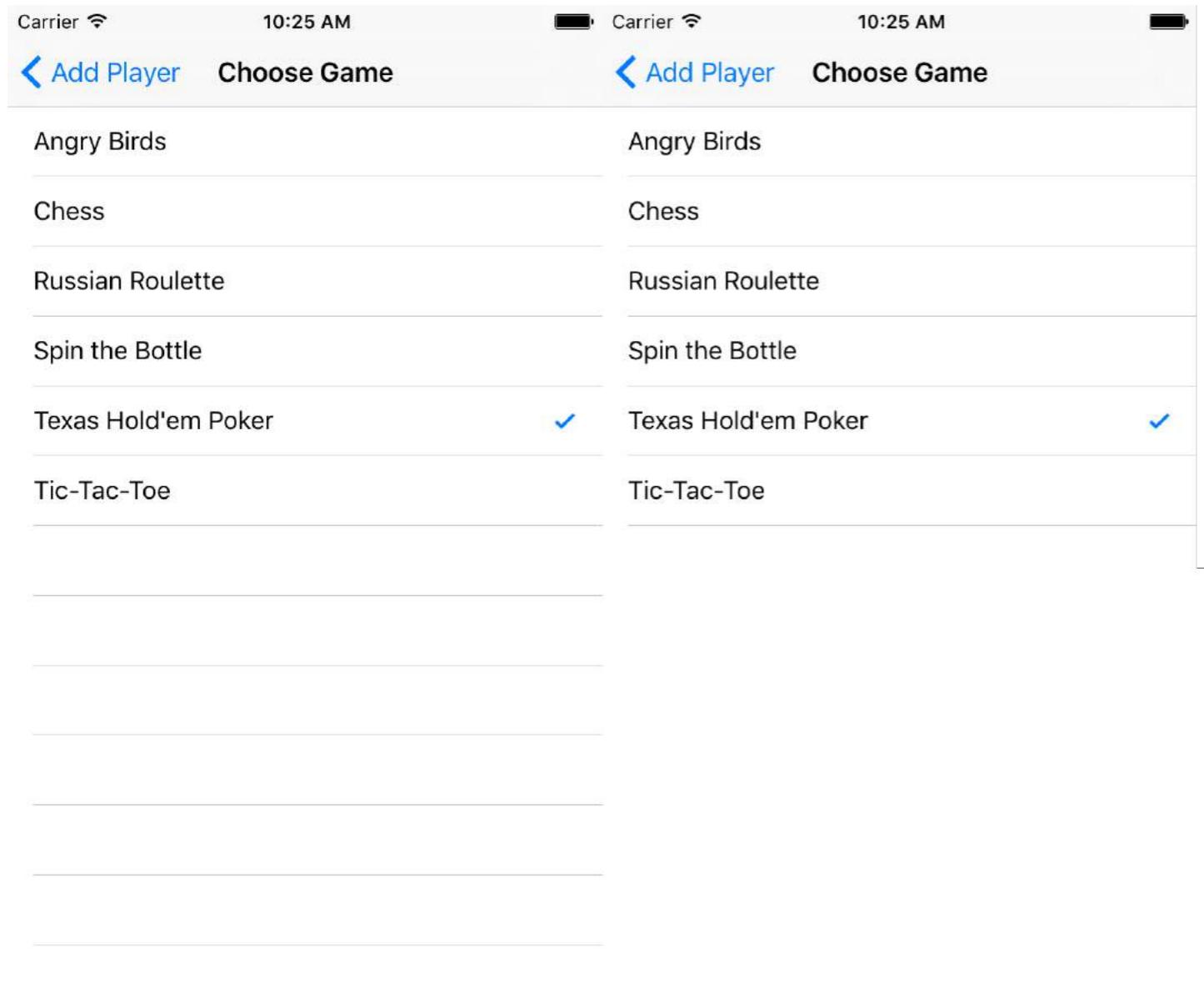
You can hide the UITableViewCell separator lines for empty cells by setting an empty footer view at the bottom of a UITableView:

#### Swift

```
tableView.tableFooterView = UIView()
```

#### Objective-C

```
tableView.tableFooterView = [[UIView alloc] initWithFrame:CGRectZero];
```



图片来自Ray Wenderlich。

## 第14.4节：代理和数据源

`UITableViewDelegate` 用于控制表格的显示方式，`UITableViewDataSource` 用于定义 `UITableView` 的数据。有两个必需的方法和许多可选的方法，可以用来定制 `UITableView` 中的大小、分区、标题和单元格。

### UITableViewDataSource

#### 必需的方法

`numberOfRowsInSection:` 此方法定义每个分区中将显示多少个单元格。

#### Objective-C

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
```

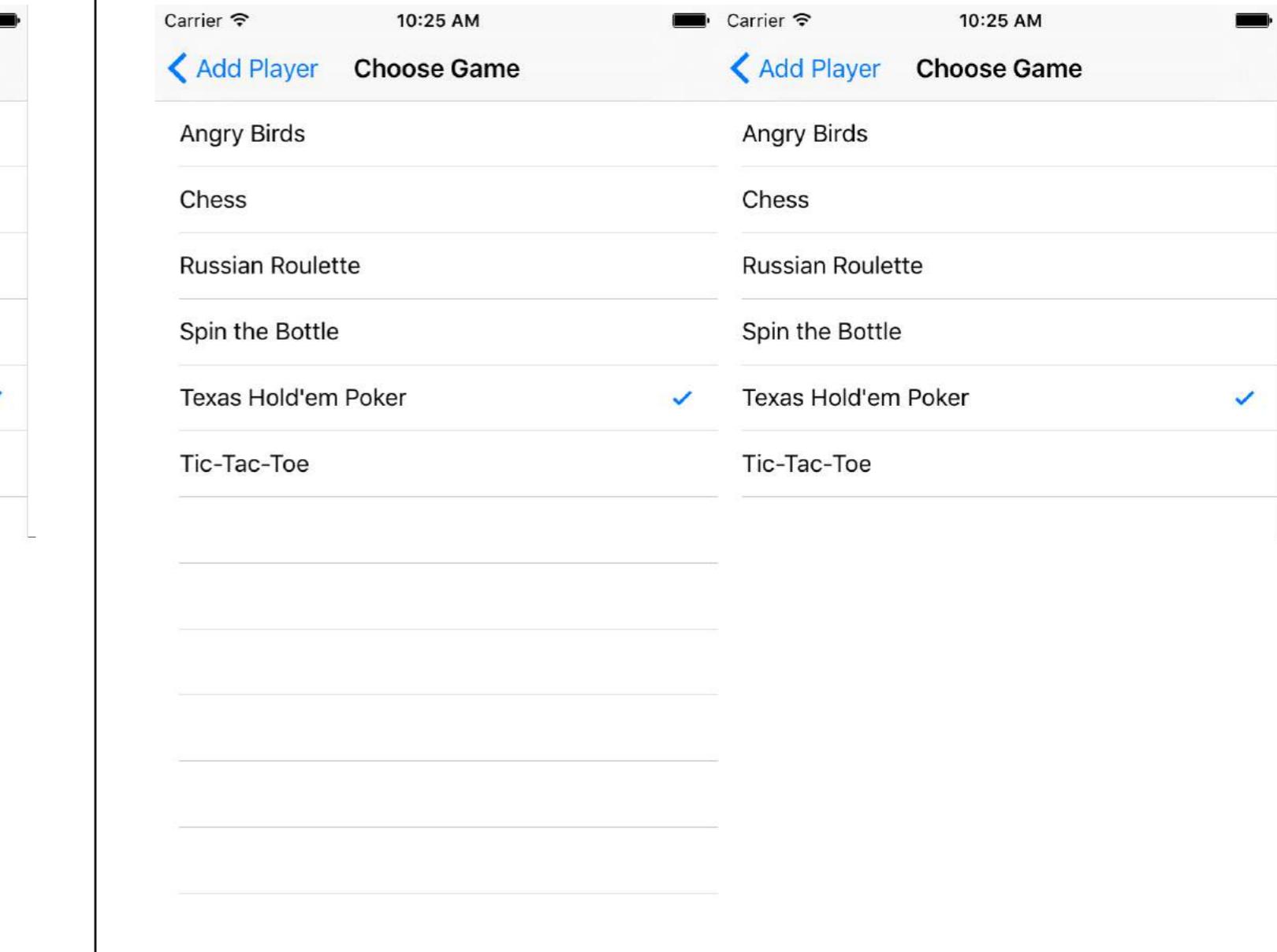


Image is from [Ray Wenderlich](#).

## Section 14.4: Delegate and Datasource

The `UITableViewDelegate` is used to control how the table is displayed, and `UITableViewDataSource` is used to define the `UITableView`'s data. There are two required methods and many optional ones which can be used to customize size, sections, headings, and cells in the `UITableView`.

### UITableViewDataSource

#### Required Methods

`numberOfRowsInSection:` This method defines how many cells will be displayed in each section of the tableview.

#### Objective-C

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
```

```
// 返回表格视图的行数。通常从数组中填充,  
// 或者可以静态定义。  
return self.myArray.count;  
}
```

### Swift 3

```
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {  
    // 返回表视图的行数。通常从数组中获取,  
    // 或者可以静态定义。  
    return self.myArray.count  
}
```

cellForRowIndexPath: 该方法用于创建和配置UITableView的单元格。  
应返回UITableViewCell或其自定义子类。

注意： 使用dequeueReusableCellWithIdentifier:forIndexPath:要求该标识符对应的类或nib已通过UITableView的registerClass:forCellReuseIdentifier:或registerNib:forCellReuseIdentifier:方法注册。通常，这会在UIViewController的viewDidLoad方法中完成。

### Objective-C

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {  
    MyCustomCell *cell = [tableView dequeueReusableCellWithIdentifier:@"MyCustomCell"  
                           forIndexPath:indexPath];  
  
    // 所有额外的自定义操作写在这里  
    cell.titleLabel.text = [NSString stringWithFormat:@"标题 行 %lu", indexPath.row];  
  
    return cell;  
}
```

### Swift 3

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->  
    UITableViewCell {  
    let cell = tableView.dequeueReusableCell(withIdentifier: "MyCustomCell", for: indexPath)  
  
    // 所有额外的自定义操作写在这里  
    cell.titleLabel.text = String(format: "标题 行 %lu", indexPath.row)  
  
    return cell  
}
```

### 可选方法

titleForHeaderInSection: 定义表视图中每个分区头部的标题字符串。该方法仅允许更改标题，进一步的自定义可以通过定义头部视图来实现。

```
// Return the number of rows for the table view. Usually populated from an array,  
// or can be statically defined.  
return self.myArray.count;  
}
```

### Swift 3

```
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {  
    // Return the number of rows for the table view. Usually populated from an array,  
    // or can be statically defined.  
    return self.myArray.count  
}
```

cellForRowIndexPath: This method is where the UITableView's cells are created and configured.  
Should return either a UITableViewCell or a custom subclass.

**Note:** Using dequeueReusableCellWithIdentifier:forIndexPath: requires that the class or nib has been registered for that identifier using the UITableView's registerClass:forCellReuseIdentifier: or registerNib:forCellReuseIdentifier: methods. Usually, this will be done in the UIViewController's viewDidLoad method.

### Objective-C

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {  
    MyCustomCell *cell = [tableView dequeueReusableCellWithIdentifier:@"MyCustomCell"  
                           forIndexPath:indexPath];  
  
    // All additional customization goes here  
    cell.titleLabel.text = [NSString stringWithFormat:@"Title Row %lu", indexPath.row];  
  
    return cell;  
}
```

### Swift 3

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->  
    UITableViewCell {  
    let cell = tableView.dequeueReusableCell(withIdentifier: "MyCustomCell", for: indexPath)  
  
    // All additional customization goes here  
    cell.titleLabel.text = String(format: "Title Row %lu", indexPath.row)  
  
    return cell  
}
```

### Optional Methods

titleForHeaderInSection: Defines a string as the title for each section header in the table view. This method only allows for changing the title, further customization can be done by defining the view for the header.

### Objective-C

```
- (NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section {
```

### Objective-C

```
- (NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section {
```

```

switch(section) {
    case 0:
        return @"标题 1";
        break;

    case 1:
        return @"标题 2";
        break;

    默认:
        return nil;
        break;
}

```

### Swift 3

```

func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {
    switch section {
        case 0:
            return "标题 1"
        case 1:
            return "标题 2"
        default:
            return nil
    }
}

```

titleForFooterInSection: 定义表视图中每个分区页脚的标题字符串。

### Objective-C

```

- (NSString *)tableView:(UITableView *)tableView titleForFooterInSection:(NSInteger)section {
    return @"页脚文本";
}

```

### Swift 3

```

func tableView(_ tableView: UITableView, titleForFooterInSection section: Int) -> String? {
    return "页脚文本"
}

```

canEditRowAtIndexPath: 用于确定是否应为指定行显示编辑界面。  
如果指定的行可以被删除或添加，则应返回YES。

### Objective-C

```

- (BOOL)tableView:(UITableView *)tableView canEditRowAtIndexPath:(NSIndexPath *)indexPath {
    return YES;
}

```

### Swift 3

```

func tableView(_ tableView: UITableView, canEditRowAtIndexPath indexPath: NSIndexPath) -> Bool {
    return true
}

```

```

switch(section) {
    case 0:
        return @"Title 1";
        break;

    case 1:
        return @"Title 2";
        break;

    default:
        return nil;
        break;
}

```

### Swift 3

```

func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {
    switch section {
        case 0:
            return "Title 1"
        case 1:
            return "Title 2"
        default:
            return nil
    }
}

```

titleForFooterInSection: Defines a string as the title for each section header in the table view.

### Objective-C

```

- (NSString *)tableView:(UITableView *)tableView titleForFooterInSection:(NSInteger)section {
    return @"Footer text";
}

```

### Swift 3

```

func tableView(_ tableView: UITableView, titleForFooterInSection section: Int) -> String? {
    return "Footer text"
}

```

canEditRowAtIndexPath: Used to determine if the editing UI should be displayed for the specified row.  
Should return YES if the specified row can be deleted or added.

### Objective-C

```

- (BOOL)tableView:(UITableView *)tableView canEditRowAtIndexPath:(NSIndexPath *)indexPath {
    return YES;
}

```

### Swift 3

```

func tableView(_ tableView: UITableView, canEditRowAtIndexPath indexPath: NSIndexPath) -> Bool {
    return true
}

```

```
}
```

`commitEditingStyle:forRowAtIndexPath` 应执行处理指定行添加或删除所需的操作。例如，从`UITableView`中带动画地移除单元格，并从表的数据模型中移除相关对象。

## Objective-C

```
- (void)tableView:(UITableView *)tableView
commitEditingStyle:(UITableViewCellEditingStyle)editingStyle forRowAtIndexPath:(NSIndexPath *)indexPath {
    switch (editingStyle) {
        case UITableViewCellEditingStyleInsert:
            // 在此处向后备数据模型插入新数据
            [self insertNewDataIntoDataModel];
            [tableView insertRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationAutomatic];
            break;
        case UITableViewCellEditingStyleDelete:
            // self 从数据模型中移除数据，索引为 indexPath.row
            [tableView deleteRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationAutomatic];
            break;
        default:
            // 如果 editingStyle 既不是插入也不是删除，则不执行任何操作
            break;
    }
}
```

## Swift 3

```
func tableView(_ tableView: UITableView, commitEditingStyle: UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {
    switch editingStyle {
        case .Insert:
            self.向数据模型中插入新数据()
            tableView.insertRow(at: indexPath, withAnimation: .Automatic)
        case .Delete:
            self.从数据模型中移除数据，索引为 indexPath.row
            tableView.deleteRow(at: indexPath, withAnimation: .Automatic)
        default:
            // 如果 editingStyle 既不是插入也不是删除，则不执行任何操作
    }
}
```

`editActions:forRowAt` 允许在`UITableView`中为某行的编辑模式添加额外的操作或按钮。例如，如果你想在用户滑动编辑该行时显示两个按钮，一个编辑按钮和一个删除按钮，那么你可以使用此方法。

## Swift 3

```
override func tableView(_ tableView: UITableView, editActionsForRowAt indexPath: IndexPath) -> [UITableViewRowAction]? {
    // 在处理器中，你将获得该操作以及正在编辑的行的 indexPath
}
```

```
}
```

`commitEditingStyle:forRowAtIndexPath` Should perform the work required to handle the addition or removal of the specified row. For example, remove the cell from the `UITableView` with animation, and remove the associated object from the table's data model.

## Objective-C

```
- (void)tableView:(UITableView *)tableView
commitEditingStyle:(UITableViewCellEditingStyle)editingStyle forRowAtIndexPath:(NSIndexPath *)indexPath {
    switch (editingStyle) {
        case UITableViewCellEditingStyleInsert:
            // Insert new data into the backing data model here
            [self insertNewDataIntoDataModel];
            [tableView insertRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationAutomatic];
            break;
        case UITableViewCellEditingStyleDelete:
            [self removeDataFromDataModelAtIndex:indexPath.row];
            [tableView deleteRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationAutomatic];
            break;
        default:
            // Nothing to perform if the editingStyle was neither Insert or Delete
            break;
    }
}
```

## Swift 3

```
func tableView(_ tableView: UITableView, commitEditingStyle: UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {
    switch editingStyle {
        case .Insert:
            self.insertNewDataIntoDataModel()
            tableView.insertRow(at: indexPath, withAnimation: .Automatic)
        case .Delete:
            self.removeDataFromDataModel(atIndex: indexPath.row)
            tableView.deleteRow(at: indexPath, withAnimation: .Automatic)
        default:
            // Nothing to perform if the editingStyle was neither Insert or Delete
    }
}
```

`editActions:forRowAt` Allows ability to add aditional actions or buttons to the edit mode of a row inside a `UITableView`. For example if you wanted two buttons, an edit and delete button when user swipes to edit the row, then you would use this method.

## Swift 3

```
override func tableView(_ tableView: UITableView, editActionsForRowAt indexPath: IndexPath) -> [UITableViewRowAction]? {
    // In the handler you will get passed the action as well as the indexPath for
```

```

// 正在编辑的行
let editAction = UITableViewRowAction(style: .normal, title: "编辑", handler: { [unowned self]
action, indexPath in
    // 当点击编辑时执行某些操作
})

// 更改编辑操作的颜色
editAction.backgroundColor = UIColor.blue

let deleteAction = UITableViewRowAction(style: .destructive, title: "删除", handler: {
[unowned self] action, indexPath in
    // 处理删除事件
})
}

return [deleteAction, editAction]
}

```

## UITableViewDelegate

UITableViewDelegate中的所有方法都是可选的，但实现这些方法的代理将为UITableView启用额外功能。

numberOfSectionsInTableView: 默认返回1，但通过返回不同的节数可以启用多节支持。

## Objective-C

```

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return self.numSections;
}

```

## Swift 3

```

func numberOfSectionsInTableView(_ tableView: UITableView) -> Int {
    return self.numSections
}

```

viewForHeaderInSection 允许配置自定义视图作为节的头部视图。

## Objective-C

```

- (UIView *)tableView:(UITableView *)tableView viewForHeaderInSection:(NSInteger)section {
    UIView *view = [[UIView alloc] initWithFrame:CGRectMake(0, 0, CGRectGetWidth(tableView.frame),
22)];
    view.backgroundColor = [UIColor groupTableViewBackgroundColor];

    UILabel *label = [[UILabel alloc] init];
    label.font = [UIFont systemFontOfSize:12];
    label.textColor = [UIColor darkGrayColor];

    switch (section) {
        case 1:
            label.text = @"标题";
    }
}

```

```

// the row that is being edited
let editAction = UITableViewRowAction(style: .normal, title: "Edit", handler: { [unowned self]
action, indexPath in
    // Do something when edit is tapped
})

// Change the color of the edit action
editAction.backgroundColor = UIColor.blue

let deleteAction = UITableViewRowAction(style: .destructive, title: "Delete", handler: {
[unowned self] action, indexPath in
    // Handel the delete event
})

return [deleteAction, editAction]
}

```

## UITableViewDelegate

All methods in UITableViewDelegate are optional, but a delegate that implements them will enable extra features for the UITableView.

numberOfSectionsInTableView: By default this returns 1, but multiple section support is enabled by returning a different number of sections.

## Objective-C

```

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return self.numSections;
}

```

## Swift 3

```

func numberOfSectionsInTableView(_ tableView: UITableView) -> Int {
    return self.numSections
}

```

viewForHeaderInSection Allows the configuration of a custom view as the header for the section.

## Objective-C

```

- (UIView *)tableView:(UITableView *)tableView viewForHeaderInSection:(NSInteger)section {
    UIView *view = [[UIView alloc] initWithFrame:CGRectMake(0, 0, CGRectGetWidth(tableView.frame),
22)];
    view.backgroundColor = [UIColor groupTableViewBackgroundColor];

    UILabel *label = [[UILabel alloc] init];
    label.font = [UIFont systemFontOfSize:12];
    label.textColor = [UIColor darkGrayColor];

    switch (section) {
        case 1:
            label.text = @"Title";
    }
}

```

```

label.frame = labelFrame;

UIButton *more = [[UIButton alloc] initWithFrame:btnFrame];
[more setTitle:@"查看更多" forState:UIControlStateNormal];
[more.titleLabel setFont:[UIFont systemFontOfSize:12]];
[view addSubview:more];
} break;

default:
label.frame = CGRectMake(0, 0, 0, 0);
break;
}

[view addSubview:label];
return view;
}

```

### Swift 3

```

func tableView(_ tableView: UITableView, viewForHeaderInSection section: Int) -> UIView? {
    let view = UIView(frame: CGRect(x: 0, y: 0, width: tableView.frame.size.width, height: 22))
    view.backgroundColor = UIColor.groupTableViewBackgroundColor()

    let label = UILabel()
    label.font = UIFont.systemFontOfSize(12)
    label.textColor = UIColor.darkGrayColor()

    switch section {
        case 1:
            label.text = "标题"
            label.frame = labelFrame

            let more = UIButton(frame: btnFrame)
            more.setTitle("查看更多", forState:.Normal)
            view.addSubview(more)

        默认:
            label.frame = CGRect.zero
    }

    view.addSubview(label)
    return view;
}

```

heightForRowAtIndexPath: 定义表视图中每个单元格的高度。

### Objective-C

```

- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath {
    return 44;
}

```

### Swift 3

```

func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
    return 44
}

```

```

label.frame = labelFrame;

```

```

UIButton *more = [[UIButton alloc] initWithFrame:btnFrame];
[more setTitle:@"See more" forState:UIControlStateNormal];
[more.titleLabel setFont:[UIFont systemFontOfSize:12]];
[view addSubview:more];
} break;

default:
label.frame = CGRectMake(0, 0, 0, 0);
break;
}

[view addSubview:label];
return view;
}

```

### Swift 3

```

func tableView(_ tableView: UITableView, viewForHeaderInSection section: Int) -> UIView? {
    let view = UIView(frame: CGRect(x: 0, y: 0, width: tableView.frame.size.width, height: 22))
    view.backgroundColor = UIColor.groupTableViewBackgroundColor()

    let label = UILabel()
    label.font = UIFont.systemFontOfSize(12)
    label.textColor = UIColor.darkGrayColor()

    switch section {
        case 1:
            label.text = "Title"
            label.frame = labelFrame

            let more = UIButton(frame: btnFrame)
            more.setTitle("See more", forState:.Normal)
            view.addSubview(more)

        default:
            label.frame = CGRect.zero
    }

    view.addSubview(label)
    return view;
}

```

heightForRowAtIndexPath: Define the height of each cell in the table view.

### Objective-C

```

- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath {
    return 44;
}

```

### Swift 3

```

func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
    return 44
}

```

}

heightForHeaderInSection: 和 heightForFooterInSection 定义表视图中每个分区的表头和表尾的高度

#### Objective-C

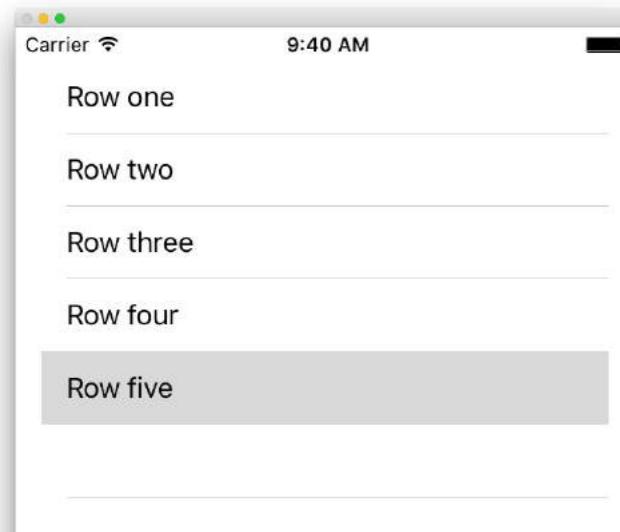
```
- (CGFloat)tableView:(UITableView *)tableView heightForHeaderInSection:(NSInteger)section {
    return 33;
}
```

#### Swift 3

```
func tableView(_ tableView: UITableView, heightForHeaderInSection section: Int) -> CGFloat {
    return 33
}
```

## 第14.5节：创建 UITableView

表视图是一列可选择的行。每一行的数据都来自数据源。此示例创建了一个简单的表视图，其中每行显示一行文本。



#### 向你的Storyboard中添加 UITableView

虽然有多种方法可以创建UITableView，但最简单的方法之一是将其添加到Storyboard中。  
打开你的Storyboard，拖动一个UITableView到你的UIViewController上。确保使用自动布局正确对齐表格（固定四边）。

#### 用数据填充你的表格

为了动态显示内容（即从数组、Core Data模型、网络服务器等数据源加载内容）到你的表视图中，你需要设置数据源。

#### 创建一个简单的数据源

数据源如上所述，可以是任何包含数据的东西。格式和内容完全由你决定。  
唯一的要求是你必须能够在以后读取它，以便在需要时用数据填充表格的每一行。

}

heightForHeaderInSection: and heightForFooterInSection Define the height for the header and footer of each section in the table view

#### Objective-C

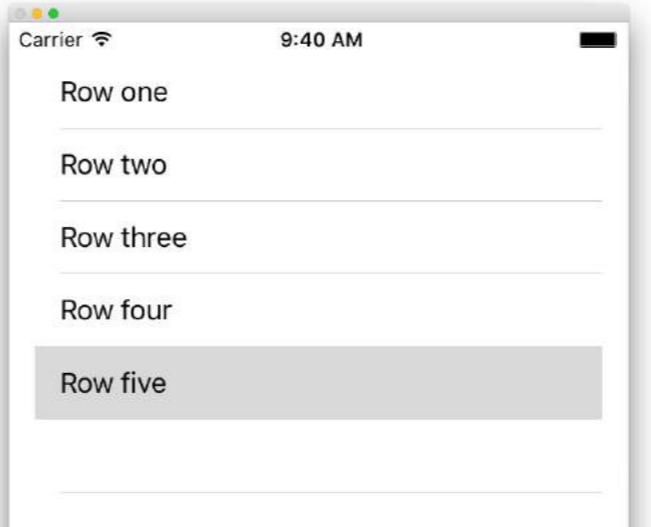
```
- (CGFloat)tableView:(UITableView *)tableView heightForHeaderInSection:(NSInteger)section {
    return 33;
}
```

#### Swift 3

```
func tableView(_ tableView: UITableView, heightForHeaderInSection section: Int) -> CGFloat {
    return 33
}
```

## Section 14.5: Creating a UITableView

A Table View is a list of rows that can be selected. Each row is populated from a data source. This example creates a simple table view in which each row is a single line of text.



#### Add a UITableView to your Storyboard

Although there are a number of ways to create a UITableView, one of the easiest is to add one to a Storyboard. Open your Storyboard and drag a UITableView onto your UIViewController. Make sure to use Auto Layout to correctly align the table (pin all four sides).

#### Populating Your Table with Data

In order to display content dynamically (i.e. load it from a data source like an array, a Core Data model, a networked server, etc.) in your table view you need to setup the data source.

#### Creating a simple data source

A data source could, as stated above, be anything with data. Its entirely up to you how to format it and what's in it. The only requirement is that you must be able to read it later so that you can populate each row of your table with data when needed.

在这个例子中，我们将设置一个包含一些字符串（文本）的数组作为数据源：

## Swift

```
let mydataArray: [String] = ["第一行", "第二行", "第三行", "第四行", "第五行"]
```

## Objective-C

```
// 你需要将此变量定义为全局变量（如@property），以便在需要时访问它。
```

```
NSArray *mydataArray = @[@"第一行", @"第二行", @"第三行", @"第四行", @"第五行"];
```

## 在你的视图控制器中设置数据源

确保你的视图控制器遵循UITableViewDataSource协议。

## Swift

```
class ViewController: UIViewController, UITableViewDataSource {
```

## Objective-C

```
@interface ViewController : UIViewController <UITableViewDataSource>
```

一旦你的视图控制器声明它将遵循UITableViewDataSource（这正是我们上面所做的），你必须在视图控制器类中实现至少以下方法：

- tableView:numberOfRowsInSection，询问你的表视图应该有多少行。

```
// Swift
```

```
func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    return self.mydataArray.count
}
```

- tableView:cellForRowIndexPath，要求你在tableView:numberOfRowsInSection中指定的每一行创建并返回一个单元格。所以，如果说需要10行，这个方法将被调用十次针对每一行，你需要为每一行创建一个单元格。

```
// Swift
```

```
func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) ->
    UITableViewCell {
    // 在这里创建一个新单元格。cellReuseIdentifier需要与Storyboard中单元格的重用标识符匹配
    let cell: UITableViewCell =
    tableView.dequeueReusableCellWithIdentifier(cellReuseIdentifier) as UITableViewCell!
    // 将单元格上的标签设置为数据数组中的文本
    cell.textLabel?.text = self.mydataArray[indexPath.row]
    return cell
}
```

In this example, we'll just set an array with some strings (text) as our data source:

## Swift

```
let mydataArray: [String] = ["Row one", "Row two", "Row three", "Row four", "Row five"]
```

## Objective-C

```
// You'll need to define this variable as a global variable (like an @property) so that you can access it later when needed.
```

```
NSArray *mydataArray = @[@"Row one", @"Row two", @"Row three", @"Row four", @"Row five"];
```

## Setting up your data source in your View Controller

Make sure your view controller conforms to the [UITableViewDataSource](#) protocol.

## Swift

```
class ViewController: UIViewController, UITableViewDataSource {
```

## Objective-C

```
@interface ViewController : UIViewController <UITableViewDataSource>
```

As soon as your view controller has declared it will **conform** to the [UITableViewDataSource](#) (that's what we've just done above), you are *required* to implement at least the following methods in your view controller class:

- tableView:numberOfRowsInSection, this asks you how many rows your table view should have.

```
// Swift
```

```
func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    return self.mydataArray.count
}
```

- tableView:cellForRowIndexPath, requests that you create and return a cell for each row you specified in tableView:numberOfRowsInSection. So, if you said you needed 10 rows, this method will be called ten times for each row, and you need to create a cell for each of those rows.

```
// Swift
```

```
func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) ->
    UITableViewCell {
    // Create a new cell here. The cellReuseIdentifier needs to match the reuse identifier from the cell in your Storyboard
    let cell: UITableViewCell =
    tableView.dequeueReusableCellWithIdentifier(cellReuseIdentifier) as UITableViewCell!
    // Set the label on your cell to the text from your data array
    cell.textLabel?.text = self.mydataArray[indexPath.row]
    return cell
}
```

**警告:** 你不能在cellForRowAtIndexPath:中返回nil。这会导致你的应用程序崩溃，并且你将在控制台看到以下错误：

未捕获的异常 'NSInternalInconsistencyException'，原因：'UITableView 数据源必须从tableView:cellForRowAtIndexPath:返回一个单元格'

## 将表视图的数据源连接到你的视图控制器

你可以通过代码将表的数据源属性设置为视图控制器的self来实现。或者，你可以在故事板中选择你的表视图，打开属性检查器，选择“Outlets”面板，然后从dataSource拖动到你的视图控制器（注意：确保你连接的是UIViewController，不是UIView或视图控制器中的其他对象）。

## 处理行选择

当用户点击表视图中的一行时，通常你会想做一些响应。在许多应用中，点击一行后，会显示该项的更多信息。想想消息应用：当你点击显示某个联系人的那一行时，屏幕上会显示与该人的对话内容。

为了实现这一点，你必须遵守UITableViewDelegate协议。这样做类似于遵守数据源协议。不过这次，你只需在UITableViewDataSource后面添加它，并用逗号分隔。代码应如下所示：

## Swift

```
class ViewController: UIViewController, UITableViewDataSource, UITableViewDelegate {
```

## Objective-C

```
@interface ViewController : UIViewController <UITableViewDataSource, UITableViewDelegate>
```

表视图的代理没有必须实现的方法。然而，要处理行选择，你需要使用以下方法：

- tableView:didSelectRowAtIndexPath，当每一行被点击时调用，允许你做出响应。以我们的示例为例，我们只会在Xcode日志中打印一条确认语句。

```
// Swift

func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
    print("你点击了第 \(indexPath.row) 个单元格。")
}

// Objective-C

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
    NSLog(@"你点击了第 %ld 个单元格。", (long)indexPath.row);
}
```

## 最终方案

下面是仅包含代码的完整设置，没有解释。

## Swift

**WARNING:** You may **NOT** return nil for any cells in cellForRowAtIndexPath:. This will cause your app to crash, and you will see the following error in the console:

Uncaught exception 'NSInternalInconsistencyException', reason: 'UITableView dataSource must **return** a cell from tableView:cellForRowAtIndexPath:'

## Connecting the table view's data source to your view controller

You can either do this via code by setting your table's dataSource property to **self** on your view controller. Or you may select your table view in your storyboard, open the Attributes Inspector, select the "Outlets" panel, and drag from dataSource to your view controller (**NOTE:** make sure you connect to the UIViewController, **not** a UIView or another object *in* your UIViewController).

## Handling row selections

When a user taps on a row in your table view, generally, you'll want to do something - to respond. In many apps, when you tap on a row, more information about that item you tapped upon is displayed. Think of the Messages app: when you tap on the row showing one of your contacts, the conversation with that person is then displayed on screen.

In order to do that, you must conform to the **UITableViewDelegate** protocol. Doing so is similar to conforming to the data source protocol. This time however, you'll just add it next to **UITableViewDataSource** and separate it with a comma. So it should look like this:

## Swift

```
class ViewController: UIViewController, UITableViewDataSource, UITableViewDelegate {
```

## Objective-C

```
@interface ViewController : UIViewController <UITableViewDataSource, UITableViewDelegate>
```

There are no required methods to implement for the table view's delegate. However, to handle row selections you'll need to use the following method:

- **tableView:didSelectRowAtIndexPath**，这是当一行被点击时调用的，允许你做出响应。对于我们的示例，我们只会在Xcode日志中打印一条确认语句。

```
// Swift

func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
    print("You tapped cell number \(indexPath.row).")
}

// Objective-C

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
    NSLog(@"You tapped cell number %ld.", (long)indexPath.row);
}
```

## The Final Solution

See below for the full setup with just code, no explanation.

## Swift

```

import UIKit
class ViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {

    // 数据模型：这些字符串将作为表视图单元格的数据
    let mydataArray: [String] = ["第一行", "第二行", "第三行", "第四行", "第五行"]

    // 单元格重用标识（滚出视图的单元格可以被重用）
    let cellReuseIdentifier = "cell"

    // 别忘了从故事板连接这个
    @IBOutlet var myTableView: UITableView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // 注册表视图单元格类及其重用标识符
        myTableView.registerClass(UITableViewCell.self, forCellReuseIdentifier: cellReuseIdentifier)

        // 该视图控制器本身将提供表视图的代理方法和行数据。
        myTableView.delegate = self
        myTableView.dataSource = self
    }

    // 表视图中的行数
    func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return self.mydataArray.count
    }

    // 为每个表视图行创建一个单元格
    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
        // 如有必要，创建一个新单元格或重用旧单元格
        let cell:UITableViewCell = tableView.dequeueReusableCellWithIdentifier(cellReuseIdentifier) as UITableViewCell!

        // 从数据模型设置文本
        cell.textLabel?.text = self.mydataArray[indexPath.row]

        return cell
    }

    // 表视图单元格被点击时运行的方法
    func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
        print("你点击了第 \(indexPath.row) 个单元格。")
    }
}

```

## Objective-C

### ViewController.h

```

#import <UIKit/UIKit.h>

@interface ViewController: UIViewController <UITableViewDelegate, UITableViewDataSource> {
    IBOutlet UITableView *myTableView;
    NSArray *mydataArray;
}

```

```

import UIKit
class ViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {

    // Data model: These strings will be the data for the table view cells
    let mydataArray: [String] = ["Row one", "Row two", "Row three", "Row four", "Row five"]

    // cell reuse id (cells that scroll out of view can be reused)
    let cellReuseIdentifier = "cell"

    // don't forget to hook this up from the storyboard
    @IBOutlet var myTableView: UITableView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Register the table view cell class and its reuse id
        myTableView.registerClass(UITableViewCell.self, forCellReuseIdentifier: cellReuseIdentifier)

        // This view controller itself will provide the delegate methods and row data for the table view.
        myTableView.delegate = self
        myTableView.dataSource = self
    }

    // number of rows in table view
    func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return self.mydataArray.count
    }

    // create a cell for each table view row
    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
        // create a new cell if needed or reuse an old one
        let cell:UITableViewCell = tableView.dequeueReusableCellWithIdentifier(cellReuseIdentifier) as UITableViewCell!

        // set the text from the data model
        cell.textLabel?.text = self.mydataArray[indexPath.row]

        return cell
    }

    // method to run when table view cell is tapped
    func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
        print("You tapped cell number \(indexPath.row).")
    }
}

```

## Objective-C

### ViewController.h

```

#import <UIKit/UIKit.h>

@interface ViewController: UIViewController <UITableViewDelegate, UITableViewDataSource> {
    IBOutlet UITableView *myTableView;
    NSArray *mydataArray;
}

```

```
@end
```

## ViewController.m

```
#import "ViewController.h"

// cell reuse id (cells that scroll out of view can be reused)
NSString * _Nonnull cellReuseIdentifier = @"cell";

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    // 数据模型：这些字符串将作为表视图单元格的数据
    mydataArray = @[@"第一行", @"第二行", @"第三行", @"第四行", @"第五行"];

    // 注册表视图单元格类及其重用标识符
    [myTableView registerClass:[UITableViewCell class] forCellReuseIdentifier:cellReuseIdentifier];

    // 该视图控制器本身将提供表视图的代理方法和行数据。
    myTableView.delegate = self;
    myTableView.dataSource = self;
}

// 表视图中的行数
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
    return mydataArray.count;
}

// 为每个表视图行创建一个单元格
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    // 如果需要，创建一个新单元格或重用旧单元格
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:cellReuseIdentifier];

    // 从数据模型设置文本
    cell.textLabel.text = mydataArray[indexPath.row];

    return cell;
}

// 表格视图单元格被点击时运行的方法
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath{
    NSLog(@"你点击了第 %ld 个单元格。", (long)indexPath.row);
}

@end
```

## 第14.6节：滑动删除行

我总觉得有一个非常简单、独立的示例是很好的，这样在学习新任务时不会有任何假设。这个答案就是关于删除UITableView行的。项目的表现如下：

```
@end
```

## ViewController.m

```
#import "ViewController.h"

// cell reuse id (cells that scroll out of view can be reused)
NSString * _Nonnull cellReuseIdentifier = @"cell";

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    // Data model: These strings will be the data for the table view cells
    mydataArray = @[@"Row one", @"Row two", @"Row three", @"Row four", @"Row five"];

    // Register the table view cell class and its reuse id
    [myTableView registerClass:[UITableViewCell class] forCellReuseIdentifier:cellReuseIdentifier];

    // This view controller itself will provide the delegate methods and row data for the table view.
    myTableView.delegate = self;
    myTableView.dataSource = self;
}

// number of rows in table view
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
    return mydataArray.count;
}

// create a cell for each table view row
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    // create a new cell if needed or reuse an old one
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:cellReuseIdentifier];

    // set the text from the data model
    cell.textLabel.text = mydataArray[indexPath.row];

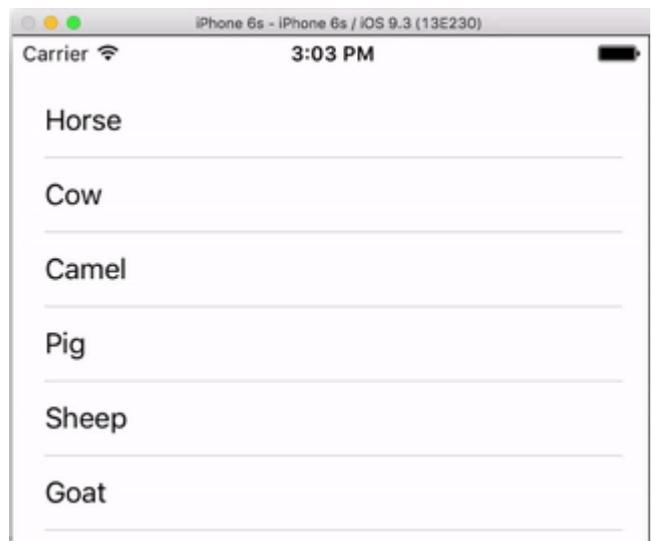
    return cell;
}

// method to run when table view cell is tapped
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath{
    NSLog(@"You tapped cell number %ld.", (long)indexPath.row);
}

@end
```

## Section 14.6: Swipe to Delete Rows

I always think it is nice to have a very simple, self-contained example so that nothing is assumed when I am learning a new task. This answer is that for deleting UITableView rows. The project performs like this:



该项目基于Swift的UITableView示例。

#### 添加代码

创建一个新项目，并将 ViewController.swift 代码替换为以下内容。

```
import UIKit
class ViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {

    // 这些字符串将作为表视图单元格的数据
    var animals: [String] = ["马", "牛", "骆驼", "猪", "羊", "山羊"]

    let cellReuseIdentifier = "cell"

    @IBOutlet var tableView: UITableView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // 如果你愿意，可以在界面构建器中完成以下三件事
        // 而不是在代码中完成。
        self.tableView.registerClass(UITableViewCell.self, forCellReuseIdentifier: cellReuseIdentifier)
        tableView.delegate = self
        tableView.dataSource = self
    }

    // 表视图中的行数
    func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return self.animals.count
    }

    // 为每个表视图行创建一个单元格
    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
        let cell:UITableViewCell =
        self.tableView.dequeueReusableCellWithIdentifier(cellReuseIdentifier) as UITableViewCell!
        cell.textLabel?.text = self.animals[indexPath.row]
        return cell
    }
}
```



This project is based on the [UITableView example for Swift](#).

#### Add the Code

Create a new project and replace the ViewController.swift code with the following.

```
import UIKit
class ViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {

    // These strings will be the data for the table view cells
    var animals: [String] = ["Horse", "Cow", "Camel", "Pig", "Sheep", "Goat"]

    let cellReuseIdentifier = "cell"

    @IBOutlet var tableView: UITableView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // It is possible to do the following three things in the Interface Builder
        // rather than in code if you prefer.
        self.tableView.registerClass(UITableViewCell.self, forCellReuseIdentifier: cellReuseIdentifier)
        tableView.delegate = self
        tableView.dataSource = self
    }

    // number of rows in table view
    func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return self.animals.count
    }

    // create a cell for each table view row
    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
        let cell:UITableViewCell =
        self.tableView.dequeueReusableCellWithIdentifier(cellReuseIdentifier) as UITableViewCell!
        cell.textLabel?.text = self.animals[indexPath.row]
        return cell
    }
}
```

```

// 表视图单元格被点击时运行的方法
func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
    print("你点击了第 \(indexPath.row) 个单元格。")
}

// 该方法处理行删除
func tableView(tableView: UITableView, commitEditingStyle editingStyle:
UITableViewCellEditingStyle, forRowAtIndexPath indexPath: NSIndexPath) {

    if editingStyle == .Delete {

        // 从数据模型中移除该项
        animals.removeAtIndex(indexPath.row)

        // 删除表视图中的行
        tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation: .Fade)
    } else if editingStyle == .Insert {
        // 在我们的示例中未使用，但如果你要添加新行，这里是实现的地方。
    }
}
}

```

上面代码中实现行删除的关键方法是最后一个。这里再次强调：

```

func tableView(tableView: UITableView, commitEditingStyle editingStyle:
UITableViewCellEditingStyle, forRowAtIndexPath indexPath: NSIndexPath) {

    if editingStyle == .Delete {

        // 从数据模型中移除该项
        animals.removeAtIndex(indexPath.row)

        // 删除表视图中的行
        tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation: .Fade)
    }
}

```

#### Storyboard (故事板)

在故事板中的视图控制器添加一个UITableView。使用自动布局将表视图的四边固定到视图控制器的边缘。从故事板中的表视图按住Control键拖动到代码中的@IBOutlet var tableView: UITableView!这一行。

#### 完成

就是这样。现在你应该可以运行你的应用，通过左滑并点击“删除”来删除行了。

#### 注意事项

- 此功能仅在iOS 8及以上版本可用。更多详情请参见[this answer](#)。
- 如果你需要更改显示按钮的数量或按钮文本，请参见[this answer](#)了解更多详情。

#### 进一步阅读

- [如何制作带操作的可滑动表格视图单元格–无需为滚动视图而抓狂](#)
- [苹果文档](#)

```

// method to run when table view cell is tapped
func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
    print("You tapped cell number \(indexPath.row).")
}

// this method handles row deletion
func tableView(tableView: UITableView, commitEditingStyle editingStyle:
UITableViewCellEditingStyle, forRowAtIndexPath indexPath: NSIndexPath) {

    if editingStyle == .Delete {

        // remove the item from the data model
        animals.removeAtIndex(indexPath.row)

        // delete the table view row
        tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation: .Fade)
    } else if editingStyle == .Insert {
        // Not used in our example, but if you were adding a new row, this is where you would
        do it.
    }
}
}

```

The single key method in the code above that enables row deletion is the last one. Here it is again for emphasis:

```

func tableView(tableView: UITableView, commitEditingStyle editingStyle:
UITableViewCellEditingStyle, forRowAtIndexPath indexPath: NSIndexPath) {

    if editingStyle == .Delete {

        // remove the item from the data model
        animals.removeAtIndex(indexPath.row)

        // delete the table view row
        tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation: .Fade)
    }
}

```

#### Storyboard

Add a [UITableView](#) to the View Controller in the storyboard. Use auto layout to pin the four sides of the table view to the edges of the View Controller. Control drag from the table view in the storyboard to the `@IBOutlet var tableView: UITableView!` line in the code.

#### Finished

That's all. You should be able to run your app now and delete rows by swiping left and tapping "Delete".

#### Notes

- This is only available from iOS 8. See [this answer](#) for more details.
- If you need to change the number of buttons displayed or the button text then see [this answer](#) for more details.

#### Further reading

- [How To Make A Swipeable Table View Cell With Actions – Without Going Nuts With Scroll Views](#)
- [Apple Documentation](#)

## 第14.7节：展开和折叠UITableView单元格

在你的Storyboard中，在UIViewController上添加一个UITableView对象，并让它覆盖整个视图。设置UITableViewDataSource和UITableViewDelegate连接。

### Objective-C

在你的.h文件中

```
NSMutableArray *arrayForBool;  
NSMutableArray *sectionTitleArray;
```

在你的.m文件中

```
- (void)viewDidLoad {  
    [super viewDidLoad];  
  
    arrayForBool = [[NSMutableArray alloc] init];  
    sectionTitleArray = @[@"Sam",@"Sanju",@"John",@"Staffy"];  
  
    for (int i=0; i<[sectionTitleArray count]; i++) {  
        [arrayForBool addObject:[NSNumber numberWithBool:NO]];  
    }  
  
    _tableView.dataSource = self;  
    _tableView.delegate = self;  
}  
  
// 声明分区中的行数  
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {  
    if ([[arrayForBool objectAtIndex:section] boolValue]) {  
        return section+2;  
    } else {  
        return 0;  
    }  
}  
  
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {  
  
    static NSString *cellid=@"hello";  
    UITableViewCell *cell=[tableView dequeueReusableCellWithIdentifier:cellid];  
    if (cell==nil) {  
        cell=[[UITableViewCell alloc]initWithStyle:UITableViewCellStyleSubtitle  
reuseIdentifier:cellid];  
    }  
    BOOL manyCells = [[arrayForBool objectAtIndex:indexPath.section] boolValue];  
  
    /** 如果该分区应该关闭 *****/  
    if(!manyCells){  
        cell.backgroundColor=[UIColor clearColor];  
        cell.textLabel.text=@"";  
    }  
    /** 如果该部分应被打开*****/  
    else{  
        cell.textLabel.text=[NSString stringWithFormat:@"%@",[sectionTitleArray  
objectAtIndex:indexPath.section],indexPath.row+1];  
        cell.backgroundColor=[UIColor whiteColor];  
        cell.selectionStyle=UITableViewCellSelectionStyleNone ;  
    }  
}
```

## Section 14.7: Expanding & Collapsing UITableViewCells

In your Storyboard, add a UITableView object on your UIViewController and let it cover the entire view. Setup the UITableViewDataSource and UITableViewDelegate connections.

### Objective-C

In your .h file

```
NSMutableArray *arrayForBool;  
NSMutableArray *sectionTitleArray;
```

In your .m file

```
- (void)viewDidLoad {  
    [super viewDidLoad];  
  
    arrayForBool = [[NSMutableArray alloc] init];  
    sectionTitleArray = @[@"Sam",@"Sanju",@"John",@"Staffy"];  
  
    for (int i=0; i<[sectionTitleArray count]; i++) {  
        [arrayForBool addObject:[NSNumber numberWithBool:NO]];  
    }  
  
    _tableView.dataSource = self;  
    _tableView.delegate = self;  
}  
  
// Declare number of rows in section  
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {  
    if ([[arrayForBool objectAtIndex:section] boolValue]) {  
        return section+2;  
    } else {  
        return 0;  
    }  
}  
  
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {  
  
    static NSString *cellid=@"hello";  
    UITableViewCell *cell=[tableView dequeueReusableCellWithIdentifier:cellid];  
    if (cell==nil) {  
        cell=[[UITableViewCell alloc]initWithStyle:UITableViewCellStyleSubtitle  
reuseIdentifier:cellid];  
    }  
    BOOL manyCells = [[arrayForBool objectAtIndex:indexPath.section] boolValue];  
  
    /** If the section supposed to be closed*****/  
    if(!manyCells){  
        cell.backgroundColor=[UIColor clearColor];  
        cell.textLabel.text=@"";  
    }  
    /** If the section supposed to be Opened*****/  
    else{  
        cell.textLabel.text=[NSString stringWithFormat:@"%@",[sectionTitleArray  
objectAtIndex:indexPath.section],indexPath.row+1];  
        cell.backgroundColor=[UIColor whiteColor];  
        cell.selectionStyle=UITableViewCellSelectionStyleNone ;  
    }  
}
```

```

cell.textLabel.textColor=[UIColor blackColor];

/* 添加自定义分隔线到单元格 */
UIView* separatorLineView = [[UIView alloc]initWithFrame:CGRectMake(15, 40,
_expandableTableView.frame.size.width-15, 1)];
separatorLineView.backgroundColor = [UIColor blackColor];
[cell.contentView addSubview:separatorLineView];
return cell;
}

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
return [sectionTitleArray count];
}

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{

***** 关闭该部分，一旦数据被选中 *****
[arrayForBool replaceObjectAtIndex:indexPath.section withObject:[NSNumber numberWithBool:NO]];

[_expandableTableView reloadSections:[NSIndexSet indexSetWithIndex:indexPath.section]
withRowAnimation:UITableViewRowAnimationAutomatic];
}

- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
if ([[arrayForBool objectAtIndex:indexPath.section] boolValue]) {
    return 40;
}
return 0;
}

- (UIView *)tableView:(UITableView *)tableView viewForHeaderInSection:(NSInteger)section
{
UIView *sectionView=[[UIView alloc]initWithFrame:CGRectMake(0, 0, 280, 40)];
sectionView.tag=section;
UILabel *viewLabel=[[UILabel alloc]initWithFrame:CGRectMake(10, 0,
_expandableTableView.frame.size.width-10, 40)];
viewLabel.backgroundColor=[UIColor clearColor];
viewLabel.textColor=[UIColor blackColor];
viewLabel.font=[UIFont systemFontOfSize:15];
viewLabel.text=[NSString stringWithFormat:@"List of %@",[sectionTitleArray objectAtIndex:section]];
[sectionView addSubview:viewLabel];
***** 添加带有分区视图的自定义分隔线 *****
UIView* separatorLineView = [[UIView alloc] initWithFrame:CGRectMake(15, 40,
_expandableTableView.frame.size.width-15, 1)];
separatorLineView.backgroundColor = [UIColor blackColor];
[sectionView addSubview:separatorLineView];

***** 给 SectionView 添加 UITapGestureRecognizer *****
UITapGestureRecognizer *headerTapped = [[UITapGestureRecognizer alloc] initWithTarget:self
action:@selector(sectionHeaderTapped:)];
[sectionView addGestureRecognizer:headerTapped];

return sectionView;
}

```

```

cell.textLabel.textColor=[UIColor blackColor];

/* Add a custom Separator with cell*/
UIView* separatorLineView = [[UIView alloc]initWithFrame:CGRectMake(15, 40,
_expandableTableView.frame.size.width-15, 1)];
separatorLineView.backgroundColor = [UIColor blackColor];
[cell.contentView addSubview:separatorLineView];
return cell;
}

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
return [sectionTitleArray count];
}

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{

***** Close the section, once the data is selected *****
[arrayForBool replaceObjectAtIndex:indexPath.section withObject:[NSNumber numberWithBool:NO]];

[_expandableTableView reloadSections:[NSIndexSet indexSetWithIndex:indexPath.section]
withRowAnimation:UITableViewRowAnimationAutomatic];
}

- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
if ([[arrayForBool objectAtIndex:indexPath.section] boolValue]) {
    return 40;
}
return 0;
}

- (UIView *)tableView:(UITableView *)tableView viewForHeaderInSection:(NSInteger)section
{
UIView *sectionView=[[UIView alloc]initWithFrame:CGRectMake(0, 0, 280, 40)];
sectionView.tag=section;
UILabel *viewLabel=[[UILabel alloc]initWithFrame:CGRectMake(10, 0,
_expandableTableView.frame.size.width-10, 40)];
viewLabel.backgroundColor=[UIColor clearColor];
viewLabel.textColor=[UIColor blackColor];
viewLabel.font=[UIFont systemFontOfSize:15];
viewLabel.text=[NSString stringWithFormat:@"List of %@",[sectionTitleArray objectAtIndex:section]];
[sectionView addSubview:viewLabel];
***** Add a custom Separator with Section view *****
UIView* separatorLineView = [[UIView alloc] initWithFrame:CGRectMake(15, 40,
_expandableTableView.frame.size.width-15, 1)];
separatorLineView.backgroundColor = [UIColor blackColor];
[sectionView addSubview:separatorLineView];

***** Add UITapGestureRecognizer to SectionView *****
UITapGestureRecognizer *headerTapped = [[UITapGestureRecognizer alloc] initWithTarget:self
action:@selector(sectionHeaderTapped:)];
[sectionView addGestureRecognizer:headerTapped];

return sectionView;
}

```

```
}
```

```
- (void)sectionHeaderTapped:(UITapGestureRecognizer *)gestureRecognizer{
    NSIndexPath *indexPath = [NSIndexPath indexPathForRow:0 inSection:gestureRecognizer.view.tag];
    if (indexPath.row == 0) {
        BOOL collapsed = [[arrayForBool objectAtIndex:indexPath.section] boolValue];
        for (int i=0; i<[sectionTitleArray count]; i++) {
            if (indexPath.section==i) {
                [arrayForBool replaceObjectAtIndex:i withObject:[NSNumber numberWithBool:!collapsed]];
            }
        }
        [_expandableTableView reloadSections:[NSSet indexSetWithIndex:gestureRecognizer.view.tag]
        withRowAnimation:UITableViewRowAnimationAutomatic];
    }
}
```

```
}
```

```
- (void)sectionHeaderTapped:(UITapGestureRecognizer *)gestureRecognizer{
    NSIndexPath *indexPath = [NSIndexPath indexPathForRow:0 inSection:gestureRecognizer.view.tag];
    if (indexPath.row == 0) {
        BOOL collapsed = [[arrayForBool objectAtIndex:indexPath.section] boolValue];
        for (int i=0; i<[sectionTitleArray count]; i++) {
            if (indexPath.section==i) {
                [arrayForBool replaceObjectAtIndex:i withObject:[NSNumber numberWithBool:!collapsed]];
            }
        }
        [_expandableTableView reloadSections:[NSSet indexSetWithIndex:gestureRecognizer.view.tag]
        withRowAnimation:UITableViewRowAnimationAutomatic];
    }
}
```

# 第15章：UITableViewController

UITableViewController 控制器对象，用于管理表视图。在某些特定场景下，建议使用 UITableViewController，例如当你有大量单元格且其中一些包含 UITextField 时。

## 第15.1节：具有动态属性的 TableView，使用 tableViewCellStyle basic

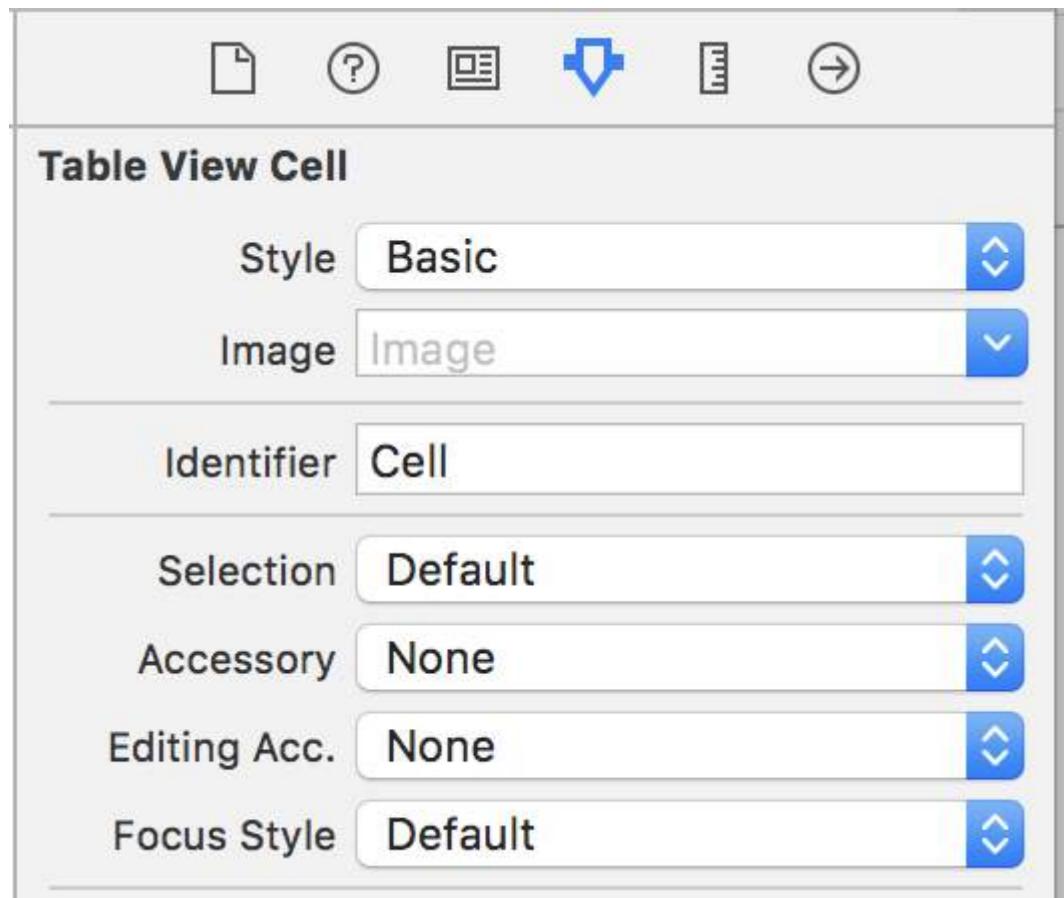
```
override func numberOfSections(in tableView: UITableView) -> Int {  
    // 你需要返回至少1，以显示表视图中的单元格  
    return 1  
}  
  
override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {  
    // 返回表视图中行的数量。  
    return 3  
}  
  
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->  
UITableViewCell {  
  
    let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath)  
    // 标识符字符串应与单元格属性检查器中输入的 identifier 相同（见图片）。  
  
    // 配置单元格...  
    cell.textLabel?.text = "单元格 \(indexPath.row) :" + "Hello"  
    // 单元格有不同的样式：自定义、基础、右侧详情、左侧详情、子标题。  
    // 对于自定义样式，你可以使用自己的对象和约束，对于其他样式，  
    // 都已准备好，只需根据设计选择即可。（见图片了解如何更改样式）  
  
    return cell  
}  
  
override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {  
    // 当你点击一个单元格时，此代理方法将被触发  
}
```

# Chapter 15: UITableViewController

UITableViewController controller object that manages a table view. For some certain scenario it will be recommended to use UITableViewController, for example if you have lot of cells and some have UITextField.

## Section 15.1: TableView with dynamic properties with tableViewCellStyle basic

```
override func numberOfSections(in tableView: UITableView) -> Int {  
    // You need to return minimum one to show the cell inside the tableview  
    return 1  
}  
  
override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {  
    // return the number of rows inside the tableview.  
    return 3  
}  
  
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->  
UITableViewCell {  
  
    let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath)  
    // identifier string should be same as what you have entered in the cell Attribute inspector ->  
    // identifier (see the image).  
  
    // Configure the cell...  
    cell.textLabel?.text = "Cell \(indexPath.row) :" + "Hello"  
    // cell have different style Custom, basic, right detail, left detail, subtitle.  
    // For custom you can use your own objects and constraints, for other styles all  
    // is ready just select according to your design. (see the image for changing the style)  
  
    return cell  
}  
  
override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {  
    // this delegate method will trigger when you click a cell  
}
```



## 第15.2节：带自定义单元格的表视图

对于自定义表视图单元格，您需要一个继承自UITableViewController的类，下面是一个示例类。

```
class TableViewCell: UITableViewCell {
    @IBOutlet weak var lblTitle: UILabel!

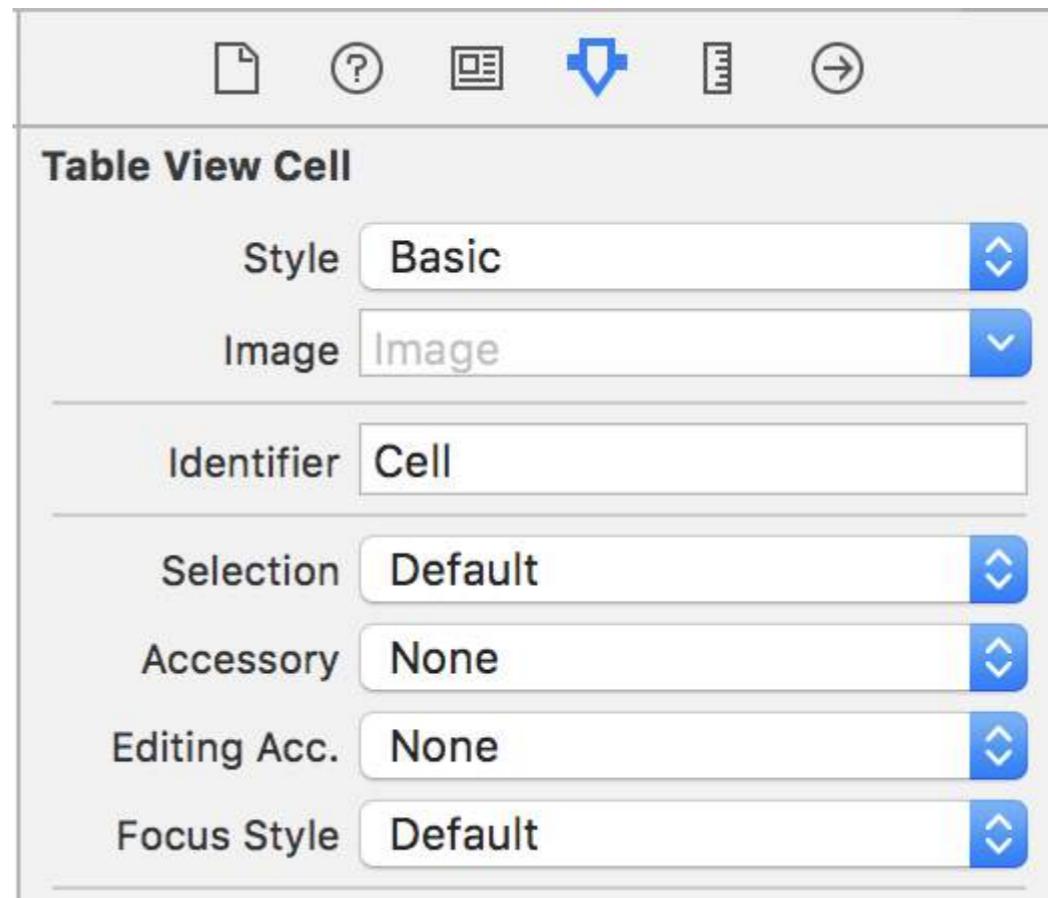
    override func awakeFromNib() {
        super.awakeFromNib()
        // 初始化代码
    }

    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)
        // 配置选中状态下的视图
    }
}
```

您的表视图代理

```
override func numberOfSections(in tableView: UITableView) -> Int {
    // 你需要返回至少1，以显示表视图中的单元格
    return 1
}

override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    // 返回表视图中行的数量。
    return 3
}
```



## Section 15.2: TableView with Custom Cell

For custom tableview cell you need a class that is subclass from `UITableViewCell`, an example class you can see below.

```
class TableViewCell: UITableViewCell {
    @IBOutlet weak var lblTitle: UILabel!

    override func awakeFromNib() {
        super.awakeFromNib()
        // Initialization code
    }

    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)
        // Configure the view for the selected state
    }
}
```

Your tableview delegates

```
override func numberOfSections(in tableView: UITableView) -> Int {
    // You need to return minimum one to show the cell inside the tableview
    return 1
}

override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    // return the number of rows inside the tableview.
    return 3
}
```

```
}
```

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath) as! TableViewCell
    // 标识符字符串应与您在单元格属性检查器 -> 标识符中输入的内容相同。
    // 配置单元格...
    cell.lblTitle.text = "单元格 \(indexPath.row) :" + "Hello"
}
override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
    // 当你点击一个单元格时，此代理方法将被触发
}
```

```
}
```

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath) as! TableViewCell
    // identifier string should be same as what you have entered in the cell Attribute inspector -> identifier.
    // Configure the cell...
    cell.lblTitle.text = "Cell \(indexPath.row) :" + "Hello"
}
override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
    // this delegate method will trigger when you click a cell
}
```

# 第16章：UIRefreshControl 表视图

`UIRefreshControl` 对象提供了一个标准控件，可用于启动表视图内容的刷新。您通过关联的表视图控制器对象将刷新控件链接到表视图。表视图控制器负责将控件添加到表视图的视觉表现中，并根据适当的用户手势管理该控件的显示。

## 第16.1节：在表视图上设置 refreshControl：

```
UIRefreshControl *refreshControl = [[UIRefreshControl alloc] init];
[refreshControl addTarget:self action:@selector(pullToRefresh:)
forControlEvents:UIControlEventValueChanged];
self.scrollView.alwaysBounceVertical = YES;
[self.scrollView addSubview:refreshControl];

- (void)pullToRefresh:(UIRefreshControl*) sender{
// 在主线程之外执行工作
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
    // 模拟网络流量（休眠2秒）
    [NSThread sleepForTimeInterval:2];
    // 更新数据
    //在主线程上调用完成
    dispatch_sync(dispatch_get_main_queue(), ^{
        //更新网络活动界面
        NSLog(@"%@", @"完成");
        [sender endRefreshing];
    });
});
}

}
```

## 第16.2节：Objective-C示例

首先在ViewController中声明一个属性，如下所示

```
@property (nonatomic) UIRefreshControl *refreshControl;
```

稍后在viewDidLoad()中按如下方式设置refreshControl：

```
self.refreshControl = [[UIRefreshControl alloc] init];
[self.tableView addSubview:self.refreshControl];
[self.refreshControl addTarget:self action:@selector(refreshTable)
forControlEvents:UIControlEventValueChanged];
//设置活动动画的tint颜色
self.refreshControl.tintColor = [UIColor redColor];
//为文本设置属性字符串
NSMutableAttributedString * string = [[NSMutableAttributedString alloc]
initWithString:@"firstsecondthird"];
[string addAttribute:NSForegroundColorAttributeName value:[UIColor redColor]
range:NSMakeRange(0, 5)];
[string addAttribute:NSForegroundColorAttributeName value:[UIColor greenColor]
range:NSMakeRange(5, 6)];
[string addAttribute:NSForegroundColorAttributeName value:[UIColor blueColor]
range:NSMakeRange(11, 5)];
self.refreshControl.attributedTitle = string;
```

现在函数 refreshTable 定义为：

# Chapter 16: UIRefreshControl UITableView

A `UIRefreshControl` object provides a standard control that can be used to initiate the refreshing of a table view's contents. You link a refresh control to a table through an associated table view controller object. The table view controller handles the work of adding the control to the table's visual appearance and managing the display of that control in response to appropriate user gestures.

## Section 16.1: Set up refreshControl on tableView:

```
UIRefreshControl *refreshControl = [[UIRefreshControl alloc] init];
[refreshControl addTarget:self action:@selector(pullToRefresh:)
forControlEvents:UIControlEventValueChanged];
self.scrollView.alwaysBounceVertical = YES;
[self.scrollView addSubview:refreshControl];

- (void)pullToRefresh:(UIRefreshControl*) sender{
//Do work off the main thread
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
    // Simulate network traffic (sleep for 2 seconds)
    [NSThread sleepForTimeInterval:2];
    //Update data
    //Call complete on the main thread
    dispatch_sync(dispatch_get_main_queue(), ^{
        //Update network activity UI
        NSLog(@"%@",@"COMPLETE");
        [sender endRefreshing];
    });
});
});
```

## Section 16.2: Objective-C Example

First declare a property like this in the ViewController

```
@property (nonatomic) UIRefreshControl *refreshControl;
```

Later in the `viewDidLoad()` set up the refreshControl as given below:

```
self.refreshControl = [[UIRefreshControl alloc] init];
[self.tableView addSubview:self.refreshControl];
[self.refreshControl addTarget:self action:@selector(refreshTable)
forControlEvents:UIControlEventValueChanged];
//Setting the tint Color of the Activity Animation
self.refreshControl.tintColor = [UIColor redColor];
//Setting the attributed String to the text
NSMutableAttributedString * string = [[NSMutableAttributedString alloc]
initWithString:@"firstsecondthird"];
[string addAttribute:NSForegroundColorAttributeName value:[UIColor redColor]
range:NSMakeRange(0, 5)];
[string addAttribute:NSForegroundColorAttributeName value:[UIColor greenColor]
range:NSMakeRange(5, 6)];
[string addAttribute:NSForegroundColorAttributeName value:[UIColor blueColor]
range:NSMakeRange(11, 5)];
self.refreshControl.attributedTitle = string;
```

Now The function `refreshTable` is defined as:

```
- (void)refreshTable {  
    //TODO: 刷新你的数据  
    [self.refreshControl endRefreshing];  
    [self.refreshControl beginRefreshing];  
    [self.tableView reloadData];  
    [self.refreshControl endRefreshing];  
}
```



```
- (void)refreshTable {  
    //TODO: refresh your data  
    [self.refreshControl endRefreshing];  
    [self.refreshControl beginRefreshing];  
    [self.tableView reloadData];  
    [self.refreshControl endRefreshing];  
}
```



# 第17章：UITableViewCell

加载自定义单元格xib文件使用单元格类别类，无需注册nib文件

## 第17.1节：UITableViewCell的Xib文件

创建一个UITableView单元格类别类。

### UITableViewCell+RRCell.h 文件

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

@interface UITableViewCell (RRCell)
-(id)initWithOwner:(id)owner;
@end
```

### UITableViewCell+RRCell.m 文件

```
#import "UITableViewCell+RRCell.h"

@implementation UITableViewCell (RRCell)

#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wobjc-designated-initializers"

-(id)initWithOwner:(id)owner {
    if (self = [super init]) {
        NSArray *nib = [[NSBundle mainBundle]loadNibNamed:NSStringFromClass([self class])
owner:self options:nil];
        self = [nib objectAtIndex:0];
    }
    return self;
}

#pragma clang diagnostic pop
```

@end

导入单元格类别类以在cellForRowAtIndex方法中使用此方法

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)
*)indexPath
{
    //创建自定义单元格xib文件，通过单元格类别类加载
    CustomCell *cell = [[CustomCell alloc] initWithOwner:self];

    return cell;
}
```

# Chapter 17: UITableViewCell

Load custom cell xib file uses the cell category class, no need to register the nib file

## Section 17.1: Xib file of UITableViewCell

Create a `UITableViewCell` cell category class.

### UITableViewCell+RRCell.h file

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

@interface UITableViewCell (RRCell)
-(id)initWithOwner:(id)owner;
@end
```

### UITableViewCell+RRCell.m file

```
#import "UITableViewCell+RRCell.h"

@implementation UITableViewCell (RRCell)

#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wobjc-designated-initializers"

-(id)initWithOwner:(id)owner {
    if (self = [super init]) {
        NSArray *nib = [[NSBundle mainBundle]loadNibNamed:NSStringFromClass([self class])
owner:self options:nil];
        self = [nib objectAtIndex:0];
    }
    return self;
}

#pragma clang diagnostic pop
```

@end

Import cell category class to use this method into the `cellForRowAtIndex` method

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)
*)indexPath
{
    //Created custom cell xib file to load by cell category class
    CustomCell *cell = [[CustomCell alloc] initWithOwner:self];

    return cell;
}
```

# 第18章：UITableViewCell的自定义选择方法

## 第18.1节：区分行的单选和双选

一个实现UITableView的示例，允许检测单元格是被单击还是双击。

```
override func viewDidLoad() {
    viewDidLoad()

    let doubleTapGestureRecognizer = UITapGestureRecognizer(target: self, action:
#selector(handleDoubleTap(sender:)))
    doubleTapGestureRecognizer.numberOfTapsRequired = 2
    tableView.addGestureRecognizer(doubleTapGestureRecognizer)

    let tapGestureRecognizer = UITapGestureRecognizer(target: self, action:
#selector(handleTapGesture(sender:)))
    tapGestureRecognizer.numberOfTapsRequired = 1
    tapGestureRecognizer.require(toFail: doubleTapGestureRecognizer)
    tableView.addGestureRecognizer(tapGestureRecognizer)
}

func handleTapGesture(sender: UITapGestureRecognizer) {
    let touchPoint = sender.location(in: tableView)
    if let indexPath = tableView.indexPathForRow(at: touchPoint) {
        print(indexPath)
    }
}

func handleDoubleTap(sender: UITapGestureRecognizer) {
    let touchPoint = sender.location(in: tableView)
    if let indexPath = tableView.indexPathForRow(at: touchPoint) {
        print(indexPath)
    }
}
```

# Chapter 18: Custom methods of selection of UITableViewCells

## Section 18.1: Distinction between single and double selection on row

An example of implementation UITableView which allows to detect if cell has been tapped single or double time.

```
override func viewDidLoad() {
    viewDidLoad()

    let doubleTapGestureRecognizer = UITapGestureRecognizer(target: self, action:
#selector(handleDoubleTap(sender:)))
    doubleTapGestureRecognizer.numberOfTapsRequired = 2
    tableView.addGestureRecognizer(doubleTapGestureRecognizer)

    let tapGestureRecognizer = UITapGestureRecognizer(target: self, action:
#selector(handleTapGesture(sender:)))
    tapGestureRecognizer.numberOfTapsRequired = 1
    tapGestureRecognizer.require(toFail: doubleTapGestureRecognizer)
    tableView.addGestureRecognizer(tapGestureRecognizer)
}

func handleTapGesture(sender: UITapGestureRecognizer) {
    let touchPoint = sender.location(in: tableView)
    if let indexPath = tableView.indexPathForRow(at: touchPoint) {
        print(indexPath)
    }
}

func handleDoubleTap(sender: UITapGestureRecognizer) {
    let touchPoint = sender.location(in: tableView)
    if let indexPath = tableView.indexPathForRow(at: touchPoint) {
        print(indexPath)
    }
}
```

# 第19章：UITableViewCell的自定义选择方法

UITableViewCell选择管理的高级方法。当简单的 didSelect...形式的UITableViewDelegate不足以实现某些功能时的示例。

## 第19.1节：区分行上的单选和双选

一个实现示例，能够检测用户是在UITableViewCell上单击还是双击。

```
override func viewDidLoad() {
    viewDidLoad()

    let doubleTapGestureRecognizer = UITapGestureRecognizer(target: self, action:
#selector(handleDoubleTap(sender:)))
    doubleTapGestureRecognizer.numberOfTapsRequired = 2
    tableView.addGestureRecognizer(doubleTapGestureRecognizer)

    let tapGestureRecognizer = UITapGestureRecognizer(target: self, action:
#selector(handleTapGesture(sender:)))
    tapGestureRecognizer.numberOfTapsRequired = 1
    tapGestureRecognizer.require(toFail: doubleTapGestureRecognizer)
    tableView.addGestureRecognizer(tapGestureRecognizer)
}

func handleTapGesture(sender: UITapGestureRecognizer) {
    let touchPoint = sender.location(in: tableView)
    if let indexPath = tableViewindexPathForRow(at: touchPoint) {
        print(indexPath)
    }
}

func handleDoubleTap(sender: UITapGestureRecognizer) {
    let touchPoint = sender.location(in: tableView)
    if let indexPath = tableViewindexPathForRow(at: touchPoint) {
        print(indexPath)
    }
}
```

# Chapter 19: Custom methods of selection of UITableViewCells

Advance ways to manage selections of UITableViewCell. Examples when simple didSelect... form UITableViewDelegate is not enough to achieve something.

## Section 19.1: Distinction between single and double selection on row

An example of implementation which give a possibility to detect if user single or double tap on UITableViewCell.

```
override func viewDidLoad() {
    viewDidLoad()

    let doubleTapGestureRecognizer = UITapGestureRecognizer(target: self, action:
#selector(handleDoubleTap(sender:)))
    doubleTapGestureRecognizer.numberOfTapsRequired = 2
    tableView.addGestureRecognizer(doubleTapGestureRecognizer)

    let tapGestureRecognizer = UITapGestureRecognizer(target: self, action:
#selector(handleTapGesture(sender:)))
    tapGestureRecognizer.numberOfTapsRequired = 1
    tapGestureRecognizer.require(toFail: doubleTapGestureRecognizer)
    tableView.addGestureRecognizer(tapGestureRecognizer)
}

func handleTapGesture(sender: UITapGestureRecognizer) {
    let touchPoint = sender.location(in: tableView)
    if let indexPath = tableViewindexPathForRow(at: touchPoint) {
        print(indexPath)
    }
}

func handleDoubleTap(sender: UITapGestureRecognizer) {
    let touchPoint = sender.location(in: tableView)
    if let indexPath = tableViewindexPathForRow(at: touchPoint) {
        print(indexPath)
    }
}
```

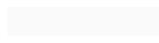
# 第20章：UIView

## 第20.1节：使视图圆角

要使UIView圆角，需为视图的layer指定cornerRadius。



这同样适用于继承自UIView的任何类，例如UIImageView。



### 编程实现

#### Swift代码

```
someImageView.layoutIfNeeded()  
someImageView.clipsToBounds = true  
someImageView.layer.cornerRadius = 10
```

#### Objective-C代码

```
[someImageView layoutIfNeeded];  
someImageView.clipsToBounds = YES;  
someImageView.layer.cornerRadius = 10;
```

### 示例

```
//Swift代码  
topImageView.layoutIfNeeded()  
bottomImageView.layoutIfNeeded()  
topImageView.clipsToBounds = true  
topImageView.layer.cornerRadius = 10  
bottomImageView.clipsToBounds = true  
bottomImageView.layer.cornerRadius = bottomImageView.frame.width / 2
```

```
//Objective-C 代码  
[topImageView layoutIfNeeded]  
[bottomImageView layoutIfNeeded];  
topImageView.clipsToBounds = YES;  
topImageView.layer.cornerRadius = 10;  
bottomImageView.clipsToBounds = YES;  
bottomImageView.cornerRadius = CGRectGetWidth(bottomImageView.frame) / 2;
```

这是结果，显示了使用指定圆角半径的圆角视图效果：

# Chapter 20: UIView

## Section 20.1: Make the view rounded

To make a rounded `UIView`, specify a `cornerRadius` for the view's layer.

*This also applies any class which inherits from `UIView`, such as `UIImageView`.*

### Programmatically

#### Swift Code

```
someImageView.layoutIfNeeded()  
someImageView.clipsToBounds = true  
someImageView.layer.cornerRadius = 10
```

#### Objective-C Code

```
[someImageView layoutIfNeeded];  
someImageView.clipsToBounds = YES;  
someImageView.layer.cornerRadius = 10;
```

### Example

```
//Swift code  
topImageView.layoutIfNeeded()  
bottomImageView.layoutIfNeeded()  
topImageView.clipsToBounds = true  
topImageView.layer.cornerRadius = 10  
bottomImageView.clipsToBounds = true  
bottomImageView.layer.cornerRadius = bottomImageView.frame.width / 2
```

```
//Objective-C code  
[topImageView layoutIfNeeded]  
[bottomImageView layoutIfNeeded];  
topImageView.clipsToBounds = YES;  
topImageView.layer.cornerRadius = 10;  
bottomImageView.clipsToBounds = YES;  
bottomImageView.layer.cornerRadius = CGRectGetWidth(bottomImageView.frame) / 2;
```

Here is the result, showing the rounded view effect using the specified corner radius:



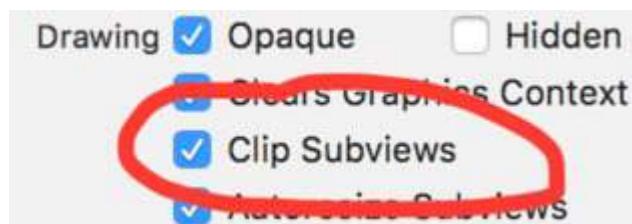
## 注意

要实现此效果，您需要包含 QuartzCore 框架。

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

## Storyboard 配置

通过在 Storyboard 中设置相应属性，也可以实现圆角视图效果，无需编程。



由于 Storyboard 中未暴露layer属性，您必须通过“用户定义的运行时属性”部分修改cornerRadius属性。



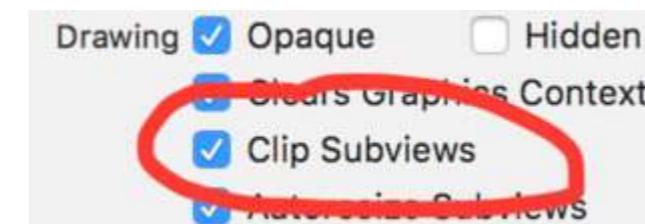
## Note

To do this you need to include the QuartzCore framework.

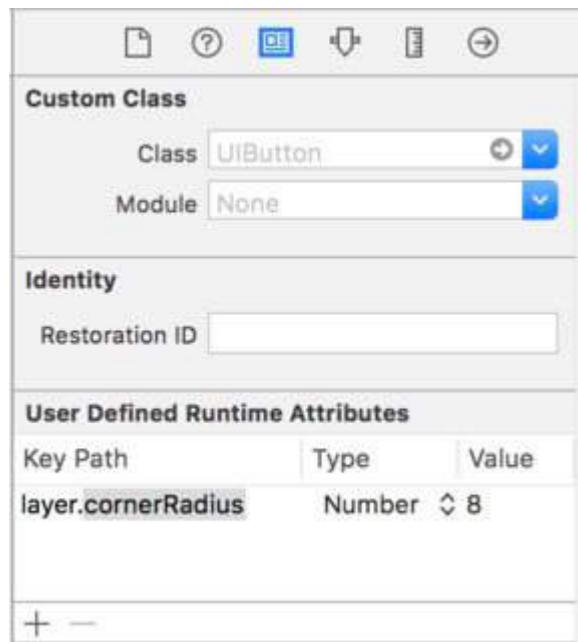
```
#import <QuartzCore/QuartzCore.h>
```

## Storyboard Configuration

A rounded view effect can also be achieved non-programmatically by setting the corresponding properties in **Storyboard**.



Since layer properties aren't exposed in Storyboard, you have to modify the cornerRadius attribute via the User Defined Runtime Attributes section.



## Swift 扩展

只要视图宽高相同，您可以使用这个方便的扩展来应用圆形视图效果。

```
extension UIView {
@discardableResult
public func setAsCircle() -> Self {
    self.clipsToBounds = true
    let frameSize = self.frame.size
    self.layer.cornerRadius = min(frameSize.width, frameSize.height) / 2.0
    return self
}
}
```

使用方法：

```
yourView.setAsCircle()
```

## 第20.2节：使用IBInspectable和IBDesignable

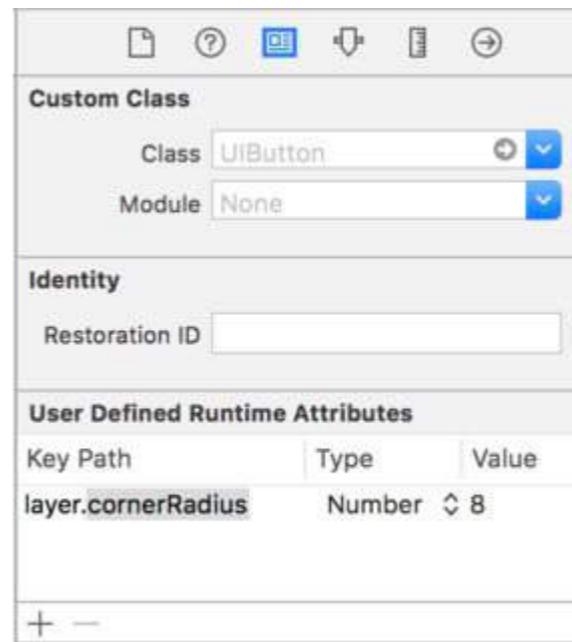
最近 Xcode 版本中最酷的新功能之一（或两个）是IBInspectable属性和IBDesignableUIView。这些功能与应用程序的功能无关，而是影响 Xcode 中的开发者体验。其目标是在不运行应用程序的情况下，能够直观地检查 iOS 应用中的自定义视图。假设你有一个继承自UIView的自定义视图，创意命名为CustomView。

在此自定义视图中，它将显示一串指定颜色的文本。您也可以选择不显示任何文本。  
我们需要三个属性：

```
var textColor: UIColor = UIColor.blackColor()
var text: String?
var showText: Bool = true
```

然后我们可以在类中重写 drawRect 函数：

```
if showText {
    如果 let text = text {
        let s = NSString(string: text)
        s.drawInRect(rect,
```



## Swift Extension

You can use this handy extension to apply rounded view as long as it has same width and height.

```
extension UIView {
@discardableResult
public func setAsCircle() -> Self {
    self.clipsToBounds = true
    let frameSize = self.frame.size
    self.layer.cornerRadius = min(frameSize.width, frameSize.height) / 2.0
    return self
}
}
```

To use it:

```
yourView.setAsCircle()
```

## Section 20.2: Using IBInspectable and IBDesignable

One (or two) of the coolest new features in recent Xcode releases are the IBInspectable properties and IBDesignable UIViews. These have nothing to do with the functionality of your application but instead impact the developer experience in Xcode. The goal is to be able to visually inspect custom views in your iOS application without running it. So assume that you have a custom view creatively named CustomView that inherits from UIView. In this custom view, it will display a string of text with a designated color. You can also choose not to display any text. We'll need three properties:

```
var textColor: UIColor = UIColor.blackColor()
var text: String?
var showText: Bool = true
```

We can then override the drawRect function in the class:

```
if showText {
    if let text = text {
        let s = NSString(string: text)
        s.drawInRect(rect,
```

```

        withAttributes: [
NSForegroundColorAttributeName: textColor,
        NSFontAttributeName: UIFont(name: "Helvetica Neue", size: 18)!
    ])
}
}

```

假设text属性已设置，当应用程序运行时，这将在视图的左上角绘制一个字符串。问题是如果不运行应用程序，我们无法知道它的显示效果。这时IBInspectable和IBDesignable就派上用场了。IBInspectable允许我们在Xcode中像使用内置控件一样直观地设置视图的属性值。IBDesignable会在故事板中显示一个可视化预览。类应该是这样的：

```

@IBDesignable
class CustomView: UIView {
    @IBInspectable var textColor: UIColor = UIColor.blackColor()
    @IBInspectable var text: String?
    @IBInspectable var showText: Bool = true

    override func drawRect(rect: CGRect) {
        // ...
    }
}

```

或者在Objective C中：

```

IB_DESIGNABLE
@interface CustomView: UIView

@property (nonatomic, strong) IBInspectable UIColor* textColor;
@property (nonatomic, strong) IBInspectable NSString* text;
@property (nonatomic, assign) IBInspectable BOOL showText;

@end

@implementation CustomView

- (instancetype)init {
    if(self = [super init]) {
        self.textColor = [UIColor blackColor];
        self.showText = YES;
    }
    return self;
}

- (void)drawRect:(CGRect)rect {
    //...
}

@end

```

接下来的截图展示了在 Xcode 中发生的情况。第一个截图是添加修改后的类之后的效果。

注意，有三个新的 UI 元素对应三个属性。Text Color 会显示一个颜色选择器，Text 只是一个输入框，Show Text 会给我们提供 Off 和 On 两个选项，分别对应 false 和 true。

```

        withAttributes: [
NSForegroundColorAttributeName: textColor,
        NSFontAttributeName: UIFont(name: "Helvetica Neue", size: 18)!
    ])
}
}

```

Assuming that the text property is set, this will draw a string in the upper left hand corner of the view when the application is run. The problem is we won't know what it looks like without running the application. This is where IBInspectable and IBDesignable come in. IBInspectable allows us to visually set property values of the view in Xcode, just like with the built in controls. IBDesignable will show us a visual preview in the storyboard. Here is how the class should look:

```

@IBDesignable
class CustomView: UIView {
    @IBInspectable var textColor: UIColor = UIColor.blackColor()
    @IBInspectable var text: String?
    @IBInspectable var showText: Bool = true

    override func drawRect(rect: CGRect) {
        // ...
    }
}

```

Or in Objective C:

```

IB_DESIGNABLE
@interface CustomView: UIView

@property (nonatomic, strong) IBInspectable UIColor* textColor;
@property (nonatomic, strong) IBInspectable NSString* text;
@property (nonatomic, assign) IBInspectable BOOL showText;

@end

@implementation CustomView

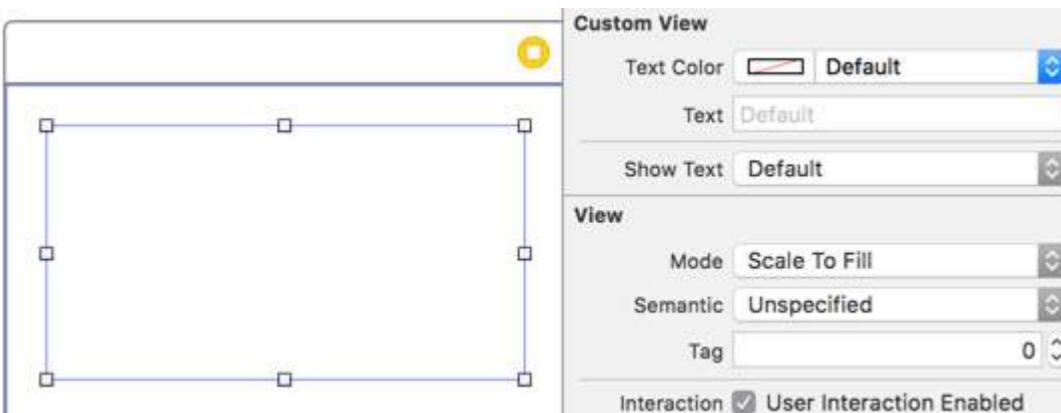
- (instancetype)init {
    if(self = [super init]) {
        self.textColor = [UIColor blackColor];
        self.showText = YES;
    }
    return self;
}

- (void)drawRect:(CGRect)rect {
    //...
}

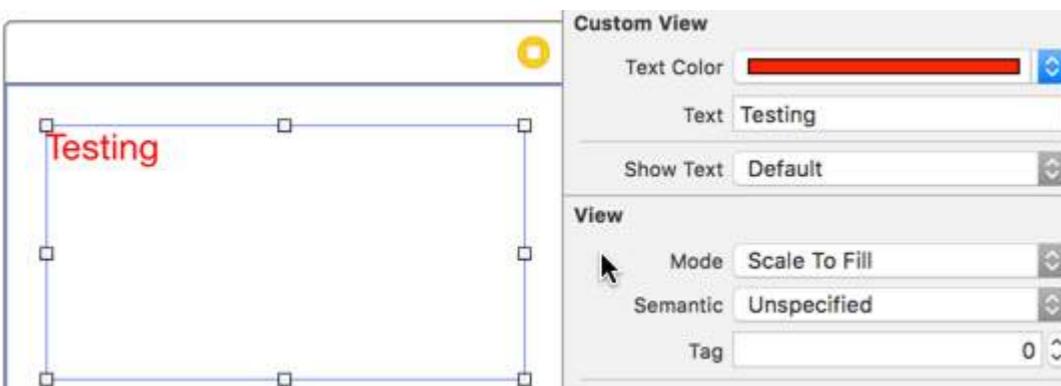
@end

```

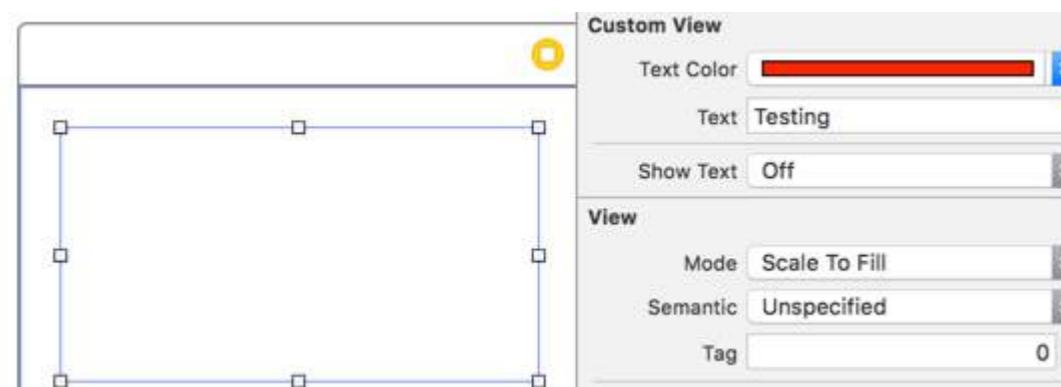
The next screenshots show what happens in Xcode. The first one is what happens after adding the revised class. Notice that there are three new UI elements for the three properties. The Text Color will display a color picker, Text is just an input box and Show Text will give us the options for Off and On which are false and true respectively.



接下来是在使用颜色选择器将文本颜色更改为红色后的效果。同时，提供了一些文本以便drawRect函数显示它。请注意，Interface Builder中的视图也已更新。

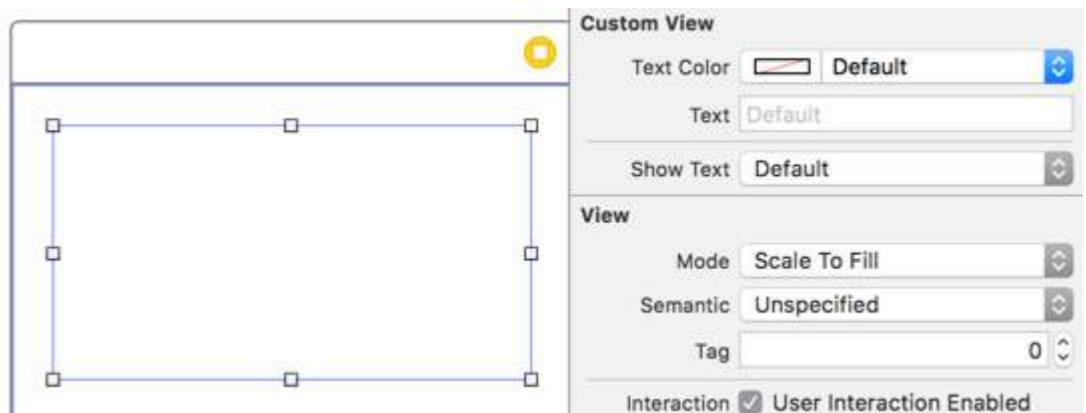


最后，在属性检查器中将显示文本设置为关闭，会使Interface Builder中的文本显示消失。

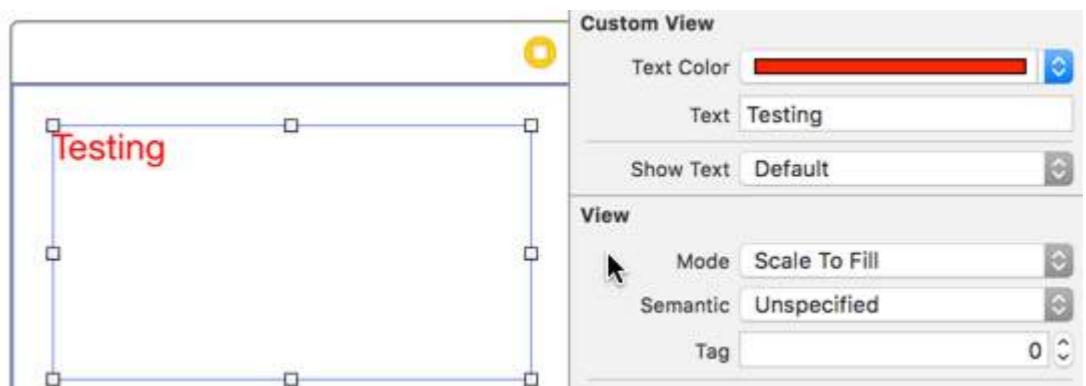


然而，我们都会遇到需要在多个视图中创建圆角UIView的情况  
Storyboard中。与其为Storyboard中的每个视图声明IBDesignable，不如创建一个UIView的扩展，通过设置圆角半径  
，为项目中的每个UIView构建一个用户界面，从而创建圆角视图。在Storyboard中创建的任何UIView上都可以配置边  
框半径。

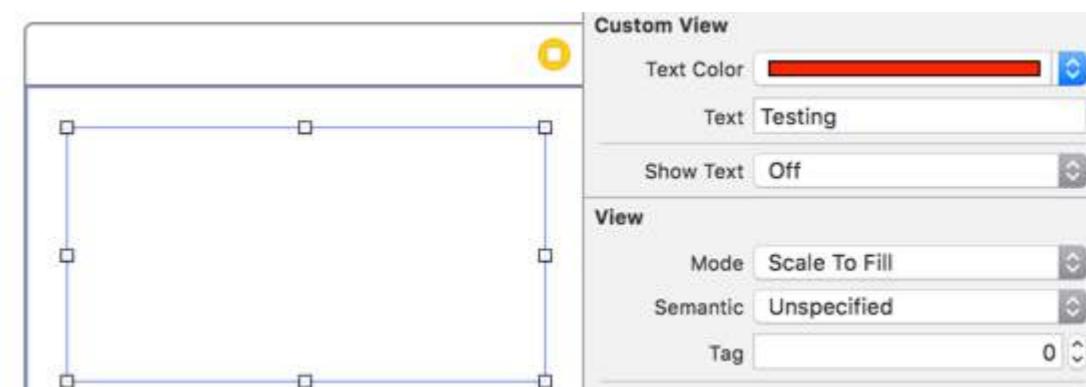
```
extension UIView {
    @IBInspectable 变量 cornerRadius:CGFloat {
        set {
            layer.cornerRadius = newValue
            clipsToBounds = newValue > 0
        }
        get {
            return layer.cornerRadius
        }
    }
}
```



The next is after changing the *Text Color* to red using the color picker. Also, some text has been provided to make the *drawRect* function display it. Notice that the view in Interface Builder has been updated as well.



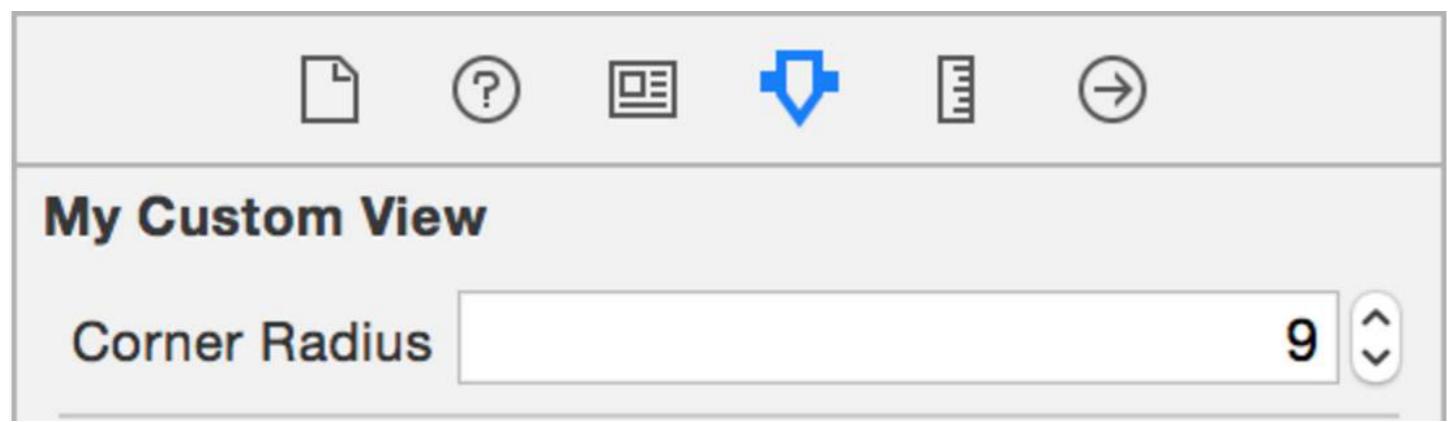
Finally, setting *Show Text* to Off in the property inspector makes the text display in Interface Builder disappear.



However, We all come up situation when we need to create rounded [UIView](#) at multiple views in your  
Storyboard.Instead of declaring IBDesignable to every views of Storyboard, its better to create an Extension of  
[UIView](#) and get a user interface built just for your every [UIView](#) through out the project to create rounded view by  
setting corner radius.A configurable border radius on any UIView you create in storyboard.

```
extension UIView {
    @IBInspectable var cornerRadius:CGFloat {
        set {
            layer.cornerRadius = newValue
            clipsToBounds = newValue > 0
        }
        get {
            return layer.cornerRadius
        }
    }
}
```

}



## 第20.3节：截取快照

你可以这样从一个UIView截取快照：

### Swift

```
let snapshot = view.snapshotView(afterScreenUpdates: true)
```

### Objective-C

```
UIView *snapshot = [view snapshotViewAfterScreenUpdates: YES];
```

## 第20.4节：创建一个UIView

### Objective-C

```
CGRect myFrame = CGRectMake(0, 0, 320, 35)
UIView *view = [[UIView alloc] initWithFrame:myFrame];
```

```
//定义frame的另一种方式
UIView *view = [[UIView alloc] init];
CGRect myFrame = view.frame;
myFrame.size.width = 320;
myFrame.size.height = 35;
myFrame.origin.x = 0;
myFrame.origin.y = 0;
view.frame = myFrame;
```

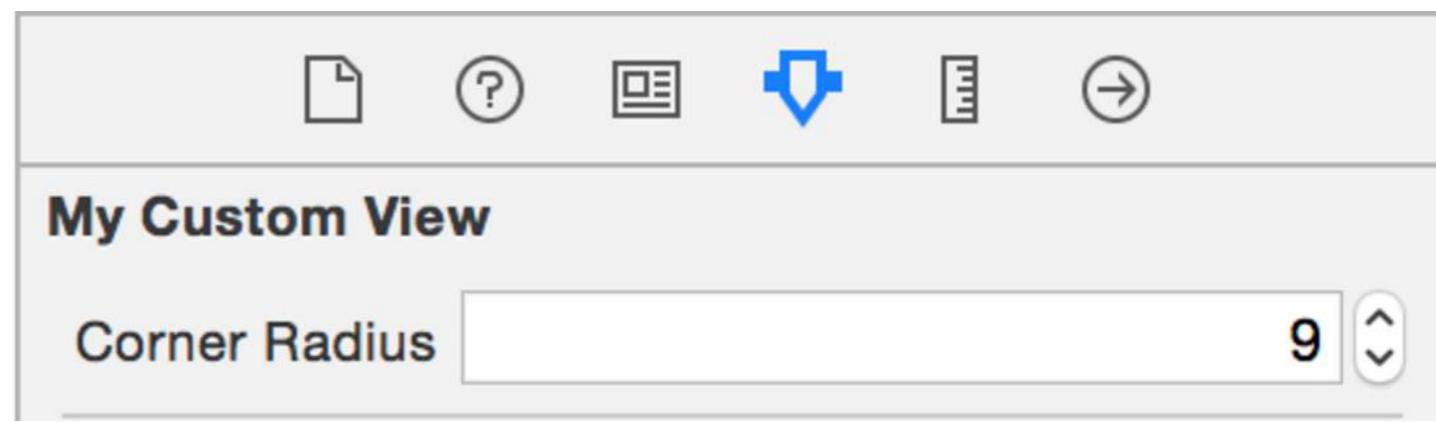
### Swift

```
let myFrame = CGRect(x: 0, y: 0, width: 320, height: 35)
let view = UIView(frame: myFrame)
```

## 第20.5节：抖动视图

```
extension UIView {
    func shake() {
        let animation = CAKeyframeAnimation(keyPath: "transform.translation.x")
        animation.timingFunction = CAMediaTimingFunction(name: kCAMediaTimingFunctionLinear)
        animation.duration = 0.6
        animation.values = [-10.0, 10.0, -7.0, 7.0, -5.0, 5.0, 0.0]
        layer.add(animation, forKey: "shake")
    }
}
```

}



## Section 20.3: Taking a snapshot

You can take a snapshot from a `UIView` like this:

### Swift

```
let snapshot = view.snapshotView(afterScreenUpdates: true)
```

### Objective-C

```
UIView *snapshot = [view snapshotViewAfterScreenUpdates: YES];
```

## Section 20.4: Create a UIView

### Objective-C

```
CGRect myFrame = CGRectMake(0, 0, 320, 35)
UIView *view = [[UIView alloc] initWithFrame:myFrame];
```

```
//Alternative way of defining the frame
UIView *view = [[UIView alloc] init];
CGRect myFrame = view.frame;
myFrame.size.width = 320;
myFrame.size.height = 35;
myFrame.origin.x = 0;
myFrame.origin.y = 0;
view.frame = myFrame;
```

### Swift

```
let myFrame = CGRect(x: 0, y: 0, width: 320, height: 35)
let view = UIView(frame: myFrame)
```

## Section 20.5: Shake a View

```
extension UIView {
    func shake() {
        let animation = CAKeyframeAnimation(keyPath: "transform.translation.x")
        animation.timingFunction = CAMediaTimingFunction(name: kCAMediaTimingFunctionLinear)
        animation.duration = 0.6
        animation.values = [-10.0, 10.0, -7.0, 7.0, -5.0, 5.0, 0.0]
        layer.add(animation, forKey: "shake")
    }
}
```

```
}
```

此函数可用于通过轻微抖动来吸引对特定视图的注意。

## 第20.6节：利用内在内容尺寸

在创建UIView子类时，内在内容尺寸有助于避免设置硬编码的高度和宽度约束

类如何利用这一点的基本示例

```
类 ImageView: UIView {
    变量 image: UIImage {
        设置后 {
            使内在内容尺寸失效()
        }
    }
    // 省略初始化方法
    // 便利初始化(image: UIImage)

    重写函数 内在内容尺寸() -> CGSize {
        返回 CGSize(宽度: image.尺寸.宽度, 高度: image.尺寸.高度)
    }
}
```

如果你只想内在地提供一个尺寸，可以为你想忽略的值提供UIViewNoIntrinsicMetric值。

```
重写函数 内在内容尺寸() -> CGSize {
    返回 CGSize(宽度: UIViewNoIntrinsicMetric, 高度: image.尺寸.宽度)
}
```

### 在使用AutoLayout和Interface Builder时的好处

可以将此 ImageView（或 UIImageView）的水平对齐设置为父视图中心 X，垂直对齐设置为父视图中心 Y。

```
}
```

This function can be used to draw attention to a specific view by shaking it a bit.

## Section 20.6: Utilizing Intrinsic Content Size

When creating a UIView subclass, intrinsic content size helps to avoid setting hardcoded height and width constraints

a basic glimpse into how a class can utilize this

```
class ImageView: UIView {
    var image: UIImage {
        didSet {
            invalidateIntrinsicContentSize()
        }
    }
    // omitting initializers
    // convenience init(image: UIImage)

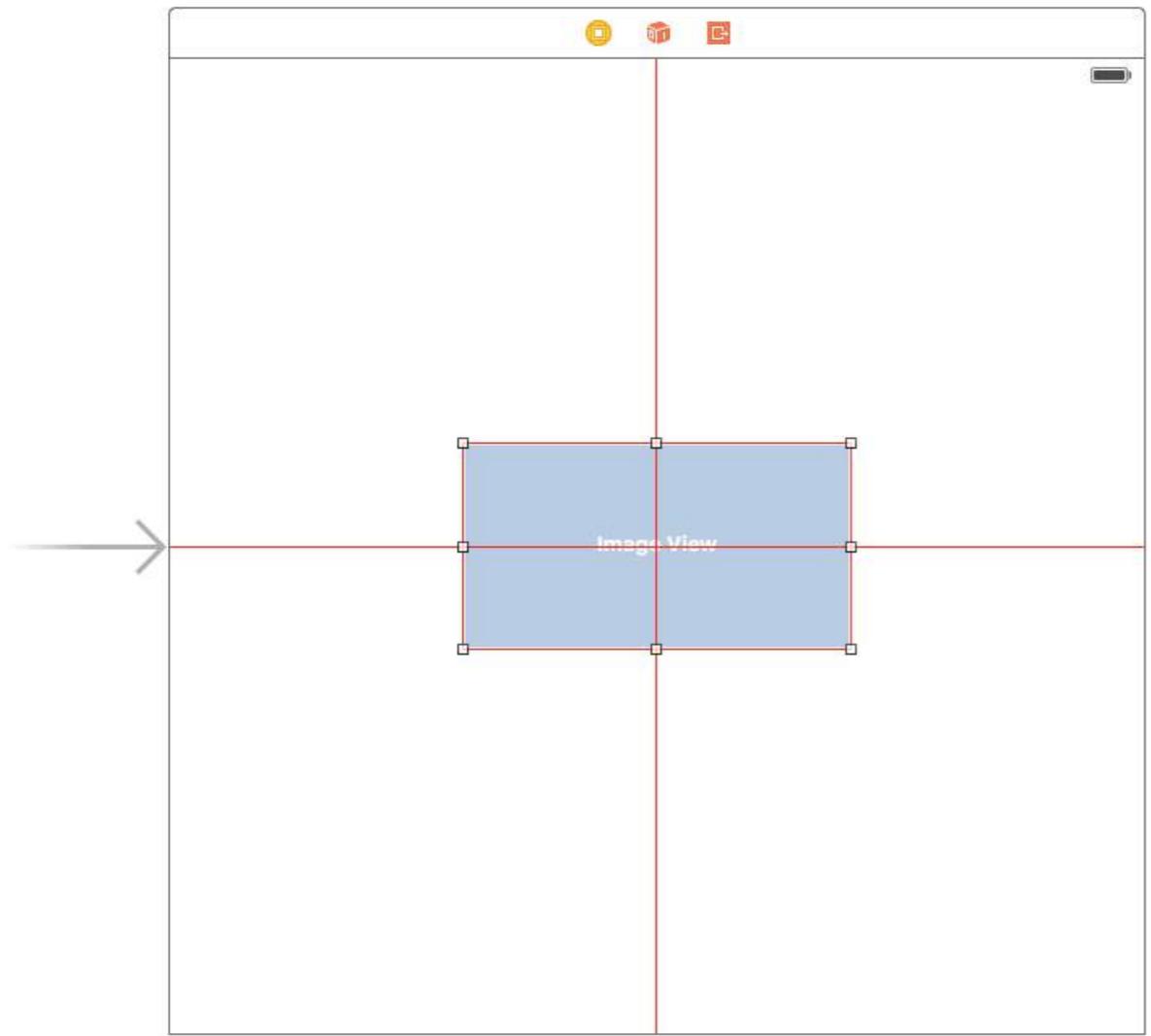
    override func intrinsicContentSize() -> CGSize {
        return CGSize(width: image.size.width, height: image.size.height)
    }
}
```

If you only want to provide one size intrinsically, you can provide the value `UIViewNoIntrinsicMetric` for the value that you wish to ignore.

```
override func intrinsicContentSize() -> CGSize {
    return CGSize(width: UIViewNoIntrinsicMetric, height: image.size.width)
}
```

### Benefits when using with AutoLayout and Interface Builder

One could take this ImageView (or UIImageView) and set the horizontal alignment to superview center X and the vertical alignment to superview center Y.

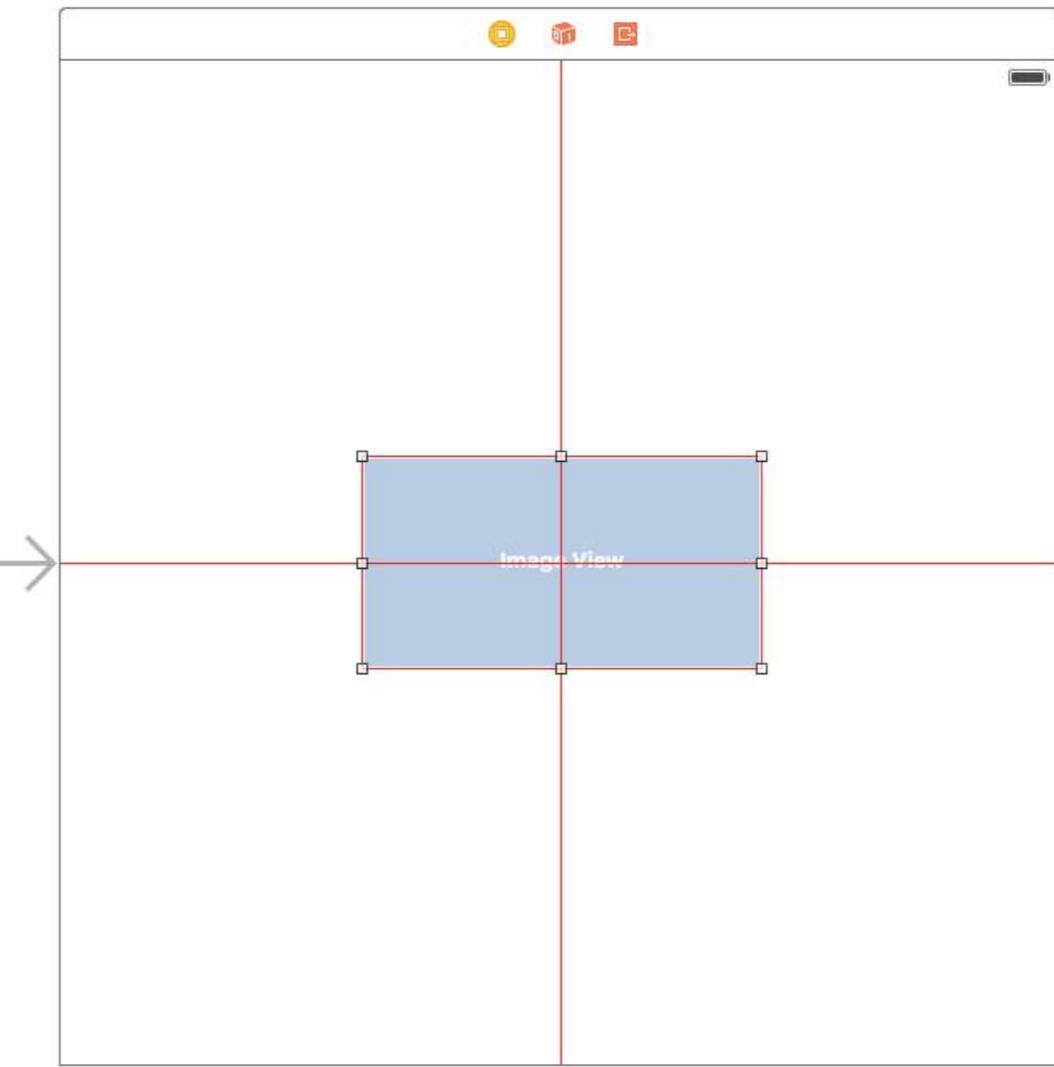


界面构建器此时会向你发出以下警告：



这就是占位符固有尺寸的作用所在。

进入尺寸检查器面板，找到固有尺寸下拉菜单，你可以将该值从默认切换为占位符。

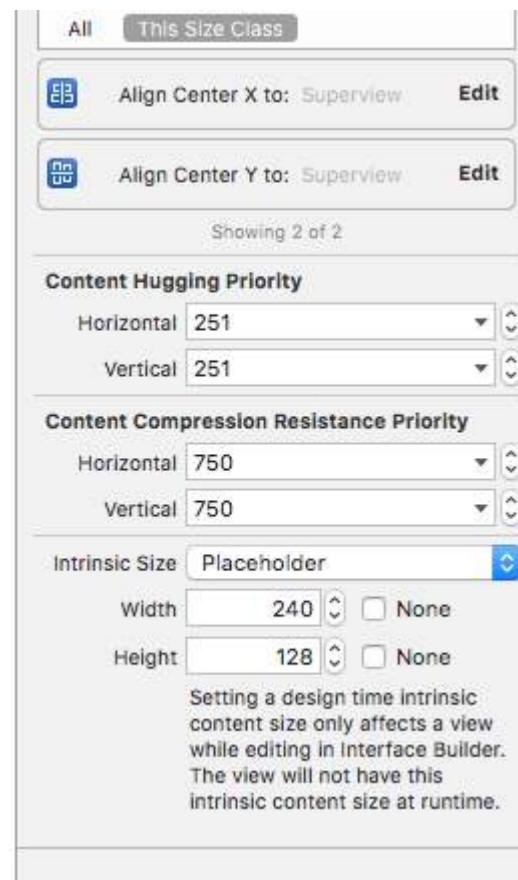


Interface builder will complain to you at this point giving the following warning:



This is where Placeholder Intrinsic Size comes in.

Going into the Size inspector panel, and down to the Intrinsic Size dropdown, you can switch this value from Default to Placeholder.



现在界面构建器会移除之前的警告，你可以使用此尺寸在界面构建器中布局动态尺寸的视图。

## 第20.7节：以编程方式管理UIView插入和从另一个UIView中删除

假设你有一个父视图，想要以编程方式插入一个新的子视图（例如，当你想将一个UIImageView插入到UIViewController的视图中），那么你可以按如下方式操作。

### Objective-C

```
[parentView addSubview:subView];
```

### Swift

```
parentView.addSubview(subView)
```

你也可以将子视图添加到另一个已经是父视图子视图的子视图2下面，使用以下代码：

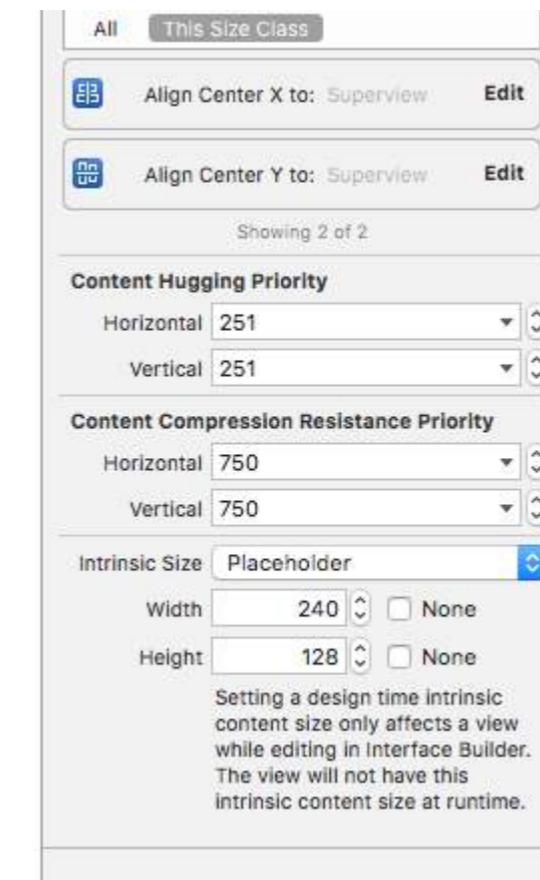
### Objective-C

```
[parentView insertSubview:subView belowSubview:subView2];
```

### Swift

```
parentView.insertSubview(subView, belowSubview: subView2)
```

如果你想将它插入到 subView2 之上，可以这样做：



and now interface builder will remove the previous warnings and you can use this size to have dynamically sized views laid out in interface builder.

## Section 20.7: Programmatically manage UIView insertion and deletion into and from another UIView

Suppose you have a parentView into which you want to insert a new subView programmatically (eg. when you want to insert an UIImageView into a UIViewController's view), than you can do it as below.

### Objective-C

```
[parentView addSubview:subView];
```

### Swift

```
parentView.addSubview(subView)
```

You can also add the subView below another subView2, which is already a sub view of parentView using the following code:

### Objective-C

```
[parentView insertSubview:subView belowSubview:subView2];
```

### Swift

```
parentView.insertSubview(subView, belowSubview: subView2)
```

If you want to insert it above subView2 you can do it this way:

## Objective-C

```
[parentView insertSubview:subView aboveSubview:subView2];
```

### Swift

```
parentView.insertSubview(subView, aboveSubview: subView2)
```

如果在代码的某处你需要将某个 subView 置于最前面，也就是置于所有其他 parentView 的子视图之上，你可以这样做：

## Objective-C

```
[parentView bringSubviewToFront:subView];
```

### Swift

```
parentView.bringSubviewToFront(subView)
```

最后，如果你想从 parentView 中移除 subView，可以按如下方式操作：

## Objective-C

```
[subView removeFromSuperview];
```

### Swift

```
subView.removeFromSuperview()
```

## 第20.8节：使用自动布局创建UIView

```
UIView *view = [[UIView alloc] init];
[self.view addSubview:view];
//如果你想使用高度作为约束，请使用此函数
[self addView:view onParentView:self.view withHeight:200.f];
//如果你想添加一个相对于父视图且应随之调整大小的视图，请使用此函数
[self addFullResizeConstraintForSubview:view addedOnParentView:self.view];
```

### 函数

#### 使用自动布局约束添加固定高度视图的函数

```
-(void)addView:(UIView*)subView onParentView:(UIView*)parentView withHeight:(CGFloat)height{
    subView.translatesAutoresizingMaskIntoConstraints = NO;
    NSLayoutConstraint *trailing =[NSLayoutConstraint
        constraintWithItem:subView
        attribute:NSLayoutAttributeTrailing
        relatedBy:NSLayoutRelationEqual
        toItem:parent
        attribute:NSLayoutAttributeTrailing
    ];
    subView.heightAnchor.constraint(equalToConstant:height).isActive = YES;
}
```

## Objective-C

```
[parentView insertSubview:subView aboveSubview:subView2];
```

### Swift

```
parentView.insertSubview(subView, aboveSubview: subView2)
```

If somewhere in your code you need to bring a certain subView to front, so above all the others parentView's subviews, you can do it like this:

## Objective-C

```
[parentView bringSubviewToFront:subView];
```

### Swift

```
parentView.bringSubviewToFront(subView)
```

Finally, if you want to remove subView from parentView, you can do as below:

## Objective-C

```
[subView removeFromSuperview];
```

### Swift

```
subView.removeFromSuperview()
```

## Section 20.8: Create UIView using Autolayout

```
UIView *view = [[UIView alloc] init];
[self.view addSubview:view];
//Use the function if you want to use height as constraint
[self addView:view onParentView:self.view withHeight:200.f];
```

```
//Use this function if you want to add view with respect to parent and should resize with it
[self addFullResizeConstraintForSubview:view addedOnParentView:self.view];
```

### Functions

#### Function to add view with fixed height using autolayout constraints

```
-(void)addView:(UIView*)subView onParentView:(UIView*)parentView withHeight:(CGFloat)height{
    subView.translatesAutoresizingMaskIntoConstraints = NO;
    NSLayoutConstraint *trailing =[NSLayoutConstraint
        constraintWithItem:subView
        attribute:NSLayoutAttributeTrailing
        relatedBy:NSLayoutRelationEqual
        toItem:parent
        attribute:NSLayoutAttributeTrailing
    ];
    subView.heightAnchor.constraint(equalToConstant:height).isActive = YES;
}
```

```

multiplier:1.0
    constant:10.f];

NSLayoutConstraint *top = [NSLayoutConstraint
    constraintWithItem:subView
    attribute:NSLayoutAttributeTop
    relatedBy:NSLayoutRelationEqual
    toItem:parent
    attribute:NSLayoutAttributeTop
    multiplier:1.0
    constant:10.f];

NSLayoutConstraint *leading = [NSLayoutConstraint
    constraintWithItem:subView
    attribute:NSLayoutAttributeLeading
    relatedBy:NSLayoutRelationEqual
    toItem:parent
    attribute:NSLayoutAttributeLeading
    multiplier:1.0
    constant:10.f];
[parent addConstraint:trailing];
[parent addConstraint:top];
[parent addConstraint:leading];

NSLayoutConstraint *heightConstraint =[NSLayoutConstraint
    constraintWithItem:subView
    attribute:NSLayoutAttributeHeight
    relatedBy:NSLayoutRelationEqual
    toItem:nil
    attribute:0
    multiplier:0.0
    constant:height];

[subView addConstraint:heightConstraint];
}

```

为创建的 `UIView` 添加完整的大小调整约束。

```

-(void)addFullResizeConstraintForSubview:(UIView*)subView addedOnParentView:(UIView*)parentView{
    subView.translatesAutoresizingMaskIntoConstraints = NO;

    NSLayoutConstraint *trailing =[NSLayoutConstraint
        constraintWithItem:subView
        attribute:NSLayoutAttributeTrailing
        relatedBy:NSLayoutRelationEqual
        toItem:parent
        attribute:NSLayoutAttributeTrailing
        multiplier:1.0
        constant:10.f];

    NSLayoutConstraint *top = [NSLayoutConstraint
        constraintWithItem:subView
        attribute:NSLayoutAttributeTop
        relatedBy:NSLayoutRelationEqual
        toItem:parent
        attribute:NSLayoutAttributeTop
        multiplier:1.0
        constant:10.f];

```

```

multiplier:1.0
    constant:10.f];

NSLayoutConstraint *top = [NSLayoutConstraint
    constraintWithItem:subView
    attribute:NSLayoutAttributeTop
    relatedBy:NSLayoutRelationEqual
    toItem:parent
    attribute:NSLayoutAttributeTop
    multiplier:1.0
    constant:10.f];

NSLayoutConstraint *leading = [NSLayoutConstraint
    constraintWithItem:subView
    attribute:NSLayoutAttributeLeading
    relatedBy:NSLayoutRelationEqual
    toItem:parent
    attribute:NSLayoutAttributeLeading
    multiplier:1.0
    constant:10.f];
[parent addConstraint:trailing];
[parent addConstraint:top];
[parent addConstraint:leading];

NSLayoutConstraint *heightConstraint =[NSLayoutConstraint
    constraintWithItem:subView
    attribute:NSLayoutAttributeHeight
    relatedBy:NSLayoutRelationEqual
    toItem:nil
    attribute:0
    multiplier:0.0
    constant:height];

[subView addConstraint:heightConstraint];
}

```

**Function add full resize constraint for created UIView.**

```

-(void)addFullResizeConstraintForSubview:(UIView*)subView addedOnParentView:(UIView*)parentView{
    subView.translatesAutoresizingMaskIntoConstraints = NO;

    NSLayoutConstraint *trailing =[NSLayoutConstraint
        constraintWithItem:subView
        attribute:NSLayoutAttributeTrailing
        relatedBy:NSLayoutRelationEqual
        toItem:parent
        attribute:NSLayoutAttributeTrailing
        multiplier:1.0
        constant:10.f];

    NSLayoutConstraint *top = [NSLayoutConstraint
        constraintWithItem:subView
        attribute:NSLayoutAttributeTop
        relatedBy:NSLayoutRelationEqual
        toItem:parent
        attribute:NSLayoutAttributeTop
        multiplier:1.0
        constant:10.f];

```

```

constant:10.f];

NSLayoutConstraint *leading = [NSLayoutConstraint
    constraintWithItem:subView
    attribute:NSLayoutAttributeLeading
    relatedBy:NSLayoutRelationEqual
    toItem:parent
    attribute:NSLayoutAttributeLeading
    multiplier:1.0
    constant:10.f];

NSLayoutConstraint *bottom =[NSLayoutConstraint
    constraintWithItem:subView
    attribute:NSLayoutAttributeBottom
    relatedBy:NSLayoutRelationEqual
    toItem:parent
    attribute:NSLayoutAttributeBottom
    multiplier:1.0
    constant:0.f];
[parent addConstraint:trailing];
[parent addConstraint:top];
[parent addConstraint:leading];
[parent addConstraint:bottom];
}

```

## 第20.9节：UIView的动画

```

let view = UIView(frame: CGRect(x: 0, y: 0, width: 100, height: 100))
view.backgroundColor = UIColor.orange
self.view.addSubview(view)
UIView.animate(withDuration: 0.75, delay: 0.5, options: .curveEaseIn, animations: {
    //这将使视图从 (0,0) 移动到
    // (self.view.frame.origin.x,self.view.frame.origin.y)
    view.frame.origin.x = self.view.frame.origin.x
    view.frame.origin.y = self.view.frame.origin.y
}) { (finished) in
    view.backgroundColor = UIColor.blueColor()
}

```

## 第20.10节：UIView扩展，用于尺寸和框架属性

如果我们想获取视图的原点x坐标，则需要这样写：

```
view.frame.origin.x
```

对于宽度，我们需要写：

```
view.frame.size.width
```

但如果我们给UIView添加一个简单的扩展，就可以非常简单地获取所有属性，如：

```

view.x
view.y
view.width
view.height

```

它还将帮助设置以下属性，例如：

```

constant:10.f];

NSLayoutConstraint *leading = [NSLayoutConstraint
    constraintWithItem:subView
    attribute:NSLayoutAttributeLeading
    relatedBy:NSLayoutRelationEqual
    toItem:parent
    attribute:NSLayoutAttributeLeading
    multiplier:1.0
    constant:10.f];

NSLayoutConstraint *bottom =[NSLayoutConstraint
    constraintWithItem:subView
    attribute:NSLayoutAttributeBottom
    relatedBy:NSLayoutRelationEqual
    toItem:parent
    attribute:NSLayoutAttributeBottom
    multiplier:1.0
    constant:0.f];
[parent addConstraint:trailing];
[parent addConstraint:top];
[parent addConstraint:leading];
[parent addConstraint:bottom];
}

```

## Section 20.9: Animating a UIView

```

let view = UIView(frame: CGRect(x: 0, y: 0, width: 100, height: 100))
view.backgroundColor = UIColor.orange
self.view.addSubview(view)
UIView.animate(withDuration: 0.75, delay: 0.5, options: .curveEaseIn, animations: {
    //This will cause view to go from (0,0) to
    // (self.view.frame.origin.x,self.view.frame.origin.y)
    view.frame.origin.x = self.view.frame.origin.x
    view.frame.origin.y = self.view.frame.origin.y
}) { (finished) in
    view.backgroundColor = UIColor.blueColor()
}

```

## Section 20.10: UIView extension for size and frame attributes

If we want to get the x-coordinate of origin of the view, then we need to write like:

```
view.frame.origin.x
```

For width, we need to write:

```
view.frame.size.width
```

But if we add a simple extension to an `UIView`, we can get all the attributes very simply, like:

```

view.x
view.y
view.width
view.height

```

It will also help setting these attributes like:

```
view.x = 10  
view.y = 10  
view.width = 100  
view.height = 200
```

简单的扩展如下：

```
extension UIView {  
  
    var x: CGFloat {  
        get {  
            return self.frame.origin.x  
        }  
        set {  
            self.frame = CGRect(x: newValue, y: self.frame.origin.y, width: self.frame.size.width,  
height: self.frame.size.height)  
        }  
    }  
  
    var y: CGFloat {  
        get {  
            return self.frame.origin.y  
        }  
        set {  
            self.frame = CGRect(x: self.frame.origin.x, y: newValue, width: self.frame.size.width,  
height: self.frame.size.height)  
        }  
    }  
  
    var width: CGFloat {  
        get {  
            return self.frame.size.width  
        }  
        set {  
            self.frame = CGRect(x: self.frame.origin.x, y: self.frame.origin.y, width: newValue,  
height: self.frame.size.height)  
        }  
    }  
  
    var height: CGFloat {  
        get {  
            return self.frame.height  
        }  
        set {  
            self.frame = CGRect(x: self.frame.origin.x, y: self.frame.origin.y, width:  
self.frame.size.width, height: newValue)  
        }  
    }  
}
```

我们需要将此类文件添加到项目中，并且它将在整个项目中可用！

```
view.x = 10  
view.y = 10  
view.width = 100  
view.height = 200
```

And the simple extension would be:

```
extension UIView {  
  
    var x: CGFloat {  
        get {  
            return self.frame.origin.x  
        }  
        set {  
            self.frame = CGRect(x: newValue, y: self.frame.origin.y, width: self.frame.size.width,  
height: self.frame.size.height)  
        }  
    }  
  
    var y: CGFloat {  
        get {  
            return self.frame.origin.y  
        }  
        set {  
            self.frame = CGRect(x: self.frame.origin.x, y: newValue, width: self.frame.size.width,  
height: self.frame.size.height)  
        }  
    }  
  
    var width: CGFloat {  
        get {  
            return self.frame.size.width  
        }  
        set {  
            self.frame = CGRect(x: self.frame.origin.x, y: self.frame.origin.y, width: newValue,  
height: self.frame.size.height)  
        }  
    }  
  
    var height: CGFloat {  
        get {  
            return self.frame.height  
        }  
        set {  
            self.frame = CGRect(x: self.frame.origin.x, y: self.frame.origin.y, width:  
self.frame.size.width, height: newValue)  
        }  
    }  
}
```

We need to add this class file in a project and it'll be available to use throughout the project!

# 第21章：UIView快照

## 第21.1节：获取快照

```
- (UIImage *)getSnapshot
{
    UIScreen *screen = [UIScreen mainScreen];
    CGRect bounds = [self.view bounds];
    UIGraphicsBeginImageContextWithOptions(bounds.size, false, screen.scale);
    CGContextRef context = UIGraphicsGetCurrentContext();
    CGContextSetInterpolationQuality(context, kCGInterpolationHigh);
    [self.view drawViewHierarchyInRect:bounds afterScreenUpdates:YES];
    UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    return image;
}
```

### Swift

```
var screenshot: UIImage
{
    UIGraphicsBeginImageContext(self.bounds.size);
    let context = UIGraphicsGetCurrentContext();
    self.layer.render(in: context)
    let screenShot = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    return screenShot
}
```

## 第21.2节：带有其他标记和

文本的子视图快照

- 支持肖像和风景两种类型的图像
- 绘图和其他子视图可以合并，在我的情况下我正在添加标签进行绘制

```
{
    CGSize fullSize = getImageForEdit.size;
    CGSize sizeInView = AVMakeRectWithAspectRatioInsideRect(imgViewFake.image.size,
    imgViewFake.bounds).size;
    CGFloat orgScale = orgScale = fullSize.width/sizeInView.width;
    CGSize newSize = CGSizeMake(orgScale * img.image.size.width, orgScale * img.image.size.height);
    if(newSize.width <= fullSize.width && newSize.height <= fullSize.height){
        newSize = fullSize;
    }
    CGRect offsetRect;
    if (getImageForEdit.size.height > getImageForEdit.size.width){
        CGFloat scale = newSize.height/fullSize.height;
        CGFloat offset = (newSize.width - fullSize.width*scale)/2;
        offsetRect = CGRectMake(offset, 0, newSize.width-offset*2, newSize.height);
    }
    else{
        CGFloat scale = newSize.width/fullSize.width;
        CGFloat offset = (newSize.height - fullSize.height*scale)/2;
        offsetRect = CGRectMake(0, offset, newSize.width, newSize.height-offset*2);
    }
    UIGraphicsBeginImageContextWithOptions(newSize, NO, getImageForEdit.scale);
    [getImageForEdit drawAtPoint:offsetRect.origin];
    // [img.image drawInRect:CGRectMake(0,0,newSize.width,newSize.height)];
}
```

# Chapter 21: Snapshot of UIView

## Section 21.1: Getting the Snapshot

```
- (UIImage *)getSnapshot
{
    UIScreen *screen = [UIScreen mainScreen];
    CGRect bounds = [self.view bounds];
    UIGraphicsBeginImageContextWithOptions(bounds.size, false, screen.scale);
    CGContextRef context = UIGraphicsGetCurrentContext();
    CGContextSetInterpolationQuality(context, kCGInterpolationHigh);
    [self.view drawViewHierarchyInRect:bounds afterScreenUpdates:YES];
    UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    return image;
}
```

### Swift

```
var screenshot: UIImage
{
    UIGraphicsBeginImageContext(self.bounds.size);
    let context = UIGraphicsGetCurrentContext();
    self.layer.render(in: context)
    let screenShot = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    return screenShot
}
```

## Section 21.2: Snapshot with subview with other markup and text

- Support Portrait and Landscape both type of image
- Drawing and other subviews can be merged in my case I'm adding label to draw

```
{
    CGSize fullSize = getImageForEdit.size;
    CGSize sizeInView = AVMakeRectWithAspectRatioInsideRect(imgViewFake.image.size,
    imgViewFake.bounds).size;
    CGFloat orgScale = orgScale = fullSize.width/sizeInView.width;
    CGSize newSize = CGSizeMake(orgScale * img.image.size.width, orgScale * img.image.size.height);
    if(newSize.width <= fullSize.width && newSize.height <= fullSize.height){
        newSize = fullSize;
    }
    CGRect offsetRect;
    if (getImageForEdit.size.height > getImageForEdit.size.width){
        CGFloat scale = newSize.height/fullSize.height;
        CGFloat offset = (newSize.width - fullSize.width*scale)/2;
        offsetRect = CGRectMake(offset, 0, newSize.width-offset*2, newSize.height);
    }
    else{
        CGFloat scale = newSize.width/fullSize.width;
        CGFloat offset = (newSize.height - fullSize.height*scale)/2;
        offsetRect = CGRectMake(0, offset, newSize.width, newSize.height-offset*2);
    }
    UIGraphicsBeginImageContextWithOptions(newSize, NO, getImageForEdit.scale);
    [getImageForEdit drawAtPoint:offsetRect.origin];
    // [img.image drawInRect:CGRectMake(0,0,newSize.width,newSize.height)];
}
```

```
CGFloat oldScale = img.contentScaleFactor;
img.contentScaleFactor = getImageForEdit.scale;
[img drawViewHierarchyInRect:CGRectMake(0, 0, newSize.width, newSize.height)
afterScreenUpdates:YES];
img.contentScaleFactor = oldScale;
UIImage *combImage = UIGraphicsGetImageFromCurrentImageContext();
UIGraphicsEndImageContext();
imageData = UIImageJPEGRepresentation(combImage, 1);
}
```

```
CGFloat oldScale = img.contentScaleFactor;
img.contentScaleFactor = getImageForEdit.scale;
[img drawViewHierarchyInRect:CGRectMake(0, 0, newSize.width, newSize.height)
afterScreenUpdates:YES];
img.contentScaleFactor = oldScale;
UIImage *combImage = UIGraphicsGetImageFromCurrentImageContext();
UIGraphicsEndImageContext();
imageData = UIImageJPEGRepresentation(combImage, 1);
}
```

# 第22章：UIAlertController

## 第22.1节：使用UIAlertController的警告视图

`UIalertView` 和 `UIActionSheet` 在 iOS 8 及以后版本中已被弃用。因此，苹果引入了一个新的控制器用于警告视图 和 操作表，称为 `UIAlertController`，通过更改 `preferredStyle`，可以在 警告视图 和 操作表 之间切换。它没有代理方法，因为所有按钮事件都在它们的 块中处理。

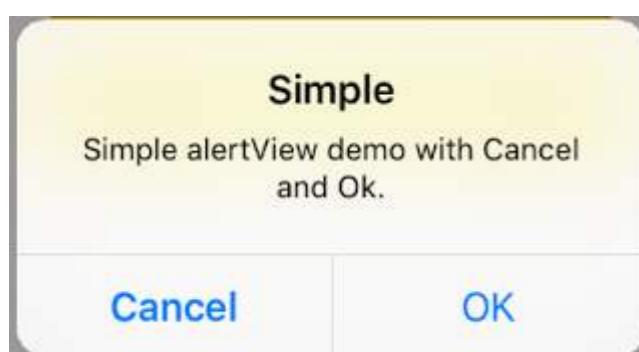
### 简单警告视图

*Swift :*

```
let alert = UIAlertController(title: "简单", message: "带有取消和  
确定按钮的简单警告视图演示。", preferredStyle alert) .  
  
alert.addAction(UIAlertAction(title: "取消", style: .cancel) { _ in  
    print("取消")  
})  
alert.addAction(UIAlertAction(title: "确定", style: .default) { _ in  
    print("确定")  
})  
  
present(alert, animated: true)
```

*Objective-C :*

```
UIAlertController *alertController = [UIAlertController alertControllerWithTitle:@"简单"  
message:@"带有取消和确定按钮的简单警告视图演示。" preferredStyle:UIAlertControllerStyleAlert];  
UIAlertAction *cancelAction = [UIAlertAction actionWithTitle:@"取消"  
style:UIAlertActionStyleCancel handler:^(UIAlertAction * action) {  
    NSLog(@"取消");  
}];  
UIAlertAction *okAction = [UIAlertAction actionWithTitle:@"确定" style:UIAlertActionStyleDefault  
handler:^(UIAlertAction * action) {  
    NSLog(@"确定");  
}];  
  
[alertController addAction:cancelAction];  
[alertController addAction:okAction];  
[self presentViewController:alertController animated: YES completion: nil];
```



破坏性警告视图

*Swift :*

# Chapter 22: UIAlertController

## Section 22.1: AlertViews with UIAlertController

`UIalertView` and `UIActionSheet` are Deprecated in iOS 8 and Later. So Apple introduced a new controller for AlertView and ActionSheet called `UIAlertController` , changing the `preferredStyle`, you can switch between AlertView and ActionSheet. There is no delegate method for it because all button events are handled in their blocks.

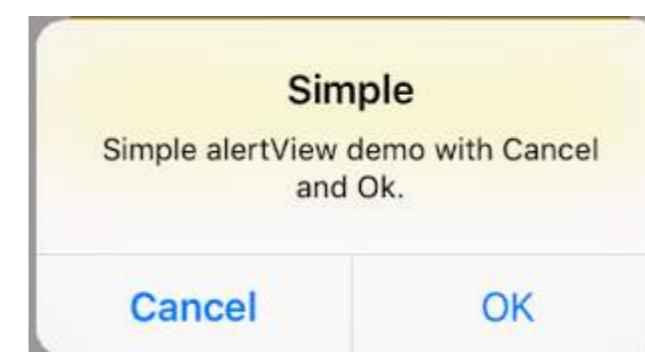
### Simple AlertView

*Swift:*

```
let alert = UIAlertController(title: "Simple", message: "Simple alertView demo with Cancel and  
OK.", preferredStyle: .alert)  
  
alert.addAction(UIAlertAction(title: "Cancel", style: .cancel) { _ in  
    print("Cancel")  
})  
alert.addAction(UIAlertAction(title: "OK", style: .default) { _ in  
    print("OK")  
})  
  
present(alert, animated: true)
```

*Objective-C :*

```
UIAlertController *alertController = [UIAlertController alertControllerWithTitle:@"Simple"  
message:@"Simple alertView demo with Cancel and OK." preferredStyle:UIAlertControllerStyleAlert];  
UIAlertAction *cancelAction = [UIAlertAction actionWithTitle:@"Cancel"  
style:UIAlertActionStyleCancel handler:^(UIAlertAction * action) {  
    NSLog(@"Cancel");  
}];  
UIAlertAction *okAction = [UIAlertAction actionWithTitle:@"OK" style:UIAlertActionStyleDefault  
handler:^(UIAlertAction * action) {  
    NSLog(@"OK");  
}];  
  
[alertController addAction:cancelAction];  
[alertController addAction:okAction];  
[self presentViewController:alertController animated: YES completion: nil];
```



Destructive AlertView

*Swift:*

```

let alert = UIAlertController(title: "简单", message: "带有取消和
确定按钮的简单警告视图演示。", preferredStyle: .alert)

alert.addAction(UIAlertAction(title: "破坏性", style: .destructive) { _ in
    print("破坏性")
})
alert.addAction(UIAlertAction(title: "确定", style: .default) { _ in
    print("确定")
})

present(alert, animated: true)

```

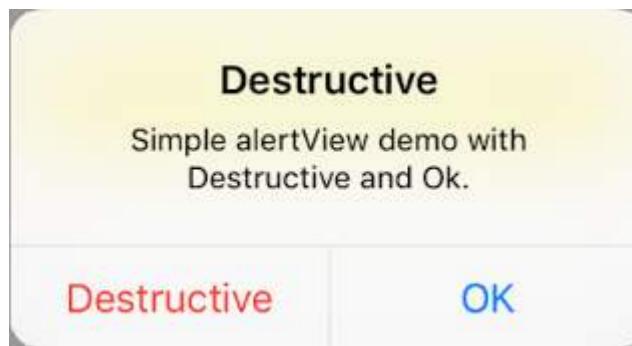
#### Objective-C :

```

UIAlertController *alertController = [UIAlertController alertControllerWithTitle:@"破坏性"
message:@"带有破坏性和确定按钮的简单警告视图演示。"
preferredStyle:UIAlertControllerStyleAlert];
UIAlertAction *destructiveAction = [UIAlertAction actionWithTitle:@"取消"
style:UIAlertActionStyleDestructive handler:^(UIAlertAction * action) {
    NSLog(@"破坏性");
}];
UIAlertAction *okAction = [UIAlertAction actionWithTitle:@"确定" style:UIAlertActionStyleDefault
handler:^(UIAlertAction * action) {
NSLog(@"确定");
}];

[alertController addAction:destructiveAction];
[alertController addAction:okAction];
[self presentViewController:alertController animated: YES completion: nil];

```



## 第22.2节：使用UIAlertController的操作表

使用UIAlertController，类似已弃用的UIActionSheet的操作表通过与AlertViews相同的API创建。

#### 带有两个按钮的简单操作表

##### Swift

```

let alertController = UIAlertController(title: "演示", message: "带有两个按钮的演示",
preferredStyle: UIAlertActionStyle.actionSheet)

```

#### Objective-C

```

UIAlertController *alertController = [UIAlertController alertControllerWithTitle:@"演示"
message:@"带有两个按钮的演示" preferredStyle:UIAlertControllerStyleActionSheet];

```

创建“取消”和“确定”按钮

```

let alert = UIAlertController(title: "Simple", message: "Simple alertView demo with Cancel and
OK.", preferredStyle: .alert)

alert.addAction(UIAlertAction(title: "Destructive", style: .destructive) { _ in
    print("Destructive")
})
alert.addAction(UIAlertAction(title: "OK", style: .default) { _ in
    print("OK")
})

present(alert, animated: true)

```

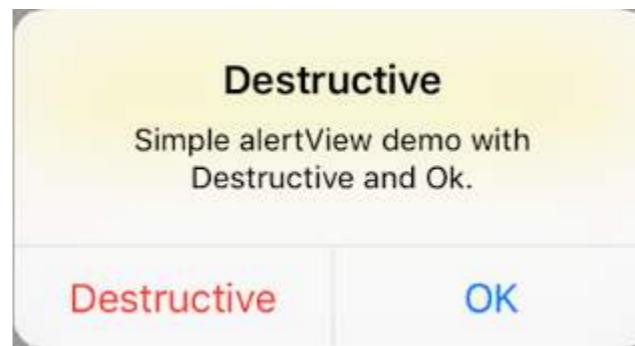
#### Objective-C:

```

UIAlertController *alertController = [UIAlertController alertControllerWithTitle:@"Destructive"
message:@"Simple alertView demo with Destructive and OK."
preferredStyle:UIAlertControllerStyleAlert];
UIAlertAction *destructiveAction = [UIAlertAction actionWithTitle:@"Cancel"
style:UIAlertActionStyleDestructive handler:^(UIAlertAction * action) {
    NSLog(@"Destructive");
}];
UIAlertAction *okAction = [UIAlertAction actionWithTitle:@"OK" style:UIAlertActionStyleDefault
handler:^(UIAlertAction * action) {
    NSLog(@"OK");
}];

[alertController addAction:destructiveAction];
[alertController addAction:okAction];
[self presentViewController:alertController animated: YES completion: nil];

```



## Section 22.2: Action Sheets with UIAlertController

With UIAlertController, action sheets like the deprecated UIActionSheet are created with the same API as you use for AlertViews.

#### Simple Action Sheet with two buttons

##### Swift

```

let alertController = UIAlertController(title: "Demo", message: "A demo with two buttons",
preferredStyle: UIAlertActionStyle.actionSheet)

```

#### Objective-C

```

UIAlertController *alertController = [UIAlertController alertControllerWithTitle:@"Demo"
message:@"A demo with two buttons" preferredStyle:UIAlertControllerStyleActionSheet];

```

Create the buttons "Cancel" and "Okay"

## Swift

```
let cancelAction = UIAlertAction(title: "取消", style: .cancel) { (result : UIAlertAction) ->
Void in
    //按钮按下时的操作
}
let okAction = UIAlertAction(title: "确定", style: .default) { (result : UIAlertAction) -> Void in
    //按钮按下时的操作
}
```

## Objective-C

```
UIAlertAction *cancelAction = [UIAlertAction actionWithTitle:@"取消"
style:UIAlertActionStyleCancel handler:^(UIAlertAction * action) {
    //按钮按下时的操作
}];

UIAlertAction * okAction = [UIAlertAction actionWithTitle:@"确定" style:UIAlertActionStyleDefault
handler:^(UIAlertAction * action) {
    //按钮按下时的操作
}];
```

然后将它们添加到操作表中：

## Swift

```
alertController.addAction(cancelAction)
alertController.addAction(okAction)
```

## Objective-C

```
[alertController addAction:cancelAction];
[alertController addAction:okAction];
```

现在展示UIAlertController：

## Swift

```
self.present(alertController, animated: true, completion: nil)
```

## Objective-C

```
[self presentViewController:alertController animated: YES completion: nil];
```

这应该是结果：

## Swift

```
let cancelAction = UIAlertAction(title: "Cancel", style: .cancel) { (result : UIAlertAction) ->
Void in
    //action when pressed button
}
let okAction = UIAlertAction(title: "Okay", style: .default) { (result : UIAlertAction) -> Void in
    //action when pressed button
}
```

## Objective-C

```
UIAlertAction *cancelAction = [UIAlertAction actionWithTitle:@"Cancel"
style:UIAlertActionStyleCancel handler:^(UIAlertAction * action) {
    //action when pressed button
}];

UIAlertAction * okAction = [UIAlertAction actionWithTitle:@"Okay" style:UIAlertActionStyleDefault
handler:^(UIAlertAction * action) {
    //action when pressed button
}];
```

And add them to the action sheet:

## Swift

```
alertController.addAction(cancelAction)
alertController.addAction(okAction)
```

## Objective-C

```
[alertController addAction:cancelAction];
[alertController addAction:okAction];
```

Now present the UIAlertController:

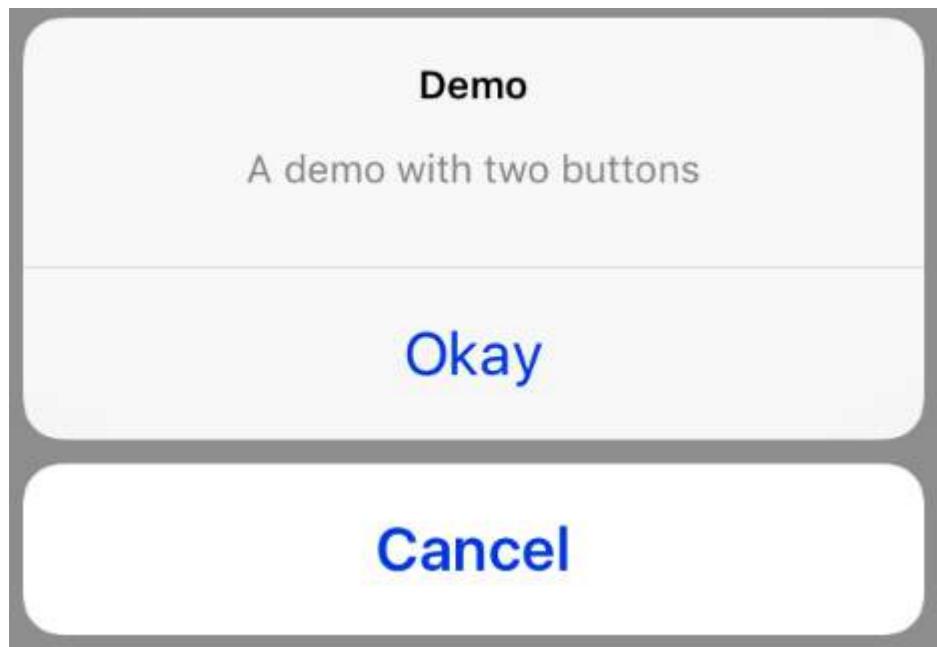
## Swift

```
self.present(alertController, animated: true, completion: nil)
```

## Objective-C

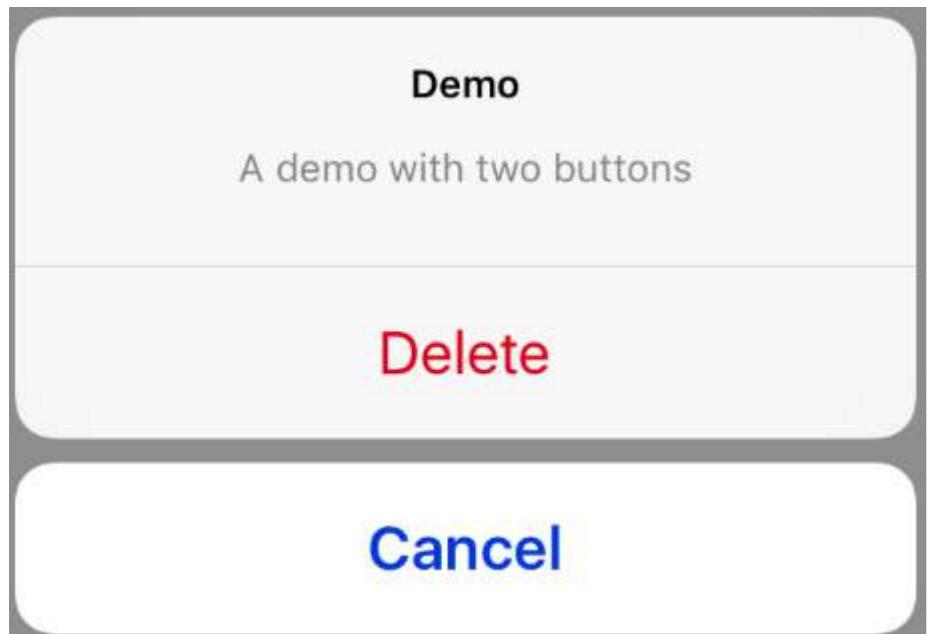
```
[self presentViewController:alertController animated: YES completion: nil];
```

This should be the result:



#### 带有破坏性按钮的操作表

使用UIAlertActionStyle .destructive为UIAlertAction将创建一个红色调的按钮。



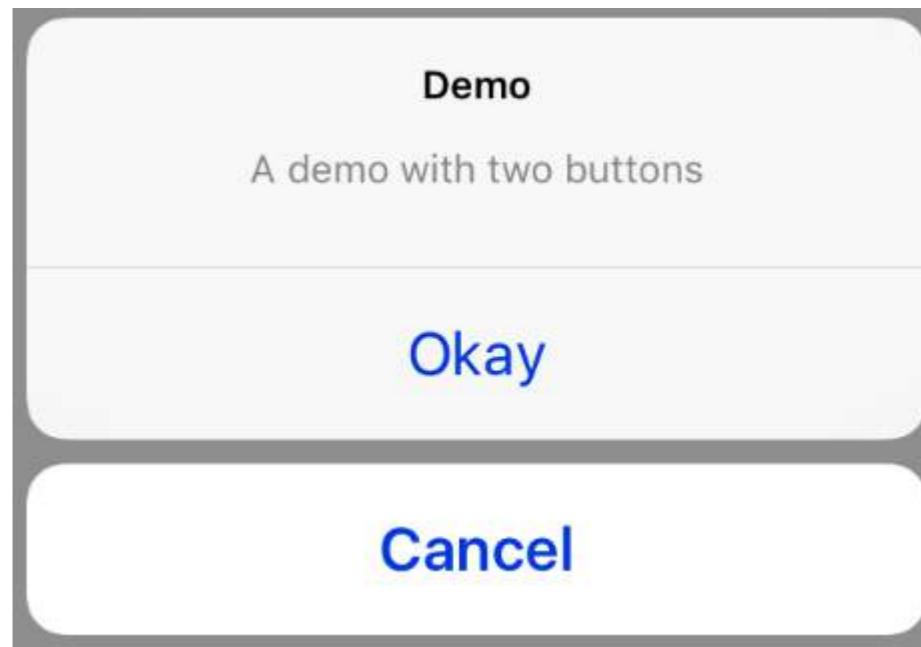
在此示例中，上面的okAction被替换为此UIAlertAction：

#### Swift

```
let destructiveAction = UIAlertAction(title: "删除", style: .destructive) { (result : UIAlertAction) -> Void in
    //按钮按下时的操作
}
```

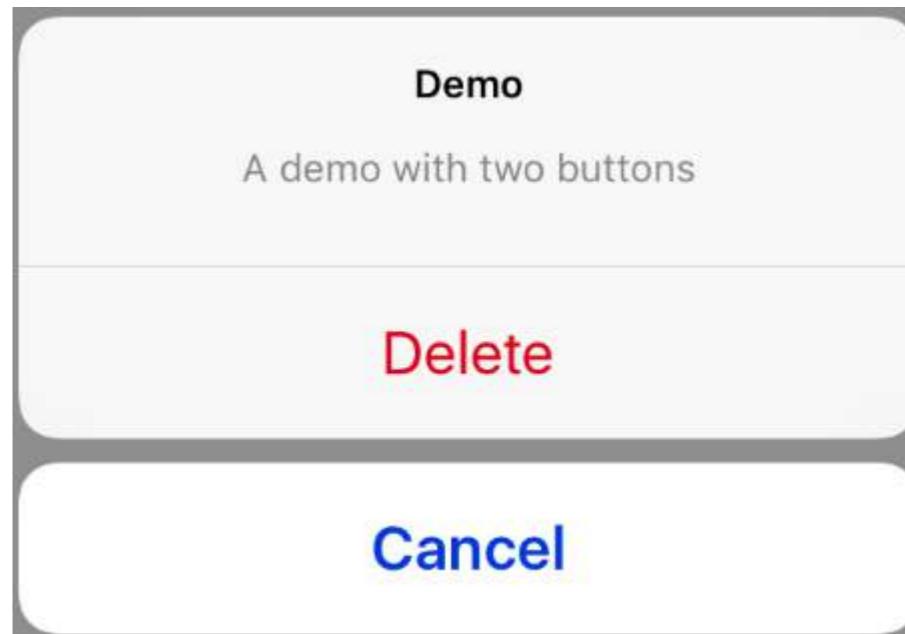
#### Objective-C

```
UIAlertAction * destructiveAction = [UIAlertAction actionWithTitle:@"删除"
style:UIAlertActionStyleDestructive handler:^(UIAlertAction * action) {
    //按钮按下时的操作
}];
```



#### Action Sheet with destructive button

Using the UIAlertActionStyle .destructive for an UIAlertAction will create a button with red tint color.



For this example, the okAction from above was replaced by this UIAlertAction:

#### Swift

```
let destructiveAction = UIAlertAction(title: "Delete", style: .destructive) { (result : UIAlertAction) -> Void in
    //action when pressed button
}
```

#### Objective-C

```
UIAlertAction * destructiveAction = [UIAlertAction actionWithTitle:@"Delete"
style:UIAlertActionStyleDestructive handler:^(UIAlertAction * action) {
    //action when pressed button
}];
```

## 第22.3节：在UIAlertController中添加文本字段，类似于提示框

### Swift

```
let alert = UIAlertController(title: "你好",
message: "欢迎来到iOS的世界",
preferredStyle: UIAlertControllerStyle.alert)
let defaultAction = UIAlertAction(title: "确定", style: UIAlertActionStyle.default) { (action) in
}
defaultAction.isEnabled = false
alert.addAction(defaultAction)

alert.addTextFieldWithConfigurationHandler { (textField) in
    textField.delegate = self
}

present(alert, animated: true, completion: nil)
```

### Objective-C

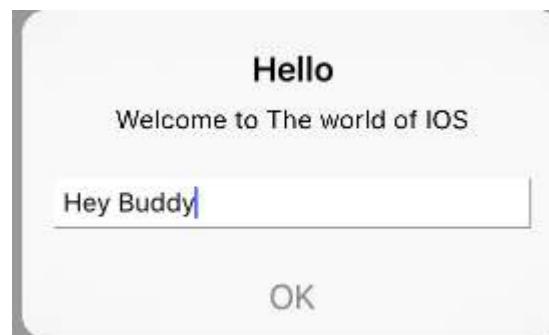
```
UIAlertController* alert = [UIAlertController alertControllerWithTitle:@"你好"
                                                               message:@"欢迎来到iOS的世界"
                                                               preferredStyle:UIAlertControllerStyleAlert];

UIAlertAction* defaultAction = [UIAlertAction actionWithTitle:@"确定"
                                                       style:UIAlertActionStyleDefault
                                                     handler:^(UIAlertAction * action) {}];

defaultAction.enabled = NO;
[alert addAction:defaultAction];

[alert addTextFieldWithConfigurationHandler:^(UITextField *textField) {
    textField.delegate = self;
}];

[self presentViewController:alert animated:YES completion:nil];
```



## 第22.4节：突出显示操作按钮

警报控制器有一个属性，用于强调添加到警报控制器中的某个操作。该属性可用于突出显示特定操作以引起用户注意。对于 Objective C；

```
@property(nonatomic, strong) UIAlertAction *preferredAction
```

已经添加到警报控制器中的操作可以赋值给此属性。警报控制器将

## Section 22.3: Adding Text Field in UIAlertController like a prompt Box

### Swift

```
let alert = UIAlertController(title: "Hello",
                             message: "Welcome to the world of iOS",
                             preferredStyle: UIAlertControllerStyle.alert)
let defaultAction = UIAlertAction(title: "OK", style: UIAlertActionStyle.default) { (action) in
}
defaultAction.isEnabled = false
alert.addAction(defaultAction)

alert.addTextFieldWithConfigurationHandler { (textField) in
    textField.delegate = self
}

present(alert, animated: true, completion: nil)
```

### Objective-C

```
UIAlertController* alert = [UIAlertController alertControllerWithTitle:@"Hello"
                                                               message:@"Welcome to the world of
                                                               iOS"
                                                               preferredStyle:UIAlertControllerStyleAlert];

UIAlertAction* defaultAction = [UIAlertAction actionWithTitle:@"OK"
                                                       style:UIAlertActionStyleDefault
                                                     handler:^(UIAlertAction * action) {}];

defaultAction.enabled = NO;
[alert addAction:defaultAction];

[alert addTextFieldWithConfigurationHandler:^(UITextField *textField) {
    textField.delegate = self;
}];

[self presentViewController:alert animated:YES completion:nil];
```



## Section 22.4: Highlighting an action button

Alert controller has a property which is used to put emphases on an action added in the alert controller. This property can be used to highlight a particular action for user attention. For objective C;

```
@property(nonatomic, strong) UIAlertAction *preferredAction
```

An action **which is already added in alert controller** can be assigned to this property. The Alert Controller will

突出显示此操作。

**此属性只能用于UIAlertControllerStyleAlert。**

以下示例演示如何使用它。

```
UIAlertController *alertController = [UIAlertController alertControllerWithTitle:@"取消编辑"
message:@"您确定要取消编辑吗?" preferredStyle:UIAlertControllerStyleAlert];

UIAlertAction *cancel = [UIAlertAction actionWithTitle:@"取消" style:UIAlertActionStyleCancel
handler:^(UIAlertAction * action) {
    NSLog(@"取消");
}];

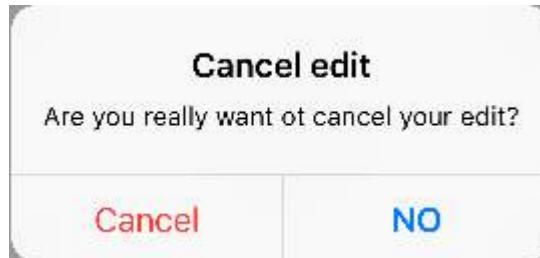
UIAlertAction *no = [UIAlertAction actionWithTitle:@"否" style:UIAlertActionStyleDefault
handler:^(UIAlertAction * action) {
    NSLog(@"高亮按钮被按下.");
}];

[alertController addAction:cancel];
[alertController addAction:no];

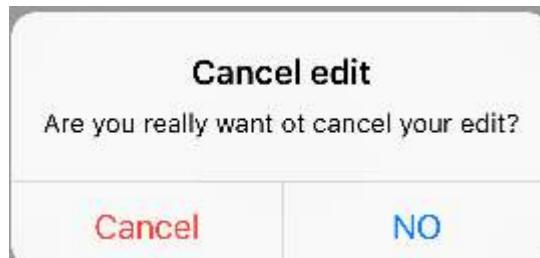
//将“否”操作设置为首选操作。
//注意
//该操作应已添加到警报控制器
alertController.preferredAction = no;

[self presentViewController:alertController animated: YES completion: nil];
```

警报控制器设置了首选操作。 NO 按钮被高亮显示。



警报控制器未设置首选操作。 NO 按钮未被高亮显示。



## 第22.5节：显示和处理警报

一个按钮

highlight this action.

**This property can only be used with UIAlertControllerStyleAlert.**

Following example shows how to use it.

```
UIAlertController *alertController = [UIAlertController alertControllerWithTitle:@"Cancel edit"
message:@"Are you really want to cancel your edit?" preferredStyle:UIAlertControllerStyleAlert];

UIAlertAction *cancel = [UIAlertAction actionWithTitle:@"Cancel" style:UIAlertActionStyleCancel
handler:^(UIAlertAction * action) {
    NSLog(@"Cancel");
}];

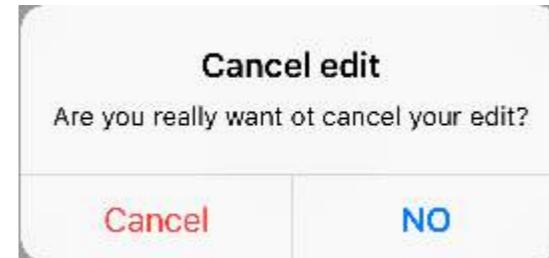
UIAlertAction *no = [UIAlertAction actionWithTitle:@"NO" style:UIAlertActionStyleDefault
handler:^(UIAlertAction * action) {
    NSLog(@"Highlighted button is pressed.");
}];

[alertController addAction:cancel];
[alertController addAction:no];

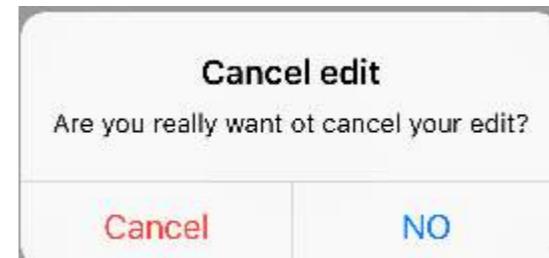
//add no action to preffered action.
//Note
//the action should already be added to alert controller
alertController.preferredAction = no;

[self presentViewController:alertController animated: YES completion: nil];
```

Alert Controller with **preferred action set**.The NO button is highlighted.

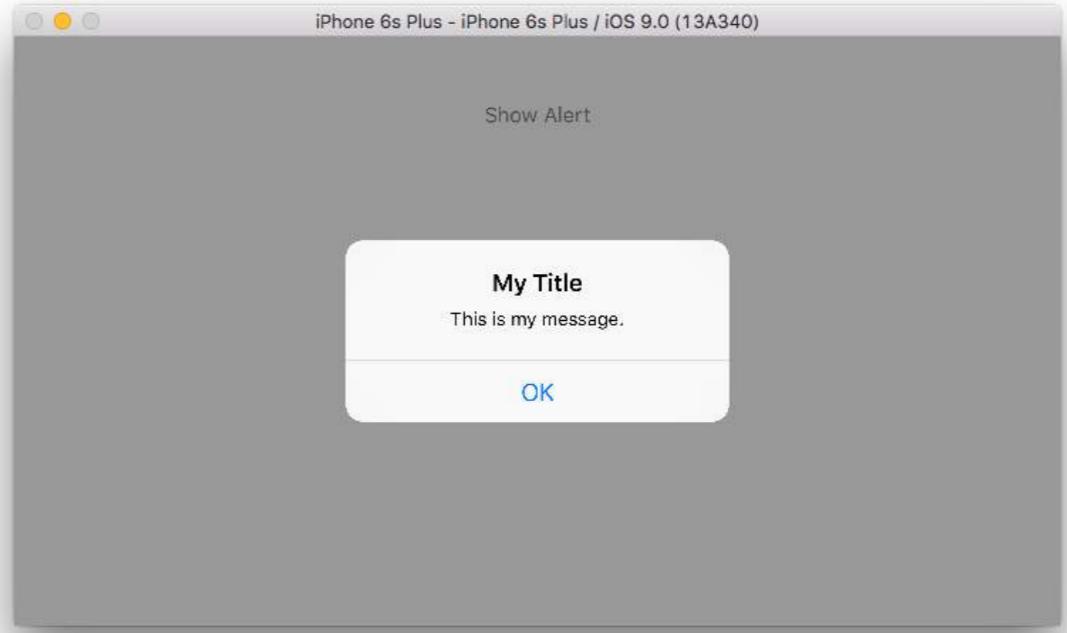
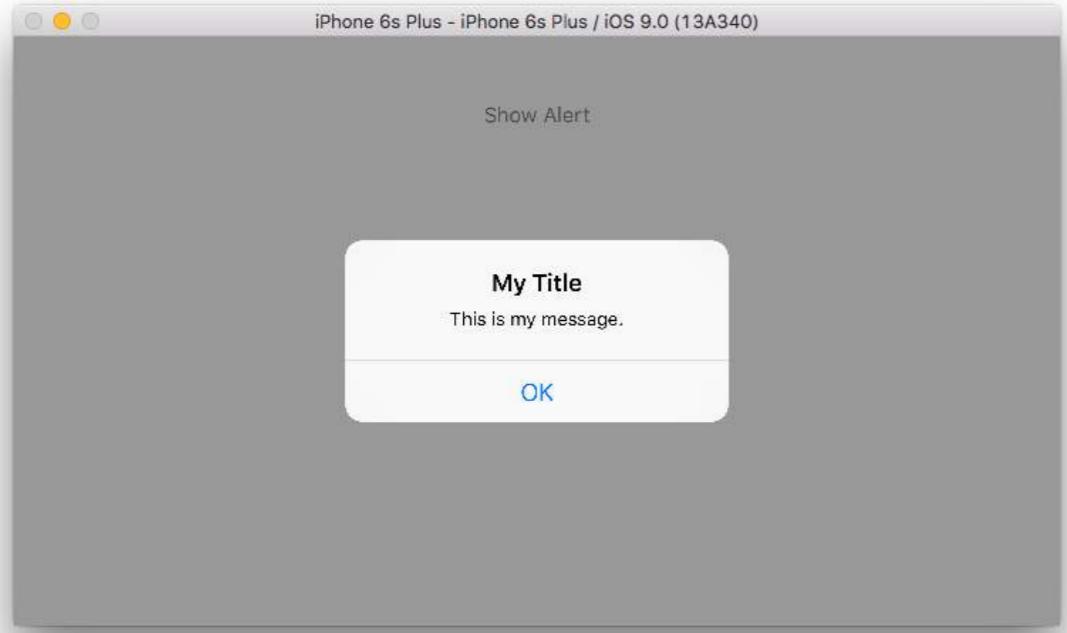


Alert Controller with **preferred action not set**.The NO button is not highlighted.



## Section 22.5: Displaying and handling alerts

One Button



## Swift

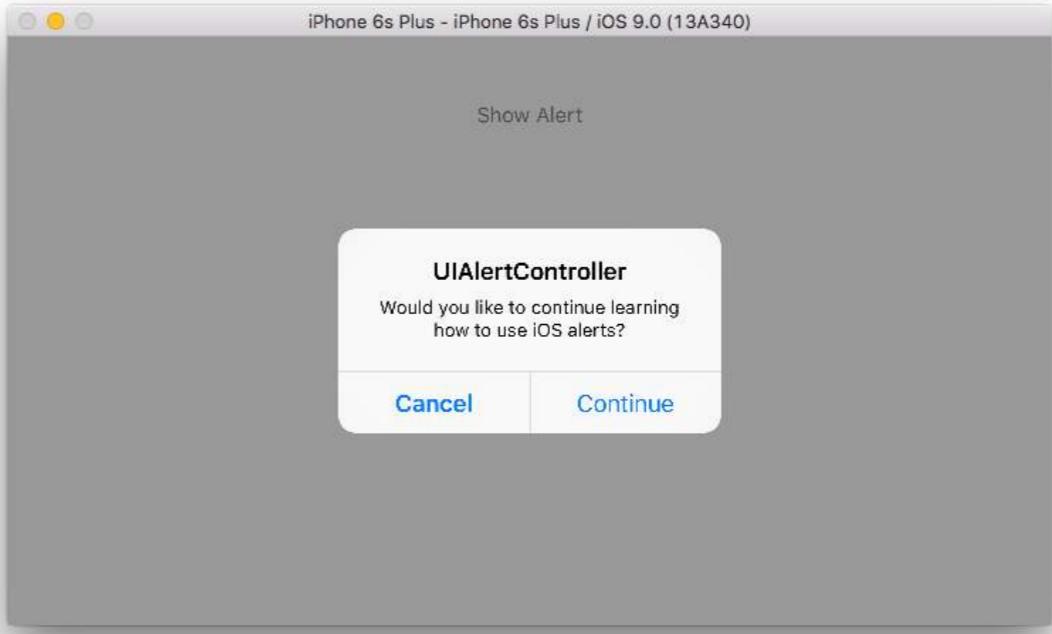
```
class ViewController: UIViewController {  
  
    @IBAction func showAlertButtonTapped(sender: UIButton) {  
  
        // 创建警报  
        let alert = UIAlertController(title: "我的标题", message: "这是我的信息。",  
        preferredStyle: UIAlertControllerStyle.Alert)  
  
        // 添加一个操作 (按钮)  
        alert.addAction(UIAlertAction(title: "确定", style: UIAlertActionStyle.Default, handler:  
        nil))  
  
        // 显示警告  
        self.presentViewController(alert, animated: true, completion: nil)  
    }  
}
```

## 两个按钮

## Swift

```
class ViewController: UIViewController {  
  
    @IBAction func showAlertButtonTapped(sender: UIButton) {  
  
        // create the alert  
        let alert = UIAlertController(title: "My Title", message: "This is my message.",  
        preferredStyle: UIAlertControllerStyle.Alert)  
  
        // add an action (button)  
        alert.addAction(UIAlertAction(title: "OK", style: UIAlertActionStyle.Default, handler:  
        nil))  
  
        // show the alert  
        self.presentViewController(alert, animated: true, completion: nil)  
    }  
}
```

## Two Buttons



## Swift

```
class ViewController: UIViewController {

    @IBAction func showAlertButtonTapped(sender: UIButton) {

        // 创建警报
        let alert = UIAlertController(title: "UIAlertController", message: "您想继续学习如何使用iOS警告吗?", preferredStyle: UIAlertControllerStyle.Alert)

        // 添加操作 (按钮)
        alert.addAction(UIAlertAction(title: "继续", style: UIAlertActionStyle.Default,
                                     handler: nil))
        alert.addAction(UIAlertAction(title: "取消", style: UIAlertActionStyle.Cancel, handler:
                                     nil))

        // 显示警告
        self.presentViewController(alert, animated: true, completion: nil)
    }
}
```

## 三个按钮

## Swift

```
class ViewController: UIViewController {

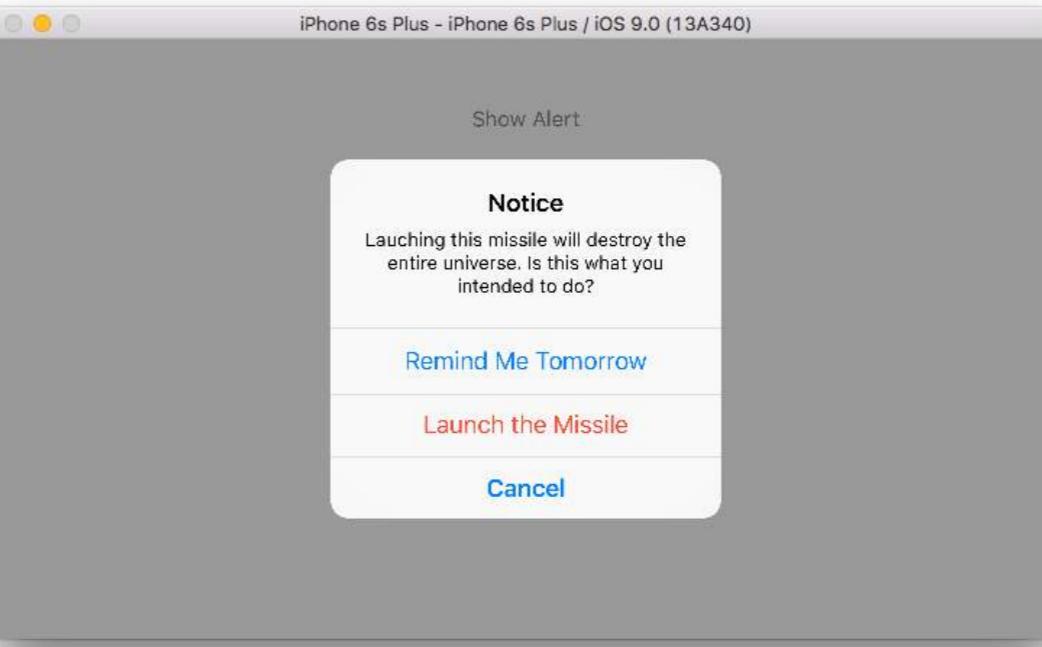
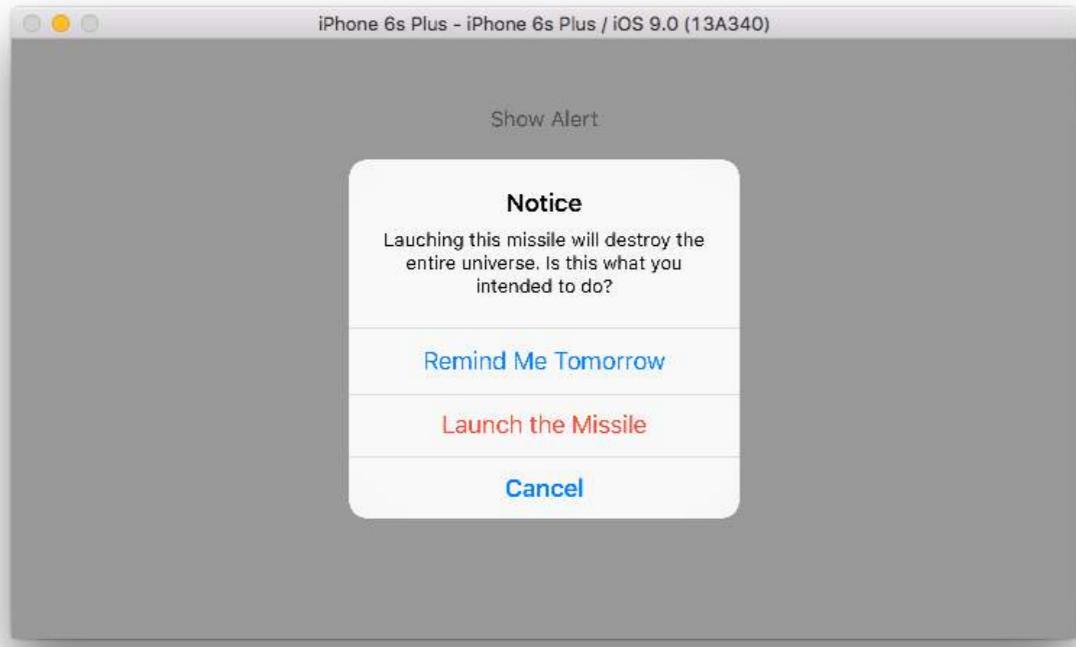
    @IBAction func showAlertButtonTapped(sender: UIButton) {

        // create the alert
        let alert = UIAlertController(title: "UIAlertController", message: "Would you like to continue learning how to use iOS alerts?", preferredStyle: UIAlertControllerStyle.Alert)

        // add the actions (buttons)
        alert.addAction(UIAlertAction(title: "Continue", style: UIAlertActionStyle.Default,
                                     handler: nil))
        alert.addAction(UIAlertAction(title: "Cancel", style: UIAlertActionStyle.Cancel, handler:
                                     nil))

        // show the alert
        self.presentViewController(alert, animated: true, completion: nil)
    }
}
```

## Three Buttons



## Swift

```
class ViewController: UIViewController {

    @IBAction func showAlertButtonTapped(sender: UIButton) {

        // 创建警报
        let alert = UIAlertController(title: "通知", message: "发射这枚导弹将摧毁整个宇宙。这是你想要做的吗?", preferredStyle: UIAlertControllerStyle.Alert)

        // 添加操作 (按钮)
        alert.addAction(UIAlertAction(title: "明天提醒我", style: UIAlertActionStyle.Default, handler: nil))
        alert.addAction(UIAlertAction(title: "取消", style: UIAlertActionStyle.Cancel, handler: nil))
        alert.addAction(UIAlertAction(title: "发射导弹", style: UIAlertActionStyle.Destructive, handler: nil))

        // 显示警告
        self.presentViewController(alert, animated: true, completion: nil)
    }
}
```

## 处理按钮点击

上面示例中的handler是 nil。你可以用一个closure替换 nil，在用户点击按钮时执行某些操作，如下面的示例：

## Swift

```
alert.addAction(UIAlertAction(title: "发射导弹", style: UIAlertActionStyle.Destructive,
handler: { action in

    // 做类似的事情...
    self.launchMissile()

}))
```

## Swift

```
class ViewController: UIViewController {

    @IBAction func showAlertButtonTapped(sender: UIButton) {

        // create the alert
        let alert = UIAlertController(title: "Notice", message: "Launching this missile will destroy the entire universe. Is this what you intended to do?", preferredStyle: UIAlertControllerStyle.Alert)

        // add the actions (buttons)
        alert.addAction(UIAlertAction(title: "Remind Me Tomorrow", style: UIAlertActionStyle.Default, handler: nil))
        alert.addAction(UIAlertAction(title: "Cancel", style: UIAlertActionStyle.Cancel, handler: nil))
        alert.addAction(UIAlertAction(title: "Launch the Missile", style: UIAlertActionStyle.Destructive, handler: nil))

        // show the alert
        self.presentViewController(alert, animated: true, completion: nil)
    }
}
```

## Handling Button Taps

The handler was nil in the above examples. You can replace nil with a closure to do something when the user taps a button, like the example below:

## Swift

```
alert.addAction(UIAlertAction(title: "Launch the Missile", style: UIAlertActionStyle.Destructive,
handler: { action in

    // do something like...
    self.launchMissile()

}))
```

## 注意事项

- 多个按钮不一定需要使用不同的UIAlertCellStyle类型。它们都可以是 `.Default`.
- 对于超过三个按钮的情况，考虑使用操作表。设置非常相似。这是一个示例。

## Notes

- Multiple buttons do not necessarily need to use different UIAlertController types. They could all be `.Default`.
- For more than three buttons consider using an Action Sheet. The setup is very similar. [Here is an example.](#)

# 第23章：UIColor

## 第23.1节：创建UIColor

有多种方法可以创建一个UIColor：

### Swift

- 使用预定义颜色之一：

```
let redColor = UIColor.redColor()
let blueColor: UIColor = .blueColor()

// 在 Swift 3 中，去掉了 "Color()" 后缀：
let redColor = UIColor.red
let blueColor: UIColor = .blue
```

如果编译器已经知道变量是 UIColor 的实例，则可以完全省略类型：

```
let view = UIView()
view.backgroundColor = .yellowColor()
```

- 使用灰度值和透明度：

```
let grayscaleColor = UIColor(white: 0.5, alpha: 1.0)
```

- 使用色调、饱和度、亮度和透明度：

```
let hsbColor = UIColor(
    hue: 0.4,
    saturation: 0.3,
    brightness: 0.7,
    alpha: 1.0
)
```

- 使用RGBA值：

```
let rgbColor = UIColor(
    red: 30.0 / 255,
    green: 70.0 / 255,
    blue: 200.0 / 255,
    alpha: 1.0
)
```

- 使用图案图像：

```
let patternColor = UIColor(patternImage: UIImage(named: "myImage")!)
```

### Objective-C

- 使用预定义颜色之一：

```
UIColor *redColor = [UIColor redColor];
```

# Chapter 23: UIColor

## Section 23.1: Creating a UIColor

There are many ways you can create a `UIColor`:

### Swift

- Using one of the predefined colors:

```
let redColor = UIColor.redColor()
let blueColor: UIColor = .blueColor()

// In Swift 3, the "Color()" suffix is removed:
let redColor = UIColor.red
let blueColor: UIColor = .blue
```

If the compiler already knows that the variable is an instance of `UIColor` you can skip the type all together:

```
let view = UIView()
view.backgroundColor = .yellowColor()
```

- Using the grayscale value and the alpha:

```
let grayscaleColor = UIColor(white: 0.5, alpha: 1.0)
```

- Using hue, saturation, brightness and alpha:

```
let hsbColor = UIColor(
    hue: 0.4,
    saturation: 0.3,
    brightness: 0.7,
    alpha: 1.0
)
```

- Using the RGBA values:

```
let rgbColor = UIColor(
    red: 30.0 / 255,
    green: 70.0 / 255,
    blue: 200.0 / 255,
    alpha: 1.0
)
```

- Using a pattern image:

```
let patternColor = UIColor(patternImage: UIImage(named: "myImage")!)
```

### Objective-C

- Using one of the predefined colors:

```
UIColor *redColor = [UIColor redColor];
```

- 使用灰度值和透明度：

```
UIColor *grayscaleColor = [UIColor colorWithWhite: 0.5 alpha: 1.0];
```

- 使用色调、饱和度、亮度和透明度：

```
UIColor *hsbColor = [UIColor
    colorWithHue: 0.4
    saturation: 0.3
    brightness: 0.7
    alpha: 1.0
];
```

- 使用RGBA值：

```
UIColor *rgbColor = [UIColor
    colorWithRed: 30.0 / 255.0
    green: 70.0 / 255.0
    blue: 200.0 / 255.0
    alpha: 1.0
];
```

- 使用图案图像：

```
UIColor *patternColor = [UIColor colorWithPatternImage:[UIImage imageNamed:@"myImage.png"]];
```

- Using the grayscale value and the alpha:

```
UIColor *grayscaleColor = [UIColor colorWithWhite: 0.5 alpha: 1.0];
```

- Using hue, saturation, brightness and alpha:

```
UIColor *hsbColor = [UIColor
    colorWithHue: 0.4
    saturation: 0.3
    brightness: 0.7
    alpha: 1.0
];
```

- Using the RGBA values:

```
UIColor *rgbColor = [UIColor
    colorWithRed: 30.0 / 255.0
    green: 70.0 / 255.0
    blue: 200.0 / 255.0
    alpha: 1.0
];
```

- Using a pattern image:

```
UIColor *patternColor = [UIColor colorWithPatternImage:[UIImage imageNamed:@"myImage.png"]];
```

## 第23.2节：从十六进制数字或字符串创建 UIColor

你可以从十六进制数字或字符串创建一个UIColor，例如 0xff00cc, "#FFFFFF"

### Swift

#### 整数值

```
extension UIColor {
    convenience init(hex: Int, alpha: CGFloat = 1.0) {
        let r = CGFloat((hex >> 16) & 0xff) / 255
        let g = CGFloat((hex >> 08) & 0xff) / 255
        let b = CGFloat((hex >> 00) & 0xff) / 255
        self.init(red: r, green: g, blue: b, alpha: alpha)
    }
}
```

#### 示例：

```
let color = UIColor(hex: 0xff00cc, alpha: 1.0)
```

注意alpha的默认值为1.0，因此可以如下使用：

```
let color = UIColor(hex: 0xff00cc)
```

#### 字符串值

## Section 23.2: Creating a UIColor from hexadecimal number or string

You can create a `UIColor` from a hexadecimal number or string, e.g. 0xff00cc, "#FFFFFF"

### Swift

#### Int Value

```
extension UIColor {
    convenience init(hex: Int, alpha: CGFloat = 1.0) {
        let r = CGFloat((hex >> 16) & 0xff) / 255
        let g = CGFloat((hex >> 08) & 0xff) / 255
        let b = CGFloat((hex >> 00) & 0xff) / 255
        self.init(red: r, green: g, blue: b, alpha: alpha)
    }
}
```

#### Example:

```
let color = UIColor(hex: 0xff00cc, alpha: 1.0)
```

Note that for alpha the default value of 1.0 is provided, so it can be used as follows:

```
let color = UIColor(hex: 0xff00cc)
```

#### String Value

```

extension UIColor {
    // 便利初始化(hexCode: String) {
    let hex =
hexCode.stringByTrimmingCharactersInSet(NSCharacterSet.alphanumericCharacterSet().invertedSet)
        var int = UInt32()
        NSScanner(string: hex).scanHexInt(&int)
        let a, r, g, b: UInt32

        switch hex.characters.count {
        case 3:
            (a, r, g, b) = (255, (int >> 8) * 17, (int >> 4 & 0xF) * 17, (int & 0xF) * 17)
        case 6:
            (a, r, g, b) = (255, int >> 16, int >> 8 & 0xFF, int & 0xFF)
        case 8:
            (a, r, g, b) = (int >> 24, int >> 16 & 0xFF, int >> 8 & 0xFF, int & 0xFF)
        default:
            (a, r, g, b) = (1, 1, 1, 0)
        }

        self.init(red: CGFloat(r) / 255, green: CGFloat(g) / 255, blue: CGFloat(b) / 255, alpha:
CGFloat(a) / 255)
    }
}

```

示例用法：

带透明度的十六进制颜色

```
let color = UIColor("#80FFFFFF")
```

不带透明度的十六进制颜色 (color 的 alpha 值将等于 1.0)

```
let color = UIColor("#FFFFFF")
let color = UIColor("#FFF")
```

## Objective-C

整数值

```

@interface UIColor (Hex)
+ (UIColor *)colorWithHex:(NSUInteger)hex alpha:(CGFloat)alpha;
@end

@implementation UIColor (Hex)
+ (UIColor *)colorWithHex:(NSUInteger)hex alpha:(CGFloat)alpha {
    return [UIColor colorWithRed:((CGFloat)((hex & 0xFF0000) >> 16))/255.0
                    green:((CGFloat)((hex & 0xFF00) >> 8))/255.0
                     blue:((CGFloat)(hex & 0xFF))/255.0
                    alpha:alpha];
}
@end

```

示例：

```
UIColor *color = [UIColor colorWithHex:0xff00cc alpha:1.0];
```

字符串值

```

extension UIColor {
    convenience init(hexCode: String) {
        let hex =
hexCode.stringByTrimmingCharactersInSet(NSCharacterSet.alphanumericCharacterSet().invertedSet)
        var int = UInt32()
        NSScanner(string: hex).scanHexInt(&int)
        let a, r, g, b: UInt32

        switch hex.characters.count {
        case 3:
            (a, r, g, b) = (255, (int >> 8) * 17, (int >> 4 & 0xF) * 17, (int & 0xF) * 17)
        case 6:
            (a, r, g, b) = (255, int >> 16, int >> 8 & 0xFF, int & 0xFF)
        case 8:
            (a, r, g, b) = (int >> 24, int >> 16 & 0xFF, int >> 8 & 0xFF, int & 0xFF)
        default:
            (a, r, g, b) = (1, 1, 1, 0)
        }

        self.init(red: CGFloat(r) / 255, green: CGFloat(g) / 255, blue: CGFloat(b) / 255, alpha:
CGFloat(a) / 255)
    }
}

```

Example Usage:

Hex with alpha

```
let color = UIColor("#80FFFFFF")
```

Hex with no alpha (color alpha will equal 1.0)

```
let color = UIColor("#FFFFFF")
let color = UIColor("#FFF")
```

## Objective-C

*Int Value*

```

@interface UIColor (Hex)
+ (UIColor *)colorWithHex:(NSUInteger)hex alpha:(CGFloat)alpha;
@end

@implementation UIColor (Hex)
+ (UIColor *)colorWithHex:(NSUInteger)hex alpha:(CGFloat)alpha {
    return [UIColor colorWithRed:((CGFloat)((hex & 0xFF0000) >> 16))/255.0
                    green:((CGFloat)((hex & 0xFF00) >> 8))/255.0
                     blue:((CGFloat)(hex & 0xFF))/255.0
                    alpha:alpha];
}
@end

```

Example:

```
UIColor *color = [UIColor colorWithHex:0xff00cc alpha:1.0];
```

*String Value*

```

- (UIColor*) hex:(NSString*)hexCode {

    NSString *noHashString = [hexCode stringByReplacingOccurrencesOfString:@"#" withString:@""];
    NSScanner *scanner = [NSScanner scannerWithString:noHashString];
    [scanner setCharactersToBeSkipped:[NSCharacterSet symbolCharacterSet]];

    unsigned hex;
    if (![scanner scanHexInt:&hex]) return nil;
    int a;
    int r;
    int g;
    int b;

    switch (noHashString.length) {
        case 3:
            a = 255;
            r = (hex >> 8) * 17;
            g = ((hex >> 4) & 0xF) * 17;
            b = ((hex >> 0) & 0xF) * 17;
            break;
        case 6:
            a = 255;
            r = (hex >> 16);
            g = (hex >> 8) & 0xFF;
            b = (hex) & 0xFF;
            break;
        case 8:
            a = (hex >> 24);
            r = (hex >> 16) & 0xFF;
            g = (hex >> 8) & 0xFF;
            b = (hex) & 0xFF;
            break;
        default:
            a = 255.0;
            r = 255.0;
            b = 255.0;
            g = 255.0;
            break;
    }

    return [UIColor colorWithRed:r / 255.0f green:g / 255.0f blue:b / 255.0f alpha:a / 255];
}

```

示例用法：

带透明度的十六进制颜色

```
UIColor* color = [self hex:@"#80FFFFFF"];
```

无透明度的十六进制 (color 透明度将等于1)

```
UIColor* color = [self hex:@"#FFFFFF"];
UIColor* color = [self hex:@"#FFF"];
```

## 第23.3节：带Alpha通道的颜色

您可以设置某个 `UIColor` 的不透明度，而无需使用 `init(red:_, green:_, blue:_, alpha:_)` 初始化器创建新的颜色。

```

- (UIColor*) hex:(NSString*)hexCode {

    NSString *noHashString = [hexCode stringByReplacingOccurrencesOfString:@"#" withString:@""];
    NSScanner *scanner = [NSScanner scannerWithString:noHashString];
    [scanner setCharactersToBeSkipped:[NSCharacterSet symbolCharacterSet]];

    unsigned hex;
    if (![scanner scanHexInt:&hex]) return nil;
    int a;
    int r;
    int g;
    int b;

    switch (noHashString.length) {
        case 3:
            a = 255;
            r = (hex >> 8) * 17;
            g = ((hex >> 4) & 0xF) * 17;
            b = ((hex >> 0) & 0xF) * 17;
            break;
        case 6:
            a = 255;
            r = (hex >> 16);
            g = (hex >> 8) & 0xFF;
            b = (hex) & 0xFF;
            break;
        case 8:
            a = (hex >> 24);
            r = (hex >> 16) & 0xFF;
            g = (hex >> 8) & 0xFF;
            b = (hex) & 0xFF;
            break;
        default:
            a = 255.0;
            r = 255.0;
            b = 255.0;
            g = 255.0;
            break;
    }

    return [UIColor colorWithRed:r / 255.0f green:g / 255.0f blue:b / 255.0f alpha:a / 255];
}

```

Example usage:

Hex with alpha

```
UIColor* color = [self hex:@"#80FFFFFF"];
```

Hex with no alpha (color alpha will equal 1)

```
UIColor* color = [self hex:@"#FFFFFF"];
UIColor* color = [self hex:@"#FFF"];
```

## Section 23.3: Color with Alpha component

You can set the opacity to a certain `UIColor` without creating a new one using the `init(red:_, green:_, blue:_, alpha:_)` initializer.

## Swift

```
let colorWithAlpha = UIColor.redColor().colorWithAlphaComponent(0.1)
```

## Swift 3

```
//在最新版本的 Swift 中
_colorWithAlpha = UIColor.red.withAlphaComponent(0.1)
```

## Objective-C

```
UIColor * colorWithAlpha = [[UIColor redColor] colorWithAlphaComponent:0.1];
```

## 第23.4节：未公开的方法

在 `UIColor` 上有各种未公开的方法，提供了替代颜色或功能。这些方法可以在 `UIColor` 的私有头文件中找到。我将记录两个私有方法的用法，`styleString()` 和 `_systemDestructiveTintColor()`。

### styleString

自 iOS 2.0 起，`UIColor` 有一个私有实例方法叫做 `styleString`，它返回颜色的 RGB 或 RGBA 字符串表示，即使是像 `whiteColor` 这样超出 RGB 空间的颜色也适用。

Objective-C：

```
@interface UIColor (Private)
- (NSString *)styleString;
@end
// ...
[[UIColor whiteColor] styleString]; // rgb(255,255,255)
[[UIColor redColor] styleString]; // rgb(255,0,0)
[[UIColor lightTextColor] styleString]; // rgba(255,255,255,0.600000)
```

在 Swift 中，你可以使用桥接头文件来暴露接口。使用纯 Swift 时，你需要创建一个带有私有方法的 `@objc` 协议，并使用 `unsafeBitCast` 将 `UIColor` 转换为该协议：

```
@objc protocol UIColorPrivate {
    func styleString() -> String
}

let white = UIColor.whiteColor()
let red = UIColor.redColor()
let lightTextColor = UIColor.lightTextColor()

let whitePrivate = unsafeBitCast(white, UIColorPrivate.self)
let redPrivate = unsafeBitCast(red, UIColorPrivate.self)
let lightTextColorPrivate = unsafeBitCast(lightTextColor, UIColorPrivate.self)

whitePrivate.styleString() // rgb(255,255,255)
redPrivate.styleString() // rgb(255,0,0)
lightTextColorPrivate.styleString() // rgba(255,255,255,0.600000)
```

`_systemDestructiveTintColor()`

## Swift

```
let colorWithAlpha = UIColor.redColor().colorWithAlphaComponent(0.1)
```

## Swift 3

```
//In Swift Latest Version
_colorWithAlpha = UIColor.red.withAlphaComponent(0.1)
```

## Objective-C

```
UIColor * colorWithAlpha = [[UIColor redColor] colorWithAlphaComponent:0.1];
```

## Section 23.4: Undocumented Methods

There are a variety of undocumented methods on `UIColor` which expose alternate colors or functionality. These can be found in the [UIColor private header file](#). I will document the use of two private methods, `styleString()` and `_systemDestructiveTintColor()`.

### styleString

Since iOS 2.0 there is a private instance method on `UIColor` called `styleString` which returns an RGB or RGBA string representation of the color, even for colors like `whiteColor` outside the RGB space.

Objective-C:

```
@interface UIColor (Private)
- (NSString *)styleString;
@end
// ...
[[UIColor whiteColor] styleString]; // rgb(255,255,255)
[[UIColor redColor] styleString]; // rgb(255,0,0)
[[UIColor lightTextColor] styleString]; // rgba(255,255,255,0.600000)
```

In Swift you could use a bridging header to expose the interface. With pure Swift, you will need to create an `@objc` protocol with the private method, and `unsafeBitCast` `UIColor` with the protocol:

```
@objc protocol UIColorPrivate {
    func styleString() -> String
}

let white = UIColor.whiteColor()
let red = UIColor.redColor()
let lightTextColor = UIColor.lightTextColor()

let whitePrivate = unsafeBitCast(white, UIColorPrivate.self)
let redPrivate = unsafeBitCast(red, UIColorPrivate.self)
let lightTextColorPrivate = unsafeBitCast(lightTextColor, UIColorPrivate.self)

whitePrivate.styleString() // rgb(255,255,255)
redPrivate.styleString() // rgb(255,0,0)
lightTextColorPrivate.styleString() // rgba(255,255,255,0.600000)
```

`_systemDestructiveTintColor()`

在UIColor上有一个未公开的类方法\_systemDestructiveTintColor，它会返回系统破坏性按钮使用的红色：

```
let red = UIColor.performSelector("_systemDestructiveTintColor").takeUnretainedValue()
```

它返回一个未管理的对象，必须调用.takeUnretainedValue()，因为颜色的所有权并未转移到我们自己的对象。

和使用任何未公开的API一样，使用此方法时应谨慎：

```
if UIColor.respondsToSelector("_systemDestructiveTintColor") {  
    if let red = UIColor.performSelector("_systemDestructiveTintColor").takeUnretainedValue() as?  
        UIColor {  
            // 使用该颜色  
        }  
}
```

或者通过使用协议：

```
@objc protocol UIColorPrivateStatic {  
    func _systemDestructiveTintColor() -> UIColor  
}  
  
let privateClass = UIColor.self as! UIColorPrivateStatic  
privateClass._systemDestructiveTintColor() // UIDeviceRGBColorSpace 1 0.231373 0.188235 1
```

## 第23.5节：基于图像模式的UIColor

您可以使用UIColor(patternImage:\_)方法通过图像模式创建一个UIColor对象。

```
btn.backgroundColor = UIColor(patternImage: UIImage(named: "image")!)
```

There is an undocumented class method on `UIColor` called `_systemDestructiveTintColor` which will return the red color used by destructive system buttons:

```
let red = UIColor.performSelector("_systemDestructiveTintColor").takeUnretainedValue()
```

It returns an unmanaged object, which you must call `.takeUnretainedValue()` on, since the color ownership has not been transferred to our own object.

As with any undocumented API, you should take caution when trying to use this method:

```
if UIColor.respondsToSelector("_systemDestructiveTintColor") {  
    if let red = UIColor.performSelector("_systemDestructiveTintColor").takeUnretainedValue() as?  
        UIColor {  
            // use the color  
        }  
}
```

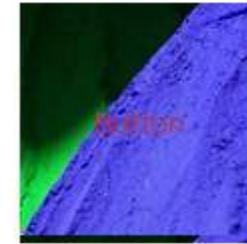
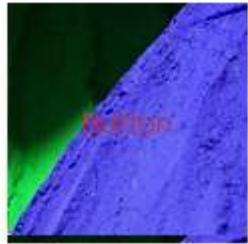
or by using a protocol:

```
@objc protocol UIColorPrivateStatic {  
    func _systemDestructiveTintColor() -> UIColor  
}  
  
let privateClass = UIColor.self as! UIColorPrivateStatic  
privateClass._systemDestructiveTintColor() // UIDeviceRGBColorSpace 1 0.231373 0.188235 1
```

## Section 23.5: UIColor from an image pattern

You can create a `UIColor` object using an image pattern by using the `UIColor(patternImage:_)` method.

```
btn.backgroundColor = UIColor(patternImage: UIImage(named: "image")!)
```



## 第23.6节：给定UIColor的更浅和更深色调

下面的代码示例演示了如何获取给定颜色的更浅和更深色调，这在具有动态主题的应用中非常有用

### 对于更深颜色

```
+ (UIColor *)darkerColorForColor:(UIColor *)c
{
    CGFloat r, g, b, a;
    if ([c getRed:&r green:&g blue:&b alpha:&a])
        return [UIColor colorWithRed:MAX(r - 0.2, 0.0)
                           green:MAX(g - 0.2, 0.0)
                           blue:MAX(b - 0.2, 0.0)
                           alpha:a];
    return nil;
}
```

### 对于较浅颜色

```
+ (UIColor *)lighterColorForColor:(UIColor *)c
{
    CGFloat r, g, b, a;
    if ([c getRed:&r green:&g blue:&b alpha:&a])
        return [UIColor colorWithRed:MIN(r + 0.2, 1.0)
                           green:MIN(g + 0.2, 1.0)
                           blue:MIN(b + 0.2, 1.0)
                           alpha:a];
    return nil;
}
```

## Section 23.6: Lighter and Darker Shade of a given UIColor

The code example below demonstrate how you can get a lighter and darker shade of a given color, useful in applications having dynamic themes

### For Darker Color

```
+ (UIColor *)darkerColorForColor:(UIColor *)c
{
    CGFloat r, g, b, a;
    if ([c getRed:&r green:&g blue:&b alpha:&a])
        return [UIColor colorWithRed:MAX(r - 0.2, 0.0)
                           green:MAX(g - 0.2, 0.0)
                           blue:MAX(b - 0.2, 0.0)
                           alpha:a];
    return nil;
}
```

### For Lighter Color

```
+ (UIColor *)lighterColorForColor:(UIColor *)c
{
    CGFloat r, g, b, a;
    if ([c getRed:&r green:&g blue:&b alpha:&a])
        return [UIColor colorWithRed:MIN(r + 0.2, 1.0)
                           green:MIN(g + 0.2, 1.0)
                           blue:MIN(b + 0.2, 1.0)
                           alpha:a];
    return nil;
}
```

```

    返回 [UIColor colorWithRed:MIN(r + 0.2, 1.0)
        green:MIN(g + 0.2, 1.0)
        blue:MIN(b + 0.2, 1.0)
        alpha:a];
    return nil;
}

```

请参见下面的视觉差异，考虑给定颜色为 [UIColor orangeColor]



```

    return [UIColor colorWithRed:MIN(r + 0.2, 1.0)
        green:MIN(g + 0.2, 1.0)
        blue:MIN(b + 0.2, 1.0)
        alpha:a];
    return nil;
}

```

See Visual differences below, considering given color is [UIColor orangeColor]



## 第23.7节：使用户自定义属性支持CGColor data类型

默认情况下，Interface Builder不支持CGColor数据类型，因此为了允许在Interface Builder中通过用户自定义属性添加CGColor，可以使用如下扩展：

Swift扩展：

```

extension CALayer {
    func borderUIColor() -> UIColor? {
        return borderColor != nil ? UIColor(CGColor: borderColor!) : nil
    }

    func setBorderUIColor(color: UIColor) {
        borderColor = color.CGColor
    }
}

```

新的用户自定义属性（borderUIColor）将被识别并正常应用。

User Defined Runtime Attributes		
Key Path	Type	Value
layer.cornerRadius	Number	6.5
layer.borderWidth	Number	1
layer.clipsToBounds	Boolean	<input checked="" type="checkbox"/>
layer.borderUIColor	Color	<input type="color"/>

## Section 23.7: Make user defined attributes apply the CGColor datatype

By default, Interface Builder doesn't accept the `CGColor` datatype, so to allow adding a `CGColor` using user defined attributes in interface builder; one may want to use an extension like this:

Swift Extension :

```

extension CALayer {
    func borderUIColor() -> UIColor? {
        return borderColor != nil ? UIColor(CGColor: borderColor!) : nil
    }

    func setBorderUIColor(color: UIColor) {
        borderColor = color.CGColor
    }
}

```

The new user defined attribute (borderUIColor) will be recognized and applied without problems.

User Defined Runtime Attributes		
Key Path	Type	Value
layer.cornerRadius	Number	6.5
layer.borderWidth	Number	1
layer.clipsToBounds	Boolean	<input checked="" type="checkbox"/>
layer.borderUIColor	Color	<input type="color"/>

# 第24章：UITextView

## 第24.1节：设置富文本

```
// 修改富文本字符串的一些属性。  
let attributedText = NSMutableAttributedString(attributedString: textView.attributedText!)  
  
// 使用NSString，这样rangeOfString的结果是NSRange。  
let text = textView.text! as NSString  
  
// 查找每个要修改元素的范围。  
let tintedRange = text.range(of: NSLocalizedString("tinted", comment: ""))  
let highlightedRange = text.range(of: NSLocalizedString("highlighted", comment: ""))  
  
// 添加色调。  
attributedText.addAttribute(NSForegroundColorAttributeName, value: UIColor.blue, range:  
tintedRange)  
  
// 添加高亮。  
attributedText.addAttribute(NSBackgroundColorAttributeName, value: UIColor.yellow, range:  
highlightedRange)  
  
textView.attributedText = attributedText
```

## 第24.2节：更改字体

### Swift

```
//系统字体  
textView.font = UIFont.systemFont(ofSize: 12)  
  
//自定义字体  
textView.font = UIFont(name: "Font Name", size: 12)
```

### Objective-C

```
//系统字体  
textView.font = [UIFont systemFontOfSize:12];  
  
//自定义字体  
textView.font = [UIFont fontWithName:@"Font Name" size:12];
```

## 第24.3节：自动检测链接、地址、日期等

UITextView内置支持自动检测多种数据。目前能够自动检测的数据包括：

```
枚举 {  
    UIDataDetectorTypePhoneNumber = 1 << 0,  
    UIDataDetectorTypeLink = 1 << 1,  
    UIDataDetectorTypeAddress = 1 << 2,  
    UIDataDetectorTypeCalendarEvent = 1 << 3,  
    UIDataDetectorTypeNone = 0,  
    UIDataDetectorTypeAll = NSUIntegerMax  
};
```

启用自动检测

# Chapter 24: UITextView

## Section 24.1: Set attributed text

```
// Modify some of the attributes of the attributed string.  
let attributedText = NSMutableAttributedString(attributedString: textView.attributedText!)  
  
// Use NSString so the result of rangeOfString is an NSRange.  
let text = textView.text! as NSString  
  
// Find the range of each element to modify.  
let tintedRange = text.range(of: NSLocalizedString("tinted", comment: ""))  
let highlightedRange = text.range(of: NSLocalizedString("highlighted", comment: ""))  
  
// Add tint.  
attributedText.addAttribute(NSForegroundColorAttributeName, value: UIColor.blue, range:  
tintedRange)  
  
// Add highlight.  
attributedText.addAttribute(NSBackgroundColorAttributeName, value: UIColor.yellow, range:  
highlightedRange)  
  
textView.attributedText = attributedText
```

## Section 24.2: Change font

### Swift

```
//System Font  
textView.font = UIFont.systemFont(ofSize: 12)  
  
//Font of your choosing  
textView.font = UIFont(name: "Font Name", size: 12)
```

### Objective-C

```
//System Font  
textView.font = [UIFont systemFontOfSize:12];  
  
//Font of your choosing  
textView.font = [UIFont fontWithName:@"Font Name" size:12];
```

## Section 24.3: Auto Detect Links, Addresses, Dates, and more

UITextView has built in support to auto detect a variety of data. The data that is able to be auto-detected currently includes:

```
enum {  
    UIDataDetectorTypePhoneNumber = 1 << 0,  
    UIDataDetectorTypeLink = 1 << 1,  
    UIDataDetectorTypeAddress = 1 << 2,  
    UIDataDetectorTypeCalendarEvent = 1 << 3,  
    UIDataDetectorTypeNone = 0,  
    UIDataDetectorTypeAll = NSUIntegerMax  
};
```

Enabling auto-detection

```
// 你可以通过在选项之间使用 `|` 运算符添加任意多个  
textView.dataDetectorTypes = (UIDataDetectorTypeLink | UIDataDetectorTypePhoneNumber);
```

如果启用，文本将在UITextView中显示为超链接

#### 可点击的数据

为了允许点击链接（根据数据类型执行不同操作），你必须确保UITextView是可选择的但不可编辑，并且启用了用户交互

```
textView.editable = NO;  
textView.selectable = YES;  
textView.userInteractionEnabled = YES; // 默认是 YES
```

## 第24.4节：更改文本

### Swift

```
textView.text = "Hello, world!"
```

### Objective-C：

```
textView.text = @"Hello, world!";
```

## 第24.5节：更改文本对齐方式

### Swift

```
textView.textAlignment = .left
```

### Objective-C

```
textView.textAlignment = NSTextAlignmentLeft;
```

## 第24.6节：UITextViewDelegate方法

### 响应编辑通知

- textViewShouldBeginEditing(\_:)
- textViewDidBeginEditing(\_:)
- textViewShouldEndEditing(\_:)
- textViewDidEndEditing(\_:)

### 响应文本变化

- textView(\_:shouldChangeTextIn:replacementText:)
- textViewDidChange(\_:)

### 响应 URL

- textView(\_: UITextView, shouldInteractWithURL: NSURL, inRange: NSRange) -> Bool

```
// you may add as many as you like by using the `|` operator between options  
textView.dataDetectorTypes = (UIDataDetectorTypeLink | UIDataDetectorTypePhoneNumber);
```

If enabled, the text will appear as a hyperlink on the UITextView

#### Clickable data

To allow the link to be clicked (which will result in different actions depending on the data type) you must ensure that the UITextView is selectable but not editable and that user interaction is enabled

```
textView.editable = NO;  
textView.selectable = YES;  
textView.userInteractionEnabled = YES; // YES by default
```

## Section 24.4: Change text

### Swift

```
textView.text = "Hello, world!"
```

### Objective-C:

```
textView.text = @"Hello, world!";
```

## Section 24.5: Change text alignment

### Swift

```
textView.textAlignment = .left
```

### Objective-C

```
textView.textAlignment = NSTextAlignmentLeft;
```

## Section 24.6: UITextViewDelegate methods

### Responding to Editing Notifications

- textViewShouldBeginEditing(\_:)
- textViewDidBeginEditing(\_:)
- textViewShouldEndEditing(\_:)
- textViewDidEndEditing(\_:)

### Responding to Text Changes

- textView(\_:shouldChangeTextIn:replacementText:)
- textViewDidChange(\_:)

### Responding to URL

- textView(\_: UITextView, shouldInteractWithURL: NSURL, inRange: NSRange) -> Bool

## 第24.7节：更改文本颜色

### Swift

```
textView.textColor = UIColor.red
```

### Objective-C

```
textView.textColor = [UIColor redColor];
```

## 第24.8节：移除额外内边距以适应精确测量的文本

`UITextView` 默认有额外的内边距。有时这很烦人，尤其是当你想在没有视图实例的情况下测量一些文本并将它们精确放置在某个区域时。

这样做可以移除这些内边距。

```
messageTextView.textContainerInset = UIEdgeInsetsZero  
messageTextView.textContainer.lineFragmentPadding = 0
```

现在你可以使用`NSAttributedString`的`boundingRectWithSize(...)`方法来测量文本大小，并调整`UITextView`的大小以适应文本。

```
let budget = getSomeCGSizeBudget()  
let text = getSomeAttributedString()  
let textSize = text.boundingRectWithSize(budget, options: [.UsesLineFragmentOrigin,  
.UsesFontLeading], context: nil).size  
messageTextView.frame.size = textSize // 刚好合适。
```

## 第24.9节：获取和设置光标位置

### 有用信息

文本字段文本的最开始位置：

```
let startPosition: UITextPosition = textView.beginningOfDocument
```

文本字段文本的最末尾位置：

```
let endPosition: UITextPosition = textView.endOfDocument
```

当前选定的范围：

```
let selectedRange: UITextRange? = textView.selectedTextRange
```

### 获取光标位置

```
if let selectedRange = textView.selectedTextRange {  
  
    let cursorPosition = textView.offsetFromPosition(textView.beginningOfDocument, toPosition:  
selectedRange.start)  
  
    print("\(cursorPosition)")  
}
```

### 设置光标位置

## Section 24.7: Change text color

### Swift

```
textView.textColor = UIColor.red
```

### Objective-C

```
textView.textColor = [UIColor redColor];
```

## Section 24.8: Remove extra paddings to fit to a precisely measured text

`UITextView` has extra paddings by default. Sometimes it's annoying especially if you want to measure some text without view instance and place them at some area precisely.

Do this to remove such paddings.

```
messageTextView.textContainerInset = UIEdgeInsetsZero  
messageTextView.textContainer.lineFragmentPadding = 0
```

Now you can measure text size using `NSAttributedString.boundingRectWithSize(...)`, and resize a `UITextView` just to fit it to the text.

```
let budget = getSomeCGSizeBudget()  
let text = getSomeAttributedString()  
let textSize = text.boundingRectWithSize(budget, options: [.UsesLineFragmentOrigin,  
.UsesFontLeading], context: nil).size  
messageTextView.frame.size = textSize // Just fits.
```

## Section 24.9: Getting and Setting the Cursor Position

### Useful information

The very beginning of the text field text:

```
let startPosition: UITextPosition = textView.beginningOfDocument
```

The very end of the text field text:

```
let endPosition: UITextPosition = textView.endOfDocument
```

The currently selected range:

```
let selectedRange: UITextRange? = textView.selectedTextRange
```

### Get cursor position

```
if let selectedRange = textView.selectedTextRange {  
  
    let cursorPosition = textView.offsetFromPosition(textView.beginningOfDocument, toPosition:  
selectedRange.start)  
  
    print("\(cursorPosition)")  
}
```

### Set cursor position

为了设置位置，所有这些方法实际上都是设置一个起始和结束值相同的范围。

[回到顶部](#)

```
let newPosition = textView.beginningOfDocument  
textView.selectedTextRange = textView.textRangeFromPosition(newPosition, toPosition: newPosition)
```

**到结尾**

```
let newPosition = textView.endOfDocument  
textView.selectedTextRange = textView.textRangeFromPosition(newPosition, toPosition: newPosition)
```

**到当前光标位置左边一个位置**

```
// 仅当当前有选中范围时  
if let selectedRange = textView.selectedTextRange {  
  
    // 并且仅当新位置有效时  
    if let newPosition = textView.positionFromPosition(selectedRange.start, inDirection:  
UITextLayoutDirection.Left, offset: 1) {  
  
        // 设置新位置  
        textView.selectedTextRange = textView.textRangeFromPosition(newPosition, toPosition:  
newPosition)  
    }  
}
```

**到任意位置**

从开头开始，向右移动5个字符。

```
let arbitraryValue: Int = 5  
if let newPosition = textView.positionFromPosition(textView.beginningOfDocument, inDirection:  
UITextLayoutDirection.Right, offset: arbitraryValue) {  
  
    textView.selectedTextRange = textView.textRangeFromPosition(newPosition, toPosition:  
newPosition)  
}
```

**相关**

**全选文本**

```
textView.selectedTextRange = textView.textRangeFromPosition(textView.beginningOfDocument,  
toPosition: textView.endOfDocument)
```

**选择一段文本**

```
// 范围：3 到 7  
let startPosition = textView.positionFromPosition(textView.beginningOfDocument, inDirection:  
UITextLayoutDirection.Right, offset: 3)  
let endPosition = textView.positionFromPosition(textView.beginningOfDocument, inDirection:  
UITextLayoutDirection.Right, offset: 7)  
  
if startPosition != nil && endPosition != nil {  
    textView.selectedTextRange = textView.textRangeFromPosition(startPosition!, toPosition:  
endPosition!)  
}
```

In order to set the position, all of these methods are actually setting a range with the same start and end values.

**To the beginning**

```
let newPosition = textView.beginningOfDocument  
textView.selectedTextRange = textView.textRangeFromPosition(newPosition, toPosition: newPosition)
```

**To the end**

```
let newPosition = textView.endOfDocument  
textView.selectedTextRange = textView.textRangeFromPosition(newPosition, toPosition: newPosition)
```

**To one position to the left of the current cursor position**

```
// only if there is a currently selected range  
if let selectedRange = textView.selectedTextRange {  
  
    // and only if the new position is valid  
    if let newPosition = textView.positionFromPosition(selectedRange.start, inDirection:  
UITextLayoutDirection.Left, offset: 1) {  
  
        // set the new position  
        textView.selectedTextRange = textView.textRangeFromPosition(newPosition, toPosition:  
newPosition)  
    }  
}
```

**To an arbitrary position**

Start at the beginning and move 5 characters to the right.

```
let arbitraryValue: Int = 5  
if let newPosition = textView.positionFromPosition(textView.beginningOfDocument, inDirection:  
UITextLayoutDirection.Right, offset: arbitraryValue) {  
  
    textView.selectedTextRange = textView.textRangeFromPosition(newPosition, toPosition:  
newPosition)  
}
```

**Related**

**Select all text**

```
textView.selectedTextRange = textView.textRangeFromPosition(textView.beginningOfDocument,  
toPosition: textView.endOfDocument)
```

**Select a range of text**

```
// Range: 3 to 7  
let startPosition = textView.positionFromPosition(textView.beginningOfDocument, inDirection:  
UITextLayoutDirection.Right, offset: 3)  
let endPosition = textView.positionFromPosition(textView.beginningOfDocument, inDirection:  
UITextLayoutDirection.Right, offset: 7)  
  
if startPosition != nil && endPosition != nil {  
    textView.selectedTextRange = textView.textRangeFromPosition(startPosition!, toPosition:  
endPosition!)  
}
```

## 在当前光标位置插入文本

```
textView.insertText("Hello")
```

### 注意事项

- 此示例最初来源于对这个Stack Overflow答案的改编。
- 该答案使用了文本字段，但相同的概念也适用于UITextView。
- 使用textView.becomeFirstResponder()来使文本框获得焦点并弹出键盘。
- 参见[this answer](#)了解如何获取某个范围内的文本。

### 相关

- [如何在Swift中创建范围](#) (间接涉及为什么这里必须使用selectedTextRange而不是直接使用selectedRange的问题)

## 第24.10节：带有HTML文本的UITextView

```
NSString *htmlString = @"><p> 这是一个<b>HTML</b>文本</p>";
NSMutableAttributedString *attributedString = [[NSMutableAttributedString alloc]
                                             initWithData: [htmlString
dataUsingEncoding:NSUTFStringEncoding]
options: @{
NSDocumentTypeDocumentAttribute: NSHTMLTextDocumentType }
documentAttributes: nil
error: nil
];
_yourTextView.attributedText = attributedString;
// 如果你想修改字体
field.font = [UIFont fontWithName:@"Raleway-Regular" size:15];
```

## 第24.11节：检查是否为空或nil

### Swift

```
if let text = self.textView.text where !text.isEmpty {
    // 处理文本的操作
} else {
    // 处理文本为空或为nil的情况
}
```

### Objective-C

```
if (self.textView.text.length > 0){
    // 处理文本的操作
} else {
    // 处理文本为空或为nil的情况
}
```

## Insert text at the current cursor position

```
textView.insertText("Hello")
```

### Notes

- This example originally comes from an adaptation of [this Stack Overflow answer](#).
- This answer uses a text field, but the same concepts apply to UITextView.
- Use textView.becomeFirstResponder() to give focus to the text field and make the keyboard appear.
- See [this answer](#) for how to get the text at some range.

### Related

- [How to Create a Range in Swift](#) (Deals indirectly with the issue of why we have to use selectedTextRange here rather than just selectedRange)

## Section 24.10: UITextView with HTML text

```
NSString *htmlString = @"><p> This is an <b>HTML</b> text</p>";
NSMutableAttributedString *attributedString = [[NSMutableAttributedString alloc]
                                             initWithData: [htmlString
dataUsingEncoding:NSUTFStringEncoding]
options: @{
NSDocumentTypeDocumentAttribute: NSHTMLTextDocumentType }
documentAttributes: nil
error: nil
];
_yourTextView.attributedText = attributedString;
// If you want to modify the font
field.font = [UIFont fontWithName:@"Raleway-Regular" size:15];
```

## Section 24.11: Check to see if empty or nil

### Swift

```
if let text = self.textView.text where !text.isEmpty {
    // Do stuff for text
} else {
    // Do stuff for nil text or empty string
}
```

### Objective-C

```
if (self.textView.text.length > 0){
    // Do stuff for text
} else {
    // Do stuff for nil text or empty string
}
```

# 第25章：UITextField代理

## 第25.1节：用户开始/结束与文本字段交互时的操作

针对Swift 3.1：

在第一个示例中，可以看到如何拦截用户在输入时与文本字段的交互。

同样，UITextFieldDelegate中有一些方法会在用户开始和结束与文本字段的交互时被调用。

要访问这些方法，您需要遵守[UITextFieldDelegate协议](#)，并且对于每个

您想要接收通知的文本框，将父类设置为代理：

```
class SomeClass: UITextFieldDelegate {  
  
    @IBOutlet var textField: UITextField!  
  
    override func viewDidLoad() {  
        super.viewDidLoad()  
        textField.delegate = self  
    }  
}
```

现在您可以实现所有 UITextFieldDelegate 方法了。

要在用户开始编辑文本框时接收通知，您可以实现[textFieldDidBeginEditing\(\\_:\)方法](#)，

示例如下：

```
func textFieldDidBeginEditing(_ textField: UITextField) {  
    // 现在您可以执行某些操作  
    // 如果类中有多个文本框，  
    // 您可以在这里进行比较，以分别处理每个文本框  
    if textField == emailTextField {  
        // 例如，验证邮箱  
    }  
    else if textField == passwordTextField {  
        // 例如，验证密码  
    }  
}
```

同样，如果需要在用户结束与文本字段的交互时收到通知，可以使用[textFieldDidEndEditing\(\\_:\)方法](#)，示例如下：

```
func textFieldDidEndEditing(_ textField: UITextField) {  
    // 现在你可以执行一些操作  
    // 如果类中有多个文本框，  
    // 您可以在这里进行比较，以分别处理每个文本框  
    if textField == emailTextField {  
        // 例如，验证邮箱  
    }  
    else if textField == passwordTextField {  
        // 例如，验证密码  
    }  
}
```

# Chapter 25: UITextField Delegate

## Section 25.1: Actions when a user has started/ended interacting with a textfield

For Swift 3.1:

In the first example one can see how you would intercept the user interacting with a textfield while writing.

Similarly, there are methods in the [UITextFieldDelegate](#) that are called when a user has started and ended his interaction with a TextField.

To be able to access these methods, you need to conform to the [UITextFieldDelegate](#) protocol, and for each textfield you want to be notified about, assign the parent class as the delegate:

```
class SomeClass: UITextFieldDelegate {  
  
    @IBOutlet var textField: UITextField!  
  
    override func viewDidLoad() {  
        super.viewDidLoad()  
        textField.delegate = self  
    }  
}
```

Now you will be able to implement all the UITextFieldDelegate methods.

To be notified when a user has started editing a textfield, you can implement [textFieldDidBeginEditing\(\\_:\) method](#) like so:

```
func textFieldDidBeginEditing(_ textField: UITextField) {  
    // now you can perform some action  
    // if you have multiple textfields in a class,  
    // you can compare them here to handle each one separately  
    if textField == emailTextField {  
        // e.g. validate email  
    }  
    else if textField == passwordTextField {  
        // e.g. validate password  
    }  
}
```

Similarly, being notified if a user has ended interaction with a textfield, you can use the [textFieldDidEndEditing\(\\_:\) method](#) like so:

```
func textFieldDidEndEditing(_ textField: UITextField) {  
    // now you can perform some action  
    // if you have multiple textfields in a class,  
    // you can compare them here to handle each one separately  
    if textField == emailTextField {  
        // e.g. validate email  
    }  
    else if textField == passwordTextField {  
        // e.g. validate password  
    }  
}
```

如果你想控制一个文本框是否应该开始/结束编辑，可以使用`textFieldShouldBeginEditing(_)`和`textFieldShouldEndEditing(_)`方法，根据你的逻辑返回true/false。

## 第25.2节：UITextField - 限制文本框输入特定字符

如果你想对文本框的用户输入进行验证，可以使用以下代码片段：

```
// MARK: - UITextFieldDelegate

let allowedCharacters =
    CharacterSet(charactersIn:"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz").inverted

func textField(_ textField: UITextField, shouldChangeCharactersIn range: NSRange, replacementString string: String) -> Bool {
    let components = string.components(separatedBy: allowedCharacters)
    let filtered = components.joined(separator: "")

    if string == filtered {
        return true
    } else {
        return false
    }
}
```

### Objective-C

```
#define ACCEPTABLE_CHARACTERS @"0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"

- (BOOL)textField:(UITextField *)textField shouldChangeCharactersInRange:(NSRange)range
replacementString:(NSString *)string
{
    NSCharacterSet *cs = [[NSCharacterSet
characterSetWithCharactersInString:ACCEPTABLE_CHARACTERS] invertedSet];

    NSString *filtered = [[string componentsSeparatedByCharactersInSet:cs]
componentsJoinedByString:@""];
    return [string isEqualToString:filtered];
}
```

此外，你还可以使用苹果提供的字符集来进行验证：

请查看 <https://developer.apple.com/reference/foundation/nscharacterset>

```
let allowedCharacters = CharacterSet.alphanumerics.inverted
let allowedCharacters = CharacterSet.capitalizedLetters.inverted
```

If you want to have control over whether a TextField should begin/end editing, the `textFieldShouldBeginEditing(_)` and `textFieldShouldEndEditing(_)` methods can be used by return true/false based on your needed logic.

## Section 25.2: UITextField - Restrict textfield to certain characters

If you want to perform a user input validation of your textfield use the following code snippet:

```
// MARK: - UITextFieldDelegate

let allowedCharacters =
CharacterSet(charactersIn:"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz").inverted

func textField(_ textField: UITextField, shouldChangeCharactersIn range: NSRange, replacementString string: String) -> Bool {
    let components = string.components(separatedBy: allowedCharacters)
    let filtered = components.joined(separator: "")

    if string == filtered {
        return true
    } else {
        return false
    }
}
```

### Objective-C

```
#define ACCEPTABLE_CHARACTERS @"0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"

- (BOOL)textField:(UITextField *)textField shouldChangeCharactersInRange:(NSRange)range
replacementString:(NSString *)string
{
    NSCharacterSet *cs = [[NSCharacterSet
characterSetWithCharactersInString:ACCEPTABLE_CHARACTERS] invertedSet];

    NSString *filtered = [[string componentsSeparatedByCharactersInSet:cs]
componentsJoinedByString:@""];
    return [string isEqualToString:filtered];
}
```

In addition you can also use character sets provided by apple to perform validation:

Take a look at <https://developer.apple.com/reference/foundation/nscharacterset>

```
let allowedCharacters = CharacterSet.alphanumerics.inverted
let allowedCharacters = CharacterSet.capitalizedLetters.inverted
```

# 第26章：UINavigationController

## 第26.1节：以编程方式将视图控制器嵌入导航控制器

Swift

```
//Swift  
let viewController = UIViewController()  
let navigationController = UINavigationController(rootViewController: viewController)  
  
//Objective-C  
UIViewController *viewController = [[UIViewController alloc] init];  
UINavigationController *navigationController = [[UINavigationController alloc]  
initWithRootViewController:viewController];
```

## 第26.2节：导航控制器中的弹出操作

返回到上一个视图控制器

要弹回到上一页，可以这样做：

Swift

```
navigationController?.popViewControllerAnimated(true)
```

Objective-C

```
[self.navigationController popViewControllerAnimated:YES];
```

返回到根视图控制器

要弹回导航堆栈的根视图控制器，可以这样做：

Swift

```
navigationController?.popToRootViewControllerAnimated(true)
```

Objective C

```
[self.navigationController popToRootViewControllerAnimated:YES];
```

## 第26.3节：目的

`UINavigationController` 用于形成视图控制器的树状层级结构，这被称为 `navigation stack`（导航栈）。

从开发者的角度来看：

你可以连接独立制作的控制器，并免费获得一个自由的层级管理器和通用的UI呈现器。`UINavigationController` 会为你自动动画过渡到新的控制器，并提供返回功能。`UINavigationController` 还可以访问导航栈中的所有其他控制器，这有助于访问某些功能或数据。

# Chapter 26: UINavigationController

## Section 26.1: Embed a view controller in a navigation controller programmatically

Swift

```
//Swift  
let viewController = UIViewController()  
let navigationController = UINavigationController(rootViewController: viewController)  
  
//Objective-C  
UIViewController *viewController = [[UIViewController alloc] init];  
UINavigationController *navigationController = [[UINavigationController alloc]  
initWithRootViewController:viewController];
```

## Section 26.2: Popping in a Navigation Controller

To previous view controller

To pop back to the previous page you can do this:

Swift

```
navigationController?.popViewControllerAnimated(true)
```

Objective-C

```
[self.navigationController popViewControllerAnimated:YES];
```

To root view controller

To pop to the root of the navigation stack, you can do this:

Swift

```
navigationController?.popToRootViewControllerAnimated(true)
```

Objective C

```
[self.navigationController popToRootViewControllerAnimated:YES];
```

## Section 26.3: Purpose

`UINavigationController` is used to form a tree-like hierarchy of view controllers, which is known as a `navigation stack`.

From developers perspective:

You can connect independently made controller and get all the benefits of a free hierarchy manager and common UI presenter gratis. `UINavigationController` animates the transition to new controllers and provides the back functionality for you automatically. `UINavigationController` also gives access to all the other controllers in the `navigation stack` which can help access to some functionality or data.

从用户的角度来看：

`UINavigationController` 帮助记住用户当前所在的位置（导航栏标题）以及如何通过嵌入的返回按钮返回到之前的某个界面。

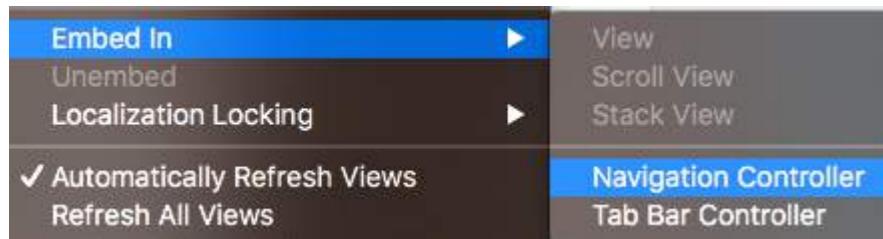
## 第26.4节：将视图控制器推入导航栈

```
//Swift  
let fooViewController = UIViewController()  
navigationController?.pushViewController(fooViewController, animated: true)  
  
//Objective-C  
UIViewController *fooViewController = [[UIViewController alloc] init];  
[navigationController pushViewController:fooViewController animated:YES];
```

## 第26.5节：创建导航控制器

在你的故事板中选择你想嵌入导航控制器的视图控制器。

然后导航到 编辑器 > 嵌入到 > 导航控制器



这样就会创建你的导航控制器



From user's perspective:

`UINavigationController` helps to remember where user is at the moment (navigation bar title) and how he can go back (embedded back button) to one of the previous screens.

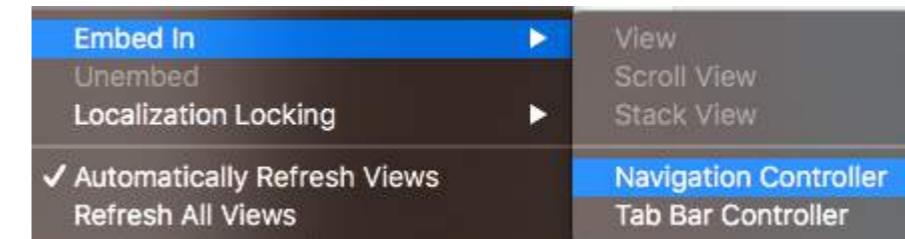
## Section 26.4: Pushing a view controller onto the navigation stack

```
//Swift  
let fooViewController = UIViewController()  
navigationController?.pushViewController(fooViewController, animated: true)  
  
//Objective-C  
UIViewController *fooViewController = [[UIViewController alloc] init];  
[navigationController pushViewController:fooViewController animated:YES];
```

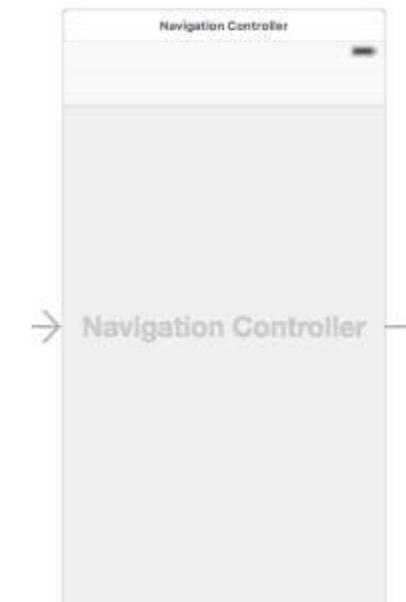
## Section 26.5: Creating a UINavigationController

In your storyboard select the ViewController that you want to embed into a Navigation Controller.

Then navigate to Editor > Embed In > Navigation Controller



And that will create your navigation controller



# 第27章：UIGestureRecognizer

## 第27.1节：UITapGestureRecognizer

使用目标（此处为self）和一个带有单个参数（UITapGestureRecognizer）的方法作为动作，初始化UITapGestureRecognizer。

初始化后，将其添加到应该识别点击的视图中。

### Swift

```
override func viewDidLoad() {
    super.viewDidLoad()
    let recognizer = UITapGestureRecognizer(target: self,
                                         action: #selector(handleTap(_:)))
    view.addGestureRecognizer(recognizer)
}

func handleTap(recognizer: UITapGestureRecognizer) { }
```

### Objective-C

```
- (void)viewDidLoad {
    [super viewDidLoad];
    UITapGestureRecognizer *recognizer =
        [[[UITapGestureRecognizer alloc] initWithTarget:self
                                              action:@selector(handleTap:)];
    [self.view addGestureRecognizer:recognizer];
}

- (void)handleTap:(UITapGestureRecognizer *)recognizer { }
```

### 通过 UITapGestureRecognizer 关闭键盘的示例：

首先，创建用于关闭键盘的函数：

```
func dismissKeyboard() {
    view.endEditing(true)
}
```

然后，在你的视图控制器中添加一个点击手势识别器，调用我们刚刚创建的方法

```
let tap: UITapGestureRecognizer = UITapGestureRecognizer(target: self, action: "dismissKeyboard")
view.addGestureRecognizer(tap)
```

### 获取手势位置的 UITapGestureRecognizer 示例（Swift 3）：

```
func handleTap(gestureRecognizer: UITapGestureRecognizer) {
    print("tap working")
    if gestureRecognizer.state == UIGestureRecognizerState.recognized
    {
        print(gestureRecognizer.location(in: gestureRecognizer.view))
```

# Chapter 27: UIGestureRecognizer

## Section 27.1: UITapGestureRecognizer

Initialize the UITapGestureRecognizer with a target, `self` in this case, and an action which is a method that has a single parameter: a UITapGestureRecognizer.

After initialization, add it to the view that it should recognize taps in.

### Swift

```
override func viewDidLoad() {
    super.viewDidLoad()
    let recognizer = UITapGestureRecognizer(target: self,
                                         action: #selector(handleTap(_:)))
    view.addGestureRecognizer(recognizer)
}

func handleTap(recognizer: UITapGestureRecognizer) { }
```

### Objective-C

```
- (void)viewDidLoad {
    [super viewDidLoad];
    UITapGestureRecognizer *recognizer =
        [[[UITapGestureRecognizer alloc] initWithTarget:self
                                              action:@selector(handleTap:)];
    [self.view addGestureRecognizer:recognizer];
}

- (void)handleTap:(UITapGestureRecognizer *)recognizer { }
```

### Example of keyboard dismissal through UITapGestureRecognizer:

First, you create the function for dismissing the keyboard:

```
func dismissKeyboard() {
    view.endEditing(true)
}
```

Then, you add a tap gesture recognizer in your view controller, calling the method we just made

```
let tap: UITapGestureRecognizer = UITapGestureRecognizer(target: self, action: "dismissKeyboard")
view.addGestureRecognizer(tap)
```

### Example of getting gesture location UITapGestureRecognizer (Swift 3):

```
func handleTap(gestureRecognizer: UITapGestureRecognizer) {
    print("tap working")
    if gestureRecognizer.state == UIGestureRecognizerState.recognized
    {
        print(gestureRecognizer.location(in: gestureRecognizer.view))
```

```
}
```

## 第27.2节：UITapGestureRecognizer (双击)

双击和单击一样，也使用UITapGestureRecognizer。只需将numberOfTapsRequired设置为2即可。

### Swift

```
override func viewDidLoad() {
    super.viewDidLoad()

    // 双击
    let doubleTapGesture = UITapGestureRecognizer(target: self, action: #selector(handleDoubleTap))
    doubleTapGesture.numberOfTapsRequired = 2
    doubleTapView.addGestureRecognizer(doubleTapGesture)
}

// 双击动作
func handleDoubleTap() {
    label.text = "检测到双击"
}
```

#### 注意事项

- 示例项目可以在这里找到。[这里](#)
- 你可以通过将numberOfTapsRequired设置为3来识别三击。

## 第27.3节：在界面构建器中添加手势识别器

从对象库中拖动一个手势识别器到你的视图上。

```
}
```

## Section 27.2: UITapGestureRecognizer (Double Tap)

The double tap, like a single tap, also uses the `UITapGestureRecognizer`. You simply set the `numberOfTapsRequired` to 2.

### Swift

```
override func viewDidLoad() {
    super.viewDidLoad()

    // Double Tap
    let doubleTapGesture = UITapGestureRecognizer(target: self, action: #selector(handleDoubleTap))
    doubleTapGesture.numberOfTapsRequired = 2
    doubleTapView.addGestureRecognizer(doubleTapGesture)
}

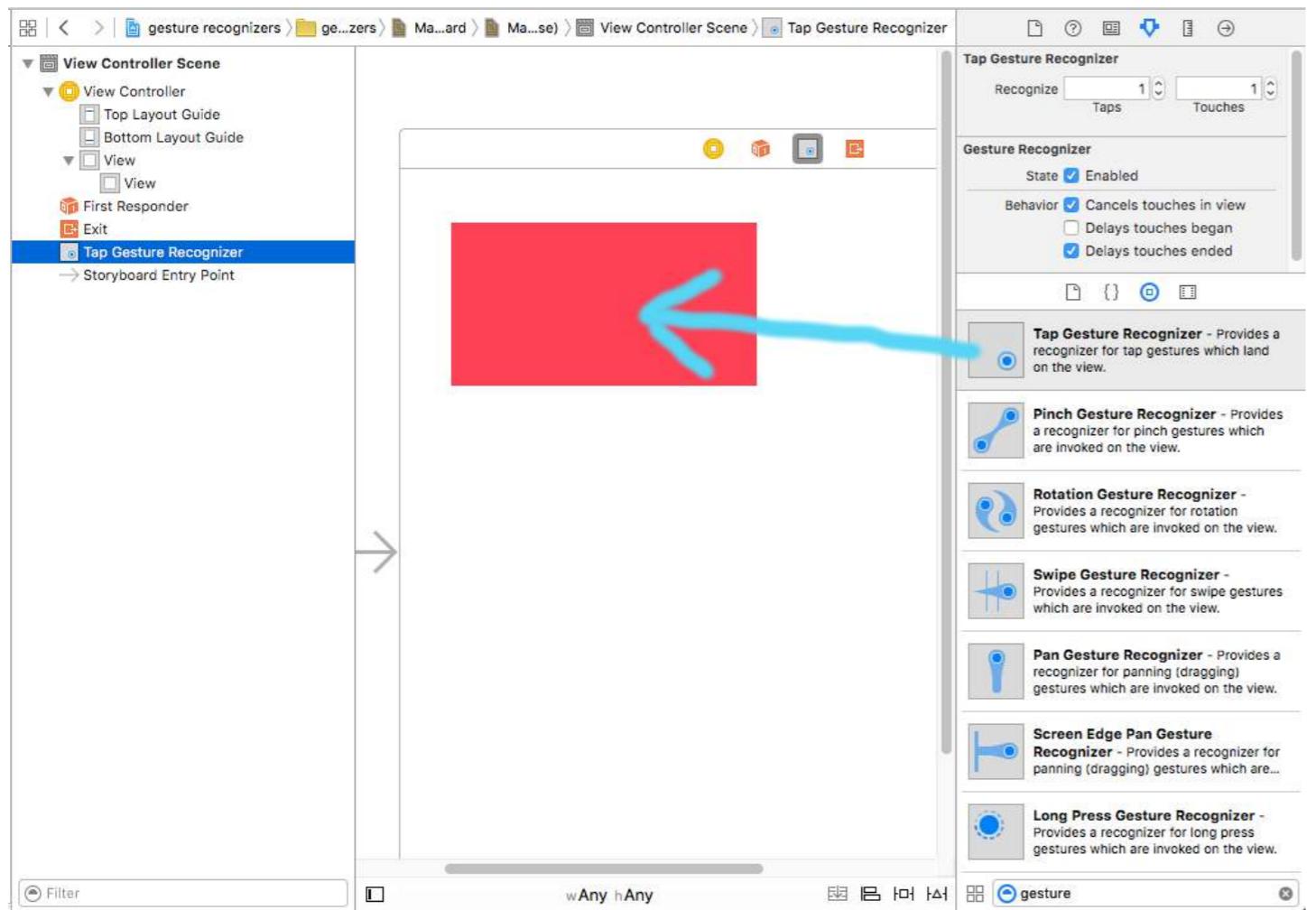
// Double tap action
func handleDoubleTap() {
    label.text = "Double tap recognized"
}
```

#### Notes

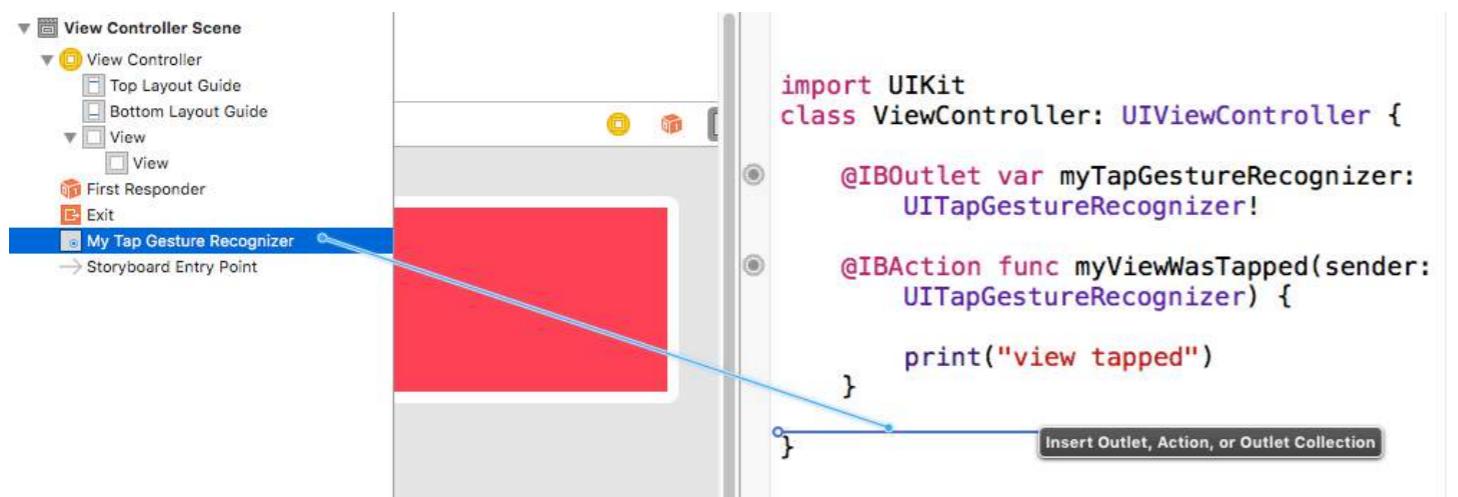
- A sample project can be found [here](#).
- You could recognize a triple tap by setting the `numberOfTapsRequired` to 3.

## Section 27.3: Adding a Gesture recognizer in the Interface Builder

Drag a gesture recognizer from the object library onto your view.



在文档大纲中从手势拖动到你的视图控制器代码，以创建一个Outlet和一个Action。



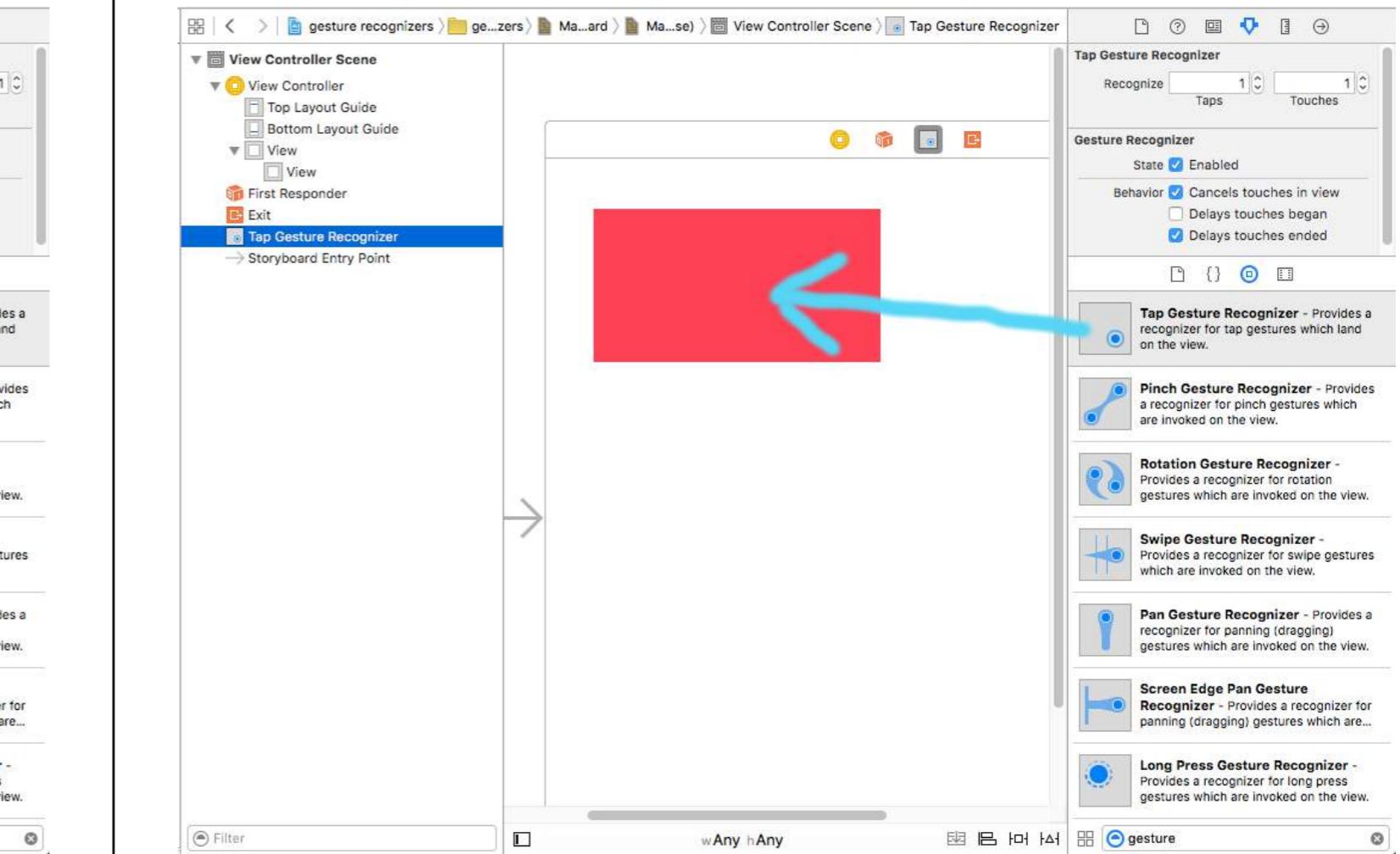
#### 注意事项

- 此示例来自这个更完整的示例项目，演示了手势识别器。

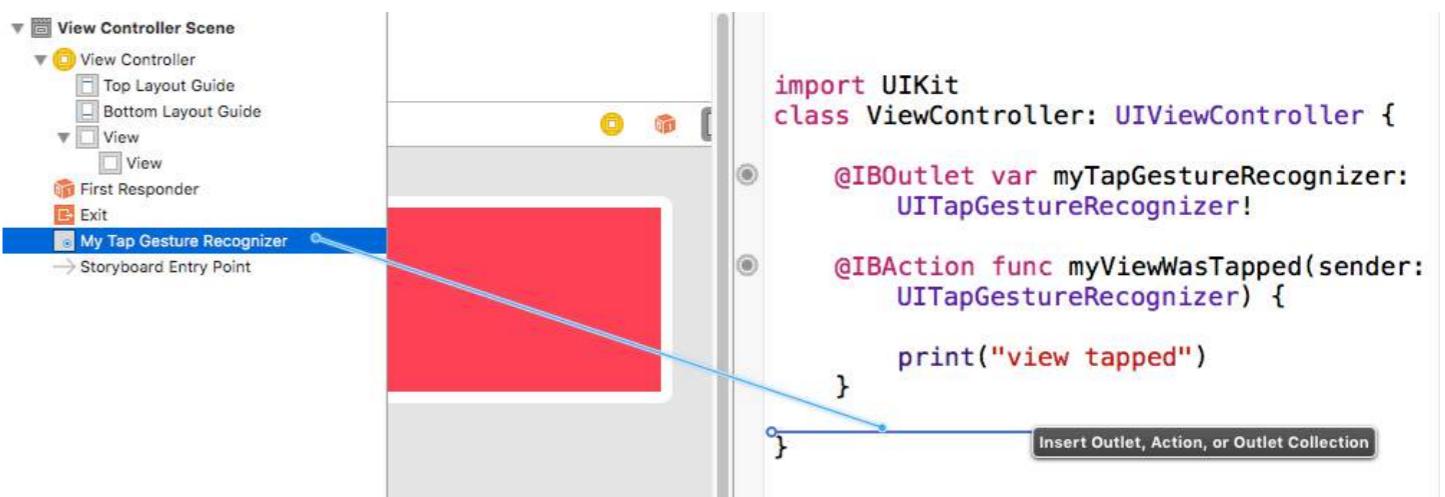
## 第27.4节：UILongPressGestureRecognizer

UILongPressGestureRecognizer允许你监听视图上的长按。你可以设置调用动作方法之前的延迟时间。

#### Swift



Control drag from the gesture in the Document Outline to your View Controller code in order to make an Outlet and an Action.



#### Notes

- This example comes from [this fuller sample project](#) demonstrating gesture recognizers.

## Section 27.4: UILongPressGestureRecognizer

The [UILongPressGestureRecognizer](#) lets you listen for a long press on a view. You can set the length of delay before the action method is called.

#### Swift

```

override func viewDidLoad() {
    super.viewDidLoad()

    // 长按
    let longPressGesture = UILongPressGestureRecognizer(target: self, action:
#selector(handleLongPress(_:)))
    longPressView.addGestureRecognizer(longPressGesture)
}

// 长按动作
func handleLongPress(gesture: UILongPressGestureRecognizer) {
    if gesture.state == UIGestureRecognizerState.Began {
        label.text = "检测到长按"
    }
}

```

#### 注意事项

- 更完整的示例项目可以在这里找到。 [here](#)
- 更改minimumPressDuration以设置长按的持续时间。

## 第27.5节：UISwipeGestureRecognizer

滑动手势允许你监听用户快速在屏幕上向某个方向滑动手指的动作。

#### Swift

```

override func viewDidLoad() {
    super.viewDidLoad()

    // 滑动 (右滑和左滑)
    let swipeRightGesture = UISwipeGestureRecognizer(target: self, action:
#selector(handleSwipe(_:)))
    let swipeLeftGesture = UISwipeGestureRecognizer(target: self, action:
#selector(handleSwipe(_:)))
    swipeRightGesture.direction = UISwipeGestureRecognizerDirection.Right
    swipeLeftGesture.direction = UISwipeGestureRecognizerDirection.Left
    swipeView.addGestureRecognizer(swipeRightGesture)
    swipeView.addGestureRecognizer(swipeLeftGesture)
}

// 滑动动作
func handleSwipe(gesture: UISwipeGestureRecognizer) {
    label.text = "检测到滑动"

    // 示例任务：将视图动画移出屏幕
    let originalLocation = swipeView.center
    if gesture.direction == UISwipeGestureRecognizerDirection.Right {
        label.text = "向右滑动"
    } else if gesture.direction == UISwipeGestureRecognizerDirection.Left {
        label.text = "向左滑动"
    }
}

```

#### Objective-C

```

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

```

```

override func viewDidLoad() {
    super.viewDidLoad()

    // Long Press
    let longPressGesture = UILongPressGestureRecognizer(target: self, action:
#selector(handleLongPress(_:)))
    longPressView.addGestureRecognizer(longPressGesture)
}

// Long press action
func handleLongPress(gesture: UILongPressGestureRecognizer) {
    if gesture.state == UIGestureRecognizerState.Began {
        label.text = "Long press recognized"
    }
}

```

#### Notes

- A fuller sample project can be found [here](#).
- Change the `minimumPressDuration` to set the length of long press.

## Section 27.5: UISwipeGestureRecognizer

Swipe gestures allow you to listen for the user moving their finger across the screen quickly in a certain direction.

#### Swift

```

override func viewDidLoad() {
    super.viewDidLoad()

    // Swipe (right and left)
    let swipeRightGesture = UISwipeGestureRecognizer(target: self, action:
#selector(handleSwipe(_:)))
    let swipeLeftGesture = UISwipeGestureRecognizer(target: self, action:
#selector(handleSwipe(_:)))
    swipeRightGesture.direction = UISwipeGestureRecognizerDirection.Right
    swipeLeftGesture.direction = UISwipeGestureRecognizerDirection.Left
    swipeView.addGestureRecognizer(swipeRightGesture)
    swipeView.addGestureRecognizer(swipeLeftGesture)
}

// Swipe action
func handleSwipe(gesture: UISwipeGestureRecognizer) {
    label.text = "Swipe recognized"

    // example task: animate view off screen
    let originalLocation = swipeView.center
    if gesture.direction == UISwipeGestureRecognizerDirection.Right {
        label.text = "Swipe right"
    } else if gesture.direction == UISwipeGestureRecognizerDirection.Left {
        label.text = "Swipe left"
    }
}

```

#### Objective-C

```

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

```

```

UISwipeGestureRecognizer *swipeLeft = [[UISwipeGestureRecognizer alloc] initWithTarget:self
action:@selector(handleSwipe:)];
UISwipeGestureRecognizer *swipeRight = [[UISwipeGestureRecognizer alloc] initWithTarget:self
action:@selector(handleSwipe:)];

// 设置滑动方向。
[swipeLeft setDirection:UISwipeGestureRecognizerDirectionLeft];
[swipeRight setDirection:UISwipeGestureRecognizerDirectionRight];

// 在图像视图上添加滑动手势
[self.view addGestureRecognizer:swipeLeft];
[self.view addGestureRecognizer:swipeRight];

}

// 处理滑动手势事件

- (void)handleSwipe:(UISwipeGestureRecognizer *)swipe {

    if (swipe.方向 == UISwipeGestureRecognizerDirectionLeft) {
        NSLog(@"左滑");
    }

    if (swipe.direction == UISwipeGestureRecognizerDirectionRight) {
        NSLog(@"右滑");
    }
}

```

#### 注意事项

- 可以在这里找到一个更完整的项目示例。[here](#)

## 第27.6节：UIPinchGestureRecognizer (捏合手势识别器)

捏合是用两根手指进行的手势，手指彼此靠近或远离。这个手势通常用于调整视图的大小。

#### Swift

```

override func viewDidLoad() {
    super.viewDidLoad()

    // 捏合
    let pinchGesture = UIPinchGestureRecognizer(target: self, action: #selector(handlePinch(_:)))
    pinchView.addGestureRecognizer(pinchGesture)
}

// 捏合动作
func handlePinch(gesture: UIPinchGestureRecognizer) {
    label.text = "检测到捏合"

    if gesture.state == UIGestureRecognizerState.Changed {
        let transform = CGAffineTransformMakeScale(gesture.scale, gesture.scale)
        pinchView.transform = transform
    }
}

```

#### 注意事项

- 可以在这里找到一个更完整的项目示例。[here](#)

```

UISwipeGestureRecognizer *swipeLeft = [[UISwipeGestureRecognizer alloc] initWithTarget:self
action:@selector(handleSwipe:)];
UISwipeGestureRecognizer *swipeRight = [[UISwipeGestureRecognizer alloc] initWithTarget:self
action:@selector(handleSwipe:)];

// Setting the swipe direction.
[swipeLeft setDirection:UISwipeGestureRecognizerDirectionLeft];
[swipeRight setDirection:UISwipeGestureRecognizerDirectionRight];

// Adding the swipe gesture on image view
[self.view addGestureRecognizer:swipeLeft];
[self.view addGestureRecognizer:swipeRight];

}

// Handling Swipe Gesture Events

- (void)handleSwipe:(UISwipeGestureRecognizer *)swipe {

    if (swipe.direction == UISwipeGestureRecognizerDirectionLeft) {
        NSLog(@"Left Swipe");
    }

    if (swipe.direction == UISwipeGestureRecognizerDirectionRight) {
        NSLog(@"Right Swipe");
    }
}

```

#### Notes

- A fuller project example can be found [here](#).

## Section 27.6: UIPinchGestureRecognizer

Pinches are a two fingered gesture where the fingers move closer or farther from each other. This gesture is generally used for resizing a view.

#### Swift

```

override func viewDidLoad() {
    super.viewDidLoad()

    // Pinch
    let pinchGesture = UIPinchGestureRecognizer(target: self, action: #selector(handlePinch(_:)))
    pinchView.addGestureRecognizer(pinchGesture)
}

// Pinch action
func handlePinch(gesture: UIPinchGestureRecognizer) {
    label.text = "Pinch recognized"

    if gesture.state == UIGestureRecognizerState.Changed {
        let transform = CGAffineTransformMakeScale(gesture.scale, gesture.scale)
        pinchView.transform = transform
    }
}

```

#### Notes

- A fuller project example can be found [here](#).

## 第27.7节：UIRotationGestureRecognizer（旋转手势识别器）

可以使用UIRotationGestureRecognizer监听围绕中心旋转的两个手指。这通常用于旋转视图。

### Swift

```
override func viewDidLoad() {
    super.viewDidLoad()

    // 旋转
    let rotateGesture = UIRotationGestureRecognizer(target: self, action:
#selector(handleRotate(_:)))
    rotateView.addGestureRecognizer(rotateGesture)
}

// 旋转动作
func handleRotate(gesture: UIRotationGestureRecognizer) {
    label.text = "检测到旋转"

    if gesture.state == UIGestureRecognizerState.Changed {
        let transform = CGAffineTransformMakeRotation(gesture.rotation)
        rotateView.transform = transform
    }
}
```

### 注意事项

- 示例项目可以在这里找到。[\\_\\_\\_\\_\\_](#)

## Section 27.7: UIRotationGestureRecognizer

Two fingers rotating around a center can be listened for with the `UIRotationGestureRecognizer`. This is generally used for rotating a view.

### Swift

```
override func viewDidLoad() {
    super.viewDidLoad()

    // Rotate
    let rotateGesture = UIRotationGestureRecognizer(target: self, action:
#selector(handleRotate(_:)))
    rotateView.addGestureRecognizer(rotateGesture)
}

// Rotate action
func handleRotate(gesture: UIRotationGestureRecognizer) {
    label.text = "Rotate recognized"

    if gesture.state == UIGestureRecognizerState.Changed {
        let transform = CGAffineTransformMakeRotation(gesture.rotation)
        rotateView.transform = transform
    }
}
```

### Notes

- A sample project can be found [here](#).

# 第28章：UIBarButtonItem

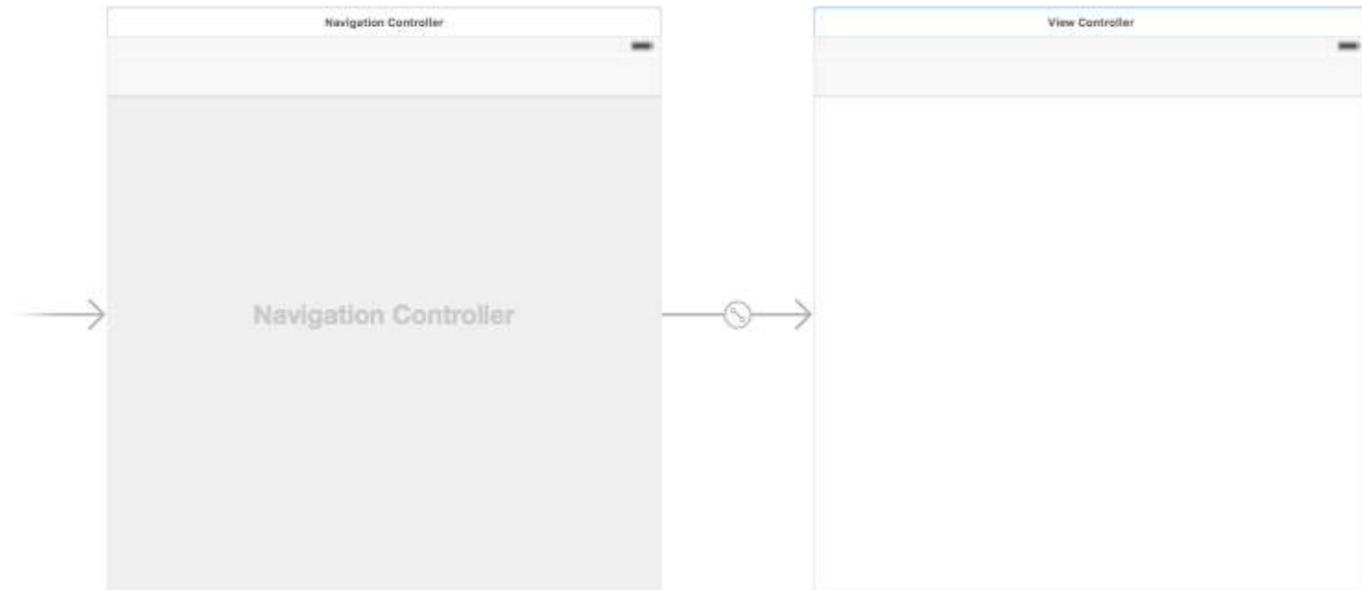
参数	描述
标题	UIBarButtonItem 标题
样式	UIBarButtonItem 的样式
目标	接收 UIBarButtonItem 操作的对象
动作	按下 UIBarButtonItem 时要执行的选择器（方法）

## 第 28.1 节：在界面构建器中创建 UIBarButtonItem

下面的示例展示了如何在界面构建器中添加一个导航栏按钮（称为UIBarButtonItem）。

### 向你的故事板添加导航控制器

选择你的视图控制器，然后在 Xcode 菜单中选择编辑 > 嵌入到 > 导航控制器。



或者，你也可以从对象库中添加一个UINavigationBar。

### 添加一个栏按钮项

从对象库中拖动一个UIBarButtonItem到顶部导航栏。

# Chapter 28: UIBarButtonItem

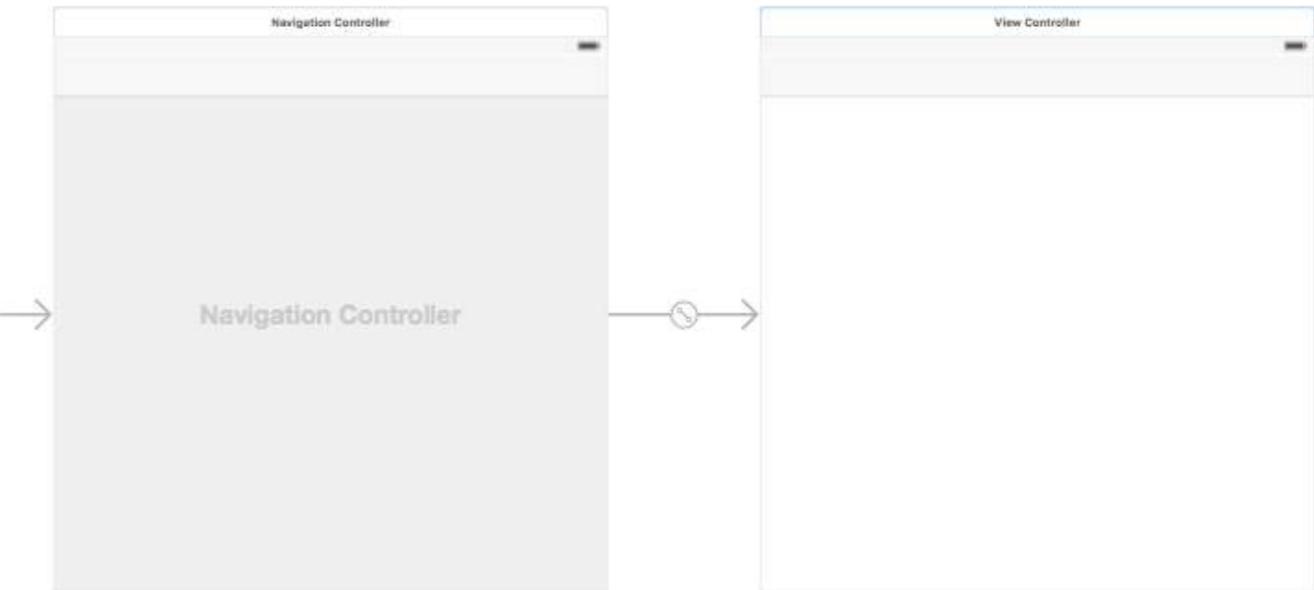
Parameter	Description
title	The UIBarButtonItem title
style	The style of the UIBarButtonItem
target	The object to receive the UIBarButtonItem action
action	The selector (method) to be performed when the UIBarButtonItem is pressed

## Section 28.1: Creating a UIBarButtonItem in the Interface Builder

The example below shows how to add a navigation bar button (called a [UIBarButtonItem](#)) in the Interface Builder.

### Add a Navigation Controller to your Storyboard

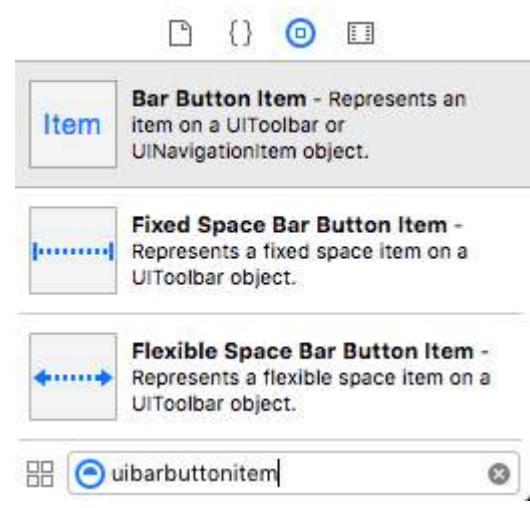
Select your View Controller and then in the Xcode menu choose **Editor > Embed In > Navigation Controller**.



Alternatively, you could add a [UINavigationBar](#) from the Object Library.

### Add a Bar Button Item

Drag a [UIBarButtonItem](#) from the Object Library to the top navigation bar.

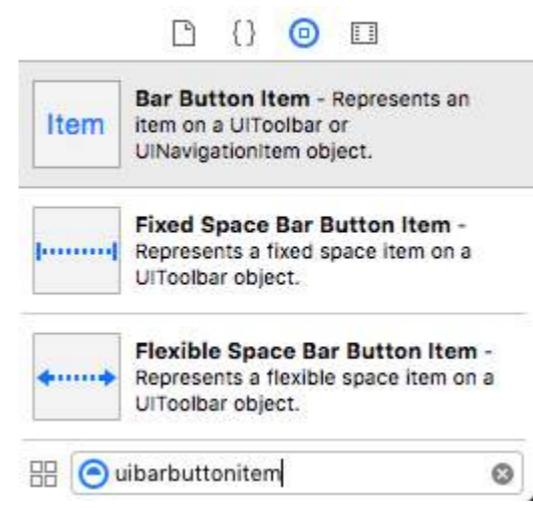


它应该看起来像这样：



#### 设置属性

你可以双击“Item”将文本改为“刷新”之类的内容，但实际上有一个刷新图标可以使用。只需选择`UIBarButtonItem`的属性检查器，然后在系统项中选择刷新。

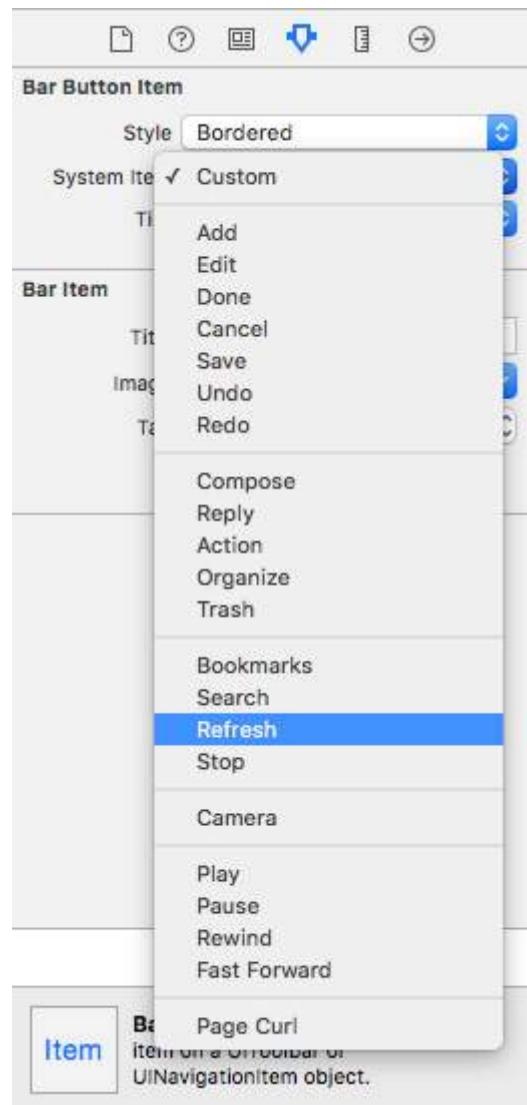


It should look like this:

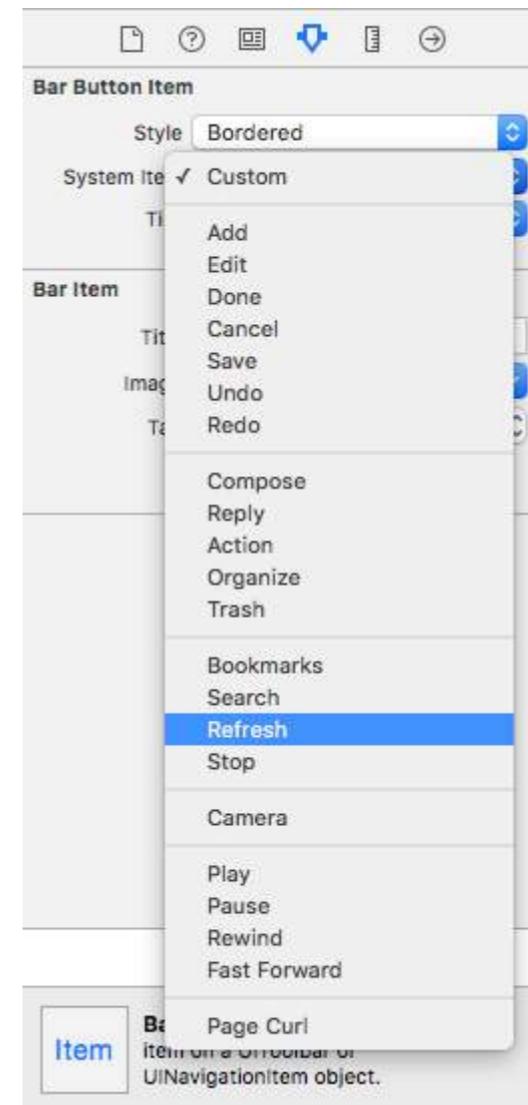


#### Set the Attributes

You could double-click "Item" to change the text to something like "Refresh", but there is an actual icon for *Refresh* that you can use. Just select the Attributes Inspector for the `UIBarButtonItem` and for **System Item** choose **Refresh**.



这将为您提供默认的刷新图标。



That will give you the default Refresh icon.



### 添加一个 IB 操作

从UIBarButtonItem拖动控制到视图控制器以添加一个@IBAction。

```
class ViewController: UIViewController {

    @IBAction func refreshBarButtonItemTap(sender: UIBarButtonItem) {
        print("多么清新！")
    }
}
```

就是这样。



### Add an IB Action

Control drag from the UIBarButtonItem to the View Controller to add an @IBAction.

```
class ViewController: UIViewController {

    @IBAction func refreshBarButtonItemTap(sender: UIBarButtonItem) {
        print("How refreshing!")
    }
}
```

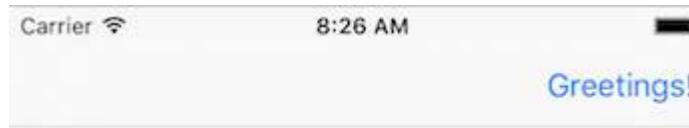
That's it.

- 此示例最初来自这个 Stack Overflow 回答。

## 第 28.2 节：创建 UIBarButtonItem

```
//Swift
let barButtonItem = UIBarButtonItem(title: "问候！", style: .Plain, target: self, action:
#selector(barButtonTapped))
self.navigationItem.rightBarButtonItem = barButtonItem

//Objective-C
UIBarButtonItem *barButtonItem = [[UIBarButtonItem alloc] initWithTitle:@"Greetings!"
style:UIBarButtonItemStylePlain target:self action:@selector(barButtonTaped)];
self.navigationItem.rightBarButtonItem = barButtonItem;
```



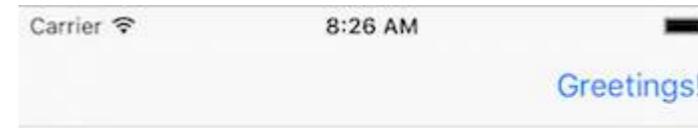
### Notes

- This example originally comes from [this Stack Overflow answer](#).

## Section 28.2: Creating a UIBarButtonItem

```
//Swift
let barButtonItem = UIBarButtonItem(title: "Greetings!", style: .Plain, target: self, action:
#selector(barButtonTapped))
self.navigationItem.rightBarButtonItem = barButtonItem

//Objective-C
UIBarButtonItem *barButtonItem = [[UIBarButtonItem alloc] initWithTitle:@"Greetings!"
style:UIBarButtonItemStylePlain target:self action:@selector(barButtonTaped)];
self.navigationItem.rightBarButtonItem = barButtonItem;
```



## 第28.3节：无色调的条形按钮原始图像

假设barButtonItem具有非空的image属性（例如，在界面构建器中设置）。

### Objective-C

```
barButtonItem.image = [barButtonItem.image
imageWithRenderingMode:UIImageRenderingModeAlwaysOriginal];
```

## Section 28.3: Bar Button Item Original Image with no Tint Color

Provided that barButtonItem has a non-null image property (e.g. set in the Interface Builder).

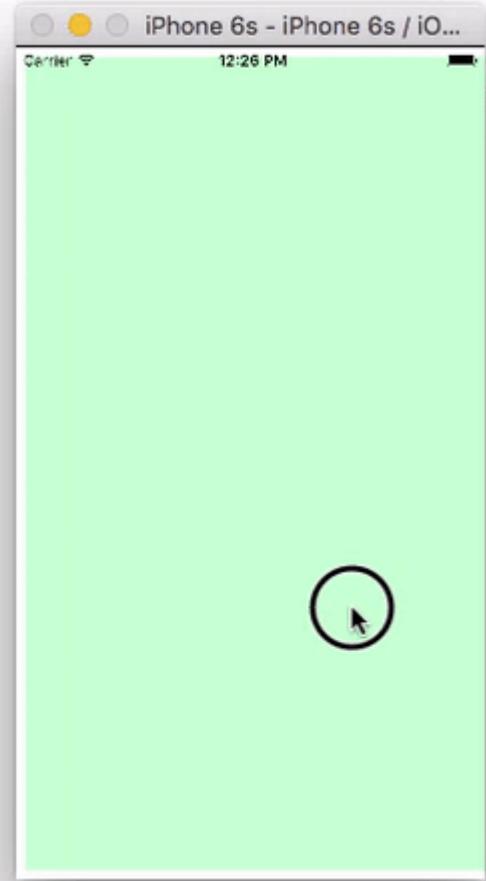
### Objective-C

```
barButtonItem.image = [barButtonItem.image
imageWithRenderingMode:UIImageRenderingModeAlwaysOriginal];
```

# 第29章：UIScrollView

## 第29.1节：启用自动布局的滚动内容

该项目是一个完全在界面构建器中完成的独立示例。你应该能在10分钟内完成学习。然后你可以将学到的概念应用到自己的项目中。



这里我只使用了UIView，但它们可以代表你喜欢的任何视图（例如按钮、标签等）。我还选择了水平滚动，因为故事板截图在这种格式下更紧凑。不过，垂直滚动的原理是相同的。

### 关键概念

- UIScrollView 应该只使用一个子视图。这个子视图是一个 UIView，作为内容视图来承载你希望滚动的所有内容。
- 使内容视图和滚动视图的父视图高度相等，以实现水平滚动。（垂直滚动时宽度相等）
- 确保所有可滚动内容都有设定的宽度，并且四边都固定。

### 开始一个新项目

可以只是一个单视图应用程序。

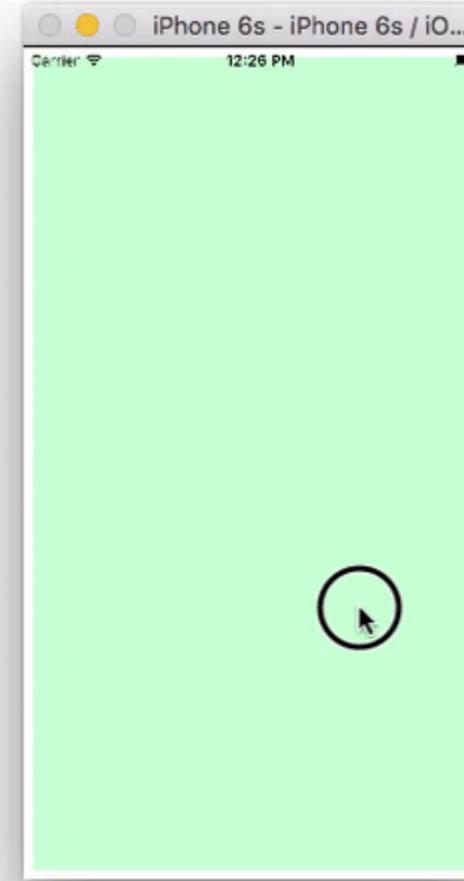
#### Storyboard (故事板)

在本例中，我们将制作一个水平滚动视图。选择视图控制器，然后在

# Chapter 29: UIScrollView

## Section 29.1: Scrolling content with Auto Layout enabled

This project is a self-contained example done completely in the Interface Builder. You should be able to work through it in 10 minutes or less. Then you can apply the concepts you learned to your own project.



Here I just use [UIViews](#) but they can represent whatever view you like (ie, button, label, etc). I also chose horizontal scrolling because the storyboard screenshots are more compact for this format. The principles are the same for vertical scrolling, though.

### Key concepts

- The [UIScrollView](#) should only use one subview. This is a 'UIView' that serves as the content view to hold everything you wish to scroll.
- Make the content view and the scroll view's *parent* have equal heights for horizontal scrolling. (Equal widths for vertical scrolling)
- Make sure that all of the scrollable content has a set width and is pinned on all sides.

### Start a new project

It can be just a single view application.

#### Storyboard

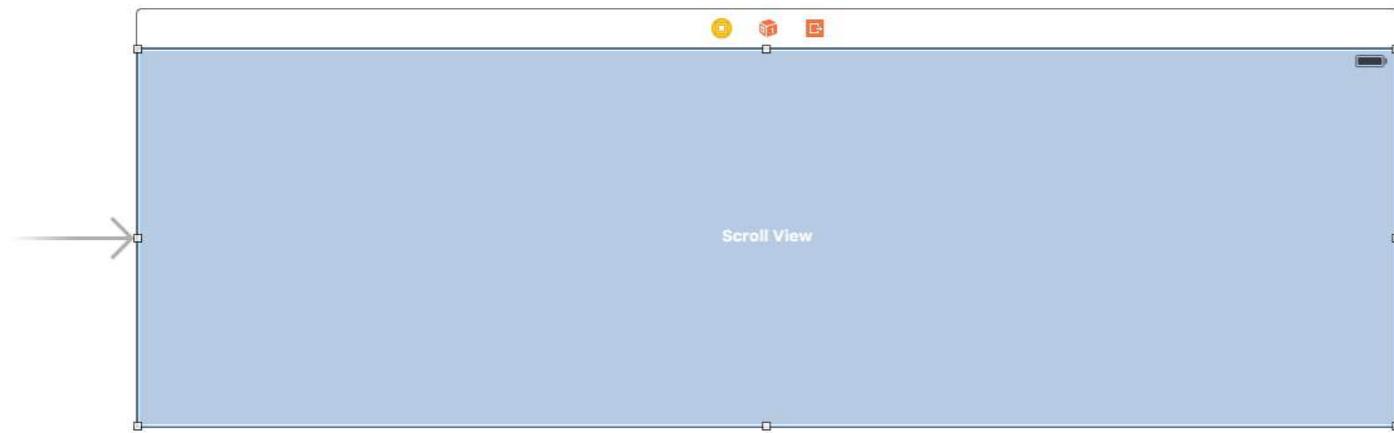
In this example, we will make a horizontal scroll view. Select the View Controller and then choose Freeform in the

尺寸检查器中选择自由形式。将宽度设为1,000，高度设为300。这只是为了在故事板上留出添加可滚动内容的空间。



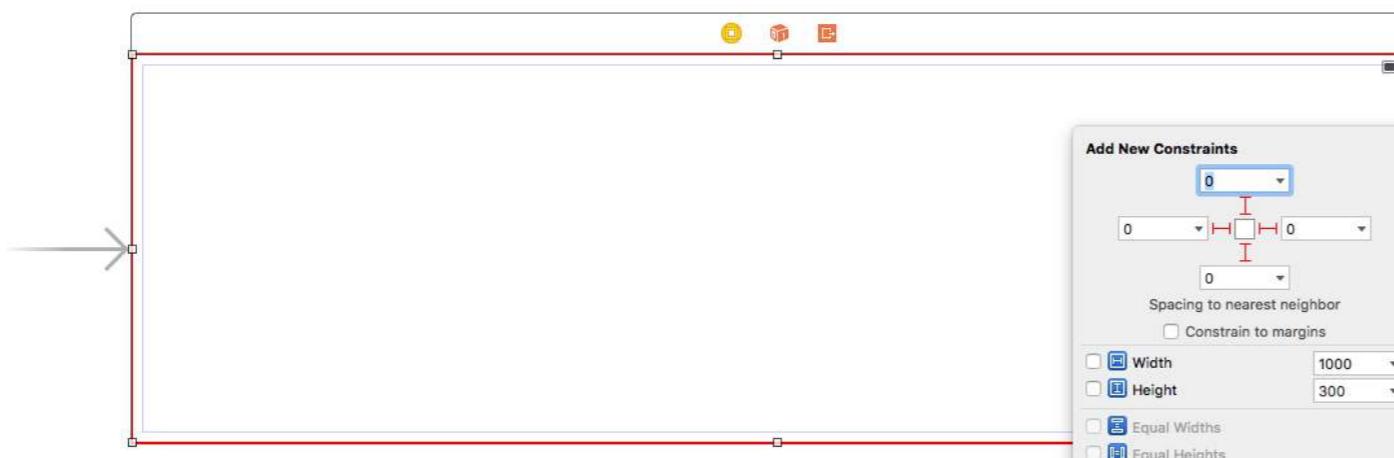
## 添加滚动视图

添加一个UIScrollView，并将四边都固定到视图控制器的根视图。



## 添加内容视图

向滚动视图添加一个UIView作为子视图。这一点很关键。不要尝试向滚动视图添加大量子视图。只需添加一个UIView。这将作为你想要滚动的其他视图的内容视图。将内容视图的四边固定到滚动视图。



## 等高

现在在文档大纲中，按住 Command 键点击内容视图和滚动视图的父视图以同时选择它们。然后设置它们的高度相等（按住 Control 键拖动从内容视图到滚动视图）。

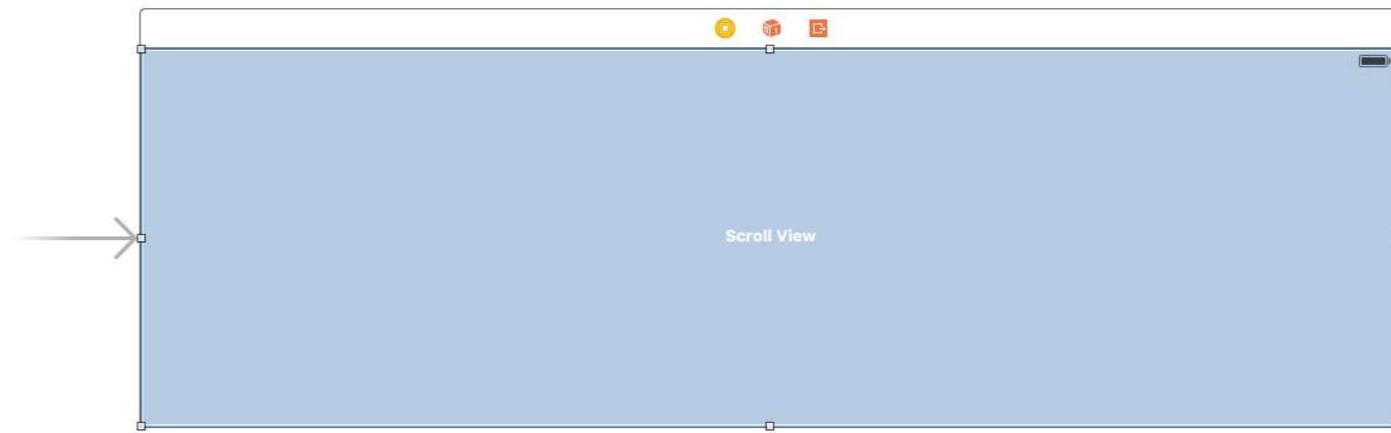
这也很关键。因为我们是水平滚动，滚动视图的内容视图不知道它应该有多高，除非我们以这种方式设置它。

Size Inspector. Make the width [1,000](#) and the height [300](#). This just gives us the room on the storyboard to add content that will scroll.



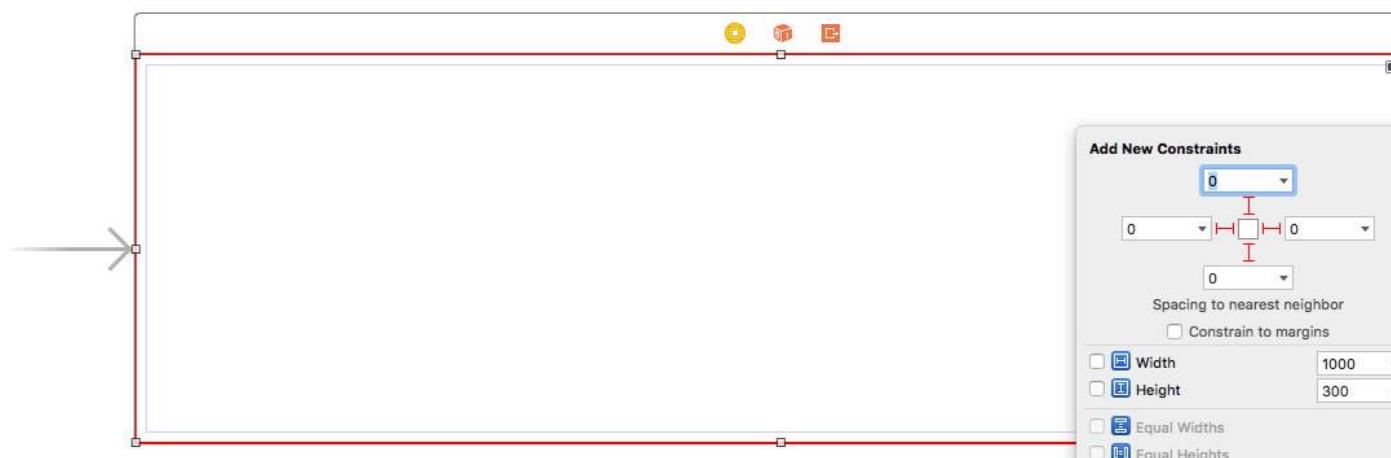
## Add a Scroll View

Add a [UIScrollView](#) and pin all four sides to the root view of the view controller.



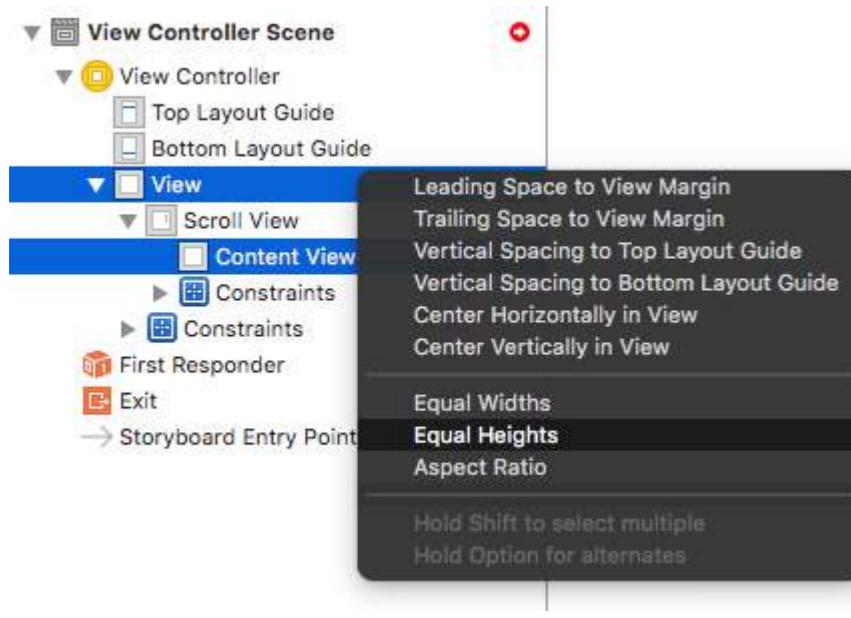
## Add a Content View

Add a [UIView](#) as a subview to the scroll view. *This is key*. Don't try to add lots of subviews to the scroll view. Just add a single [UIView](#). This will be your content view for the other views you want to scroll. Pin the content view to the scroll view on all four sides.



## Equal Heights

Now in the Document Outline, Command click both the content view and the scroll view's *parent view* in order to select them both. Then set the heights to be equal (Control/k drag from the Content View to the Scroll View>). *This is also key*. Because we are scrolling horizontally, the scroll view's content view won't know how high it should be unless we set it in this way.



注意：

- 如果我们制作的是垂直滚动的内容，那么我们会将内容视图的宽度设置为与滚动视图的父视图宽度相等。

## 添加内容

添加三个UIImageView，并为它们全部设置约束。我对所有边距都使用了8点。



约束条件：

- 绿色视图：固定顶部、左侧和底部边缘。宽度设为400。
- 红色视图：固定顶部、左侧和底部边缘。宽度设为300。
- 紫色视图：固定四个边缘。宽度设为剩余空间（本例中为268）。

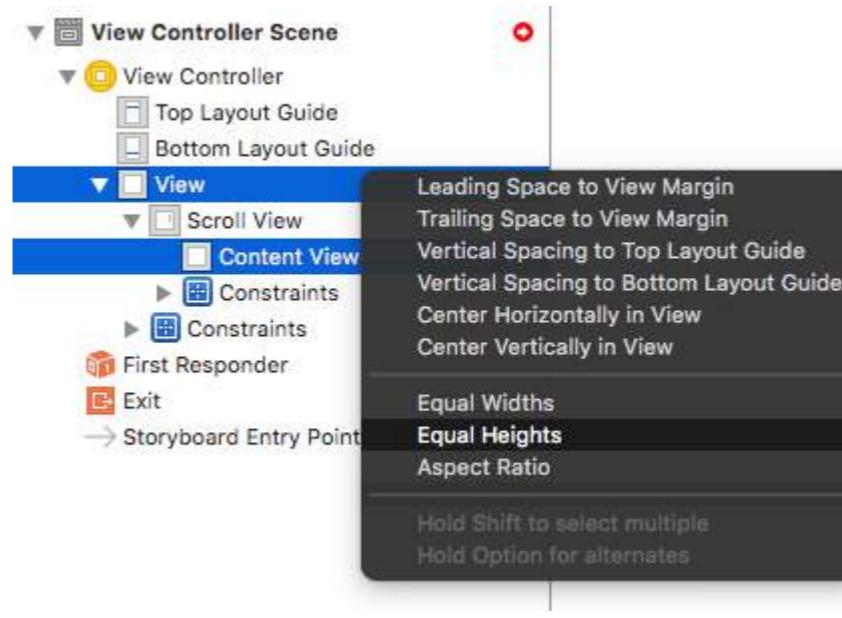
设置宽度约束也很关键，这样滚动视图才能知道其内容视图的宽度。

## 完成

就这些。你现在可以运行你的项目了。它应该像本答案顶部的滚动图片一样表现。

## 进一步学习

- [iOS：如何让AutoLayout在ScrollView上生效](#)
- [如何在Interface Builder中配置带有Auto\\_Layout的UIScrollView](#)

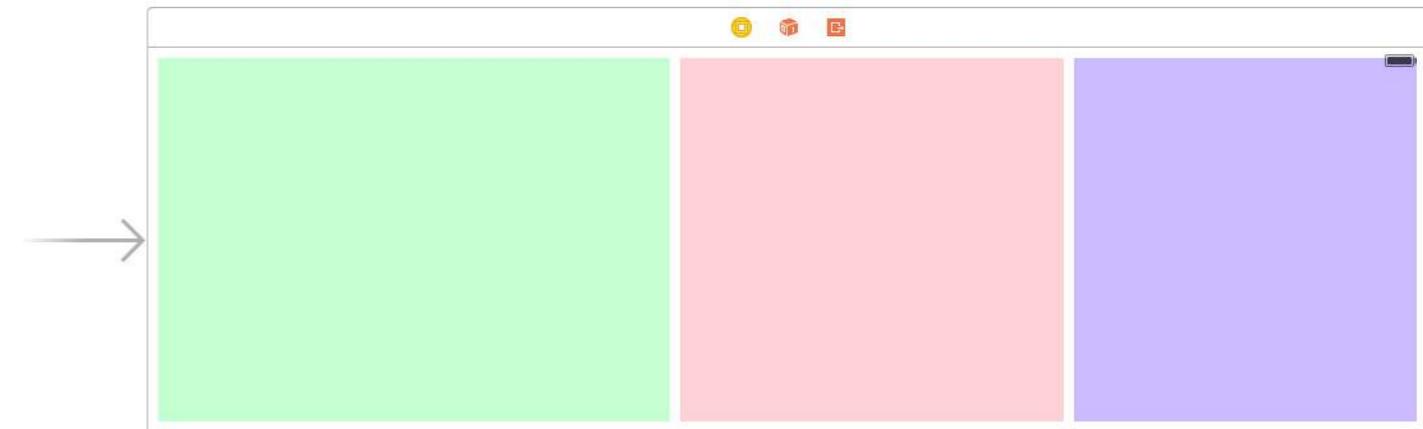


Note:

- If we were making the content scroll vertically, then we would set the content view's width to be equal to the scroll view's parent's width.

## Add content

Add three UIImageViews and give them all constraints. I used 8 point margins for everything.



Constraints:

- Green view: pin the top, left, and bottom edges. Make the width 400.
- Red view: pin the top, left, and bottom edges. Make the width 300.
- Purple view: pin all four edges. Make the width whatever the remaining space is (268 in this case).

Setting the width constraints is also key so that the scroll view knows how wide its content view will be.

## Finished

That's all. You can run your project now. It should behave like the scrolling image at the top of this answer.

## Further Study

- [iOS: How To Make AutoLayout Work On A ScrollView](#)
- [How to configure a UIScrollView with Auto Layout in Interface Builder](#)

## 第29.2节：创建UIScrollView

创建一个以CGRect为框架的UIScrollView实例。

### Swift

```
let scrollview = UIScrollView(frame: CGRect(x: 0, y: 0, width: 320, height: 400))
```

### Objective-C

```
UIScrollView *scrollview = [[UIScrollView alloc] initWithFrame:CGRectMake(0, 0, 320, 400)];
```

## Section 29.2: Create a UIScrollView

Create an instance of `UIScrollView` with a `CGRect` as frame.

### Swift

```
let scrollview = UIScrollView(frame: CGRect(x: 0, y: 0, width: 320, height: 400))
```

### Objective-C

```
UIScrollView *scrollview = [[UIScrollView alloc] initWithFrame:CGRectMake(0, 0, 320, 400)];
```

## 第29.3节：使用AutoLayout的ScrollView

使用AutoLayout的scrollview的简单步骤。

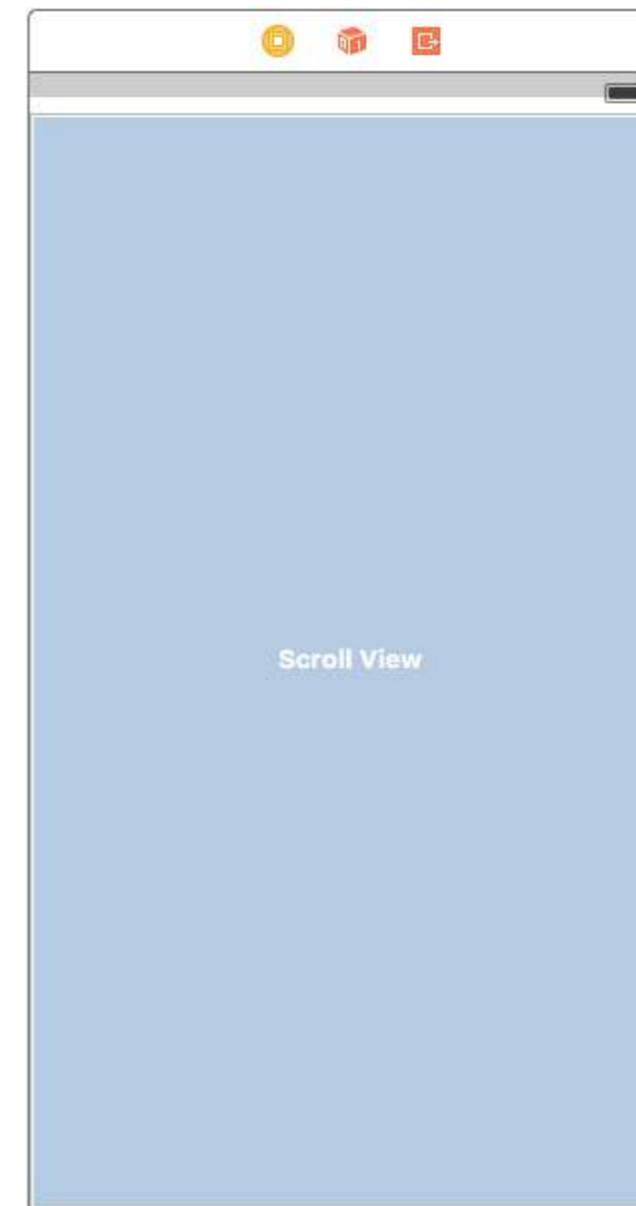


- 创建一个单视图应用的新项目
- 选择默认的视图控制器，并在属性检查器中将其屏幕尺寸更改为iPhone-4英寸。
- 向视图控制器的视图中添加一个scrollview，并将背景色设置为蓝色

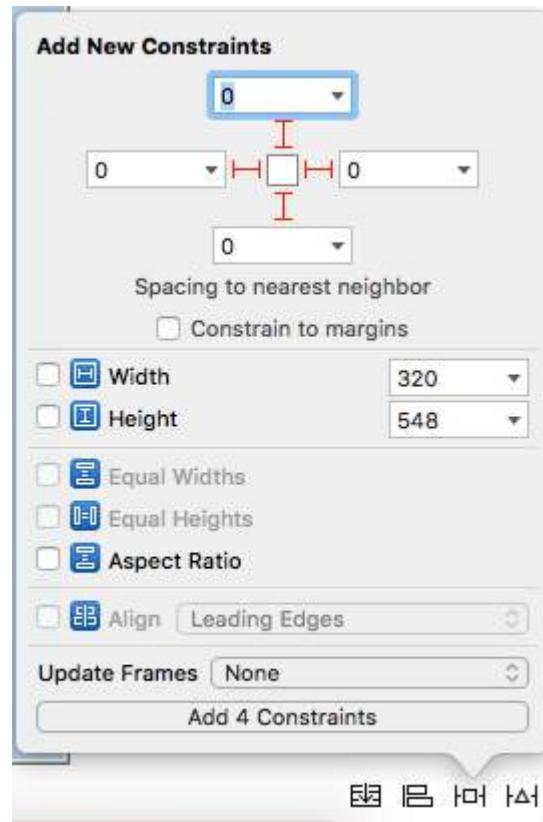
## Section 29.3: ScrollView with AutoLayout

Simple steps to use scrollview with autolayout.

- Create a new project with single view application
- Select the default viewcontroller and change its screen size to iPhone-4inch from attributes inspector.
- Add a scrollview to your viewcontroller's view as follows and set background color to blue



- 如下面图片所示为其添加约束



这将做的是，将滚动视图的每个边缘简单地固定到视图控制器的视图上

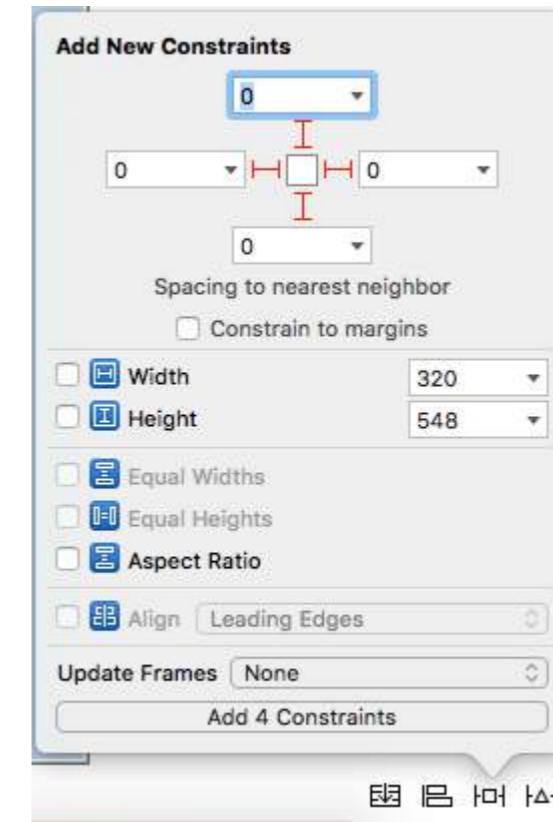
#### 场景 1：

**现在假设我们的内容非常庞大，我们希望它既能水平滚动，也能垂直滚动。**

为此，

- 向滚动视图中添加一个大小为(0,0,700,700)的 UIView。我们给它设置橙色背景色以便区分。

- Add constraints on it as shown in below image



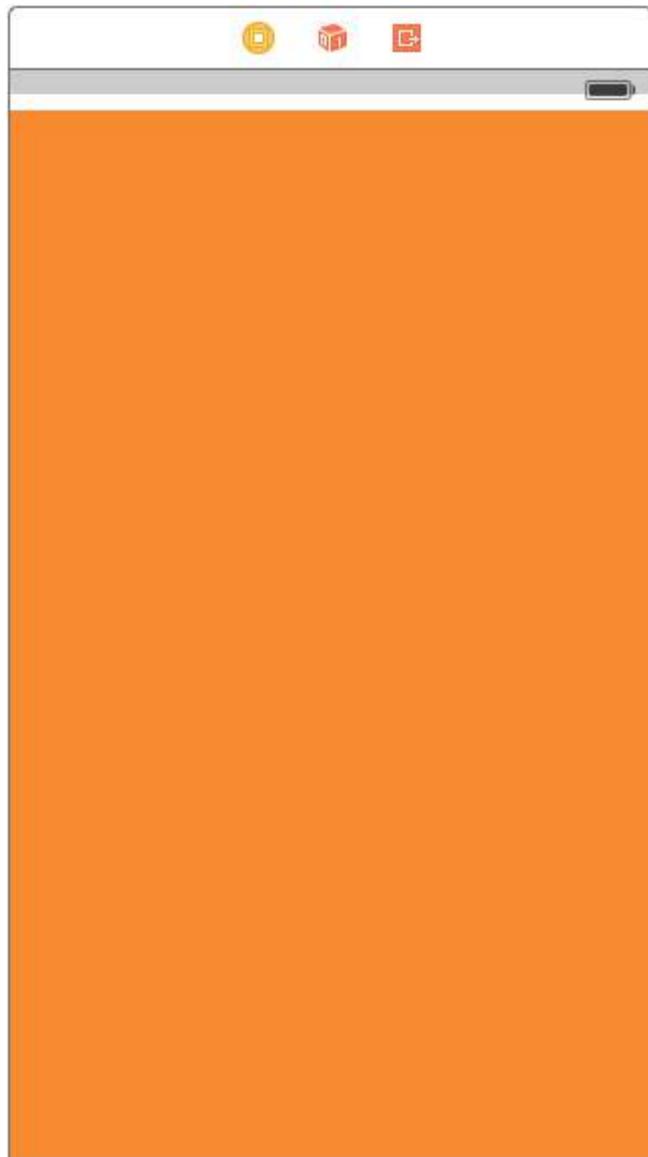
What this will do is, simply stick every edge of scrollview to viewcontroller's view

#### Scenario 1:

**Now lets say our content is huge, and we want it to scroll horizontally as well as vertically.**

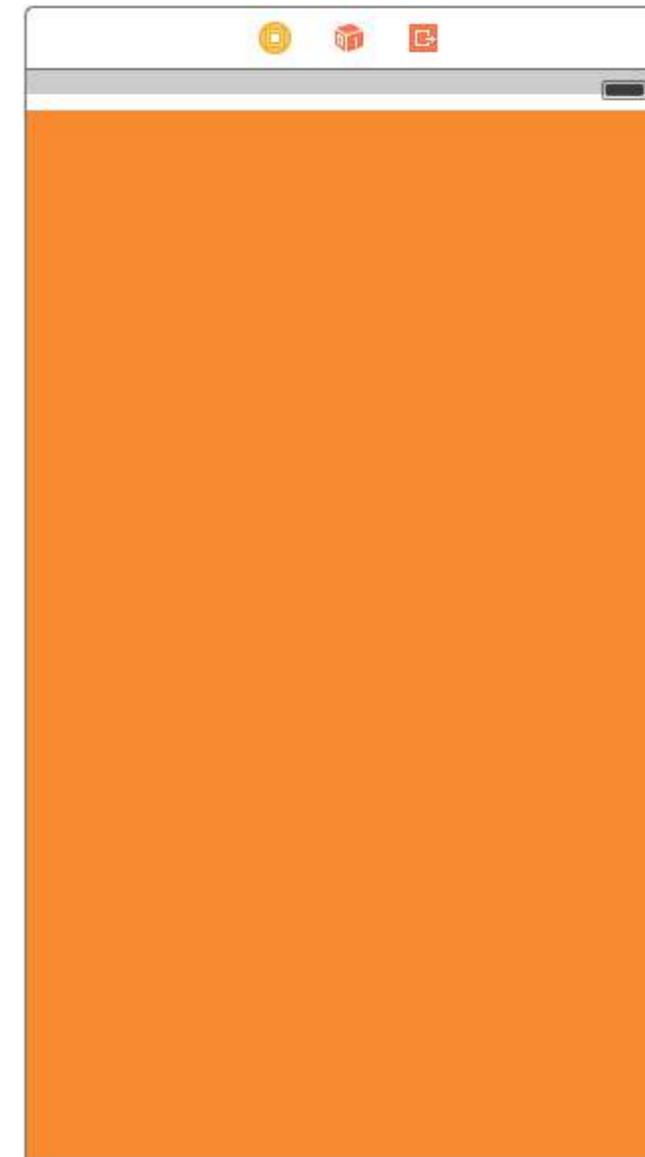
For this,

- Add a UIView to the scrollview of frame(0,0,700,700). Lets give it orange background color to identify it differently.



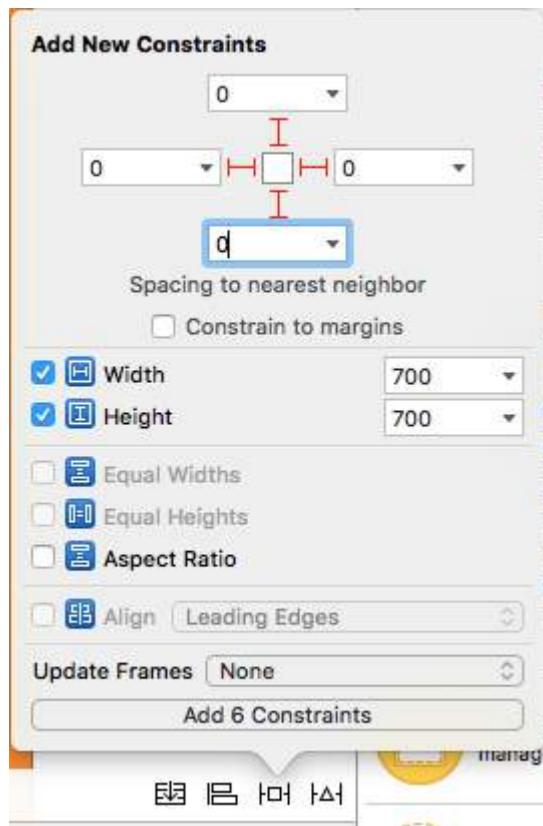
接下来是重要部分，我们需要它既能水平滚动，也能垂直滚动。

- 选择橙色视图并添加以下约束



Next comes the important part, we need it to scroll horizontally and vertically.

- Select the orange view and add the following constraints



让我解释一下我们在上一步做了什么。

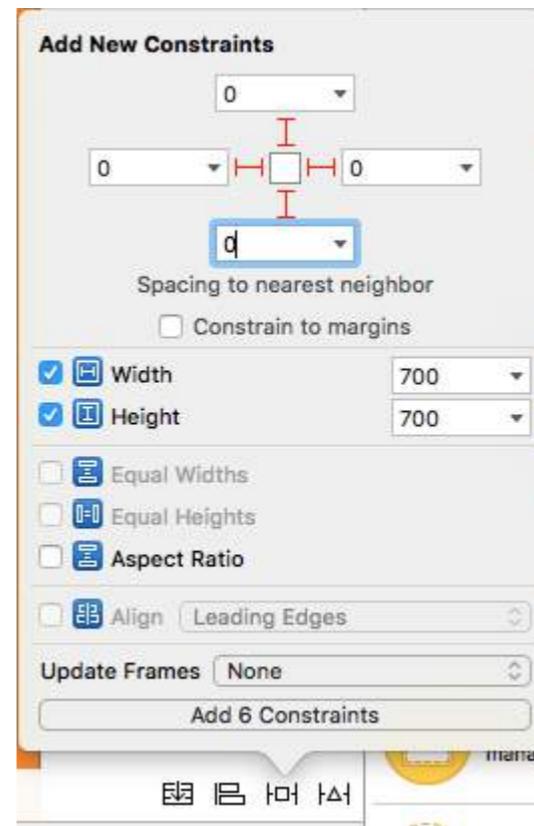
- 我们将高度和宽度固定为700。
- 我们将滚动视图的尾部间距设置为0，这告诉滚动视图内容可以水平滚动。
- 我们将滚动视图的底部间距设置为0，这告诉滚动视图内容可以垂直滚动。

现在运行项目并检查。

**场景2：假设内容宽度将与滚动宽度相同  
宽度，但高度大于滚动视图。**

按照步骤垂直滚动内容。

- 删除上述情况下的宽度约束。
- 将橙色视图的宽度更改为与滚动视图宽度匹配。
- 按住Ctrl从橙色视图拖动到滚动视图，添加等宽约束。



Let me explain what we did in above step.

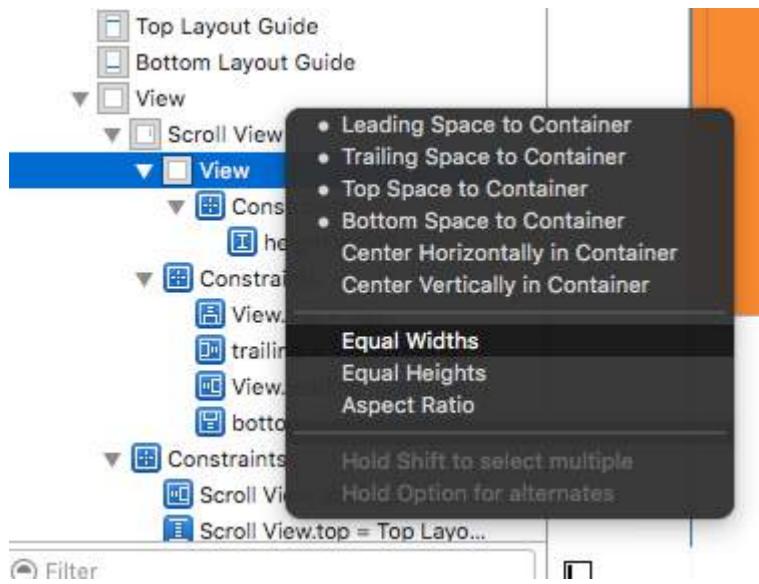
- We fixed the height and width to 700.
- We set trailing space to scrollview = 0 which tells the scrollview that content is horizontally scrollable.
- We set bottom space to scrollview = 0 which tells the scrollview that content is vertically scrollable.

Now run the project and check.

**Scenario 2: Lets consider a scenario where we know that content width is going to be same as scroll width width, but height is larger than scrollview.**

Follow the steps to scroll content vertically.

- Delete the width constraint in above case.
- Change the width of orange view to match to scrollview width.
- Ctrl-drag from orange view to scroll view and add **equal widths** constraint.



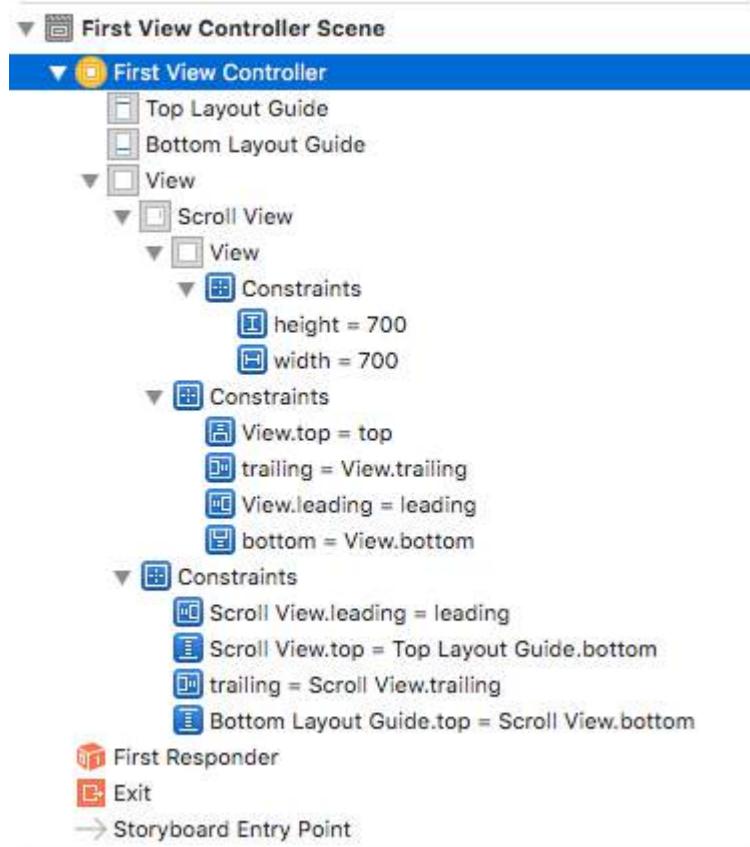
- 完成！！！只需运行并检查是否可以垂直滚动

### 情景3：

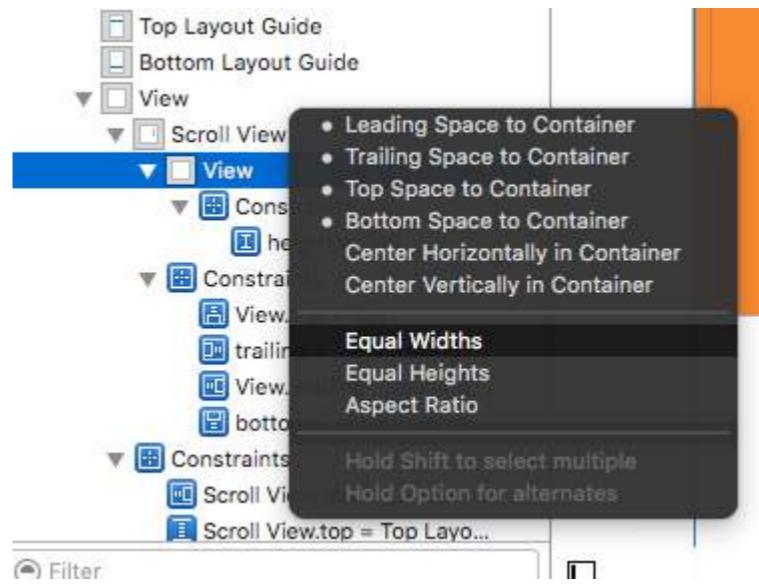
现在我们只想水平滚动，而不垂直滚动。

按照步骤水平滚动内容。

- 撤销所有更改以实现以下约束（即恢复实现了垂直和水平滚动的原始约束）。



- 检查橙色视图的框架，应为(0,0,700,700)
- 删除橙色视图的高度约束。
- 将橙色视图的高度更改为与滚动视图高度匹配。
- 按住Ctrl从橙色视图拖动到滚动视图并添加等高约束。



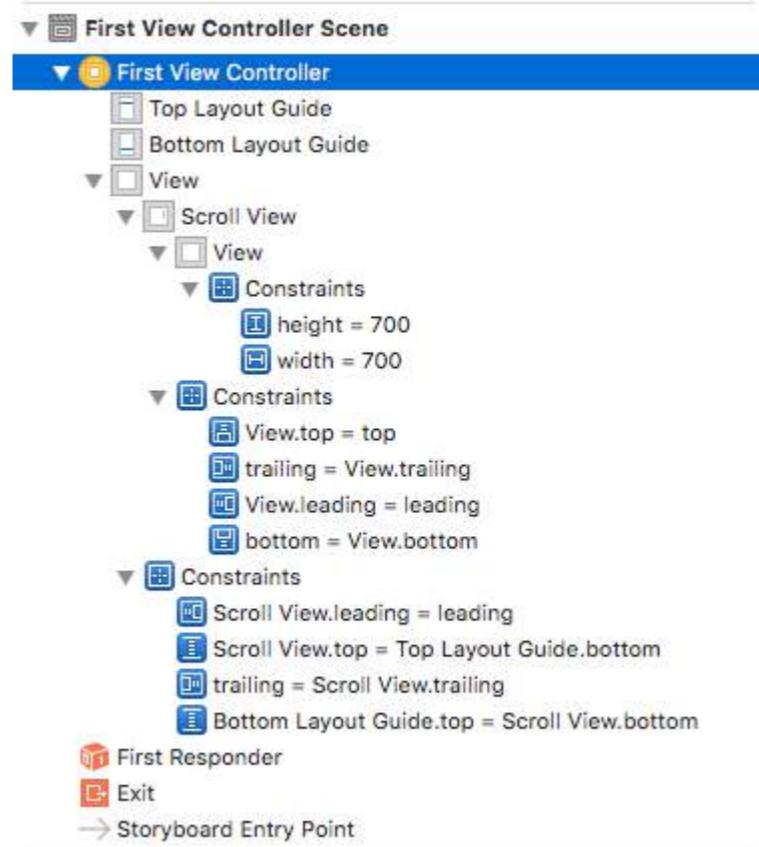
- And Done!!! Simply run and check if it scrolls vertically

### Scenario 3:

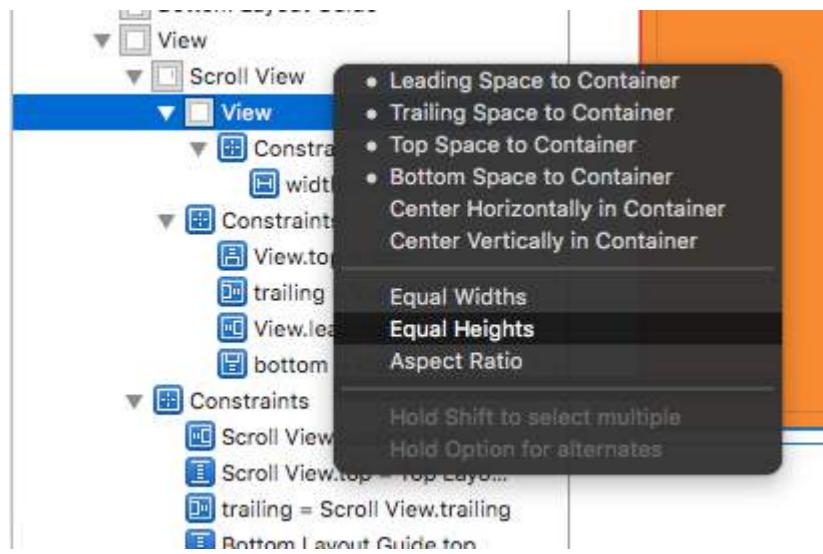
**Now we want to scroll only horizontally and not vertically.**

Follow the steps to horizontally scroll the content.

- Undo all the changes to achieve constraints as below(i.e **restore original constraints which achieved vertical and horizontal scroll**)



- Check frame of orange view, which should be (0,0,700,700)
- Delete height constraint of orange view.
- Change the height of orange view to match the scrollview height.
- Ctrl-drag from orange view to scroll view and add **equal heights** constraint.



- 完成！！！只需运行并检查是否可以垂直滚动

## 第29.4节：使用代理方法检测UIScrollView何时完成滚动

**scrollViewDidEndDecelerating:** 告诉代理滚动视图已结束减速滚动移动。

### Objective C:

```
- (void)scrollViewDidEndDecelerating:(UIScrollView *)scrollView {
    [self stoppedScrolling];
}

- (void)scrollViewDidEndDragging:(UIScrollView *)scrollView willDecelerate:(BOOL)decelerate {
    if (!decelerate) {
        [self stoppedScrolling];
    }
}

- (void)stoppedScrolling {
    // 完成, 执行相应操作
}
```

### Swift :

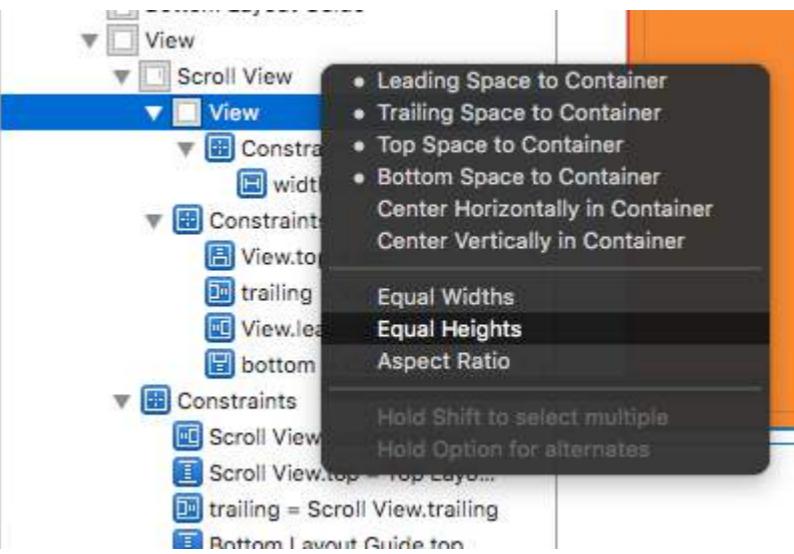
```
func scrollViewDidEndDragging(scrollView: UIScrollView, willDecelerate decelerate: Bool) {
    if !decelerate {
        stoppedScrolling()
    }
}

func scrollViewDidEndDecelerating(scrollView: UIScrollView) {
    stoppedScrolling()
}

func stoppedScrolling() {
    // 完成, 执行相应操作
}
```

## 第29.5节：启用/禁用滚动

属性 `scrollEnabled` 存储一个 Boolean 值，用于确定是否启用滚动。



- And Done!!! Simply run and check if it scrolls vertically

## Section 29.4: Detecting when UIScrollView finished scrolling with delegate methods

**scrollViewDidEndDecelerating:** this tells the delegate that the scroll view has ended decelerating the scrolling movement.

### Objective C:

```
- (void)scrollViewDidEndDecelerating:(UIScrollView *)scrollView {
    [self stoppedScrolling];
}

- (void)scrollViewDidEndDragging:(UIScrollView *)scrollView willDecelerate:(BOOL)decelerate {
    if (!decelerate) {
        [self stoppedScrolling];
    }
}

- (void)stoppedScrolling {
    // done, do whatever
}
```

### Swift:

```
func scrollViewDidEndDragging(scrollView: UIScrollView, willDecelerate decelerate: Bool) {
    if !decelerate {
        stoppedScrolling()
    }
}

func scrollViewDidEndDecelerating(scrollView: UIScrollView) {
    stoppedScrolling()
}

func stoppedScrolling() {
    // done, do whatever
}
```

## Section 29.5: Enable/Disable Scrolling

Property `scrollEnabled` stores a Boolean value that determines whether scrolling is enabled or not.

如果该属性的值为 true/YES，则启用滚动，否则不启用。默认值为 true

## Swift

```
scrollview.isScrollEnabled = true
```

## Objective-C

```
scrollview.scrollEnabled = YES;
```

# 第29.6节：UIImageView缩放

### 创建UIScrollView实例

```
let scrollview = UIScrollView.init(frame: self.view.bounds)
```

然后设置以下属性：

```
scrollView.minimumZoomScale = 0.1  
scrollView.maximumZoomScale = 4.0  
scrollView.zoomScale = 1.0  
scrollview.delegate = self as? UIScrollViewDelegate
```

为了实现图片的缩放，我们必须指定用户可以放大和缩小的范围。通过设置滚动视图的minimumZoomScale和maximumZoomScale属性来实现。默认这两个值均为1.0。

并将zoomScale设置为1.0，指定最小和最大缩放的缩放因子。

为了支持缩放，必须为滚动视图设置代理。代理对象必须遵守 `UIScrollViewDelegate` 协议。该代理类必须实现 `viewForZoomingInScrollView()` 方法并返回要缩放的视图。

按如下方式修改你的视图控制器

```
class ViewController: UIViewController, UIScrollViewDelegate
```

然后向类中添加以下代理函数。

```
func viewForZoomingInScrollView(scrollView: UIScrollView) -> UIView? {  
    return imageView  
}
```

### 现在创建 UIImageView 实例

将此变量设为类变量

```
var imageView:UIImageView = UIImageView.init(image: UIImage.init(named: "someImage.jpg"))
```

然后将其添加到 scrollview 中

```
scrollView?.addSubview(imageView)
```

### 参考文献

- [iOS滚动视图编程指南](#)

If the value of this property is true/YES, scrolling is enabled, otherwise not. The default value is true

## Swift

```
scrollview.isScrollEnabled = true
```

## Objective-C

```
scrollview.scrollEnabled = YES;
```

# Section 29.6: Zoom In/Out UIImageView

### Create UIScrollView instance

```
let scrollview = UIScrollView.init(frame: self.view.bounds)
```

And then set these properties:

```
scrollView.minimumZoomScale = 0.1  
scrollView.maximumZoomScale = 4.0  
scrollView.zoomScale = 1.0  
scrollview.delegate = self as? UIScrollViewDelegate
```

To zoom in and out image we must specify the amount the user can zoom in and out. We do this by setting values of the scroll view's `minimumZoomScale` and `maximumZoomScale` properties. Both of these are set to 1.0 by default.

And `zoomScale` to 1.0 which specify the zoom factor for the minimum and maximum zooming.

To support zooming, we must set a delegate for your scroll view. The delegate object must conform to the `UIScrollViewDelegate` protocol. That delegate class must implement the `viewForZoomingInScrollView()` method and return the view to zoom.

Modify your ViewController as shown

```
class ViewController: UIViewController, UIScrollViewDelegate
```

Then add the following delegate function to the class.

```
func viewForZoomingInScrollView(scrollView: UIScrollView) -> UIView? {  
    return imageView  
}
```

### Now create UIImageView instance

Make this variable as class variable

```
var imageView:UIImageView = UIImageView.init(image: UIImage.init(named: "someImage.jpg"))
```

And then add it to scrollview

```
scrollView?.addSubview(imageView)
```

### Reference

- [Scroll View Programming Guide for iOS](#)

## 第29.7节：滚动视图内容大小

contentSize 属性必须设置为可滚动内容的大小。这指定了可滚动区域，即contentSize大于UIScrollView的框架大小时，滚动才会可见。

### 使用自动布局：

当使用自动布局设置滚动视图的内容时，必须明确设置其垂直和水平方向的大小，并且将所有4个边缘固定到包含的滚动视图。这样，contentSize会根据滚动视图的内容自动计算，并且在内容布局更改时也会更新。

### 手动设置：

#### Swift

```
scrollview.contentSize = CGSize(width: 640, height: 800)
```

#### Objective-C

```
scrollview.contentSize = CGSizeMake(640, 800);
```

## 第29.8节：限制滚动方向

您可以使用以下代码限制用户能够滚动的方向：

```
func scrollViewDidScroll(_ scrollView: UIScrollView) {
    if scrollView.contentOffset.x != 0 {
        scrollView.contentOffset.x = 0
    }
}
```

每当用户在 x 轴上滚动时，scrollView 的内容偏移量会被重置为 0。  
您显然可以将 x 改为 y，从而将滚动方向锁定为仅水平。

您还需要确保将此代码放入 `scrollViewDidScroll(_ scrollView: UIScrollView)` 代理方法中。否则，代码将无法生效。

另外，确保在类声明中导入了 `UIScrollViewDelegate`，如下所示：

```
class ViewController: UIViewController, UIScrollViewDelegate
```

...并在某个方法中（如 `viewDidLoad(_:)`）将 scrollView 的代理设置为 `self`

```
scrollView.delegate = self
```

## Section 29.7: Scroll View Content Size

The `contentSize` property must be set to the size of the scrollable content. This specifies the size of the scrollable area. Scrolling is visible when scrollable area i.e. `contentSize` is larger than the `UIScrollView` frame size.

### With Autolayout:

When the scroll view's content is set up using autolayout, it must be explicitly sized both vertically and horizontally and have all 4 edges pinned to the containing scroll view. That way, the `contentSize` is calculated automatically based on the scroll view's contents and also gets updated when the content's layout is changed.

### Manually:

#### Swift

```
scrollview.contentSize = CGSize(width: 640, height: 800)
```

#### Objective-C

```
scrollview.contentSize = CGSizeMake(640, 800);
```

## Section 29.8: Restrict scrolling direction

You can restrict the directions the user is able to scroll to using the following code:

```
func scrollViewDidScroll(_ scrollView: UIScrollView) {
    if scrollView.contentOffset.x != 0 {
        scrollView.contentOffset.x = 0
    }
}
```

Every time the user scrolls on the x-axis, the scrollview's content offset is set back to 0.  
You can obviously change the xs to ys and therefore lock the direction to be horizontal-only.

You also need to make sure you put this code into the `scrollViewDidScroll(_ scrollView: UIScrollView)` delegate method. Otherwise, you won't get it to work.

Also, be sure to have imported the `UIScrollViewDelegate` in your class declaration, like so:

```
class ViewController: UIViewController, UIScrollViewDelegate
```

...and set the scrollview's delegate to self in some method like `viewDidLoad(_:)`

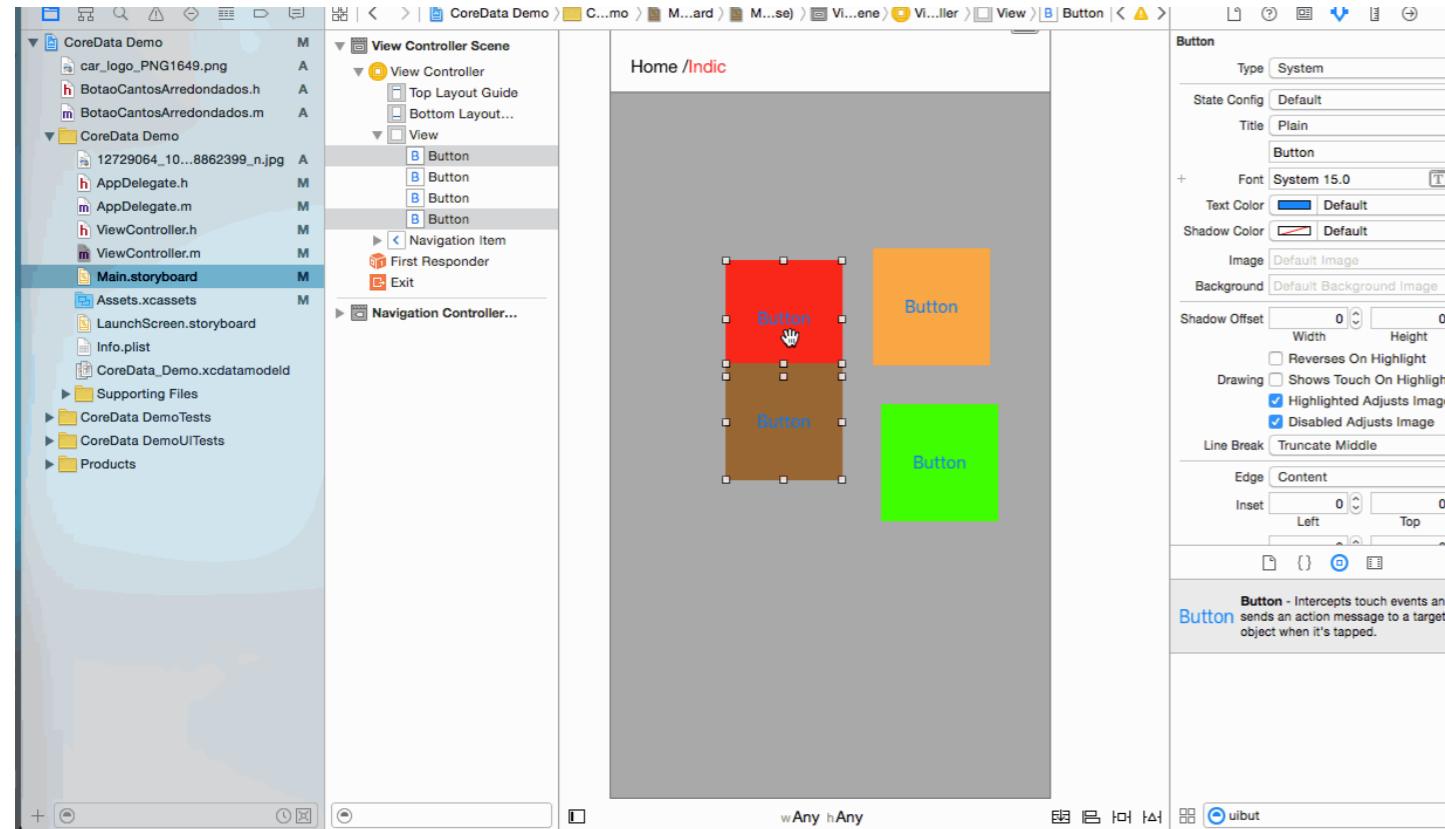
```
scrollView.delegate = self
```

# 第30章：UIStackView

## 第30.1节：使用UIStackview的中心按钮

步骤1：在你的Storyboard中放置4个按钮。按钮1，按钮2，按钮3，按钮4

步骤2：给所有按钮设置固定的高度和宽度。



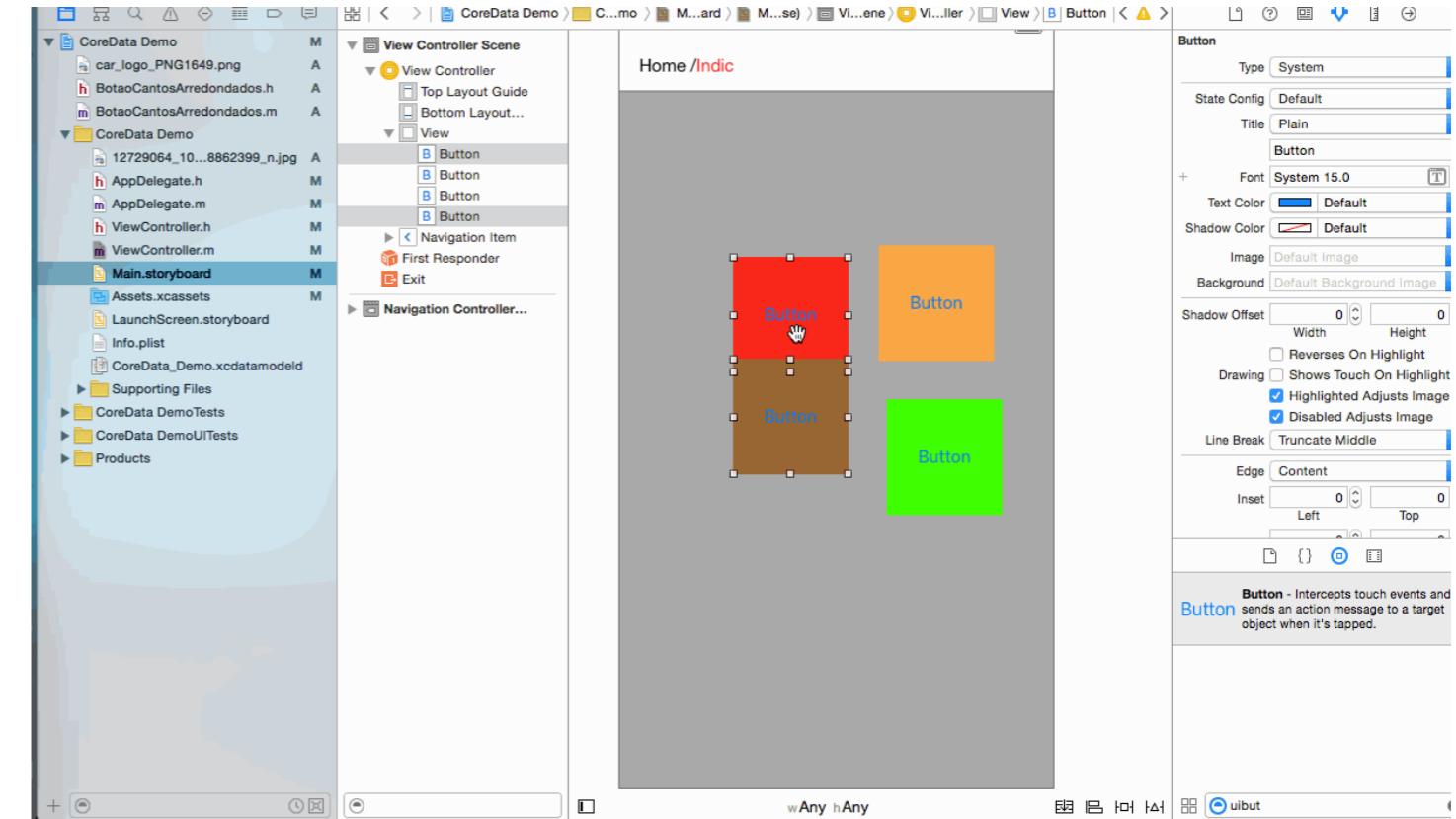
步骤3：将所有按钮两两分成2个堆栈视图（stackview）。

# Chapter 30: UIStackView

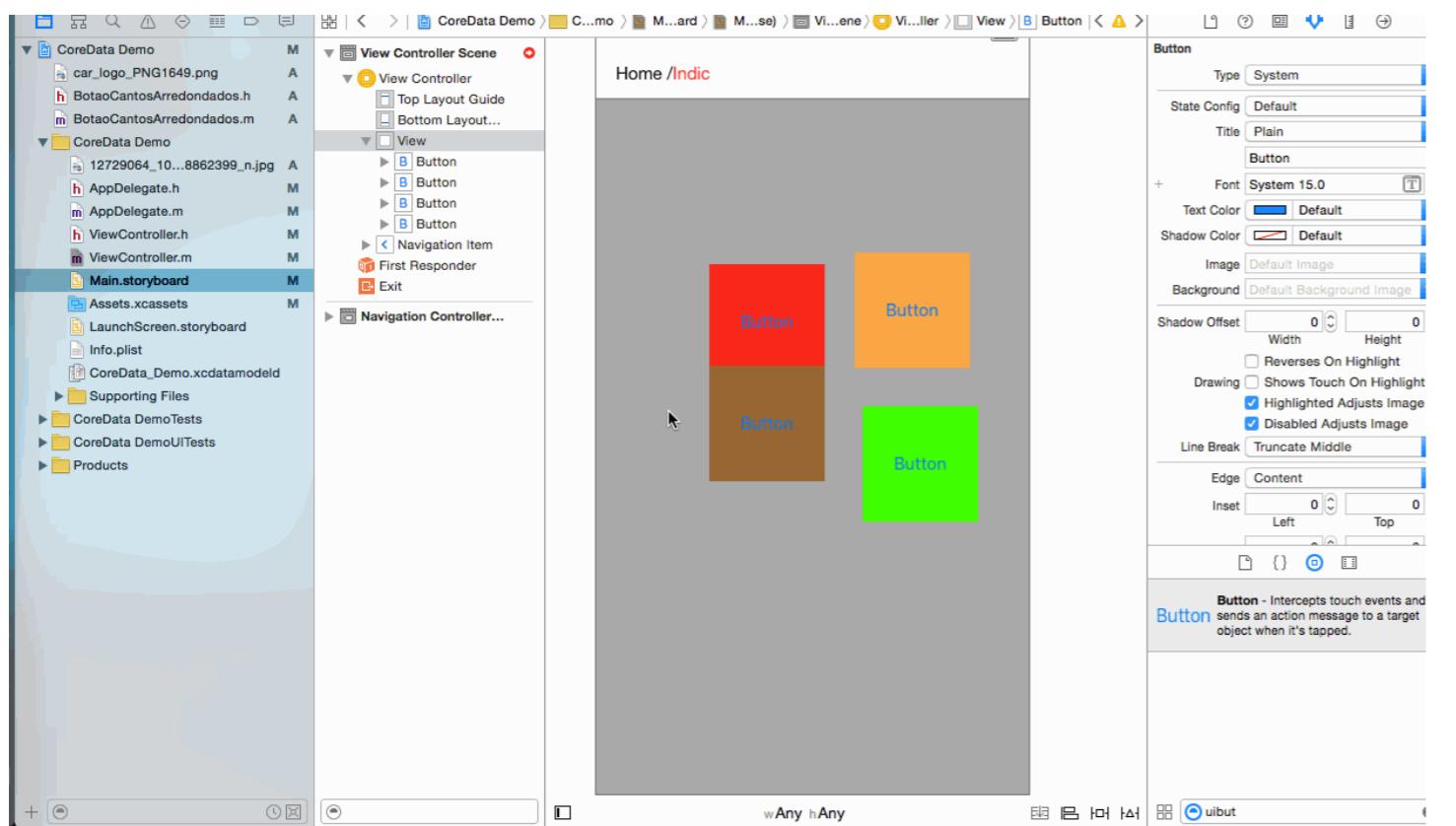
## Section 30.1: Center Buttons with UIStackview

**Step 1:** take 4 button in your Storyboard. Button1 , Button2 , Button 3 , Button4

**Step 2:** Give Fixed Height and width to All buttons .



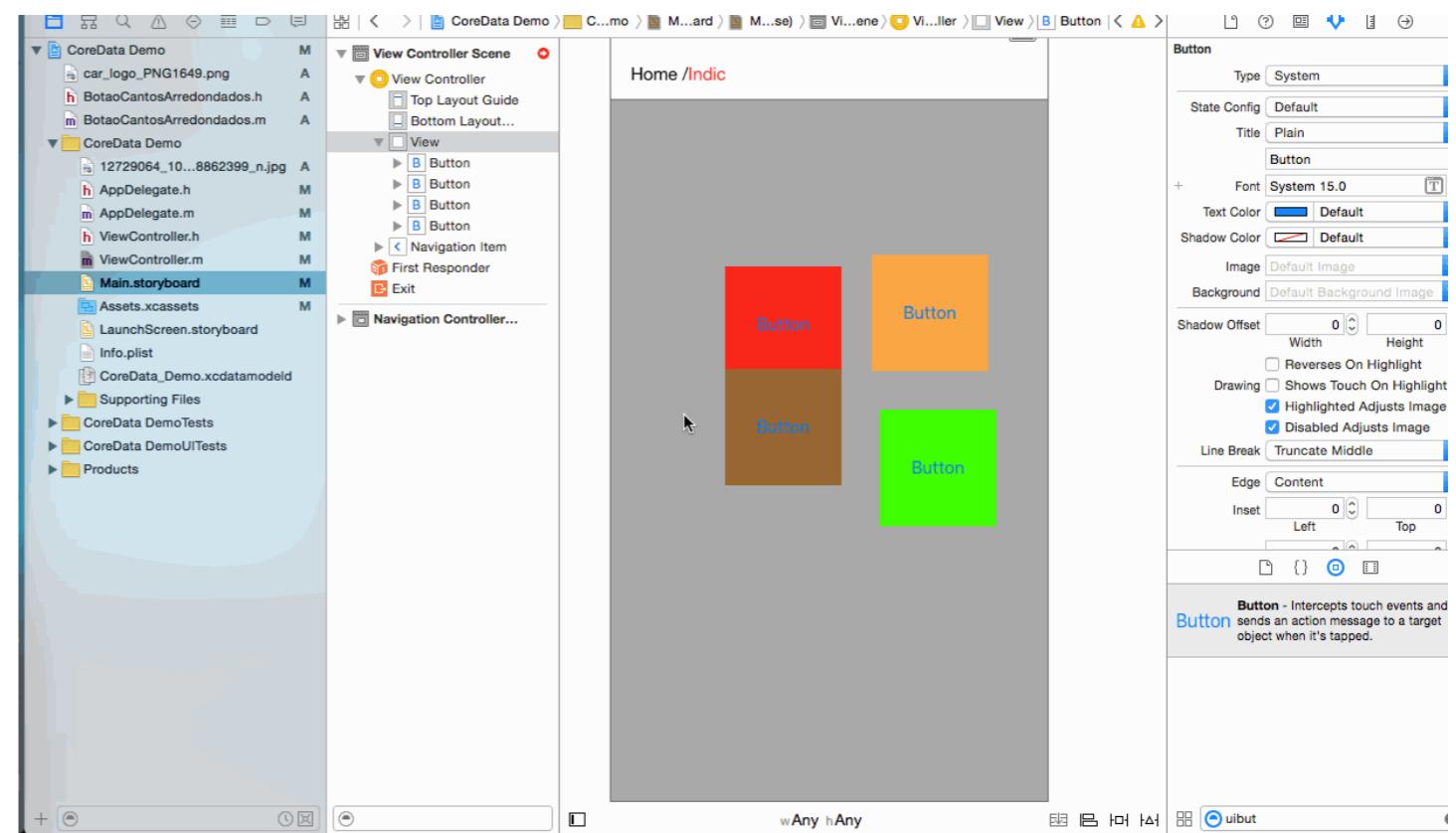
**Step 3:** All 2 - 2 button's pair in 2 stackview.



步骤4：设置两个UIStackview的属性。

Distribution -> 等分填充

Spacing -> 5 (根据你的需求)

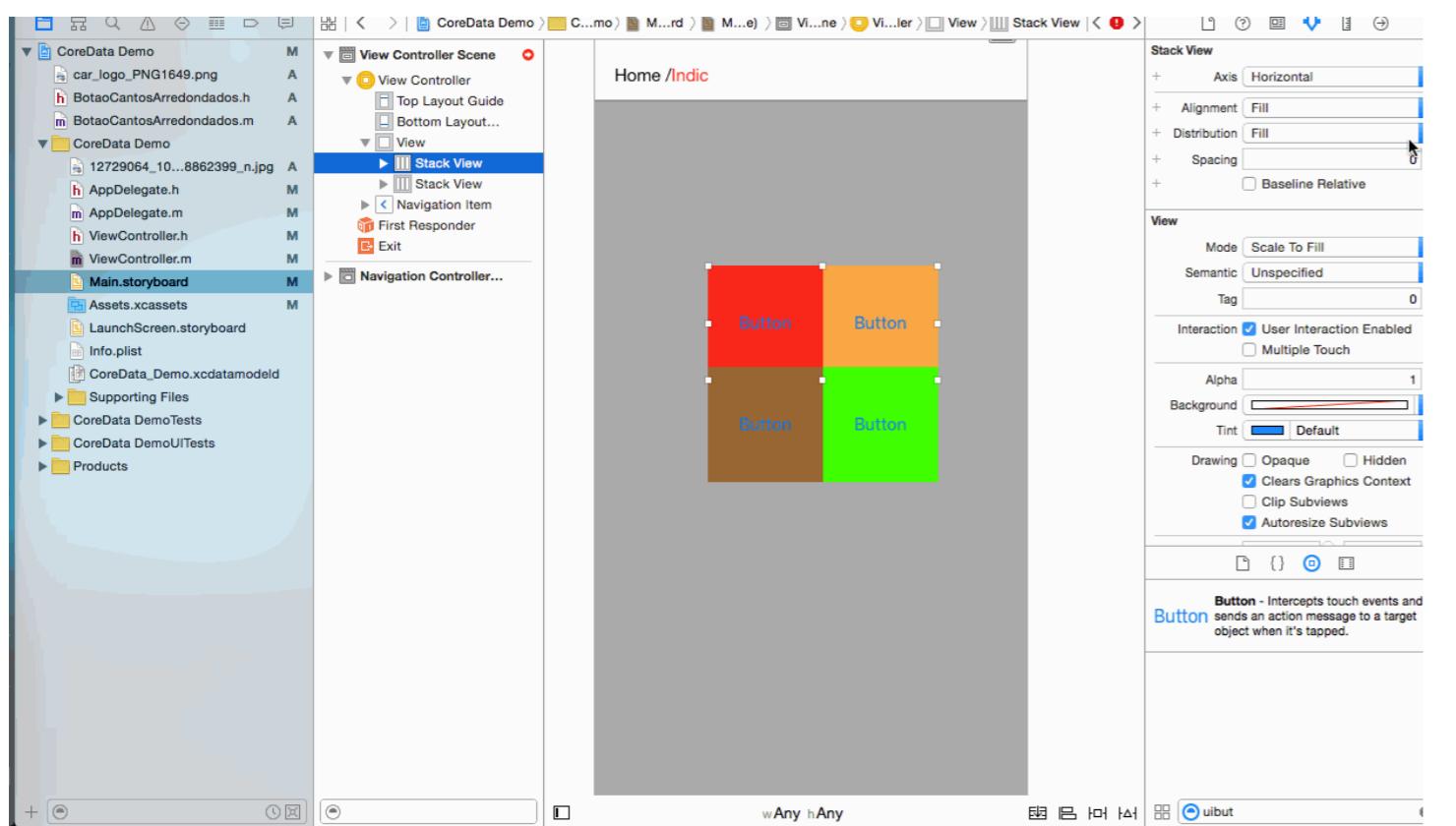


Step 4: Set UIStackview Property for both .

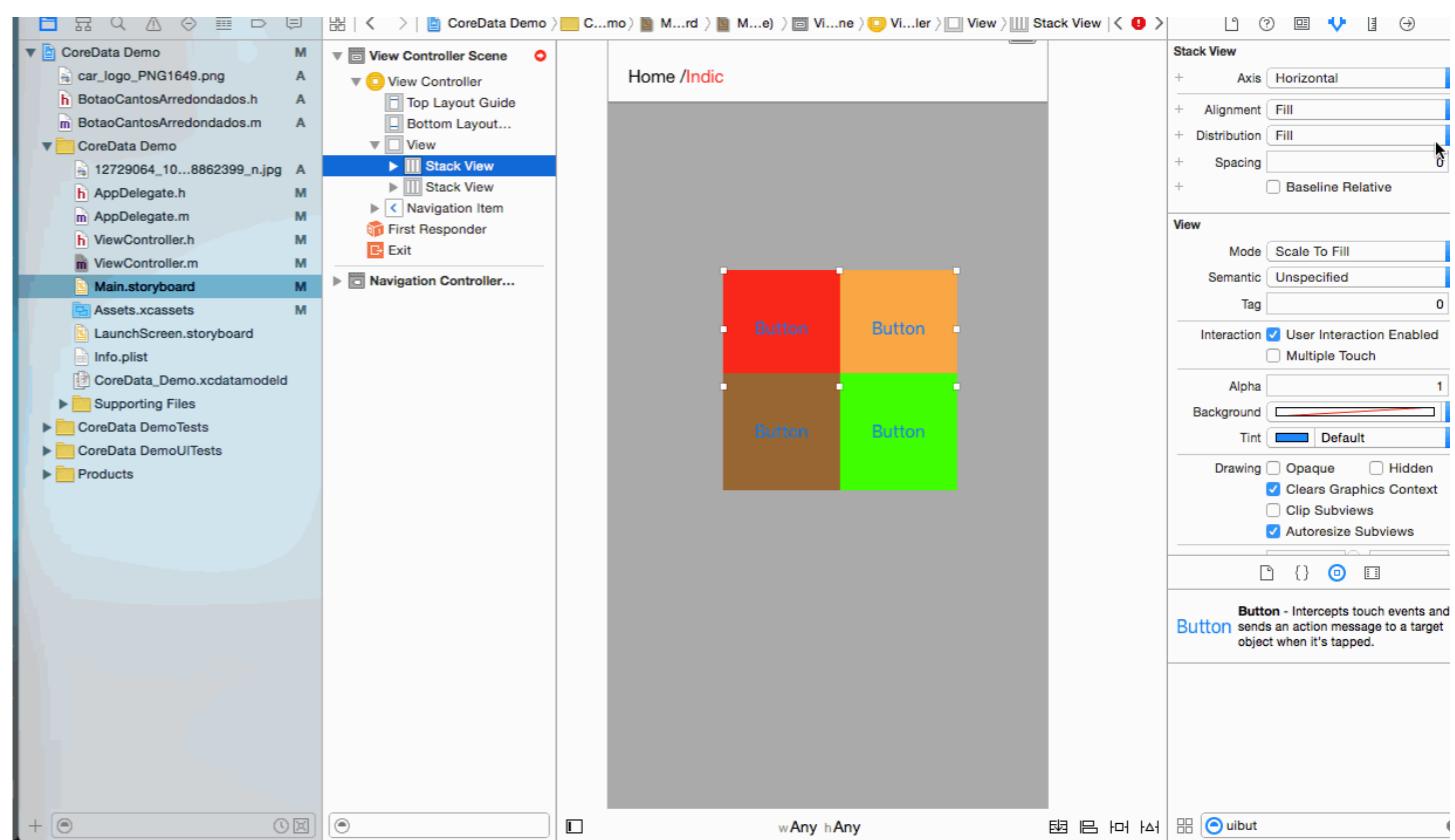
Distribution -> Fill Equally

Spacing -> 5 (as per your requirement)

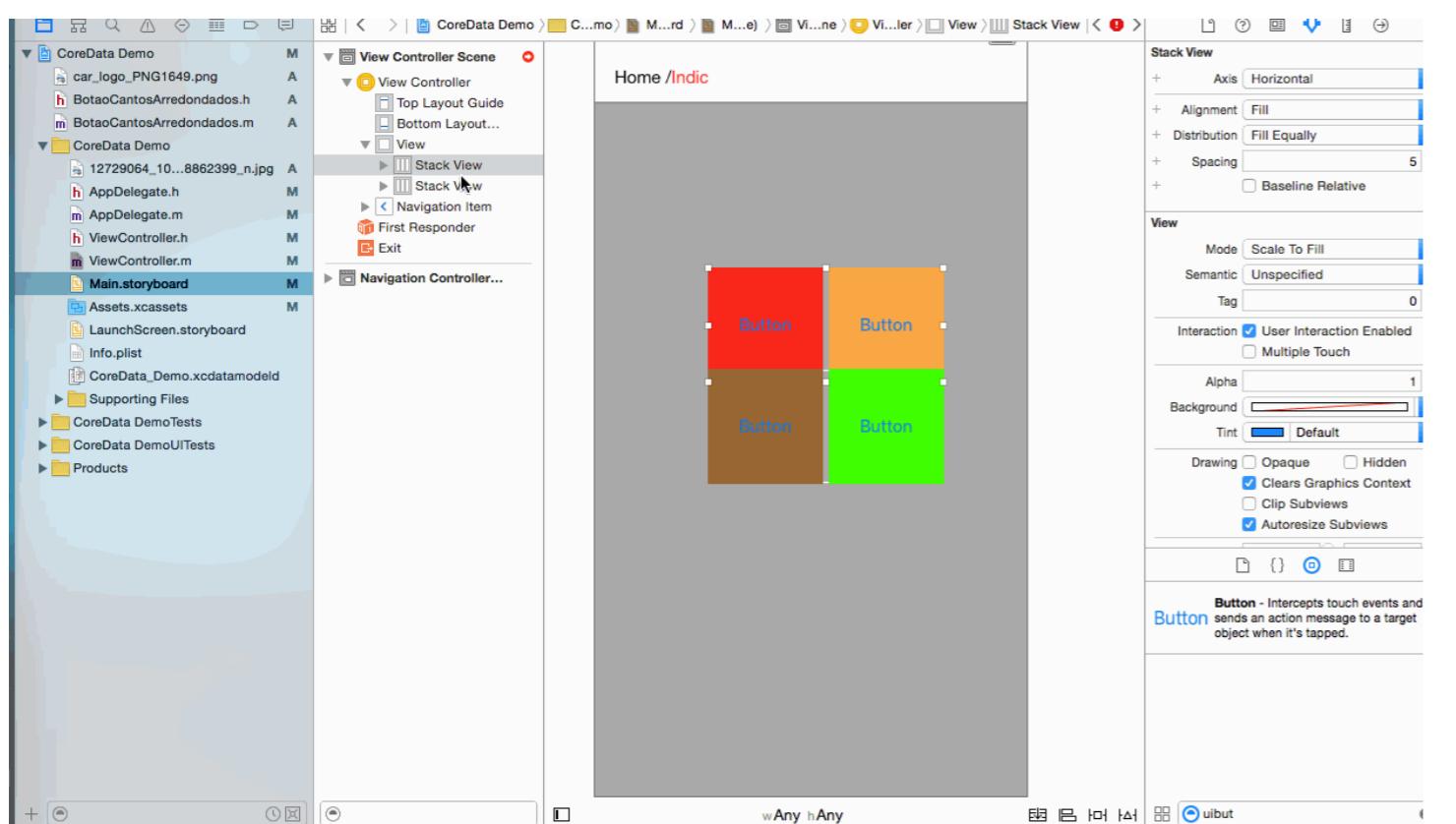




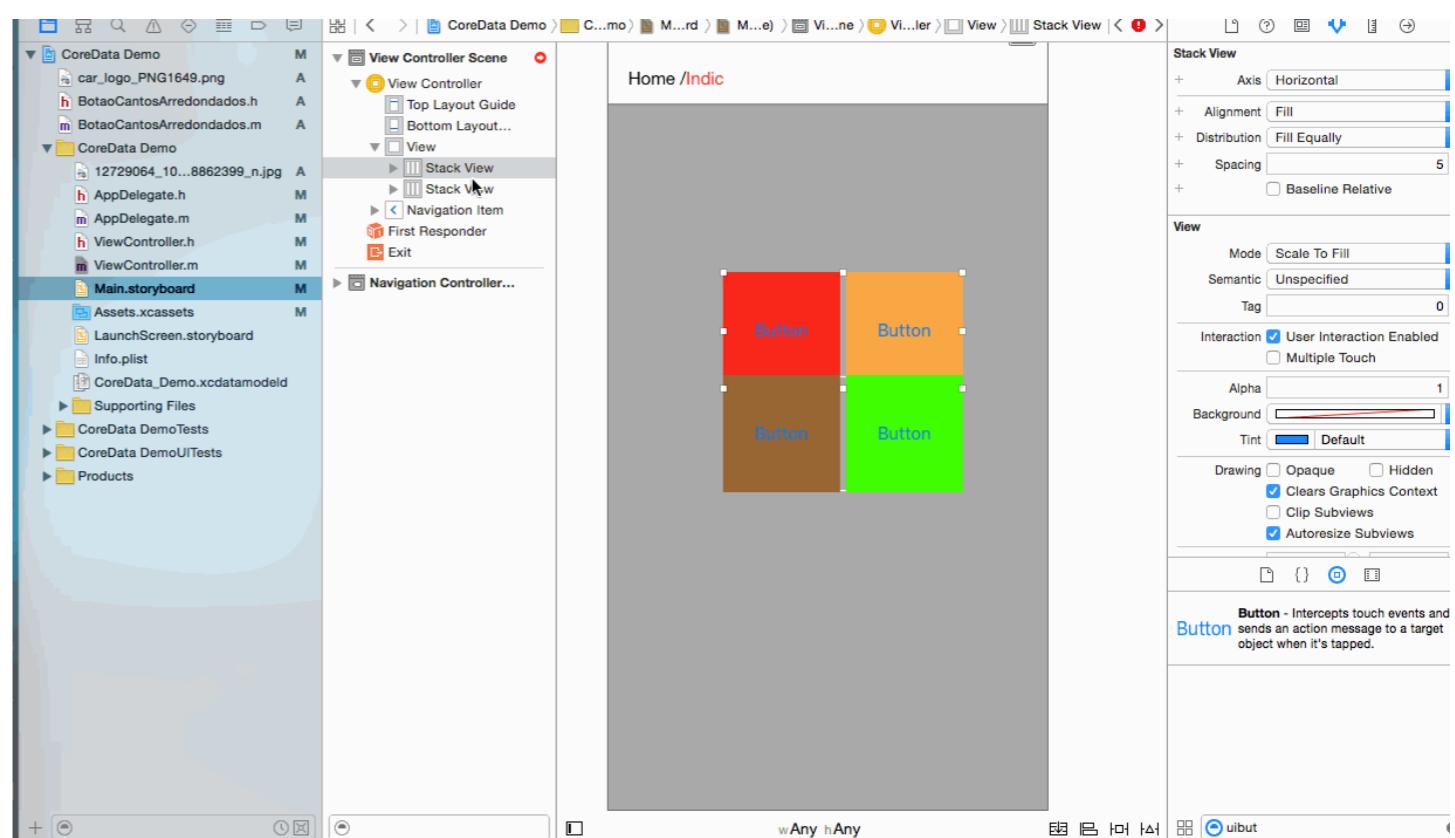
步骤5：将两个堆栈视图添加到一个堆栈视图中



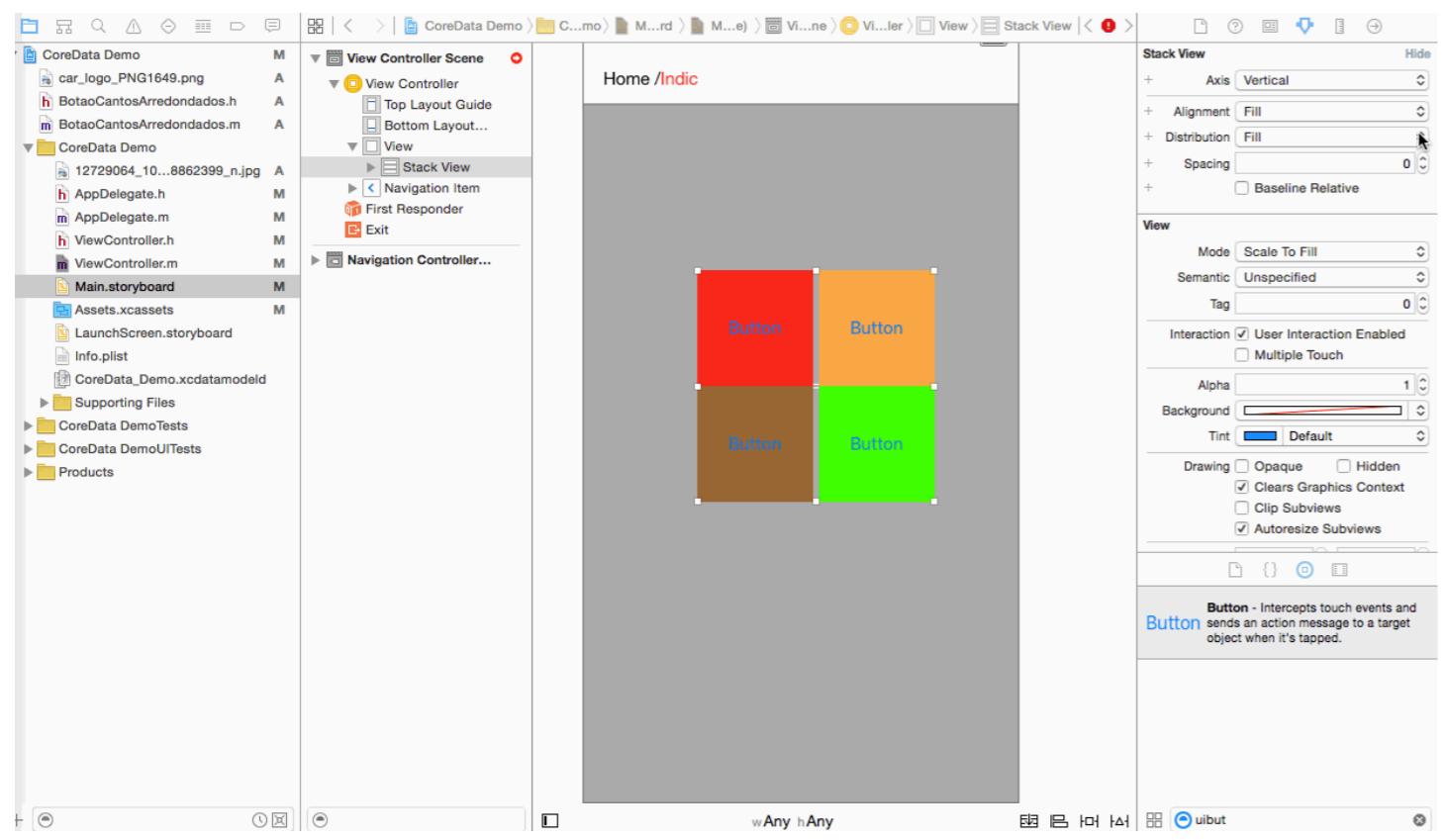
Step 5: Add both Stackview in one Stackview



步骤6：在主堆栈视图中设置Distribution = 等分填充 Spacing =5 (根据你的需求设置)



Step 6: Set Distribution = Fill equally Spacing =5 in main stackview (set According to your requirement)

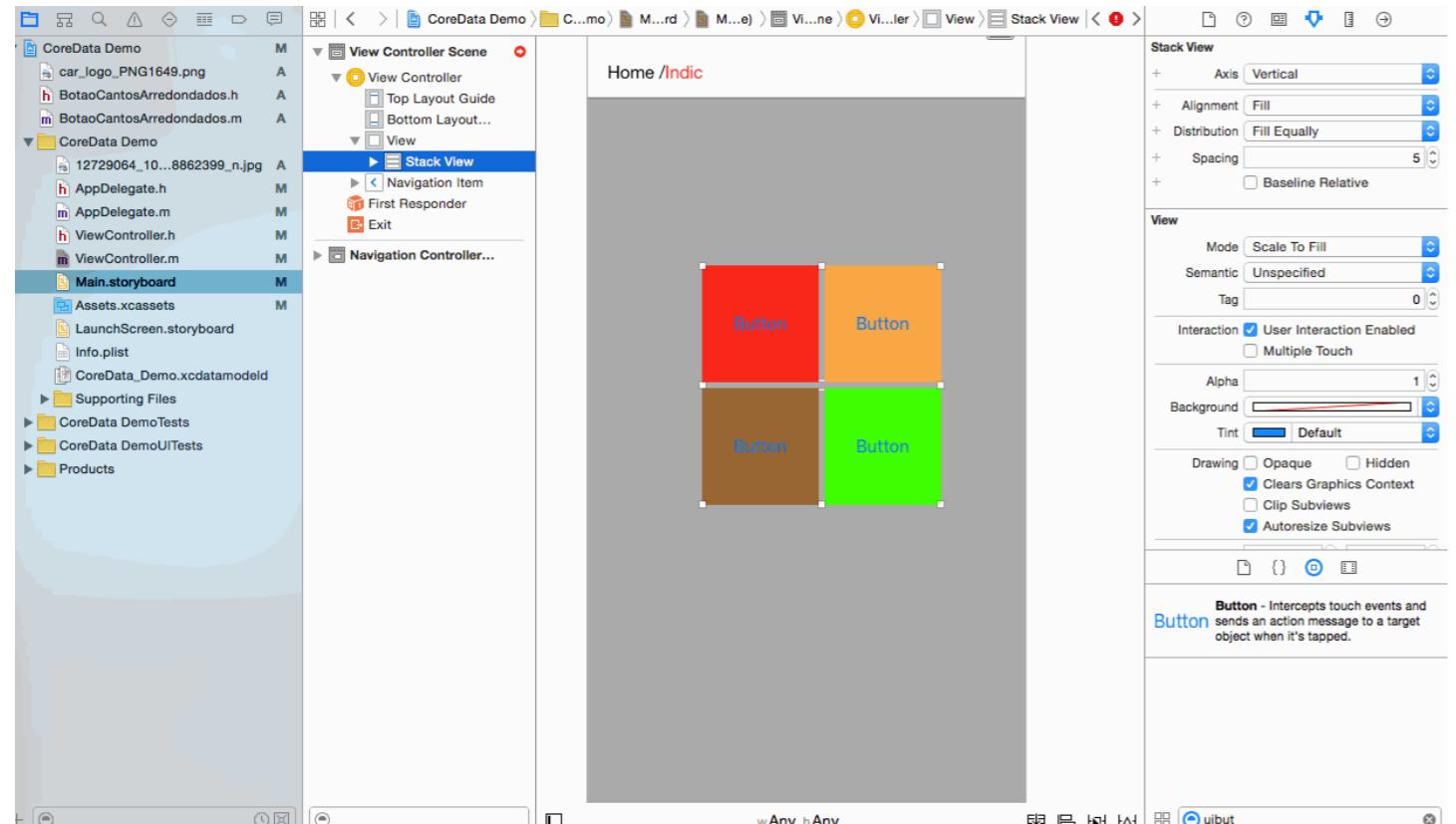


步骤7：现在将约束设置到主堆栈视图

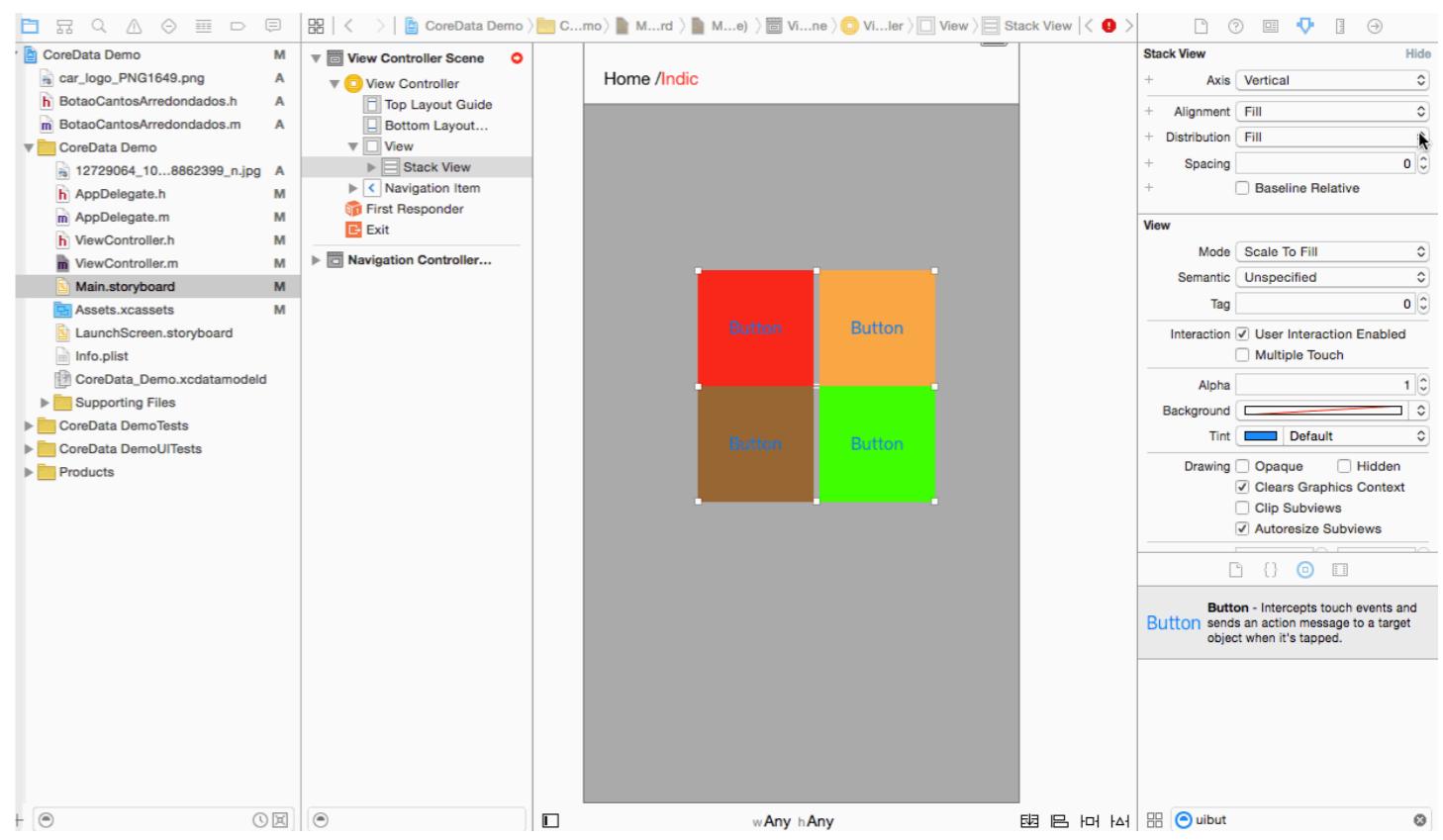
在容器中水平居中

在容器中垂直居中

并选择更新框架。



步骤8：现在是为所有设备输出的时候了。

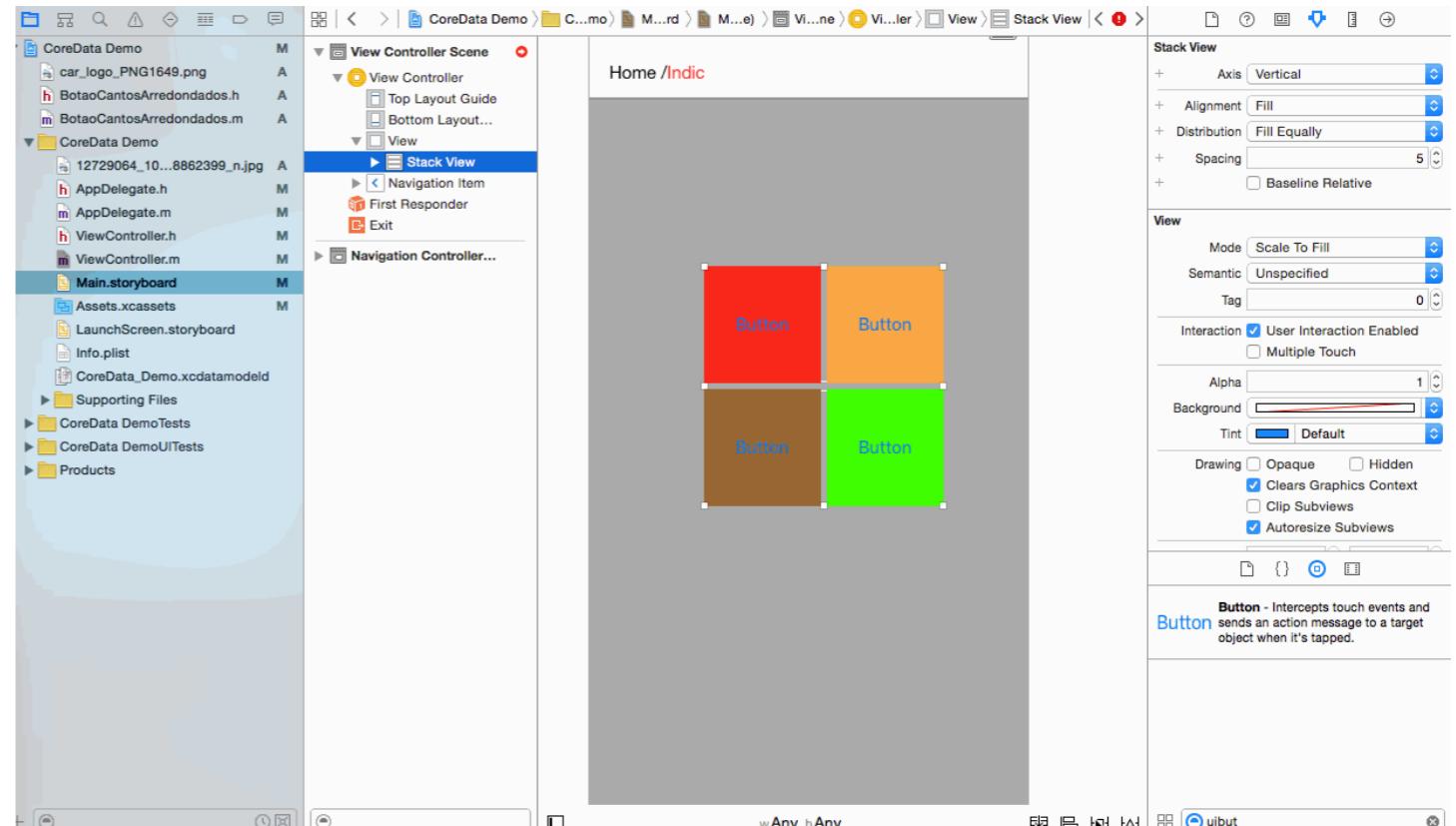


Step 7: Now set Constraint to main stackview

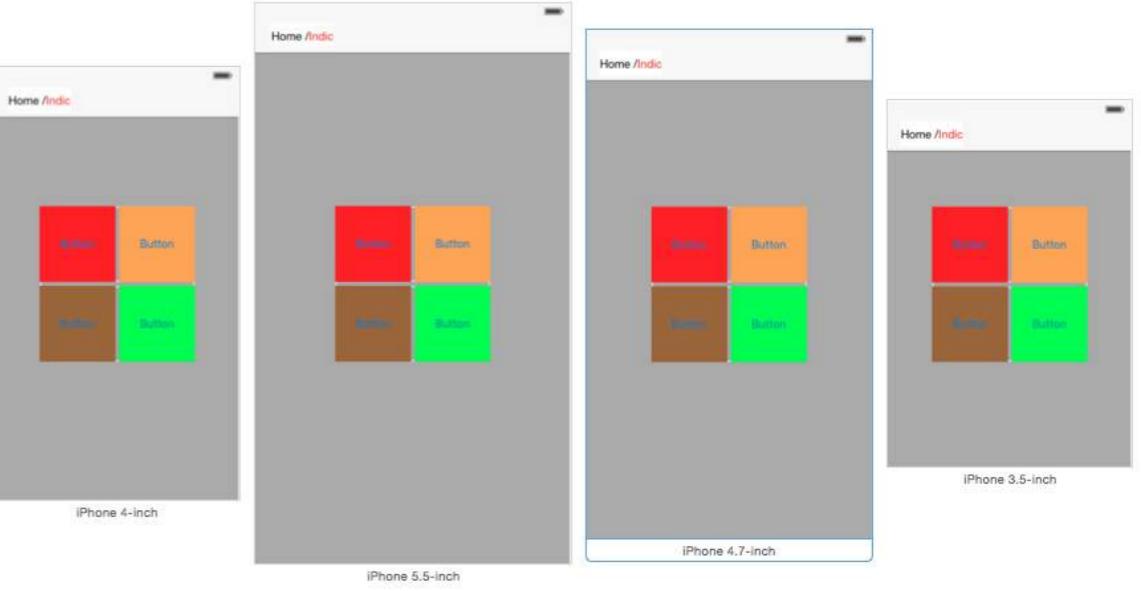
center Horizontally in container

center vertically in container

and select Update Frame.



Step 8: It's time for Output for All device .



## 第30.2节：以编程方式创建水平堆栈视图

### Swift 3

```
let stackView = UIStackView()
stackView.axis = .horizontal
stackView.alignment = .fill // .leading .firstBaseline .center .trailing .lastBaseline
stackView.distribution = .fill // .fillEqually .fillProportionally .equalSpacing .equalCentering

let label = UILabel()
label.text = "Text"
stackView.addArrangedSubview(label)
// 对于水平堆栈视图，你可能需要为标签或你添加的任何视图添加宽度约束。
```

### Swift

```
let stackView = UIStackView()
stackView.axes = .horizontal
stackView.alignment = .Fill // .Leading .FirstBaseline .Center .Trailing .LastBaseline
stackView.distribution = .Fill // .FillEqually .FillProportionally .EqualSpacing .EqualCentering

let label = UILabel(frame: CGRectZero)
label.text = "Label"
stackView.addArrangedSubview(label)
// 对于水平堆栈视图，你可能需要为标签或你添加的任何视图添加宽度约束。
```

### Objective-C

```
UIStackView *stackView = [[UIStackView alloc] init];
stackView.axis = UILayoutConstraintAxisHorizontal;
stackView.alignment = UIStackViewAlignmentFill; //UIStackViewAlignmentLeading,
UIStackViewAlignmentFirstBaseline, UIStackViewAlignmentCenter, UIStackViewAlignmentTrailing,
UIStackViewAlignmentLastBaseline
stackView.distribution = UIStackViewDistributionFill; //UIStackViewDistributionFillEqually,
```

## Section 30.2: Create a horizontal stack view programmatically

### Swift 3

```
let stackView = UIStackView()
stackView.axis = .horizontal
stackView.alignment = .fill // .leading .firstBaseline .center .trailing .lastBaseline
stackView.distribution = .fill // .fillEqually .fillProportionally .equalSpacing .equalCentering

let label = UILabel()
label.text = "Text"
stackView.addArrangedSubview(label)
// for horizontal stack view, you might want to add width constraint to label or whatever view
you're adding.
```

### Swift

```
let stackView = UIStackView()
stackView.axis = .Horizontal
stackView.alignment = .Fill // .Leading .FirstBaseline .Center .Trailing .LastBaseline
stackView.distribution = .Fill // .FillEqually .FillProportionally .EqualSpacing .EqualCentering

let label = UILabel(frame: CGRectZero)
label.text = "Label"
stackView.addArrangedSubview(label)
// for horizontal stack view, you might want to add width constraint to label or whatever view
you're adding.
```

### Objective-C

```
UIStackView *stackView = [[UIStackView alloc] init];
stackView.axis = UILayoutConstraintAxisHorizontal;
stackView.alignment = UIStackViewAlignmentFill; //UIStackViewAlignmentLeading,
UIStackViewAlignmentFirstBaseline, UIStackViewAlignmentCenter, UIStackViewAlignmentTrailing,
UIStackViewAlignmentLastBaseline
stackView.distribution = UIStackViewDistributionFill; //UIStackViewDistributionFillEqually,
```

```
UIStackViewDistributionFillProportionally, UIStackViewDistributionEqualSpacing,  
UIStackViewDistributionEqualCentering
```

```
UILabel *label = [[UILabel alloc] initWithFrame:CGRectMakeZero];  
label.text = @"Label";  
[stackView addArrangedSubview:label];  
//对于水平堆叠视图，您可能需要为您的标签或您添加的任何视图添加宽度约束。
```

## 第30.3节：通过代码创建垂直堆叠视图

### Swift

```
let stackView = UIStackView()  
stackView.axis = .Vertical  
stackView.alignment = .Fill // .Leading .FirstBaseline .Center .Trailing .LastBaseline  
stackView.distribution = .Fill // .FillEqually .FillProportionally .EqualSpacing .EqualCentering  
  
let label = UILabel(frame: CGRectZero)  
label.text = "Label"  
stackView.addArrangedSubview(label)  
// 对于垂直堆叠视图，您可能需要为标签或您添加的任何视图添加高度约束。
```

### Objective-C

```
UIStackView *stackView = [[UIStackView alloc] init];  
stackView.axis = UILayoutConstraintAxisVertical;  
stackView.alignment = UIStackViewAlignmentFill; //UIStackViewAlignmentLeading,  
UIStackViewAlignmentFirstBaseline, UIStackViewAlignmentCenter, UIStackViewAlignmentTrailing,  
UIStackViewAlignmentLastBaseline  
stackView.distribution = UIStackViewDistributionFill; //UIStackViewDistributionFillEqually,  
UIStackViewDistributionFillProportionally, UIStackViewDistributionEqualSpacing,  
UIStackViewDistributionEqualCentering  
  
UILabel *label = [[UILabel alloc] initWithFrame:CGRectMakeZero];  
label.text = @"Label";  
[stackView addArrangedSubview:label];  
// 对于垂直堆叠视图，您可能需要为标签或您添加的任何视图添加高度约束。
```

```
UIStackViewDistributionFillProportionally, UIStackViewDistributionEqualSpacing,  
UIStackViewDistributionEqualCentering
```

```
UILabel *label = [[UILabel alloc] initWithFrame:CGRectMakeZero];  
label.text = @"Label";  
[stackView addArrangedSubview:label];  
//For horizontal stack view, you might want to add a width constraint to your label or whatever  
view you are adding.
```

## Section 30.3: Create a vertical stack view programmatically

### Swift

```
let stackView = UIStackView()  
stackView.axis = .Vertical  
stackView.alignment = .Fill // .Leading .FirstBaseline .Center .Trailing .LastBaseline  
stackView.distribution = .Fill // .FillEqually .FillProportionally .EqualSpacing .EqualCentering  
  
let label = UILabel(frame: CGRectZero)  
label.text = "Label"  
stackView.addArrangedSubview(label)  
// for vertical stack view, you might want to add height constraint to label or whatever view  
you're adding.
```

### Objective-C

```
UIStackView *stackView = [[UIStackView alloc] init];  
stackView.axis = UILayoutConstraintAxisVertical;  
stackView.alignment = UIStackViewAlignmentFill; //UIStackViewAlignmentLeading,  
UIStackViewAlignmentFirstBaseline, UIStackViewAlignmentCenter, UIStackViewAlignmentTrailing,  
UIStackViewAlignmentLastBaseline  
stackView.distribution = UIStackViewDistributionFill; //UIStackViewDistributionFillEqually,  
UIStackViewDistributionFillProportionally, UIStackViewDistributionEqualSpacing,  
UIStackViewDistributionEqualCentering  
  
UILabel *label = [[UILabel alloc] initWithFrame:CGRectMakeZero];  
label.text = @"Label";  
[stackView addArrangedSubview:label];  
//For vertical stack view, you might want to add a height constraint to your label or whatever view  
you are adding.
```

# 第31章：动态更新 UIStackView

第31.1节：将UISwitch连接到一个动作，我们可以通过动画切换图像视图的水平或垂直布局

```
@IBAction func axisChange(sender: UISwitch) {  
    UIView.animateWithDuration(1.0) {  
        self.updateConstraintsForAxis()  
    }  
}
```

updateConstraintForAxis函数仅设置包含两个图像视图的堆叠视图的轴方向：

```
private func updateConstraintsForAxis() {  
    if (axisSwitch.on) {  
        stackView.axis = .Horizontal  
    } else {  
        stackView.axis = .Vertical  
    }  
}
```

下面的动画GIF可以让你了解其显示效果：



# Chapter 31: Dynamically updating a UIStackView

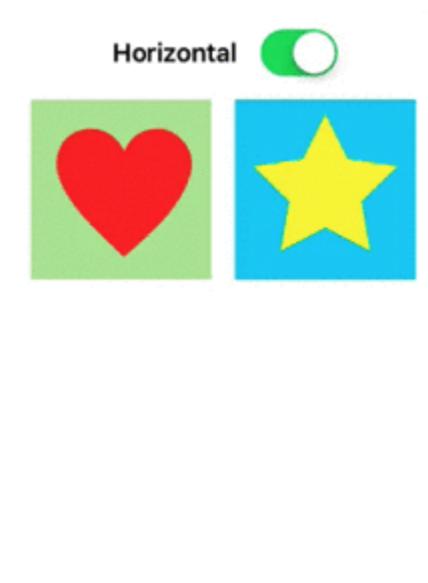
Section 31.1: Connect the UISwitch to an action we can animate switching between a horizontal or vertical layout of the image views

```
@IBAction func axisChange(sender: UISwitch) {  
    UIView.animateWithDuration(1.0) {  
        self.updateConstraintsForAxis()  
    }  
}
```

The updateConstraintForAxis function just sets the axis of the stack view containing the two image views:

```
private func updateConstraintsForAxis() {  
    if (axisSwitch.on) {  
        stackView.axis = .Horizontal  
    } else {  
        stackView.axis = .Vertical  
    }  
}
```

The animated gif below gives you an idea of how this appears:



# 第32章：带有StackView子视图的UIScrollView

## 第32.1节：ScrollView内的复杂StackView示例

下面是一个嵌套StackView的示例，给用户一种使用复杂用户界面元素或对齐方式实现连续滚动体验的感觉。

# Chapter 32: UIScrollView with StackView child

## Section 32.1: A complex StackView inside Scrollview Example

Here follows a example of what can be done with nested StackViews, giving the user the impression of a continuous scrolling experience using complex user interface elements or alignments.



## 第32.2节：防止布局歧义

关于ScrollView内StackView的一个常见问题是界面构建器中出现宽度/高度歧义警告。正如这个回答所解释的，必须  
：

## Section 32.2: Preventing ambiguous layout

A frequent question about StackViews inside Scrollviews comes from ambiguous width/height alerts on interface builder. As [this answer](#) explained, it is necessary to:

- 在UIScrollView中添加一个UIView (contentScrollView)；
- 在该contentScrollView中，将上下左右边距设置为0
- 还要设置水平和垂直居中对齐；

## 第32.3节：滚动到嵌套StackView内的内容

滚动的关键是确定偏移量，以便显示（例如）位于ScrollView内的StackView中的Textfield。

如果你尝试获取Textfield.frame.minY的位置，可能会是0，因为minY只考虑元素与StackView顶部之间的距离。所以你必须考虑所有其他父级stackviews/views。

一个好的解决方法是：

1 - 实现ScrollView扩展

```
extension UIScrollView {
    func scrollToShowView(view: UIView) {
        var offset = view.frame.minY
        var superview = view.superview
        while((superview != nil)){
            offset += (superview?.frame.minY)!
            superview = superview?.superview
        }
        offset -= 100 //可选的偏移量边距
        self.contentOffset = CGPoint.init(x: 0, y: offset)
    }
}
```

这将考虑所有父视图并累加滚动视图显示所需视图的必要偏移量（例如一个不能被用户键盘遮挡的Textfield）。

使用示例：

```
func textViewDidBeginEditing(_ textView: UITextView) {
    self.contentView.scrollToShowView(view: textView)
}
```

- Add in the UIScrollView a UIView (the contentScrollView);
- In this contentScrollView, set top, bottom, left and right margins to 0
- Set also align center horizontally and vertically;

## Section 32.3: Scrolling to content inside nested StackViews

The big gotcha about scrolling is to determine the offset necessary to present (for instance) a **Textfield inside a StackView with is inside the ScrollView**.

If you try to get **the position of Textfield.frame.minY can be 0**, because the minY frame is only considering the distance between the element and the top of the StackView. So you have to **consider all other parent stackviews/views**.

A good workaround for this is:

1 - Implement the ScrollView Extension

```
extension UIScrollView {
    func scrollToShowView(view: UIView) {
        var offset = view.frame.minY
        var superview = view.superview
        while((superview != nil)){
            offset += (superview?.frame.minY)!
            superview = superview?.superview
        }
        offset -= 100 //optional margin added on offset
        self.contentOffset = CGPoint.init(x: 0, y: offset)
    }
}
```

This will consider all parent view and sum the necessary offset for the scrollview present the necessary view on screen (for example a Textfield which cannot stay behind the user keyboard)

Usage example:

```
func textViewDidBeginEditing(_ textView: UITextView) {
    self.contentView.scrollToShowView(view: textView)
}
```

# 第33章：UIScrollView自动布局

## 第33.1节：ScrollableController

使用UIScrollView的自动布局时，它不会根据内容或

子视图的大小正确调整尺寸。

为了让UIScrollView在内容过大无法完全显示时自动滚动，  
我们需要添加一个ContentView和一些约束，使UIScrollView能够确定其  
内容的大小以及其在父视图中的宽度和高度。

```
import Foundation
import UIKit

class ScrollableController : UIViewController {

    private var scrollView: UIScrollView!
    private var contentView: UIView!

    override func viewDidLoad() {
        super.viewDidLoad()

        //设置
        self.initControls()
        self.setTheme()
        self.layoutScrollView()
        self.layoutContentView()

        //添加子视图
        self.addChildViews()
    }

    func initControls() {
        self.scrollView = UIScrollView()
        self.contentView = UIView()
    }

    func setTheme() {
        self.scrollView.backgroundColor = UIColor.blue()
        self.contentView.backgroundColor = UIColor.orange()
    }

    func layoutScrollView() {
        self.view.addSubview(self.scrollView)

        let views: NSDictionary = ["scrollView": self.scrollView]
        var constraints = Array<String>()

        //将scrollView约束到控制器的self.view上。
        constraints.append("H:|-0-[scrollView]-0-|")
        constraints.append("V:|-0-[scrollView]-0-|")

        for 约束 in 约束集合 {
            self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat: 约束,
options: NSLayoutFormatOptions(rawValue: 0), metrics: nil, views: views as! [String : AnyObject]))
        }

        self.scrollView.translatesAutoresizingMaskIntoConstraints = false
    }
}
```

# Chapter 33: UIScrollView AutoLayout

## Section 33.1: ScrollableController

When using Autolayout with a UIScrollView, it does NOT resize properly depending on the size of its contents or subviews.

In order to get a UIScrollView to automatically scroll when its contents become too large to fit in the visible area, we need to add a ContentView and some constraints that allow the UIScrollView to determine the size of its contents AND its width and height in its parent view.

```
import Foundation
import UIKit

class ScrollableController : UIViewController {

    private var scrollView: UIScrollView!
    private var contentView: UIView!

    override func viewDidLoad() {
        super.viewDidLoad()

        //Setup
        self.initControls()
        self.setTheme()
        self.layoutScrollView()
        self.layoutContentView()

        //Add child views
        self.addChildViews()
    }

    func initControls() {
        self.scrollView = UIScrollView()
        self.contentView = UIView()
    }

    func setTheme() {
        self.scrollView.backgroundColor = UIColor.blue()
        self.contentView.backgroundColor = UIColor.orange()
    }

    func layoutScrollView() {
        self.view.addSubview(self.scrollView)

        let views: NSDictionary = ["scrollView": self.scrollView]
        var constraints = Array<String>()

        //Constrain the scrollView to our controller's self.view.
        constraints.append("H:|-0-[scrollView]-0-|")
        constraints.append("V:|-0-[scrollView]-0-|")

        for constraint in constraints {
            self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat: constraint,
options: NSLayoutFormatOptions(rawValue: 0), metrics: nil, views: views as! [String : AnyObject]))
        }

        self.scrollView.translatesAutoresizingMaskIntoConstraints = false
    }
}
```

```

func 布局内容视图() {
    self.scrollView.addSubview(self.contentView)

    let views: NSDictionary = ["contentView": self.contentView, "view": self.view]
    var constraints = Array<String>()

    //将ContentView约束到scrollView。
    constraints.append("H:|-0-[contentView]-0-|")
    constraints.append("V:|-0-[contentView]-0-|")

    for 约束 in 约束集合 {
        self.scrollView.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
            约束, options: NSLayoutConstraintOptions(rawValue: 0), metrics: nil, views: views as! [String : AnyObject]))
    }

    //通过使contentView与我们控制器的self.view (ScrollView的父视图) 等宽, 禁用水平滚动。
    self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
        H:[contentView(==view)]", options: NSLayoutConstraintOptions(rawValue: 0), metrics: nil, views: views
        as! [String : AnyObject]))

    self.contentView.translatesAutoresizingMaskIntoConstraints = false
}

func addChildViews() {
    //初始化
    let greenView = UIView()
    let whiteView = UIView()

    //主题
    greenView.backgroundColor = UIColor.green()
    whiteView.backgroundColor = UIColor.orange()

    //布局 -- 子视图被添加到 'ContentView'
    self.contentView.addSubview(greenView)
    self.contentView.addSubview(whiteView)

    let views: NSDictionary = ["greenView": greenView, "whiteView": whiteView];
    var constraints = Array<String>()

    //将greenView约束到contentView, 四周间距15, 高度为400。
    constraints.append("H:|-15-[greenView]-15-|")
    constraints.append("V:|-15-[greenView(400)]")

    //将whiteView约束在greenView下方, 四周间距15, 高度为
    500.
    constraints.append("H:|-15-[whiteView]-15-|")
    constraints.append("V:[greenView]-15-[whiteView(500)]-15-|")

    for 约束 in 约束集合 {
        self.contentView.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
            约束, options: NSLayoutConstraintOptions(rawValue: 0), metrics: nil, views: views as! [String : AnyObject]))
    }

    greenView.translatesAutoresizingMaskIntoConstraints = false
    whiteView.translatesAutoresizingMaskIntoConstraints = false
}

```

```

func layoutContentView() {
    self.scrollView.addSubview(self.contentView)

    let views: NSDictionary = ["contentView": self.contentView, "view": self.view]
    var constraints = Array<String>()

    //Constrain the contentView to the scrollView.
    constraints.append("H:|-0-[contentView]-0-|")
    constraints.append("V:|-0-[contentView]-0-|")

    for constraint in constraints {
        self.scrollView.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
            constraint, options: NSLayoutConstraintOptions(rawValue: 0), metrics: nil, views: views as! [String : AnyObject]))
    }

    //Disable Horizontal Scrolling by making the contentView EqualWidth with our controller's
    self.view (ScrollView's parentView).
    self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
        "H:[contentView(==view)]", options: NSLayoutConstraintOptions(rawValue: 0), metrics: nil, views: views
        as! [String : AnyObject]))

    self.contentView.translatesAutoresizingMaskIntoConstraints = false
}

func addChildViews() {
    //Init
    let greenView = UIView()
    let whiteView = UIView()

    //Theme
    greenView.backgroundColor = UIColor.green()
    whiteView.backgroundColor = UIColor.orange()

    //Layout -- Child views are added to the 'ContentView'
    self.contentView.addSubview(greenView)
    self.contentView.addSubview(whiteView)

    let views: NSDictionary = ["greenView": greenView, "whiteView": whiteView];
    var constraints = Array<String>()

    //Constrain the greenView to the contentView with a height of 400 and 15 spacing all
    around.
    constraints.append("H:|-15-[greenView]-15-|")
    constraints.append("V:|-15-[greenView(400)]")

    //Constrain the whiteView below the greenView with 15 spacing all around and a height of
    500.
    constraints.append("H:|-15-[whiteView]-15-|")
    constraints.append("V:[greenView]-15-[whiteView(500)]-15-|")

    for constraint in constraints {
        self.contentView.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
            constraint, options: NSLayoutConstraintOptions(rawValue: 0), metrics: nil, views: views as! [String : AnyObject]))
    }

    greenView.translatesAutoresizingMaskIntoConstraints = false
    whiteView.translatesAutoresizingMaskIntoConstraints = false
}

```

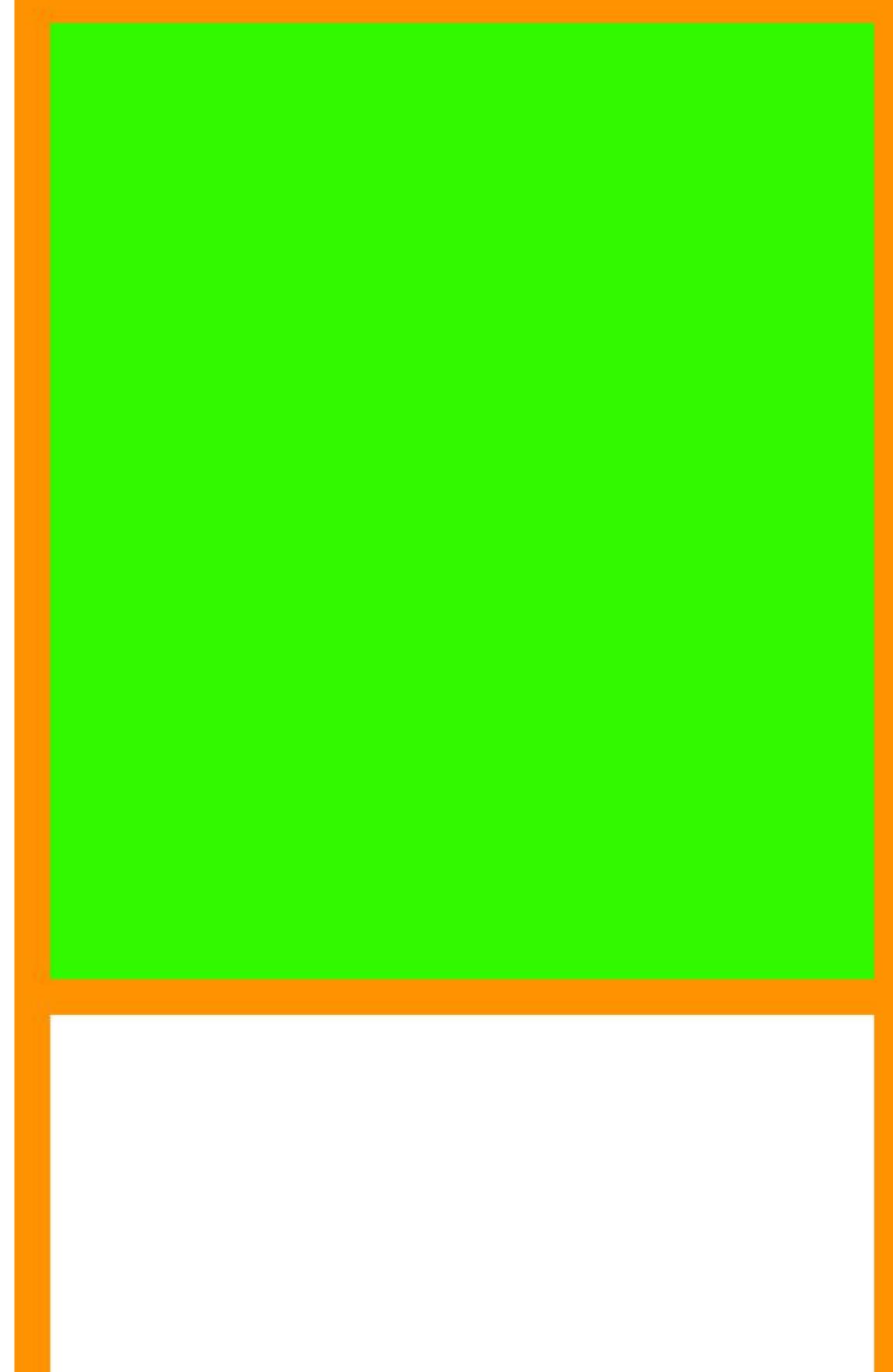
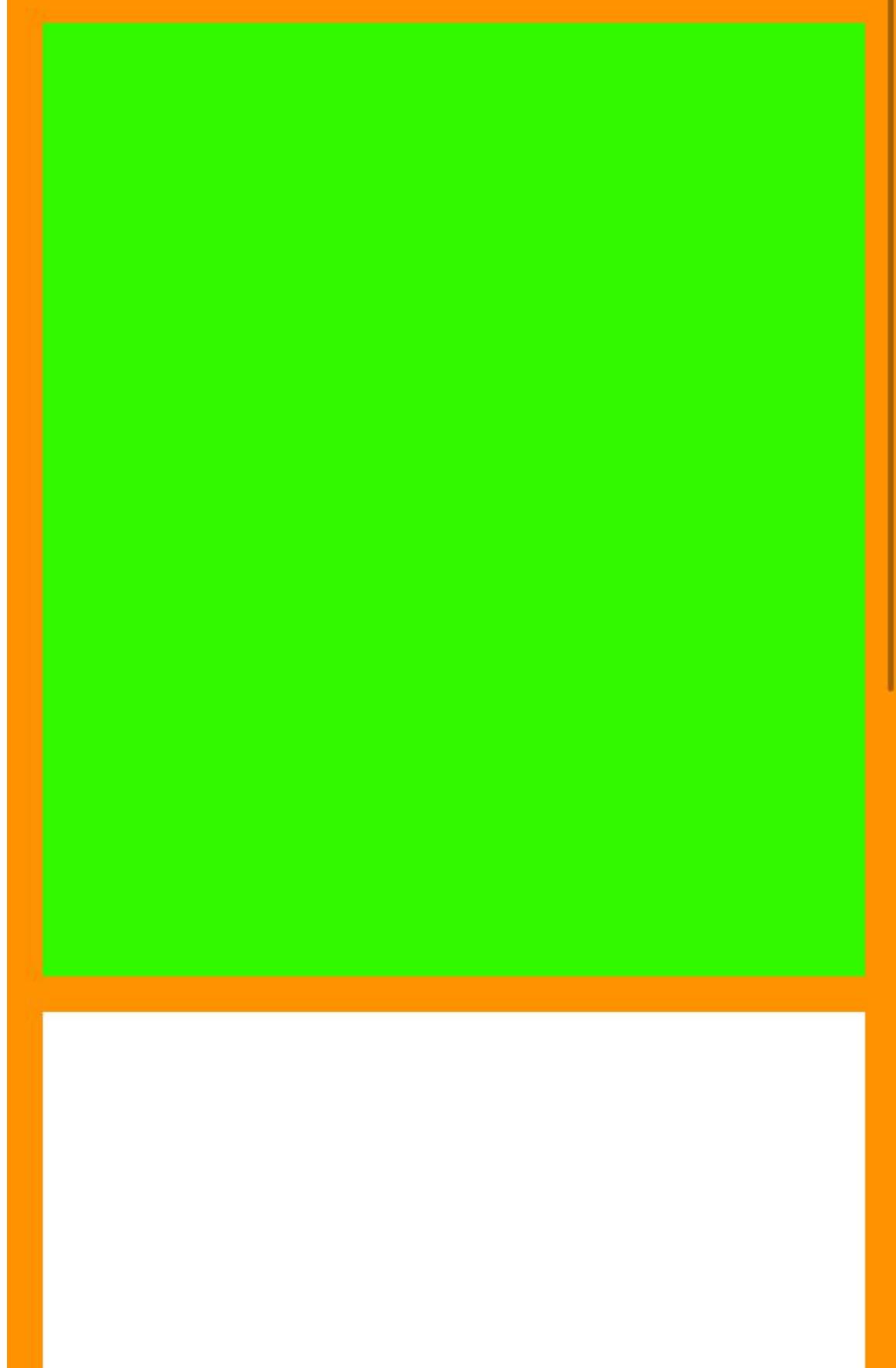
现在我们可以看到greenView（高度400）+ whiteView（高度500）大于我们的屏幕。这将导致ScrollView的contentSize增长以适应两个视图，从而允许垂直滚动。

我们使用EqualWidth约束禁用了contentView和self.view之间的水平滚动



Now we can see that the greenView (400 height) + the whiteView (500 height) is larger than our screen. This will cause the ScrollView's contentSize to grow to fit BOTH views, allowing it to scroll vertically.

We disabled horizontal scrolling using the EqualWidth constraint on the contentView and `self.view`



## 第33.2节：通过

Storyboard动态设置UIScrollView内容大小

在Storyboard中使用滚动视图时，最好根据滚动视图中存在的视图数量来计算内容大小，而不是用静态值以编程方式设置内容大小。

以下是动态获取内容大小的步骤。

步骤1：

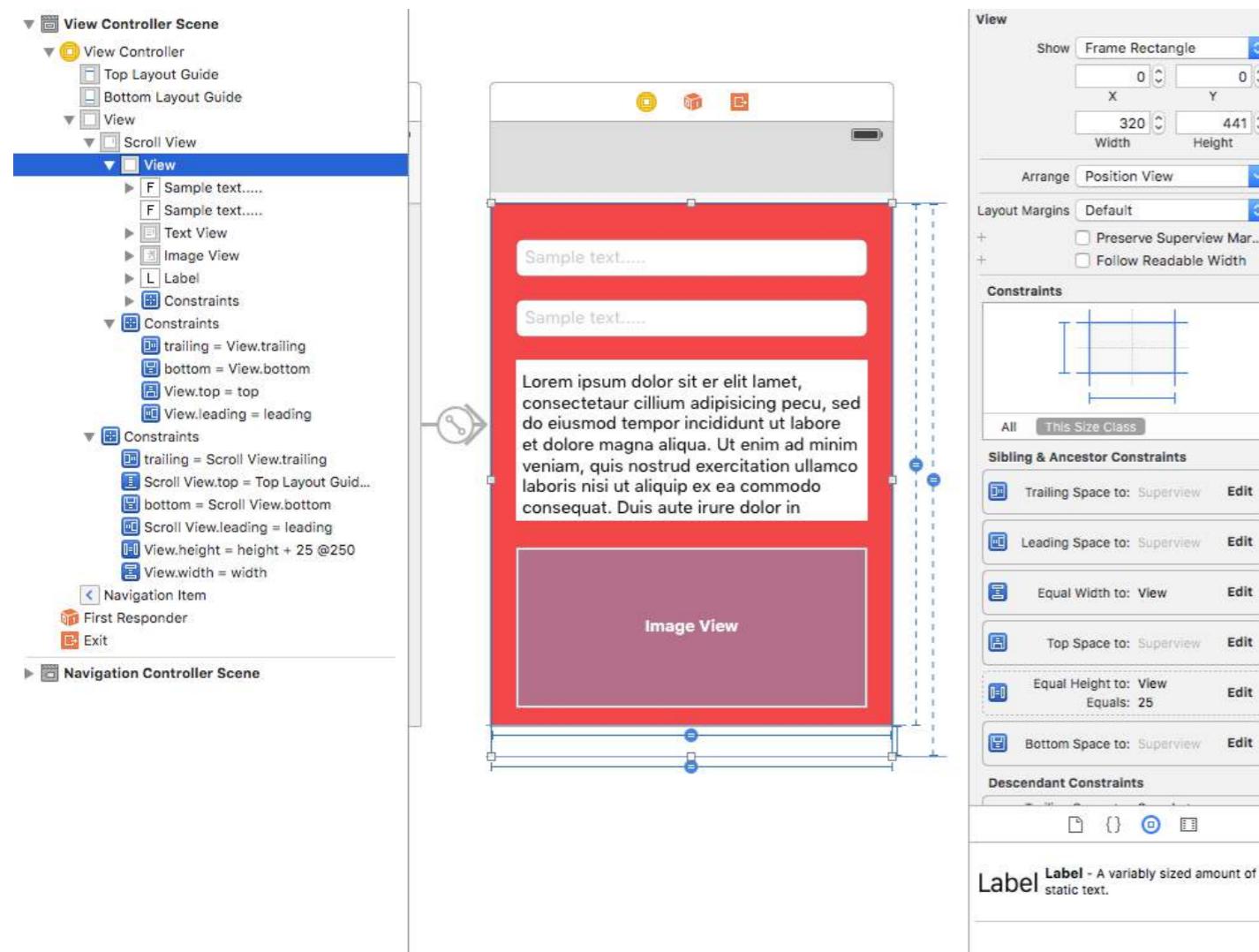
在Storyboard中向视图添加滚动视图，并添加前导、尾随、顶部和底部约束（所有值均为零）。

步骤2：

不要直接向滚动视图添加所需的视图，先向滚动视图添加一个视图（这将作为我们所有UI元素的内容视图）。向该视图添加以下约束。

1. 前导、尾随、顶部和底部约束（所有值均为零）。
2. 为主视图（即包含滚动视图的视图）添加等高、等宽约束。对于等高约束，将优先级设置为低。（这是设置内容尺寸的重要步骤）。
3. 此内容视图的高度将根据添加到视图中的视图数量而定。假设您添加的最后一个视图是一个标签，其Y位置为420，高度为20，那么您的内容视图高度将是440。

步骤3：根据您的需求，为内容视图中添加的所有视图添加约束。



## Section 33.2: UIScrollView dynamic content size through Storyboard

While using scrollviews in storyboard it's better to calculate content size according to number of views present in scrollview rather than giving content size programmatically with static value.

Here are the steps to get content size dynamically.

Step 1 :

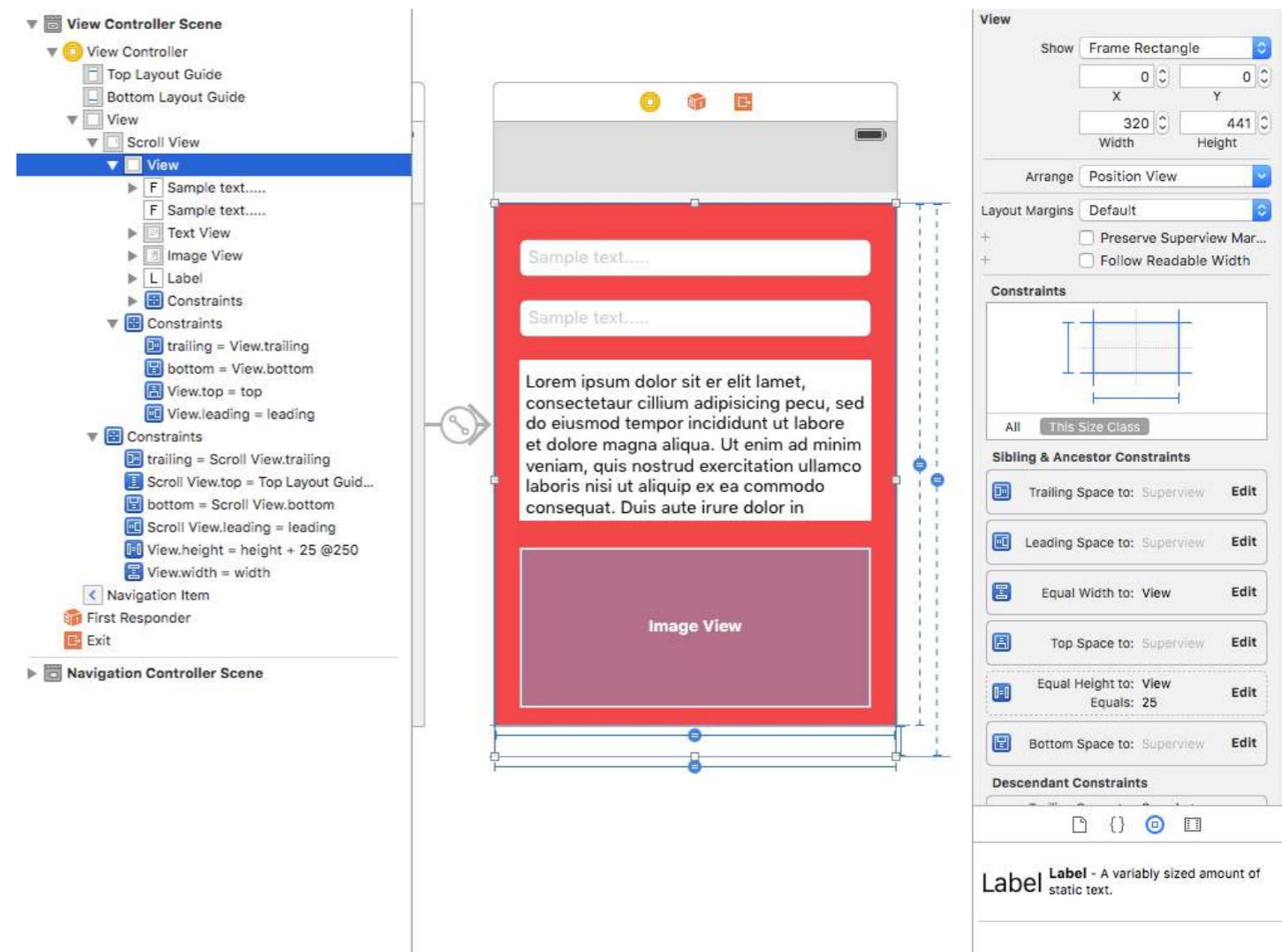
Add Scrollview to view in storyboard and add leading, trailing, top and bottom constraints (All values are zero).

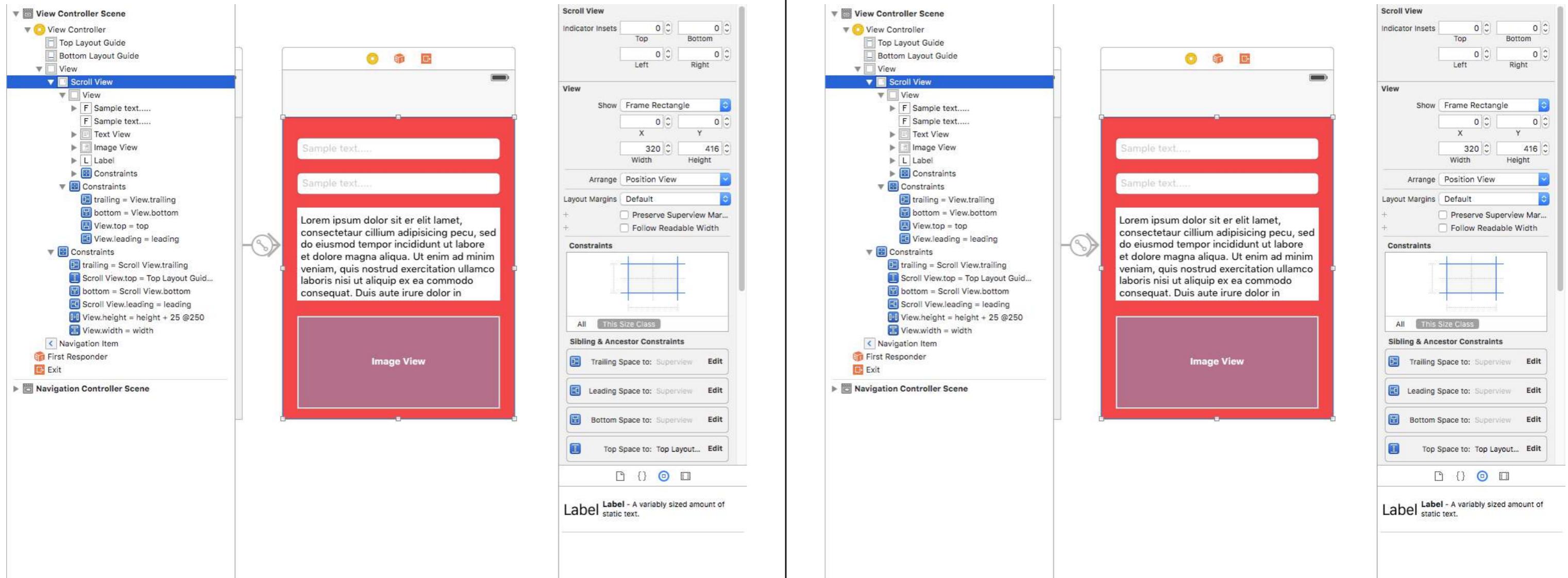
Step 2 :

Don't add directly views which you need on directly scrollview, First add one view to scrollview (that will be our content view for all UI elements). Add below constraints to this view.

1. Leading, trailing, top and bottom constraints (All values are zero).
2. Add equal height, equal width to Main view (i.e. which contains scrollview). For equal height set priority to low. (This is the important step for setting content size).
3. Height of this content view will be according to the number of views added to the view. let say if you added last view is one label and his Y position is 420 and height is 20 then your content view will be 440.

Step 3 : Add constraints to all of views which you added within content view as per your requirement.





# 第34章：UITextField

UITextField是UIKit框架的一部分，用于显示一个区域，通过屏幕键盘收集用户的文本输入

## 第34.1节：获取键盘焦点和隐藏键盘

### 获取焦点

#### Swift

```
textField.becomeFirstResponder()
```

#### Objective-C

```
[textField becomeFirstResponder];
```

### 放弃焦点

#### Swift

```
textField.resignFirstResponder()
```

#### Objective-C

```
[textField resignFirstResponder];
```

## 第34.2节：当用户按下回车键时收起键盘

设置你的视图控制器以管理文本字段的编辑。

```
class MyViewController: UITextFieldDelegate {  
  
    override viewDidLoad() {  
        super.viewDidLoad()  
  
        textField.delegate = self  
    }  
  
}
```

每次按下键盘上的回车键时都会调用 `textFieldShouldReturn`。

#### Swift:

```
func textFieldShouldReturn(textField: UITextField) -> Bool {  
    textField.resignFirstResponder()  
    return true;  
}
```

#### Objective-C :

```
- (BOOL)textFieldShouldReturn:(UITextField *)textField {  
    [textField resignFirstResponder];  
    return true;  
}
```

# Chapter 34: UITextField

UITextField is part of UIKit framework and is used to display an area to collect text input from the user using the onscreen keyboard

## Section 34.1: Get Keyboard Focus and Hide Keyboard

### Get Focus

#### Swift

```
textField.becomeFirstResponder()
```

#### Objective-C

```
[textField becomeFirstResponder];
```

### Resign

#### Swift

```
textField.resignFirstResponder()
```

#### Objective-C

```
[textField resignFirstResponder];
```

## Section 34.2: Dismiss keyboard when user pushes the return button

Setup your view controller to manage editing of text for the text field.

```
class MyViewController: UITextFieldDelegate {  
  
    override viewDidLoad() {  
        super.viewDidLoad()  
  
        textField.delegate = self  
    }  
  
}
```

`textFieldShouldReturn` is called every time the return button on the keyboard is pressed.

#### Swift:

```
func textFieldShouldReturn(textField: UITextField) -> Bool {  
    textField.resignFirstResponder()  
    return true;  
}
```

#### Objective-C:

```
- (BOOL)textFieldShouldReturn:(UITextField *)textField {  
    [textField resignFirstResponder];  
    return true;  
}
```

}

## 第34.3节：隐藏闪烁的插入点

要隐藏闪烁的插入点，需要重写UITextField的caretRectForPosition方法并返回CGRectZero。

**Swift 2.3 &lt;**

```
public override func caretRectForPosition(position: UITextPosition) -> CGRect {
    return CGRectZero
}
```

**Swift 3**

```
override func caretRect(for position: UITextPosition) -> CGRect {
    return CGRect.zero
}
```

**Objective-C**

```
- (CGRect) caretRectForPosition:(UITextPosition*) position{
    return CGRectZero;
}
```

## 第34.4节：输入辅助视图（工具栏）

在键盘上方添加辅助视图。通常用于添加“下一个/上一个”按钮，或额外的按钮如“完成/提交”（特别是对于没有内置回车键的数字/电话/小数点键盘类型）。

**Swift**

```
let textField = UITextField() // 初始化方式任意

let toolbar = UIToolbar(frame: CGRect(x: 0, y: 0, width: view.frame.size.width, height: 0))

let flexibleSpace = UIBarButtonItem(barButtonSystemItem: .FlexibleSpace, target: nil, action: nil)

let doneButton = UIBarButtonItem(barButtonSystemItem: .Done, target: self, action:
Selector("done"))

let items = [flexibleSpace, doneButton] // 将完成按钮推到右侧

toolbar.setItems(items, animated: false) // 或者 toolbar.items = ...
toolbar.sizeToFit()

textField.inputAccessoryView = toolbar
```

**Objective-C**

```
UITextField *textField = [[UITextField alloc] init];

UIToolbar *toolbar = [[UIToolbar alloc] initWithFrame:CGRectMake(0, 0, self.view.frame.size.width,
0)];

UIBarButtonItem *flexibleSpace = [[UIBarButtonItem alloc]
initWithBarButtonSystemItem:UIBarButtonSystemItemFlexibleSpace target:nil action:nil];
UIBarButtonItem *doneButton = [[UIBarButtonItem alloc]
initWithBarButtonSystemItem:UIBarButtonSystemItemDone target:self action:@selector(done)];
NSArray *items = @[

```

}

## Section 34.3: Hide blinking caret

To hide the blinking caret, you need to override caretRectForPosition of a UITextField and return CGRectZero.

**Swift 2.3 <**

```
public override func caretRectForPosition(position: UITextPosition) -> CGRect {
    return CGRectZero
}
```

**Swift 3**

```
override func caretRect(for position: UITextPosition) -> CGRect {
    return CGRect.zero
}
```

**Objective-C**

```
- (CGRect) caretRectForPosition:(UITextPosition*) position{
    return CGRectZero;
}
```

## Section 34.4: Input accessory view (toolbar)

Add an accessory view above the keyboard. This is commonly used for adding next/previous buttons, or additional buttons like Done/Submit (especially for the number/phone/decimal pad keyboard types which don't have a built-in return key).

**Swift**

```
let textField = UITextField() // initialized however

let toolbar = UIToolbar(frame: CGRect(x: 0, y: 0, width: view.frame.size.width, height: 0))

let flexibleSpace = UIBarButtonItem(barButtonSystemItem: .FlexibleSpace, target: nil, action: nil)

let doneButton = UIBarButtonItem(barButtonSystemItem: .Done, target: self, action:
Selector("done"))

let items = [flexibleSpace, doneButton] // pushes done button to right side

toolbar.setItems(items, animated: false) // or toolbar.items = ...
toolbar.sizeToFit()

textField.inputAccessoryView = toolbar
```

**Objective-C**

```
UITextField *textField = [[UITextField alloc] init];

UIToolbar *toolbar = [[UIToolbar alloc] initWithFrame:CGRectMake(0, 0, self.view.frame.size.width,
0)];

UIBarButtonItem *flexibleSpace = [[UIBarButtonItem alloc]
initWithBarButtonSystemItem:UIBarButtonSystemItemFlexibleSpace target:nil action:nil];
UIBarButtonItem *doneButton = [[UIBarButtonItem alloc]
initWithBarButtonSystemItem:UIBarButtonSystemItemDone target:self action:@selector(done)];
NSArray *items = @[

```

```

flexibleSpace,
doneButton
];

[toolbar setItems:items];
[toolbar sizeToFit];

textField.inputAccessoryView = toolbar;

```

## 第34.5节：当UITextView成为第一响应者时移动滚动视图

监听通知 UIKeyboardWillShowNotification 和 UIKeyboardWillHideNotification，根据键盘高度更新 scrollView 的内容内边距，然后滚动到聚焦控件。

```

- (void)viewDidLoad
{
    [super viewDidLoad];

    // 注册键盘通知
    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(keyboardWillShow:)
                                             name:UIKeyboardWillShowNotification
                                             object:self.view.window];

    // 注册键盘通知
    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(keyboardWillHide:)
                                             name:UIKeyboardWillHideNotification
                                             object:self.view.window];

}

// 当发送 UIKeyboardWillShowNotification 时调用
- (void)keyboardWillShow:(NSNotification*)notification
{
    // 如果没有视图或视图未显示在任何窗口中，则不处理
    if (!self.isViewLoaded || !self.view.window) {
        return;
    }

    NSDictionary *userInfo = [notification userInfo];

    CGRect keyboardFrameInWindow;
    [[userInfo objectForKey:UIKeyboardFrameEndUserInfoKey] getValue:&keyboardFrameInWindow];

    // 键盘框架以窗口坐标系指定。此处计算框架，假设它是视图的子视图，使其成为滚动视图的同级视图
    CGRect keyboardFrameInView = [self.view convertRect:keyboardFrameInWindow fromView:nil];

    CGRect scrollViewKeyboardIntersection = CGRectIntersection(_scrollView.frame,
                                                                keyboardFrameInView);
    UIEdgeInsets newContentInsets = UIEdgeInsetsMake(0, 0,
                                                    scrollViewKeyboardIntersection.size.height, 0);

    // 这是一个旧的动画方法，但它是唯一一个在参数（持续时间、曲线）和 userInfo 字典中包含的值之间保持兼容性的方法。
    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:[userInfo objectForKey:UIKeyboardAnimationDurationUserInfoKey]
doubleValue]];
    [UIView setAnimationCurve:[userInfo objectForKey:UIKeyboardAnimationCurveUserInfoKey]
intValue]];

```

```

flexibleSpace,
doneButton
];

[toolbar setItems:items];
[toolbar sizeToFit];

textField.inputAccessoryView = toolbar;



## Section 34.5: Moving scroll when UITextView becomes first responder



Observe the notifications UIKeyboardWillShowNotification and UIKeyboardWillHideNotification, update the scrollView content insets according to keyboard height, then scroll to the focused control.



```

- (void)viewDidLoad
{
    [super viewDidLoad];

    // register for keyboard notifications
    [[NSNotificationCenter defaultCenter] addObserver:self
   selector:@selector(keyboardWillShow:)
   name:UIKeyboardWillShowNotification
   object:self.view.window];

    // register for keyboard notifications
    [[NSNotificationCenter defaultCenter] addObserver:self
   selector:@selector(keyboardWillHide:)
   name:UIKeyboardWillHideNotification
   object:self.view.window];

}

// Called when UIKeyboardWillShowNotification is sent
- (void)keyboardWillShow:(NSNotification*)notification
{
    // if we have no view or are not visible in any window, we don't care
    if (!self.isViewLoaded || !self.view.window) {
        return;
    }

    NSDictionary *userInfo = [notification userInfo];

    CGRect keyboardFrameInWindow;
    [[userInfo objectForKey:UIKeyboardFrameEndUserInfoKey] getValue:&keyboardFrameInWindow];

    // the keyboard frame is specified in window-level coordinates. this calculates the frame as if it were a subview of our view, making it a sibling of the scroll view
    CGRect keyboardFrameInView = [self.view convertRect:keyboardFrameInWindow fromView:nil];

    CGRect scrollViewKeyboardIntersection = CGRectIntersection(_scrollView.frame,
  keyboardFrameInView);
    UIEdgeInsets newContentInsets = UIEdgeInsetsMake(0, 0,
  scrollViewKeyboardIntersection.size.height, 0);

    // this is an old animation method, but the only one that retains compatibility between parameters (duration, curve) and the values contained in the userInfo-Dictionary.
    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:[userInfo objectForKey:UIKeyboardAnimationDurationUserInfoKey]
doubleValue]];
    [UIView setAnimationCurve:[userInfo objectForKey:UIKeyboardAnimationCurveUserInfoKey]
intValue]];

```


```

```

_scrollView.contentInset = newContentInsets;
_scrollView.scrollIndicatorInsets = newContentInsets;

/*
* 根据视觉布局, _focusedControl 应该是输入字段
(UITextField, 等) 或另一个元素
* 应该是可见的, 例如金额文本字段下方的购买按钮 * 如果有多个输入字段, 在代理方法如 -textField
ShouldBeginEditing: 中设置 _focusedControl 是合理的

*/
if (_focusedControl) {
    CGRect controlFrameInScrollView = [_scrollView convertRect:_focusedControl.bounds
fromView:_focusedControl]; // 如果控件在滚动视图层级中较深的位置, 这将计算出该控件作为直接子视图时的框架
controlFrameInScrollView = CGRectInset(controlFrameInScrollView, 0, -10); // 用10替换
以实现控件与键盘或控件与滚动视图顶部之间的合适视觉偏移。

CGFloat controlVisualOffsetToTopOfScrollView = controlFrameInScrollView.origin.y -
_scrollView.contentOffset.y;
CGFloat controlVisualBottom = controlVisualOffsetToTopOfScrollView +
controlFrameInScrollView.size.height;

// 这是滚动视图中未被键盘遮挡的可见部分高度
CGFloat scrollViewVisibleHeight = _scrollView.frame.size.height -
scrollViewKeyboardIntersection.size.height;

if (controlVisualBottom > scrollViewVisibleHeight) { // 检查键盘是否会遮挡目标控件

    // 向上滚动直到控件到位
    CGPoint newContentOffset = _scrollView.contentOffset;
    newContentOffset.y += (controlVisualBottom - scrollViewVisibleHeight);

    // 确保不会因为“合适的视觉偏移”而设置不可能的偏移量// 如果控件位于滚动视图底部, 它最终会位于键盘正
上方, 以消除滚动不一致性

newContentOffset.y = MIN(newContentOffset.y, _scrollView.contentSize.height -
scrollViewVisibleHeight);

[_scrollView setContentOffset:newContentOffset animated:NO]; // animated:NO 因为我们
已经在这段代码周围创建了自己的动画上下文
} else if (controlFrameInScrollView.origin.y < _scrollView.contentOffset.y) {
    // 如果控件没有完全可见, 使其可见 (当用户点击部分可见的输入框时很有用)

    CGPoint newContentOffset = _scrollView.contentOffset;
    newContentOffset.y = controlFrameInScrollView.origin.y;

    [_scrollView setContentOffset:newContentOffset animated:NO]; // animated:NO 因为我们
已经在这段代码周围创建了自己的动画上下文
}

[UIView commitAnimations];
}

// 当收到 UIKeyboardWillHideNotification 时调用
- (void)keyboardWillHide:(NSNotification*)notification
{
    // 如果没有视图或视图未显示在任何窗口中, 则不处理
    if (!self.isViewLoaded || !self.view.window) {
        return;
    }
}

```

```

_scrollView.contentInset = newContentInsets;
_scrollView.scrollIndicatorInsets = newContentInsets;

/*
* Depending on visual layout, _focusedControl should either be the input field
(UITextField,...) or another element
* that should be visible, e.g. a purchase button below an amount text field
* it makes sense to set _focusedControl in delegates like -textFieldShouldBeginEditing: if you
have multiple input fields
*/
if (_focusedControl) {
    CGRect controlFrameInScrollView = [_scrollView convertRect:_focusedControl.bounds
fromView:_focusedControl]; // if the control is a deep in the hierarchy below the scroll view, this
will calculate the frame as if it were a direct subview
controlFrameInScrollView = CGRectInset(controlFrameInScrollView, 0, -10); // replace 10
with any nice visual offset between control and keyboard or control and top of the scroll view.

CGFloat controlVisualOffsetToTopOfScrollView = controlFrameInScrollView.origin.y -
_scrollView.contentOffset.y;
CGFloat controlVisualBottom = controlVisualOffsetToTopOfScrollView +
controlFrameInScrollView.size.height;

// this is the visible part of the scroll view that is not hidden by the keyboard
CGFloat scrollViewVisibleHeight = _scrollView.frame.size.height -
scrollViewKeyboardIntersection.size.height;

if (controlVisualBottom > scrollViewVisibleHeight) { // check if the keyboard will hide the
control in question
    // scroll up until the control is in place
    CGPoint newContentOffset = _scrollView.contentOffset;
    newContentOffset.y += (controlVisualBottom - scrollViewVisibleHeight);

    // make sure we don't set an impossible offset caused by the "nice visual offset"
    // if a control is at the bottom of the scroll view, it will end up just above the
keyboard to eliminate scrolling inconsistencies
    newContentOffset.y = MIN(newContentOffset.y, _scrollView.contentSize.height -
scrollViewVisibleHeight);

    [_scrollView setContentOffset:newContentOffset animated:NO]; // animated:NO because we
have created our own animation context around this code
} else if (controlFrameInScrollView.origin.y < _scrollView.contentOffset.y) {
    // if the control is not fully visible, make it so (useful if the user taps on a
partially visible input field
    CGPoint newContentOffset = _scrollView.contentOffset;
    newContentOffset.y = controlFrameInScrollView.origin.y;

    [_scrollView setContentOffset:newContentOffset animated:NO]; // animated:NO because we
have created our own animation context around this code
}
}

[UIView commitAnimations];
}

// Called when the UIKeyboardWillHideNotification is sent
- (void)keyboardWillHide:(NSNotification*)notification
{
    // if we have no view or are not visible in any window, we don't care
    if (!self.isViewLoaded || !self.view.window) {
        return;
    }
}

```

```

}

NSDictionary *userInfo = notification.userInfo;

[UIView beginAnimations:nil context:NULL];
[UIView setAnimationDuration:[userInfo valueForKey:UIKeyboardAnimationDurationUserInfoKey]
doubleValue]];
[UIView setAnimationCurve:[userInfo valueForKey:UIKeyboardAnimationCurveUserInfoKey]
intValue]];

// 撤销所有 keyboardWillShow 的操作
// 滚动视图将适当调整其 contentOffset
_scrollView.contentInset = UIEdgeInsetsZero;
_scrollView.scrollIndicatorInsets = UIEdgeInsetsZero;

[UIView commitAnimations];
}

```

## 第34.6节：键盘类型

要更改键盘的外观，可以在每个UITextField中

属性上单独设置以下类型：keyboardType

```

typedef NS_ENUM(NSUInteger, UIKeyboardType) {
    UIKeyboardTypeDefault,           // 当前输入法的默认类型。
    UIKeyboardTypeASCII Capable,    // 显示可输入ASCII字符的键盘，  
非ASCII键盘仍然保持激活状态
    UIKeyboardTypeNumbersAndPunctuation, // 数字和各种标点符号。
    UIKeyboardTypeURL,              // 优化用于URL输入的类型（突出显示 . / .com）。

    UIKeyboardTypeNumberPad,        // 数字键盘（0-9）。适合输入PIN码。
    UIKeyboardTypePhonePad,         // 电话键盘（1-9, *, 0, #，数字下方带字母）。

    UIKeyboardTypeNamePhonePad,     // 优化用于输入人名或电话号码的类型。

    UIKeyboardTypeEmailAddress,     // 优化用于输入多个电子邮件地址的类型（突出显示空格 @ .）。

    UIKeyboardTypeDecimalPad NS_ENUM_AVAILABLE_IOS(4_1), // 带小数点的数字键盘。
    UIKeyboardTypeTwitter NS_ENUM_AVAILABLE_IOS(5_0),    // 针对Twitter文本输入优化的类型（方便访问 @ #）

    UIKeyboardTypeWebSearch NS_ENUM_AVAILABLE_IOS(7_0),   // 带有URL导向附加功能的默认键盘类型（突出显示空格和  
点）。

    UIKeyboardTypeAlphabet = UIKeyboardTypeASCII Capable, // 已弃用
};

```

## 第34.7节：更改占位符颜色和字体

我们可以通过设置attributedPlaceholder（一个NSAttributedString）来更改占位符的样式。

```

var placeholderAttributes = [String: AnyObject]()
placeholderAttributes[NSForegroundColorAttributeName] = color
placeholderAttributes[NSFontAttributeName] = font

if let placeholder = textField.placeholder {
    let newAttributedPlaceholder = NSAttributedString(string: placeholder, attributes:
placeholderAttributes)
    textField.attributedPlaceholder = newAttributedPlaceholder
}

```

```

}

NSDictionary *userInfo = notification.userInfo;

[UIView beginAnimations:nil context:NULL];
[UIView setAnimationDuration:[userInfo valueForKey:UIKeyboardAnimationDurationUserInfoKey]
doubleValue]];
[UIView setAnimationCurve:[userInfo valueForKey:UIKeyboardAnimationCurveUserInfoKey]
intValue]];

// undo all that keyboardWillShow-magic
// the scroll view will adjust its contentOffset appropriately
_scrollView.contentInset = UIEdgeInsetsZero;
_scrollView.scrollIndicatorInsets = UIEdgeInsetsZero;

[UIView commitAnimations];
}

```

## Section 34.6: KeyboardType

To change the appearance of the keyboard, the following types can be set individually on every UITextField property: keyboardType

```

typedef NS_ENUM(NSUInteger, UIKeyboardType) {
    UIKeyboardTypeDefault,           // Default type for the current input method.
    UIKeyboardTypeASCII Capable,    // Displays a keyboard which can enter ASCII characters,  
non-ASCII keyboards remain active
    UIKeyboardTypeNumbersAndPunctuation, // Numbers and assorted punctuation.
    UIKeyboardTypeURL,              // A type optimized for URL entry (shows . / .com  
prominently).
    UIKeyboardTypeNumberPad,        // A number pad (0-9). Suitable for PIN entry.
    UIKeyboardTypePhonePad,         // A phone pad (1-9, *, 0, #, with letters under the  
numbers).
    UIKeyboardTypeNamePhonePad,     // A type optimized for entering a person's name or phone  
number.
    UIKeyboardTypeEmailAddress,     // A type optimized for multiple email address entry  
(shows space @ . prominently).
    UIKeyboardTypeDecimalPad NS_ENUM_AVAILABLE_IOS(4_1), // A number pad with a decimal point.
    UIKeyboardTypeTwitter NS_ENUM_AVAILABLE_IOS(5_0),    // A type optimized for twitter text  
entry (easy access to @ #)
    UIKeyboardTypeWebSearch NS_ENUM_AVAILABLE_IOS(7_0),   // A default keyboard type with URL-  
oriented addition (shows space . prominently).

    UIKeyboardTypeAlphabet = UIKeyboardTypeASCII Capable, // Deprecated
};

```

## Section 34.7: Change placeholder color and font

We can change the style of the placeholder by setting attributedPlaceholder (a NSAttributedString).

```

var placeholderAttributes = [String: AnyObject]()
placeholderAttributes[NSForegroundColorAttributeName] = color
placeholderAttributes[NSFontAttributeName] = font

if let placeholder = textField.placeholder {
    let newAttributedPlaceholder = NSAttributedString(string: placeholder, attributes:
placeholderAttributes)
    textField.attributedPlaceholder = newAttributedPlaceholder
}

```

在此示例中，我们仅更改了颜色和字体。您还可以更改其他属性，例如下划线或删除线样式。有关可更改的属性，请参阅[NSAttributedString](#)。

## 第34.8节：用UIPickerView替换键盘

在某些情况下，您可能希望为UITextField显示一个带有预定义内容的UIPickerView，而不是键盘。

### 创建自定义UIPickerView

首先，您需要一个自定义的UIPickerView包装类，该类遵循协议UIPickerViewDataSource和UIPickerViewDelegate。

```
class MyPickerView: UIPickerView, UIPickerViewDataSource, UIPickerViewDelegate
```

您需要为数据源和代理实现以下方法：

```
public func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {
    if data != nil {
        return data!.count
    } else {
        return 0
    }
}

public func numberOfComponents(in pickerView: UIPickerView) -> Int {
    return 1
}

public func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int) -> String? {
    如果 data != nil {
        返回 data![row]
    } 否则 {
        返回 ""
    }
}
```

为了处理数据，MyPickerView 需要属性 data、selectedValue 和 textFieldBeingEdited：

```
/**
`UIPickerViewDelegate` 的数据
必须始终是 `String` 数组！`UIPickerView` 只能显示字符串
*/
public var data: [String]? {
    设置后 {
        super.delegate = self
        super.dataSource = self
        self.reloadAllComponents()
    }
}

/**
存储目前正在编辑的 UITextField
*/
public var textFieldBeingEdited: UITextField?

/**
```

In this example we change only the color and font. You could change other properties such as underline or strikethrough style. Refer to [NSAttributedString](#) for the properties that can be changed.

## Section 34.8: Replace keyboard with UIPickerView

In some cases, you want to show your users a UIPickerView with predefined contents for a UITextField instead of a keyboard.

### Create a custom UIPickerView

At first, you need a custom wrapper-class for UIPickerView conforming to the protocols UIPickerViewDataSource and UIPickerViewDelegate.

```
class MyPickerView: UIPickerView, UIPickerViewDataSource, UIPickerViewDelegate
```

You need to implement the following methods for the DataSource and Delegate:

```
public func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {
    if data != nil {
        return data!.count
    } else {
        return 0
    }
}

public func numberOfComponents(in pickerView: UIPickerView) -> Int {
    return 1
}

public func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int) -> String? {
    if data != nil {
        return data![row]
    } else {
        return ""
    }
}
```

To handle the data, MyPickerView needs the properties data, selectedValue and textFieldBeingEdited:

```
/**
The data for the `UIPickerViewDelegate`
Always needs to be an array of `String`! The `UIPickerView` can ONLY display Strings
*/
public var data: [String]? {
    didSet {
        super.delegate = self
        super.dataSource = self
        self.reloadAllComponents()
    }
}

/**
Stores the UITextField that is being edited at the moment
*/
public var textFieldBeingEdited: UITextField?

/**
```

```

获取选择器的选中值
*/
public var selectedValue: String {
    get {
        if data != nil {
            return data![selectedRow(inComponent: 0)]
        } else {
            返回 ""
        }
    }
}

```

## 准备你的视图控制器

包含你的文本框的视图控制器需要有一个自定义UIPickerView的属性。  
(假设你已经有另一个属性或@IBOutlet包含你的文本框)

```

/***
作为键盘呈现的选择器视图
*/
var picker: MyPickerView?

```

在你的viewDidLoad()中，你需要初始化picker并进行一些配置：

```

picker = MyPickerView()
picker?.autoresizingMask = [.flexibleHeight, .flexibleWidth]
picker?.backgroundColor = UIColor.white()

picker?.data = ["One", "Two", "Three", "Four", "Five"] // 显示在选择器中的数据

```

现在，你可以将MyPicker作为UITextField的inputView添加：

```
textField.inputView = picker
```

## 关闭选择器键盘

现在，你已经用UIPickerView替换了键盘，但没有办法关闭它。这可以通过自定义.inputAccessoryView来实现：

向你的ViewController添加属性pickerAccessory。

```

/***
当显示`picker`时，添加到键盘上的工具栏。
*/
var pickerAccessory: UIToolbar?

```

在viewDidLoad()中，您需要为inputAccessoryView创建一个UIToolbar：

```

pickerAccessory = UIToolbar()
pickerAccessory?.autoresizingMask = .flexibleHeight

//此自定义为可选项
pickerAccessory?.barStyle = .default
pickerAccessory?.barTintColor = UIColor.red()
pickerAccessory?.backgroundColor = UIColor.red()
pickerAccessory?.isTranslucent = false

```

您应该设置工具栏的框架。为了符合iOS的设计，建议使用高度为44.0：

```

Get the selected Value of the picker
*/
public var selectedValue: String {
    get {
        if data != nil {
            return data![selectedRow(inComponent: 0)]
        } else {
            return ""
        }
    }
}

```

## Prepare your ViewController

The ViewController that contains your textField, needs to have a property for your custom UIPickerView.  
(Assuming, that you already have another property or @IBOutlet containing your textField)

```

/***
The picker view to present as keyboard
*/
var picker: MyPickerView?

```

In your viewDidLoad(), you need to initialize picker and configure it a bit:

```

picker = MyPickerView()
picker?.autoresizingMask = [.flexibleHeight, .flexibleWidth]
picker?.backgroundColor = UIColor.white()

picker?.data = ["One", "Two", "Three", "Four", "Five"] //The data shown in the picker

```

Now, you can add the MyPicker as inputView of your UITextField:

```
textField.inputView = picker
```

## Dismissing the picker-keyboard

Now, you have replaced the keyboard by an UIPickerView, but there is no possibility to dismiss it. This can be done with a custom .inputAccessoryView:

Add the property pickerAccessory to your ViewController.

```

/***
A toolbar to add to the keyboard when the `picker` is presented.
*/
var pickerAccessory: UIToolbar?

```

In viewDidLoad(), you need to create an UIToolbar for the inputAccessoryView:

```

pickerAccessory = UIToolbar()
pickerAccessory?.autoresizingMask = .flexibleHeight

//this customization is optional
pickerAccessory?.barStyle = .default
pickerAccessory?.barTintColor = UIColor.red()
pickerAccessory?.backgroundColor = UIColor.red()
pickerAccessory?.isTranslucent = false

```

You should set the frame of your toolbar. To fit in the design of iOS, it's recommended to use a height of 44.0:

```
var frame = pickerAccessory?.frame  
frame?.size.height = 44.0  
pickerAccessory?.frame = frame!
```

为了良好的用户体验，您应该添加两个按钮（“完成”和“取消”），但只添加一个用于收起键盘的按钮也可以。

```
let cancelButton = UIBarButtonItem(barButtonSystemItem: .cancel, target: self, action:  
#selector(ViewController.cancelBtnClicked(_)))  
cancelButton.tintColor = UIColor.white()  
let flexSpace = UIBarButtonItem(barButtonSystemItem: .flexibleSpace, target: nil, action: nil) //a  
两个按钮之间的灵活间距  
let doneButton = UIBarButtonItem(barButtonSystemItem: .done, target: self, action:  
#selector(ViewController.doneBtnClicked(_)))  
doneButton.tintColor = UIColor.white()  
  
//将项目添加到工具栏  
pickerAccessory?.items = [cancelButton, flexSpace, doneButton]
```

现在你可以将工具栏作为inputAccessoryView添加

```
textField.inputAccessoryView = pickerAccessory
```

在构建项目之前，你需要实现按钮调用的方法：

```
/**  
当点击`pickerAccessory`的取消按钮时调用。关闭选择器  
*/  
func cancelBtnClicked(_ button: UIBarButtonItem?) {  
    textField?.resignFirstResponder()  
}  
  
/**  
当点击 `pickerAccessory` 的完成按钮时调用。关闭选择器并将选中的值放入文本框中  
*/  
func doneBtnClicked(_ button: UIBarButtonItem?) {  
    textField?.resignFirstResponder()  
    textField.text = picker?.selectedValue  
}
```

运行你的项目，点击textField，你应该会看到一个选择器，而不是键盘：

```
var frame = pickerAccessory?.frame  
frame?.size.height = 44.0  
pickerAccessory?.frame = frame!
```

For a good user experience, you should add two buttons ("Done" and "Cancel"), but it would also work with only one that dismisses the keyboard.

```
let cancelButton = UIBarButtonItem(barButtonSystemItem: .cancel, target: self, action:  
#selector(ViewController.cancelBtnClicked(_)))  
cancelButton.tintColor = UIColor.white()  
let flexSpace = UIBarButtonItem(barButtonSystemItem: .flexibleSpace, target: nil, action: nil) //a  
flexible space between the two buttons  
let doneButton = UIBarButtonItem(barButtonSystemItem: .done, target: self, action:  
#selector(ViewController.doneBtnClicked(_)))  
doneButton.tintColor = UIColor.white()  
  
//Add the items to the toolbar  
pickerAccessory?.items = [cancelButton, flexSpace, doneButton]
```

Now you can add the toolbar as inputAccessoryView

```
textField.inputAccessoryView = pickerAccessory
```

Before you can build your project, you need to implement the methods, the buttons are calling:

```
/**  
Called when the cancel button of the `pickerAccessory` was clicked. Dismisses the picker  
*/  
func cancelBtnClicked(_ button: UIBarButtonItem?) {  
    textField?.resignFirstResponder()  
}  
  
/**  
Called when the done button of the `pickerAccessory` was clicked. Dismisses the picker and puts  
the selected value into the textField  
*/  
func doneBtnClicked(_ button: UIBarButtonItem?) {  
    textField?.resignFirstResponder()  
    textField.text = picker?.selectedValue  
}
```

Run your project, tap the textField and you should see a picker like this instead of the keyboard:

Cancel

Done

One

Two

Three

Four

以编程方式选择一个值（可选）

如果你不想让第一行自动被选中，可以像在UIPickerView中那样设置选中的行：

```
picker?.selectRow(3, inComponent: 0, animated: false) //将选择索引为3的行
```

## 第34.9节：创建一个UITextField

用CGRect作为框架初始化UITextField：

**Swift**

```
let textfield = UITextField(frame: CGRect(x: 0, y: 0, width: 200, height: 21))
```

**Objective-C**

```
UITextField *textField = [[UITextField alloc] initWithFrame:CGRectMake(0, 0, 200, 21)];
```

你也可以在界面构建器中创建一个UITextField：



Cancel

Done

One

Two

Three

Four

Select a value programmatically (optional)

If you don't want to have the first row selected automatically, you can set the selected row as in UIPickerView:

```
picker?.selectRow(3, inComponent: 0, animated: false) //Will select the row at index 3
```

## Section 34.9: Create a UITextField

Initialize the UITextField with a CGRect as a frame:

**Swift**

```
let textfield = UITextField(frame: CGRect(x: 0, y: 0, width: 200, height: 21))
```

**Objective-C**

```
UITextField *textField = [[UITextField alloc] initWithFrame:CGRectMake(0, 0, 200, 21)];
```

You can also create a UITextField in Interface Builder:



## 第34.10节：获取和设置光标位置

### 有用信息

文本字段文本的最开始位置：

```
let startPosition: UITextPosition = textField.beginningOfDocument
```

文本字段文本的最末尾位置：

```
let endPosition: UITextPosition = textField.endOfDocument
```

当前选定的范围：

```
let selectedRange: UITextRange? = textField.selectedTextRange
```

### 获取光标位置

```
if let selectedRange = textField.selectedTextRange {  
    let cursorPosition = textField.offsetFromPosition(textField.beginningOfDocument, toPosition:  
selectedRange.start)  
    print("\(cursorPosition)")  
}
```

### 设置光标位置

为了设置位置，所有这些方法实际上都是设置一个起始和结束值相同的范围。

### 回到顶部

```
let newPosition = textField.beginningOfDocument  
textField.selectedTextRange = textField.textRangeFromPosition(newPosition, toPosition: newPosition)
```

### 到结尾

```
let newPosition = textField.endOfDocument  
textField.selectedTextRange = textField.textRangeFromPosition(newPosition, toPosition: newPosition)
```

### 到当前光标位置左边一个位置

```
// 仅当当前有选定范围时  
if let selectedRange = textField.selectedTextRange {  
    // 并且仅当新位置有效时  
    if let newPosition = textField.positionFromPosition(selectedRange.start, inDirection:  
UITextLayoutDirection.Left, offset: 1) {  
        // 设置新位置  
        textField.selectedTextRange = textField.textRangeFromPosition(newPosition, toPosition:  
newPosition)  
    }  
}
```

### 到任意位置

从开头开始，向右移动5个字符。

## Section 34.10: Getting and Setting the Cursor Position

### Useful information

The very beginning of the text field text:

```
let startPosition: UITextPosition = textField.beginningOfDocument
```

The very end of the text field text:

```
let endPosition: UITextPosition = textField.endOfDocument
```

The currently selected range:

```
let selectedRange: UITextRange? = textField.selectedTextRange
```

### Get cursor position

```
if let selectedRange = textField.selectedTextRange {  
    let cursorPosition = textField.offsetFromPosition(textField.beginningOfDocument, toPosition:  
selectedRange.start)  
    print("\(cursorPosition)")  
}
```

### Set cursor position

In order to set the position, all of these methods are actually setting a range with the same start and end values.

### To the beginning

```
let newPosition = textField.beginningOfDocument  
textField.selectedTextRange = textField.textRangeFromPosition(newPosition, toPosition: newPosition)
```

### To the end

```
let newPosition = textField.endOfDocument  
textField.selectedTextRange = textField.textRangeFromPosition(newPosition, toPosition: newPosition)
```

### To one position to the left of the current cursor position

```
// only if there is a currently selected range  
if let selectedRange = textField.selectedTextRange {  
    // and only if the new position is valid  
    if let newPosition = textField.positionFromPosition(selectedRange.start, inDirection:  
UITextLayoutDirection.Left, offset: 1) {  
        // set the new position  
        textField.selectedTextRange = textField.textRangeFromPosition(newPosition, toPosition:  
newPosition)  
    }  
}
```

### To an arbitrary position

Start at the beginning and move 5 characters to the right.

```
let arbitraryValue: Int = 5
if let newPosition = textField.positionFromPosition(textField.beginningOfDocument, inDirection: UITextLayoutDirection.Right, offset: arbitraryValue) {

    textField.selectedTextRange = textField.textRangeFromPosition(newPosition, toPosition: newPosition)
}
```

## 相关

### 全选文本

```
textField.selectedTextRange = textField.textRangeFromPosition(textField.beginningOfDocument, toPosition: textField.endOfDocument)
```

### 选择一段文本

```
// 范围：3 到 7
let startPosition = textField.positionFromPosition(textField.beginningOfDocument, inDirection: UITextLayoutDirection.Right, offset: 3)
let endPosition = textField.positionFromPosition(textField.beginningOfDocument, inDirection: UITextLayoutDirection.Right, offset: 7)

if startPosition != nil && endPosition != nil {
    textField.selectedTextRange = textField.textRangeFromPosition(startPosition!, toPosition: endPosition!)
}
```

### 在当前光标位置插入文本

```
textField.insertText("Hello")
```

#### 注意事项

- 此示例最初来自这个 [Stack Overflow 回答](#)。
- 该答案使用了文本字段，但相同的概念也适用于 UITextView。
- 使用 `textField.becomeFirstResponder()` 来使文本框获得焦点并弹出键盘。
- 参见 [this answer](#) 了解如何获取某个范围内的文本。

## 相关

- [如何在Swift中创建范围](#) (间接涉及为什么这里必须使用 `selectedTextRange` 而不是直接使用 `selectedRange` 的问题)

## 第34.11节：隐藏键盘

### Swift

按住Ctrl键，从MainStoryboard中的UITextField拖动到ViewController类，创建一个UITextField的Outlet

```
let arbitraryValue: Int = 5
if let newPosition = textField.positionFromPosition(textField.beginningOfDocument, inDirection: UITextLayoutDirection.Right, offset: arbitraryValue) {

    textField.selectedTextRange = textField.textRangeFromPosition(newPosition, toPosition: newPosition)
}
```

## Related

### Select all text

```
textField.selectedTextRange = textField.textRangeFromPosition(textField.beginningOfDocument, toPosition: textField.endOfDocument)
```

### Select a range of text

```
// Range: 3 to 7
let startPosition = textField.positionFromPosition(textField.beginningOfDocument, inDirection: UITextLayoutDirection.Right, offset: 3)
let endPosition = textField.positionFromPosition(textField.beginningOfDocument, inDirection: UITextLayoutDirection.Right, offset: 7)

if startPosition != nil && endPosition != nil {
    textField.selectedTextRange = textField.textRangeFromPosition(startPosition!, toPosition: endPosition!)
}
```

### Insert text at the current cursor position

```
textField.insertText("Hello")
```

#### Notes

- This example originally comes from [this Stack Overflow answer](#).
- This answer uses a text field, but the same concepts apply to UITextView.
- Use `textField.becomeFirstResponder()` to give focus to the text field and make the keyboard appear.
- See [this answer](#) for how to get the text at some range.

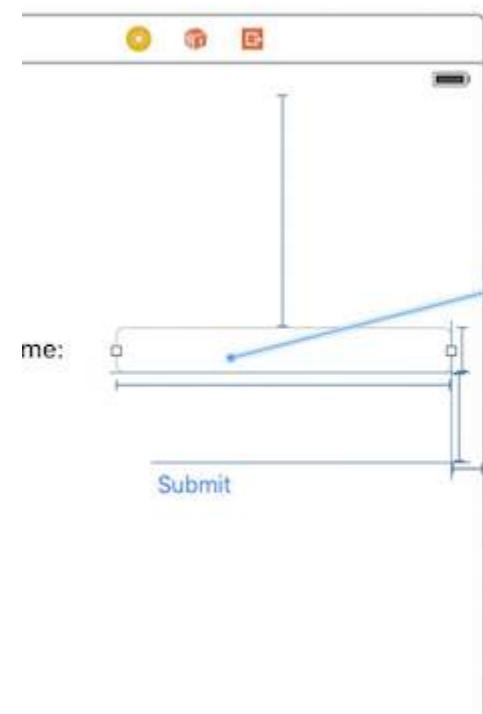
## Related

- [How to Create a Range in Swift](#) (Deals indirectly with the issue of why we have to use `selectedTextRange` here rather than just `selectedRange`)

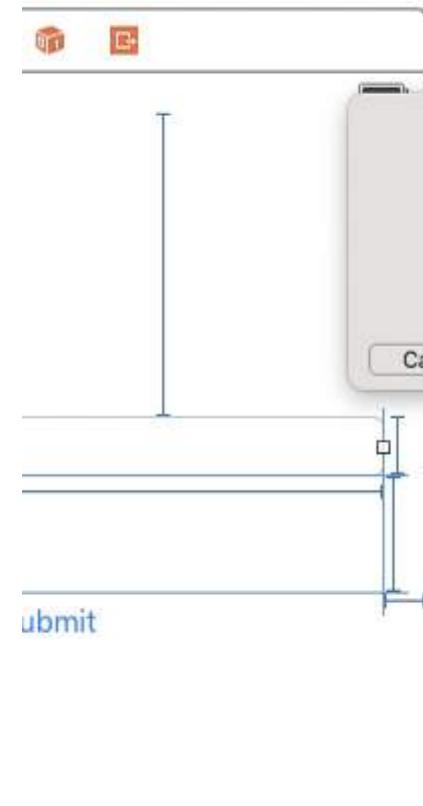
## Section 34.11: Dismiss Keyboard

### Swift

Ctrl + Drag from the UITextField in MainStoryboard to the ViewController Class and create a UITextField Outlet



```
4 //  
5 // Created by Ali on 7/21/16.  
6 // Copyright © 2016 mag. All  
rights reserved.  
7 //  
8 import UIKit  
9  
10 class ViewController: UIViewController {  
11     // Do any additional setup  
12     // Insert Outlet, Action, or Outlet Collection  
13     // after loading the view,  
14     // typically from a nib.  
15     //  
16     // Do any additional setup  
17     // after loading the view,  
18     // typically from a nib.  
19     //  
20     // DidReceiveMemoryWarning()  
21     //  
22     // Dispose of any resources  
23     // that can be recreated.  
24     //  
25     //  
26     //  
27     //  
28 }
```

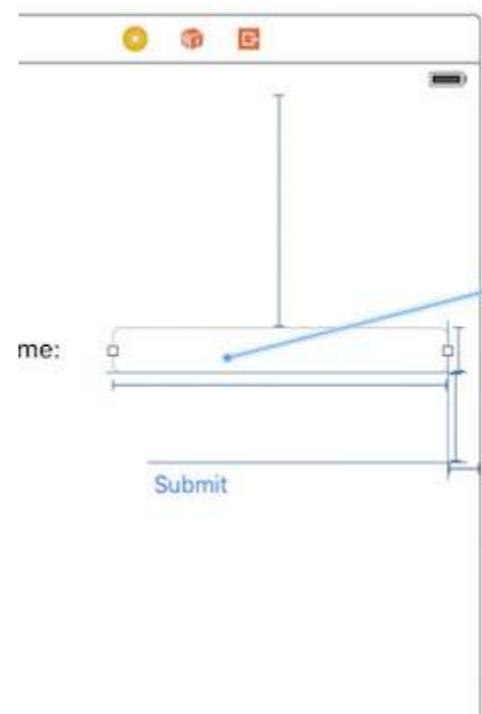


```
2 // ViewControl' test  
3 //  
4 // Created by :  
5 // Copyright © rights rese  
6 //  
7 //  
8 import UIKit  
9  
10 class ViewController: UIViewController {  
11     // Do any additional setup  
12     // Insert Outlet, Action, or Outlet Collection  
13     // after loading the view,  
14     // typically from a nib.  
15     //  
16     // Do any additional setup  
17     // after loading the view,  
18     // typically from a nib.  
19     //  
20     // DidReceiveMemoryWarning()  
21     //  
22     // Dispose of any resources  
23     // that can be recreated.  
24     //  
25     //  
26     //  
27     //  
28 }
```

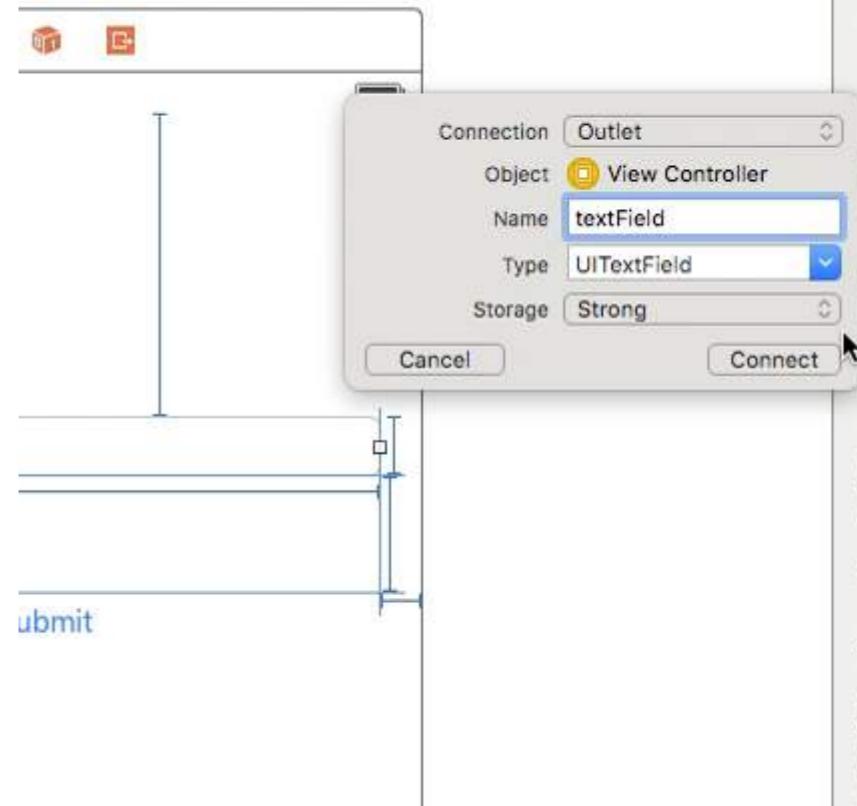
A connection dialog is open, showing the following settings:

- Connection: Outlet
- Object: View Controller
- Name: textField
- Type: UITextField
- Storage: Strong

The "Connect" button is highlighted.



```
4 //  
5 // Created by Ali on 7/21/16.  
6 // Copyright © 2016 mag. All  
rights reserved.  
7 //  
8 import UIKit  
9  
10 class ViewController: UIViewController {  
11     // Do any additional setup  
12     // Insert Outlet, Action, or Outlet Collection  
13     // after loading the view,  
14     // typically from a nib.  
15     //  
16     // Do any additional setup  
17     // after loading the view,  
18     // typically from a nib.  
19     //  
20     // DidReceiveMemoryWarning()  
21     //  
22     // Dispose of any resources  
23     // that can be recreated.  
24     //  
25     //  
26     //  
27     //  
28 }
```

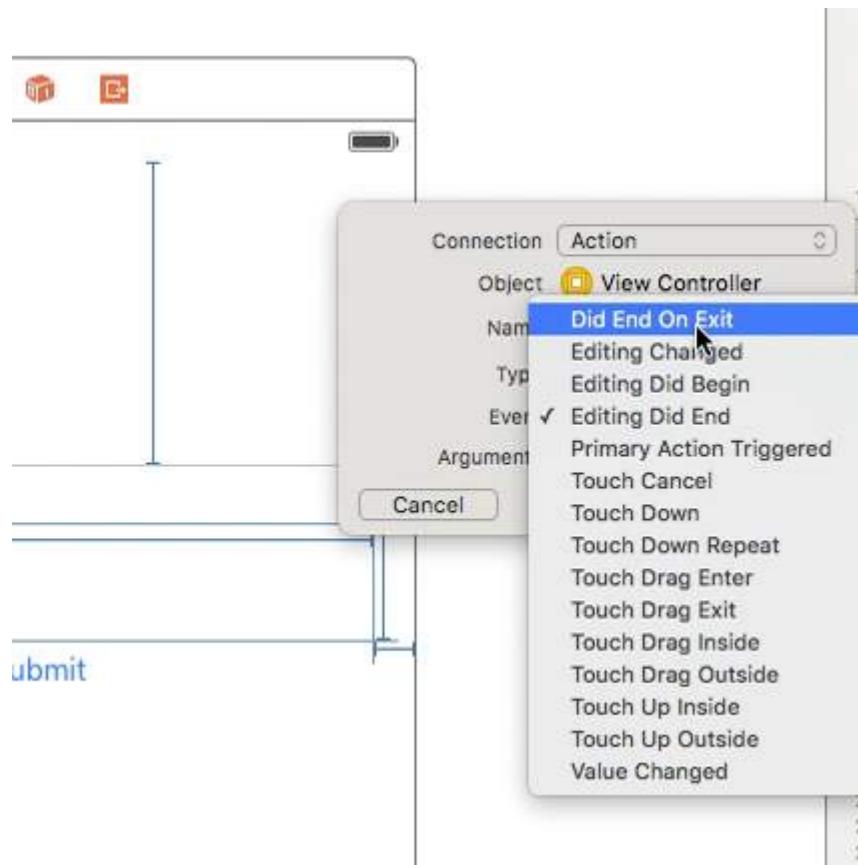


```
2 // ViewControl' test  
3 //  
4 // Created by :  
5 // Copyright © rights rese  
6 //  
7 //  
8 import UIKit  
9  
10 class ViewController: UIViewController {  
11     // Do any additional setup  
12     // Insert Outlet, Action, or Outlet Collection  
13     // after loading the view,  
14     // typically from a nib.  
15     //  
16     // Do any additional setup  
17     // after loading the view,  
18     // typically from a nib.  
19     //  
20     // DidReceiveMemoryWarning()  
21     //  
22     // Dispose of any resources  
23     // that can be recreated.  
24     //  
25     //  
26     //  
27     //  
28 }
```

A connection dialog is open, showing the following settings:

- Connection: Outlet
- Object: View Controller
- Name: textField
- Type: UITextField
- Storage: Strong

The "Connect" button is highlighted.



之后再次选择UITextField，按住Ctrl键拖动到ViewController类，但这次选择**Action**连接，在存储选项中选择**Did End On Exit**，然后点击连接。

在刚创建的action中输入你的UITextField名称.resignFirstResponder()

```
@IBAction func textFieldResign(sender: AnyObject) {
    yourTextFieldName.resignFirstResponder()
}
```

这样就能在按下键盘上的回车键时隐藏键盘。

**另一个按下回车键隐藏键盘的示例：**

我们在UIViewController后添加UITextFieldDelegate协议

在viewDidLoad函数中添加`self.yourTextFieldName.delegate = self`

最后添加以下代码

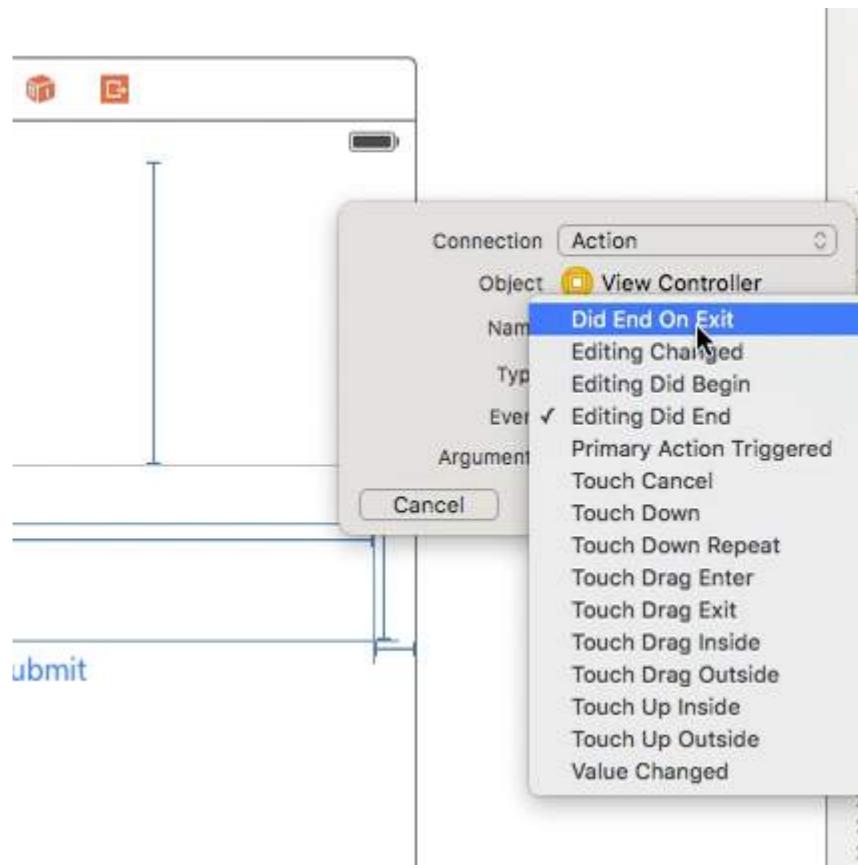
```
func textFieldShouldReturn(textField: UITextField) -> Bool {
    yourTextFieldName.resignFirstResponder()
    return true
}
```

最终代码如下：

```
class ViewController: UIViewController, UITextFieldDelegate {

@IBOutlet var textField: UITextField!

func textFieldShouldReturn(textField: UITextField) -> Bool {
    textField.resignFirstResponder()
}
```



After that select the UITextField again and Ctrl+drag in ViewController class but this time select **Action** connection and on storage select **Did End On Exit** then click connect.

in the action you just created type the name of your UITextField .`resignFirstResponder()`

```
@IBAction func textFieldResign(sender: AnyObject) {
    yourTextFieldName.resignFirstResponder()
}
```

This will take care of hiding the keyboard when pressing the return key on keyboard.

**Another example of hiding the keyboard when return key is pressed:**

we add `UITextFieldDelegate` protocol next to `UIViewController`

in the viewDidLoad function we add `self.yourTextFieldName.delegate = self`

And Finally we add this

```
func textFieldShouldReturn(textField: UITextField) -> Bool {
    yourTextFieldName.resignFirstResponder()
    return true
}
```

The final code is this:

```
class ViewController: UIViewController, UITextFieldDelegate {

@IBOutlet var textField: UITextField!

func textFieldShouldReturn(textField: UITextField) -> Bool {
    textField.resignFirstResponder()
}
```

```

        return true
    }

override func touchesBegan(touches: Set<UITouch>, withEvent event: UIEvent?){
    view.endEditing(true)
    super.touchesBegan(touches, withEvent: event)
}

override func viewDidLoad() {
    super.viewDidLoad()
    self.textField.delegate = self
}

```

#### Objective-C

```
[textField resignFirstResponder];
```

## 第34.12节：初始化文本框

#### Swift

```
let frame = CGRect(x: 0, y: 0, width: 100, height: 100)
let textField = UITextField(frame: frame)
```

#### Objective-C

```
CGRect *frame = CGRectMake(0, 0, 100, 100);
UITextField *textField = [[UITextField alloc] initWithFrame:frame];
```

#### 界面构建器

你也可以通过从对象库拖动UITextField到故事板中来添加它。



## 第34.13节：自动大写

#### Swift

```
textField.autocapitalizationType = .None
```

#### Objective-C

```
textField.autocapitalizationType = UITextAutocapitalizationTypeNone;
```

所有选项：

- .`None` \ `UITextAutocapitalizationTypeNone` : 不自动大写任何内容
- .`Words` \ `UITextAutocapitalizationTypeWords` : 自动大写每个单词

```

        return true
    }

override func touchesBegan(touches: Set<UITouch>, withEvent event: UIEvent?){
    view.endEditing(true)
    super.touchesBegan(touches, withEvent: event)
}

override func viewDidLoad() {
    super.viewDidLoad()
    self.textField.delegate = self
}

```

#### Objective-C

```
[textField resignFirstResponder];
```

## Section 34.12: Initialize text field

#### Swift

```
let frame = CGRect(x: 0, y: 0, width: 100, height: 100)
let textField = UITextField(frame: frame)
```

#### Objective-C

```
CGRect *frame = CGRectMake(0, 0, 100, 100);
UITextField *textField = [[UITextField alloc] initWithFrame:frame];
```

#### Interface Builder

You can also add a `UITextField` to a storyboard by dragging it from Object Library.



## Section 34.13: Autocapitalization

#### Swift

```
textField.autocapitalizationType = .None
```

#### Objective-C

```
textField.autocapitalizationType = UITextAutocapitalizationTypeNone;
```

All options:

- .`None` \ `UITextAutocapitalizationTypeNone` : Don't autocapitalize anything
- .`Words` \ `UITextAutocapitalizationTypeWords` : Autocapitalize every word

- `.句子` \ `UITextAutocapitalizationTypeSentences` : 自动大写句子的第一个单词
- `.所有字符` \ `UITextAutocapitalizationTypeAllCharacters` : 自动大写每个字母 (即大写锁定)

- `.Sentences` \ `UITextAutocapitalizationTypeSentences` : Autocapitalize the first word in a sentence
- `.AllCharacters` \ `UITextAutocapitalizationTypeAllCharacters` : Autocapitalize every letter (i.e. caps lock)

## 第34.14节：设置对齐方式

### Swift

```
textField.textAlignment = .居中
```

### Objective-C

```
[textField setTextAlignment: NSTextAlignmentCenter];
```

在示例中，我们将NSTextAlignment设置为居中。你也可以设置为.[左对齐](#)、[右对齐](#)、[两端对齐](#)和[自然对齐](#)。

[.自然对齐](#)是当前本地化的默认对齐方式。这意味着对于从左到右的语言（例如英语），对齐方式是[.左对齐](#)；对于从右到左的语言，则是[.右对齐](#)。

## Section 34.14: Set Alignment

### Swift

```
textField.textAlignment = .Center
```

### Objective-C

```
[textField setTextAlignment: NSTextAlignmentCenter];
```

In the example, we have set the NSTextAlignment to center. You can also set to [.Left](#), [.Right](#), [.Justified](#) and [.Natural](#).

[.Natural](#) is the default alignment for the current localization. That means for left-to-right languages (eg. English), the alignment is [.Left](#); for right-to-left languages, it is [.Right](#).

# 第35章：自定义UITextField

使用自定义UITextField，我们可以操控文本框的行为！

## 第35.1节：用于过滤输入文本的自定义UITextField

这里有一个自定义UITextField的示例，它只接受数字文本，丢弃所有其他内容。

注意：对于iPhone，使用数字键盘很容易实现这一点，但对于iPad，没有仅包含数字的键盘

```
class NumberTextField: UITextField {

required init(coder aDecoder: NSCoder) {
    super.init(coder: aDecoder)
registerForTextFieldNotifications()
}

override init(frame: CGRect) {
    super.init(frame: frame)
}

override func awakeFromNib() {
    super.awakeFromNib()
keyboardType = .numberPad//仅适用于iPhone

private func registerForTextFieldNotifications() {
    NotificationCenter.default.addObserver(self, selector:
#selector(NumberTextField.textDidChange), name: NSNotification.Name(rawValue:
"UITextFieldTextDidChangeNotification"), object: self)
}

deinit {
    NotificationCenter.default.removeObserver(self)
}

func textDidChange() {
    text = filteredText()
}

private func filteredText() -> String {
    let inverseSet = CharacterSet(charactersIn:"0123456789").inverted
    let components = text!.components(separatedBy: inverseSet)
    return components.joined(separator: "")
}
}
```

所以，无论何时我们需要一个只接受数字输入的文本框，都可以使用这个自定义的UITextField

## 第35.2节：自定义UITextField以禁止所有操作，如复制、粘贴等

如果我们想禁用所有诸如复制、粘贴、替换、选择等操作，从UITextField中，则可以使用以下自定义文本字段：

```
类 CustomTextField: UITextField {
```

# Chapter 35: Custom UITextField

Using custom UITextField, we can manipulate the behavior of text field!

## Section 35.1: Custom UITextField for Filtering Input Text

Here is an example of custom UITextField that takes only numerical text and discards all other.

**NOTE:** For iPhone it is easy to do this using Number type keyboard, but for iPad there is no keyboard with Numbers only

```
class NumberTextField: UITextField {

required init(coder aDecoder: NSCoder) {
    super.init(coder: aDecoder)
registerForTextFieldNotifications()
}

override init(frame: CGRect) {
    super.init(frame: frame)
}

override func awakeFromNib() {
    super.awakeFromNib()
keyboardType = .numberPad//useful for iPhone only
}

private func registerForTextFieldNotifications() {
    NotificationCenter.default.addObserver(self, selector:
#selector(NumberTextField.textDidChange), name: NSNotification.Name(rawValue:
"UITextFieldTextDidChangeNotification"), object: self)
}

deinit {
    NotificationCenter.default.removeObserver(self)
}

func textDidChange() {
    text = filteredText()
}

private func filteredText() -> String {
    let inverseSet = CharacterSet(charactersIn:"0123456789").inverted
    let components = text!.components(separatedBy: inverseSet)
    return components.joined(separator: "")
}
}
```

So, wherever we want text field which would take only numbers as input text, then we can use this custom UITextField

## Section 35.2: Custom UITextField to Disallow All Actions like Copy, Paste, etc

If we want to disable all the actions like Copy, Paste, Replace, Select, etc from UITextField then we can use following custom text field:

```
class CustomTextField: UITextField {
```

```
变量 enableLongPressActions = false

必需的初始化(coder aDecoder: NSCoder) {
    调用父类.初始化(coder: aDecoder)!
}

重写初始化(frame: CGRect) {
    调用父类.初始化(frame: frame)
}

重写函数 canPerformAction(_ action: Selector, withSender sender: Any?) -> Bool {
    返回 enableLongPressActions
}
}
```

使用

```
var enableLongPressActions = false

required init(coder aDecoder: NSCoder) {
    super.init(coder: aDecoder)!
}

override init(frame: CGRect) {
    super.init(frame: frame)
}

override func canPerformAction(_ action: Selector, withSender sender: Any?) -> Bool {
    return enableLongPressActions
}
```

Using enableLongPressActions property, we can enable all actions any time later, if needed.

# 第36章：UIViewController

## 第36.1节：子类化

子类化UIControl使我们可以访问以下方法：

- beginTrackingWithTouch在手指首次触摸控件边界内时调用。
- continueTrackingWithTouch在手指滑过控件及其边界外时会反复调用。
- endTrackingWithTouch在手指离开屏幕时调用。

*MyCustomControl.swift*

```
import UIKit

// 这些是我们自定义的与其他类通信的规则
protocol ViewControllerCommunicationDelegate: class {
    func myTrackingBegan()
    func myTrackingContinuing(location: CGPoint)
    func myTrackingEnded()
}

class MyCustomControl: UIControl {

    // 任何想要接收触摸事件通知的类必须将代理(delegate)设置为自身
    weak var delegate: ViewControllerCommunicationDelegate?

    override func beginTrackingWithTouch(touch: UITouch, withEvent event: UIEvent?) -> Bool {
        // 通知代理（即视图控制器）
        delegate?.myTrackingBegan()

        // 返回 true 表示后续事件（如 continueTrackingWithTouch 和
        // endTrackingWithTouch）将继续被触发
        return true
    }

    override func continueTrackingWithTouch(touch: UITouch, withEvent event: UIEvent?) -> Bool {
        // 获取触摸点在自定义控件自身坐标系中的位置
        let point = touch.locationInView(self)

        // 使用新的坐标点更新代理（即视图控制器）
        delegate?.myTrackingContinuing(point)

        // 返回 true 表示后续事件将继续被触发
        return true
    }

    override func endTrackingWithTouch(touch: UITouch?, withEvent event: UIEvent?) {
        // 通知代理（即视图控制器）
        delegate?.myTrackingEnded()
    }
}
```

*ViewController.swift*

# Chapter 36: UIViewController

## Section 36.1: Subclassing

Subclassing `UIControl` gives us access to the following methods:

- `beginTrackingWithTouch` is called when the finger first touches down within the control's bounds.
- `continueTrackingWithTouch` is called repeatedly as the finger slides across the control and even outside of the control's bounds.
- `endTrackingWithTouch` is called when the finger lifts off the screen.

*MyCustomControl.swift*

```
import UIKit

// These are our self-defined rules for how we will communicate with other classes
protocol ViewControllerCommunicationDelegate: class {
    func myTrackingBegan()
    func myTrackingContinuing(location: CGPoint)
    func myTrackingEnded()
}

class MyCustomControl: UIControl {

    // whichever class wants to be notified of the touch events must set the delegate to itself
    weak var delegate: ViewControllerCommunicationDelegate?

    override func beginTrackingWithTouch(touch: UITouch, withEvent event: UIEvent?) -> Bool {
        // notify the delegate (i.e. the view controller)
        delegate?.myTrackingBegan()

        // returning true means that future events (like continueTrackingWithTouch and
        // endTrackingWithTouch) will continue to be fired
        return true
    }

    override func continueTrackingWithTouch(touch: UITouch, withEvent event: UIEvent?) -> Bool {
        // get the touch location in our custom control's own coordinate system
        let point = touch.locationInView(self)

        // Update the delegate (i.e. the view controller) with the new coordinate point
        delegate?.myTrackingContinuing(point)

        // returning true means that future events will continue to be fired
        return true
    }

    override func endTrackingWithTouch(touch: UITouch?, withEvent event: UIEvent?) {
        // notify the delegate (i.e. the view controller)
        delegate?.myTrackingEnded()
    }
}
```

*ViewController.swift*

这是视图控制器被设置为代理并响应来自自定义控件的触摸事件的方式。

```
import UIKit
class ViewController: UIViewController, ViewControllerCommunicationDelegate {

    @IBOutlet weak var myCustomControl: MyCustomControl!
    @IBOutlet weak var trackingBeganLabel: UILabel!
    @IBOutlet weak var trackingEndedLabel: UILabel!
    @IBOutlet weak var xLabel: UILabel!
    @IBOutlet weak var yLabel: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()
        myCustomControl.delegate = self
    }

    func myTrackingBegan() {
        trackingBeganLabel.text = "Tracking began"
    }

    func myTrackingContinuing(location: CGPoint) {
        xLabel.text = "x: \(location.x)"
        yLabel.text = "y: \(location.y)"
    }

    func myTrackingEnded() {
        trackingEndedLabel.text = "跟踪结束"
    }
}
```

#### 注意事项

- 实现相同结果的替代方法而不使用子类化包括添加目标或使用手势识别器。
- 如果这些方法仅在自定义控件内部使用，则不必使用代理。我们本可以仅添加一个print语句来显示事件是如何被调用的。在这种情况下，代码将简化为

```
import UIKit
class MyCustomControl: UIControl {

    override func beginTrackingWithTouch(touch: UITouch, withEvent event: UIEvent?) -> Bool {
        print("开始跟踪")
        return true
    }

    override func continueTrackingWithTouch(touch: UITouch, withEvent event: UIEvent?) -> Bool {
        let point = touch.locationInView(self)
        print("x: \(point.x), y: \(point.y)")
        return true
    }

    override func endTrackingWithTouch(touch: UITouch?, withEvent event: UIEvent?) {
        print("结束跟踪")
    }
}
```

This is how the view controller is set up to be the delegate and respond to touch events from our custom control.

```
import UIKit
class ViewController: UIViewController, ViewControllerCommunicationDelegate {

    @IBOutlet weak var myCustomControl: MyCustomControl!
    @IBOutlet weak var trackingBeganLabel: UILabel!
    @IBOutlet weak var trackingEndedLabel: UILabel!
    @IBOutlet weak var xLabel: UILabel!
    @IBOutlet weak var yLabel: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()
        myCustomControl.delegate = self
    }

    func myTrackingBegan() {
        trackingBeganLabel.text = "Tracking began"
    }

    func myTrackingContinuing(location: CGPoint) {
        xLabel.text = "x: \(location.x)"
        yLabel.text = "y: \(location.y)"
    }

    func myTrackingEnded() {
        trackingEndedLabel.text = "Tracking ended"
    }
}
```

#### Notes

- Alternate methods of achieving the same result without subclassing include adding a target or using a gesture recognizer.
- It is not necessary to use a delegate with these methods if they are only being used within the custom control itself. We could have just added a `print` statement to show how the events are being called. In that case, the code would be simplified to

```
import UIKit
class MyCustomControl: UIControl {

    override func beginTrackingWithTouch(touch: UITouch, withEvent event: UIEvent?) -> Bool {
        print("Began tracking")
        return true
    }

    override func continueTrackingWithTouch(touch: UITouch, withEvent event: UIEvent?) -> Bool {
        let point = touch.locationInView(self)
        print("x: \(point.x), y: \(point.y)")
        return true
    }

    override func endTrackingWithTouch(touch: UITouch?, withEvent event: UIEvent?) {
        print("Ended tracking")
    }
}
```

## 第36.2节：访问容器视图控制器

当视图控制器在标签栏控制器中呈现时，可以这样访问标签栏控制器：

### Swift

```
let tabBarController = viewController.tabBarController
```

### Objective-C

```
UITabBarController *tabBarController = self.tabBarController;
```

当视图控制器是导航栈的一部分时，可以这样访问导航控制器：

### Swift

```
let navigationController = viewController.navigationController
```

### Objective-C

```
UINavigationController *navigationController = self.navigationController;
```

## 第36.3节：以编程方式设置视图

### Swift

```
class FooViewController: UIViewController {  
  
    override func loadView() {  
        view = FooView()  
    }  
}
```

## 第36.4节：从Storyboard实例化

```
UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"Main" bundle:nil];
```

### 带有标识符：

在故事板的身份检查器中为场景指定一个Storyboard ID。

## Section 36.2: Access the container view controller

When the view controller is presented within a tab bar controller, you can access the tab bar controller like this:

### Swift

```
let tabBarController = viewController.tabBarController
```

### Objective-C

```
UITabBarController *tabBarController = self.tabBarController;
```

When the view controller is part on an navigation stack, you can access the navigation controller like this:

### Swift

```
let navigationController = viewController.navigationController
```

### Objective-C

```
UINavigationController *navigationController = self.navigationController;
```

## Section 36.3: Set the view programmatically

### Swift

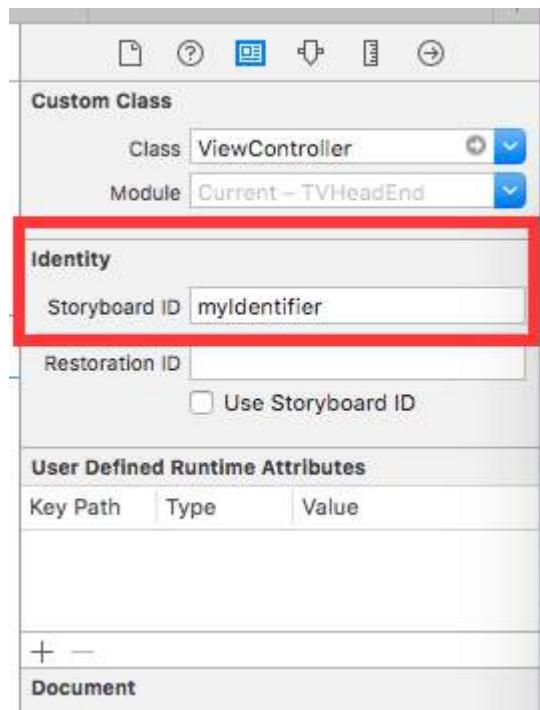
```
class FooViewController: UIViewController {  
  
    override func loadView() {  
        view = FooView()  
    }  
}
```

## Section 36.4: Instantiate from a Storyboard

```
UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"Main" bundle:nil];
```

### With an Identifier:

Give the scene a Storyboard ID within the identity inspector of the storyboard.

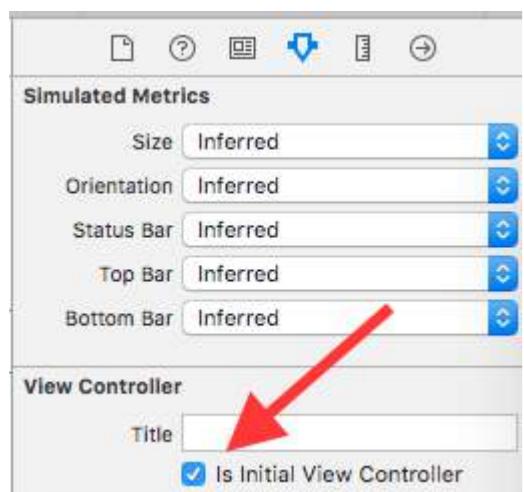


在代码中实例化：

```
UIViewController *controller = [storyboard
instantiateViewControllerWithIdentifier:@"myIdentifier"];
```

**实例化初始视图控制器：**

在故事板中选择视图控制器，然后选择属性检查器，勾选“Is Initial View Controller”框。



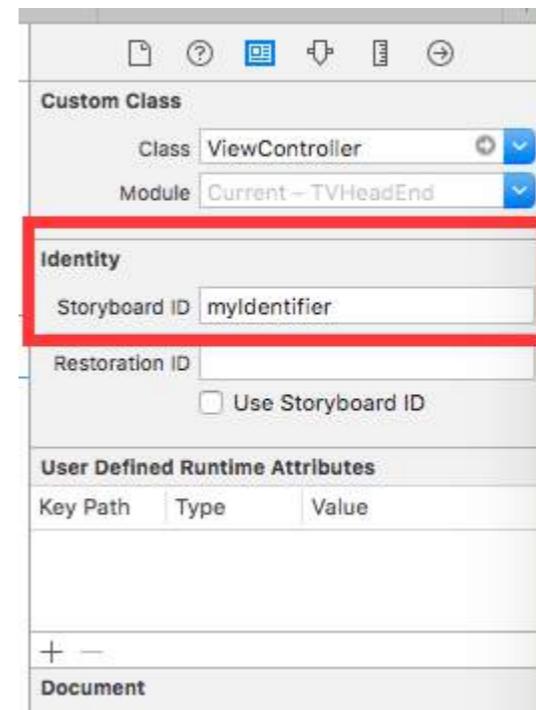
```
UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"Main" bundle:nil];
UIViewController *controller = [storyboard instantiateInitialViewController];
```

## 第36.5节：创建实例

**Swift**

```
let viewController = UIViewController()
```

**Objective-C**

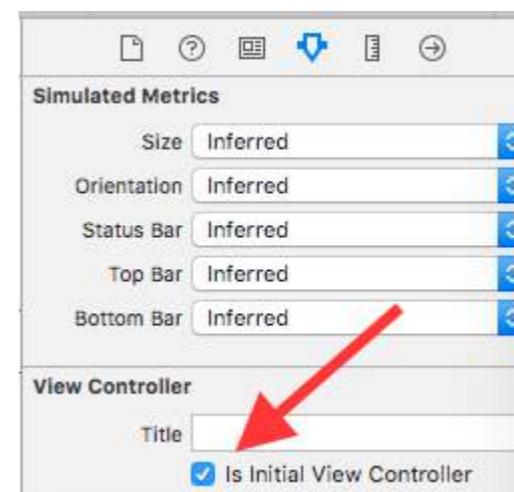


Instantiate in code:

```
UIViewController *controller = [storyboard
instantiateViewControllerWithIdentifier:@"myIdentifier"];
```

**Instantiate an initial viewController:**

Within the storyboard select the view controller, then select the attribute inspector, check the "Is Initial View Controller" box.



```
UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"Main" bundle:nil];
UIViewController *controller = [storyboard instantiateInitialViewController];
```

## Section 36.5: Create an instance

**Swift**

```
let viewController = UIViewController()
```

**Objective-C**

```
UIViewController *viewController = [UIViewController new];
```

## 第36.6节：添加/移除子视图控制器

添加子视图控制器：

```
- (void)displayContentController:(UIViewController *)vc {
    [self addChildViewController:vc];
    vc.view.frame = self.view.frame;
    [self.view addSubview:vc.view];
    [vc didMoveToParentViewController:self];
}
```

移除子视图控制器：

```
- (void)hideContentController:(UIViewController *)vc {
    [vc willMoveToParentViewController:nil];
    [vc.view removeFromSuperview];
    [vc removeFromParentViewController];
}
```

```
UIViewController *viewController = [UIViewController new];
```

## Section 36.6: Adding/removing a child view controller

To add a child view controller:

```
- (void)displayContentController:(UIViewController *)vc {
    [self addChildViewController:vc];
    vc.view.frame = self.view.frame;
    [self.view addSubview:vc.view];
    [vc didMoveToParentViewController:self];
}
```

To remove a child view controller:

```
- (void)hideContentController:(UIViewController *)vc {
    [vc willMoveToParentViewController:nil];
    [vc.view removeFromSuperview];
    [vc removeFromParentViewController];
}
```

# 第37章：UISwitch

## 第37.1节：设置开/关状态的图片

### Objective-C

```
//设置关闭状态图片  
mySwitch.offImage = [UIImage imageNamed:@"off_image"];  
[mySwitch setOffImage:[UIImage imageNamed:@"off_image"]];  
  
//设置开启状态图片  
mySwitch.onImage = [UIImage imageNamed:@"on_image"];  
[mySwitch setOnImage:[UIImage imageNamed:@"on_image"]];
```

### Swift

```
//设置关闭状态图片  
mySwitch.offImage = UIImage(named: "off_image")  
  
//设置开启状态图片  
mySwitch.onImage = UIImage(named: "on_image")
```

## 第37.2节：设置开/关

### Objective-C

```
[mySwitch setOn:YES];  
//或者  
[mySwitch setOn:YES animated:YES];
```

### Swift

```
mySwitch.setOn(false)  
//或者  
mySwitch.setOn(false, animated: false)
```

## 第37.3节：设置背景颜色

### Objective-C

```
mySwitch.backgroundColor = [UIColor yellowColor];  
[mySwitch setBackgroundColor: [UIColor yellowColor]];  
mySwitch.backgroundColor =[UIColor colorWithRed:255/255.0 green:0/255.0 blue:0/255.0 alpha:1.0];  
mySwitch.backgroundColor= [UIColor colorWithWhite: 0.5 alpha: 1.0];  
mySwitch.backgroundColor=[UIColor colorWithHue: 0.4 saturation: 0.3 brightness:0.7 alpha: 1.0];
```

### Swift

```
mySwitch.backgroundColor = UIColor.黄色  
mySwitch.backgroundColor = UIColor(red: 255.0/255, green: 0.0/255, blue: 0.0/255, alpha: 1.0)  
mySwitch.backgroundColor = UIColor(white: 0.5, alpha: 1.0)  
mySwitch.backgroundColor = UIColor(hue: 0.4,saturation: 0.3,brightness: 0.7,alpha: 1.0)
```

# Chapter 37: UISwitch

## Section 37.1: Set Image for On/Off state

### Objective-C

```
//set off-image  
mySwitch.offImage = [UIImage imageNamed:@"off_image"];  
[mySwitch setOffImage:[UIImage imageNamed:@"off_image"]];  
  
//set on-image  
mySwitch.onImage = [UIImage imageNamed:@"on_image"];  
[mySwitch setOnImage:[UIImage imageNamed:@"on_image"]];
```

### Swift

```
//set off-image  
mySwitch.offImage = UIImage(named: "off_image")  
  
//set on-image  
mySwitch.onImage = UIImage(named: "on_image")
```

## Section 37.2: Set On / Off

### Objective-C

```
[mySwitch setOn:YES];  
//or  
[mySwitch setOn:YES animated:YES];
```

### Swift

```
mySwitch.setOn(false)  
//or  
mySwitch.setOn(false, animated: false)
```

## Section 37.3: Set Background Color

### Objective-C

```
mySwitch.backgroundColor = [UIColor yellowColor];  
[mySwitch setBackgroundColor: [UIColor yellowColor]];  
mySwitch.backgroundColor =[UIColor colorWithRed:255/255.0 green:0/255.0 blue:0/255.0 alpha:1.0];  
mySwitch.backgroundColor= [UIColor colorWithWhite: 0.5 alpha: 1.0];  
mySwitch.backgroundColor=[UIColor colorWithHue: 0.4 saturation: 0.3 brightness:0.7 alpha: 1.0];
```

### Swift

```
mySwitch.backgroundColor = UIColor.yellow  
mySwitch.backgroundColor = UIColor(red: 255.0/255, green: 0.0/255, blue: 0.0/255, alpha: 1.0)  
mySwitch.backgroundColor = UIColor(white: 0.5, alpha: 1.0)  
mySwitch.backgroundColor = UIColor(hue: 0.4,saturation: 0.3,brightness: 0.7,alpha: 1.0)
```

## 第37.4节：设置色调颜色

### Objective-C

```
//关闭状态  
mySwitch.tintColor = [UIColor blueColor];  
[mySwitch setTintColor: [UIColor blueColor]];  
  
//开启状态  
mySwitch.onTintColor = [UIColor cyanColor];  
[mySwitch setOnTintColor: [UIColor cyanColor]];
```

### Swift

```
//关闭状态  
mySwitch.tintColor = UIColor.blueColor()  
  
//开启状态  
mySwitch.onTintColor = UIColor.cyanColor()
```

## Section 37.4: Set Tint Color

### Objective-C

```
//for off-state  
mySwitch.tintColor = [UIColor blueColor];  
[mySwitch setTintColor: [UIColor blueColor]];  
  
//for on-state  
mySwitch.onTintColor = [UIColor cyanColor];  
[mySwitch setOnTintColor: [UIColor cyanColor]];
```

### Swift

```
//for off-state  
mySwitch.tintColor = UIColor.blueColor()  
  
//for on-state  
mySwitch.onTintColor = UIColor.cyanColor()
```

# 第38章：UICollectionView

## 第38.1节：创建UICollectionView

初始化一个带有CGRect框架的UICollectionView：

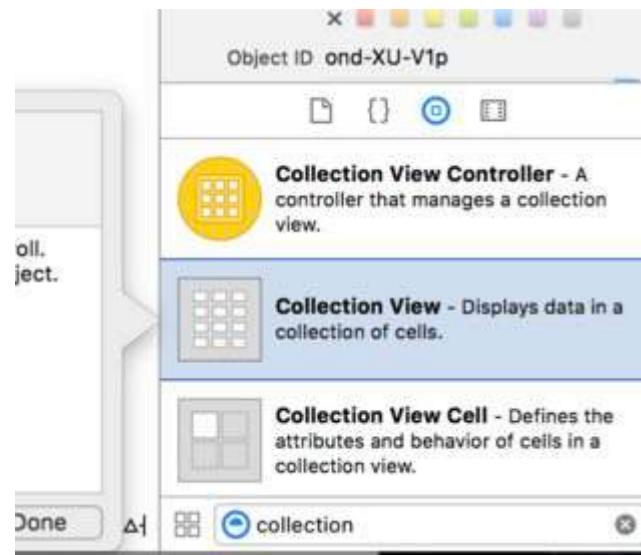
Swift:

```
let collection = UICollectionView(frame: CGRect(x: 0, y: 0, width: 200, height: 21))
```

Objective C:

```
UICollectionView *collection = [[UICollectionView alloc] initWithFrame:CGRectMake(0, 0, 200, 21)];
```

你也可以在界面构建器（Interface Builder）中创建一个UICollectionView



## 第38.2节：UICollectionView - 数据源

每个集合视图必须有一个Datasource对象。该Datasource对象是你的应用将在UICollectionView中显示的内容。

至少，所有Datasource对象必须实现

collectionView:numberOfItemsInSection:和collectionView:cellForItemAtIndexPath:方法。

必需的方法

Swift

```
func collectionView(collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
    // 返回该节中的项目数量
    let sectionArray = _data[section]
    return sectionArray.count
}

func collectionView(collectionView: UICollectionView, cellForItemAtIndexPath indexPath: NSIndexPath) -> UICollectionViewCell {
    let cell = collectionView.dequeueReusableCellWithReuseIdentifier(MyCellID)
    // 如果你使用自定义的单元格类，则将返回的单元格进行类型转换，例如：
    // as! MyCollectionViewCellClass
    // 否则当你尝试使用该类的功能时会出现错误。
}
```

# Chapter 38: UICollectionView

## Section 38.1: Create a UICollectionView

Initialize a UICollectionView with a CGRect frame:

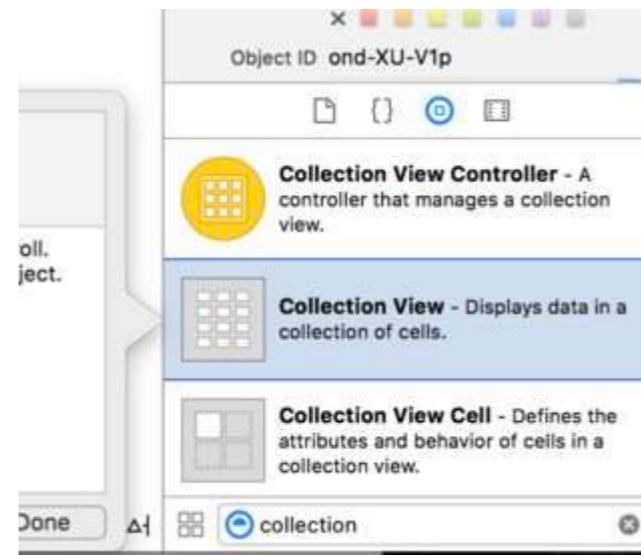
Swift:

```
let collection = UICollectionView(frame: CGRect(x: 0, y: 0, width: 200, height: 21))
```

Objective C:

```
UICollectionView *collection = [[UICollectionView alloc] initWithFrame:CGRectMake(0, 0, 200, 21)];
```

You can also create a UICollectionView in Interface Builder



## Section 38.2: UICollectionView - Datasource

Every collection view must have a Datasource object. The Datasource object is the content that your app will display within the UICollectionView. At a minimum, all Datasource objects must implement the collectionView:numberOfItemsInSection: and collectionView:cellForItemAtIndexPath: methods.

Required Methods

Swift

```
func collectionView(collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
    // Return how many items in section
    let sectionArray = _data[section]
    return sectionArray.count
}

func collectionView(collectionView: UICollectionView, cellForItemAtIndexPath indexPath: NSIndexPath) -> UICollectionViewCell {
    let cell = collectionView.dequeueReusableCellWithReuseIdentifier(MyCellID)
    // If you use a custom cell class then cast the cell returned, like:
    // as! MyCollectionViewCellClass
    // or you will have errors when you try to use features of that class.
}
```

```
// 在这里自定义你的单元格，默认的 UICollectionViewCell 不包含任何内置的
//文本或图像视图（如 UITableView），但可以添加一些，
//或者可以使用自定义的 UICollectionViewCell 子类
return cell
}
```

## Objective C

```
- (NSInteger)collectionView:(UICollectionView*)collectionView
numberOfItemsInSection:(NSInteger)section {
    // 返回该节中的项目数量
    NSArray *sectionArray = [_data objectAtIndex:section];
    return [sectionArray count];
}

- (UICollectionViewCell *)collectionView:(UICollectionView *)collectionView
cellForItemAtIndexPath:(NSIndexPath *)indexPath {
    // 返回一个单元格
    UICollectionViewCell *newCell = [self.collectionView
        dequeueReusableCellWithReuseIdentifier:MyCellID
        forIndexPath:indexPath];
    // 在这里自定义你的单元格，默认的 UICollectionViewCell 不包含任何内置的
    //文本或图像视图（如 UITableView），但可以添加一些，
    //或者可以使用自定义的 UICollectionViewCell 子类
    return newCell;
}
```

## 第38.3节：Collection View 的基本 Swift 示例

### 创建一个新项目

可以只是一个单视图应用程序。

### 添加代码

创建一个新的 Cocoa Touch 类文件（文件 > 新建 > 文件... > iOS > Cocoa Touch 类）。命名为 MyCollectionViewCell。该类将保存你在故事板中添加到单元格的视图的出口（outlets）。

```
import UIKit
class MyCollectionViewCell: UICollectionViewCell {

    @IBOutlet weak var myLabel: UILabel!
}
```

我们稍后会连接这个出口。

打开 ViewController.swift 并确保包含以下内容：

```
import UIKit
class ViewController: UIViewController, UICollectionViewDataSource, UICollectionViewDelegate {

    let reuseIdentifier = "cell" // 同时在故事板中将此字符串作为单元格标识符输入
    var items = ["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15",
    "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30", "31",
    "32", "33", "34", "35", "36", "37", "38", "39", "40", "41", "42", "43", "44", "45", "46", "47",
    "48"]
```

```
//Customize your cell here, default UICollectionViewCells do not contain any inherent
//text or image views (like UITableView), but some could be added,
//or a custom UICollectionViewCell sub-class could be used
return cell
}
```

## Objective C

```
- (NSInteger)collectionView:(UICollectionView*)collectionView
numberOfItemsInSection:(NSInteger)section {
    // Return how many items in section
    NSArray *sectionArray = [_data objectAtIndex:section];
    return [sectionArray count];
}

- (UICollectionViewCell *)collectionView:(UICollectionView *)collectionView
cellForItemAtIndexPath:(NSIndexPath *)indexPath {
    // Return a cell
    UICollectionViewCell *newCell = [self.collectionView
        dequeueReusableCellWithReuseIdentifier:MyCellID
        forIndexPath:indexPath];
    // Customize your cell here, default UICollectionViewCells do not contain any inherent
    //text or image views (like UITableView), but some could be added,
    //or a custom UICollectionViewCell sub-class could be used
    return newCell;
}
```

## Section 38.3: Basic Swift example of a Collection View

### Create a new project

It can be just a Single View Application.

### Add the code

Create a new Cocoa Touch Class file (File > New > File... > iOS > Cocoa Touch Class). Name it MyCollectionViewCell. This class will hold the outlets for the views that you add to your cell in the storyboard.

```
import UIKit
class MyCollectionViewCell: UICollectionViewCell {

    @IBOutlet weak var myLabel: UILabel!
}
```

We will connect this outlet later.

Open ViewController.swift and make sure you have the following content:

```
import UIKit
class ViewController: UIViewController, UICollectionViewDataSource, UICollectionViewDelegate {

    let reuseIdentifier = "cell" // also enter this string as the cell identifier in the storyboard
    var items = ["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15",
    "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30", "31",
    "32", "33", "34", "35", "36", "37", "38", "39", "40", "41", "42", "43", "44", "45", "46", "47",
    "48"]
```

```

// MARK: - UICollectionViewDataSource 协议

// 告诉集合视图需要创建多少个单元格
func collectionView(collectionView: UICollectionView, numberOfItemsInSection section: Int) ->
Int {
    return self.items.count
}

// 为每个单元格索引路径创建一个单元格
func collectionView(collectionView: UICollectionView, cellForItemAtIndexPath indexPath:
NSIndexPath) -> UICollectionViewCell {

    // 获取对故事板中单元格的引用
    let cell = collectionView.dequeueReusableCellWithReuseIdentifier(reuseIdentifier,
forIndexPath: indexPath) as! MyCollectionViewCell

    // 在自定义类中使用 outlet 获取单元格中 UILabel 的引用
    cell.myLabel.text = self.items[indexPath.item]
    cell.backgroundColor = UIColor.yellowColor() // 使单元格在示例中更明显
project

    return cell
}

// MARK: - UICollectionViewDelegate 协议

func collectionView(collectionView: UICollectionView, didSelectItemAtIndexPath indexPath:
NSIndexPath) {
    // 处理点击事件
    print("你选择了第 #\((indexPath.item)) 个单元格！")
}

```

#### 注意事项

- UICollectionViewDataSource 和 UICollectionViewDelegate 是集合视图遵循的协议。你也可以添加 UICollectionViewDelegateFlowLayout 协议来编程更改视图的大小，但这不是必须的。
- 我们这里只是在网格中放置简单的字符串，但你以后当然也可以放置图片。

#### 设置故事板

将一个集合视图拖到故事板中的视图控制器。如果需要，可以添加约束使其填满父视图。

```

// MARK: - UICollectionViewDataSource protocol

// tell the collection view how many cells to make
func collectionView(collectionView: UICollectionView, numberOfItemsInSection section: Int) ->
Int {
    return self.items.count
}

// make a cell for each cell index path
func collectionView(collectionView: UICollectionView, cellForItemAtIndexPath indexPath:
NSIndexPath) -> UICollectionViewCell {

    // get a reference to our storyboard cell
    let cell = collectionView.dequeueReusableCellWithReuseIdentifier(reuseIdentifier,
forIndexPath: indexPath) as! MyCollectionViewCell

    // Use the outlet in our custom class to get a reference to the UILabel in the cell
    cell.myLabel.text = self.items[indexPath.item]
    cell.backgroundColor = UIColor.yellowColor() // make cell more visible in our example
project

    return cell
}

// MARK: - UICollectionViewDelegate protocol

func collectionView(collectionView: UICollectionView, didSelectItemAtIndexPath indexPath:
NSIndexPath) {
    // handle tap events
    print("You selected cell #\((indexPath.item)!")
}

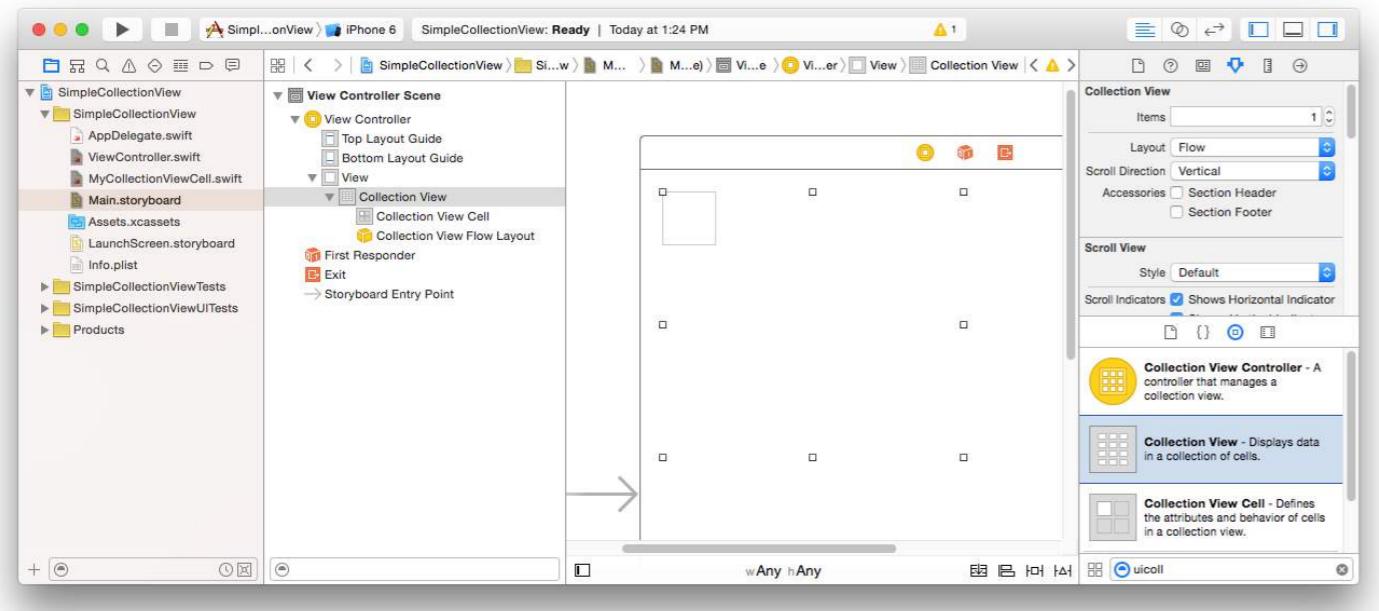
```

#### Notes

- `UICollectionViewDataSource` and `UICollectionViewDelegate` are the protocols that the collection view follows. You could also add the `UICollectionViewDelegateFlowLayout` protocol to change the size of the views programmatically, but it isn't necessary.
- We are just putting simple strings in our grid, but you could certainly do images later.

#### Setup the storyboard

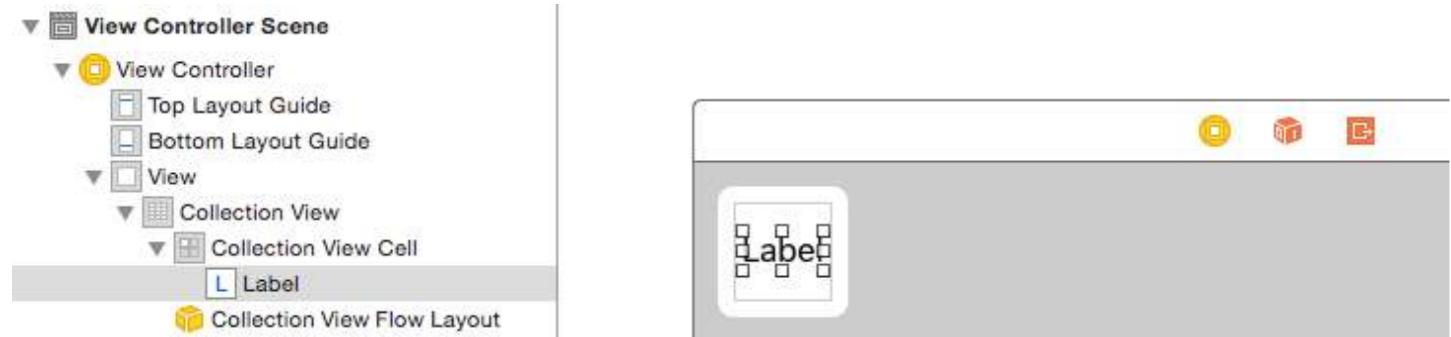
Drag a Collection View to the View Controller in your storyboard. You can add constraints to make it fill the parent view if you like.



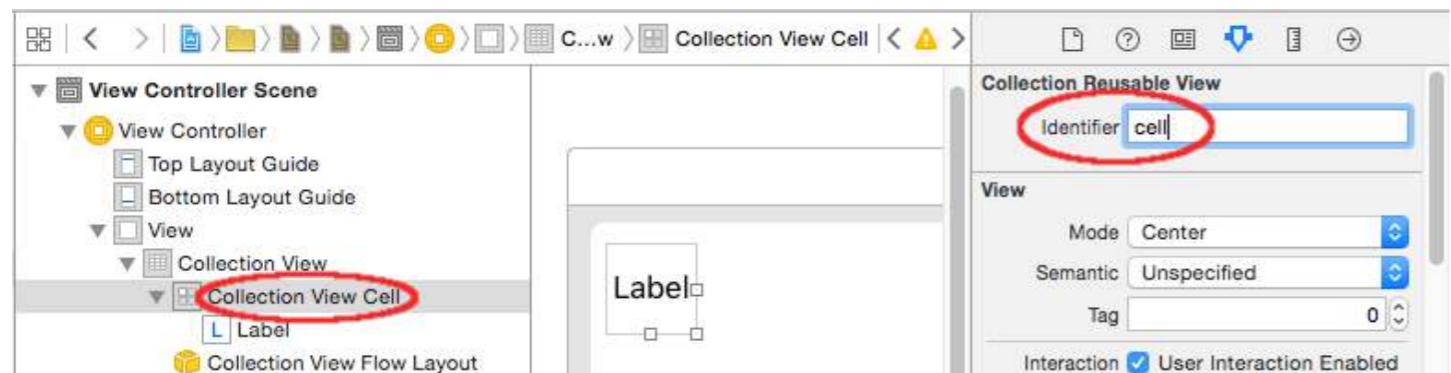
确保属性检查器中的默认设置也是如此

- 项目：1
- 布局：流式布局

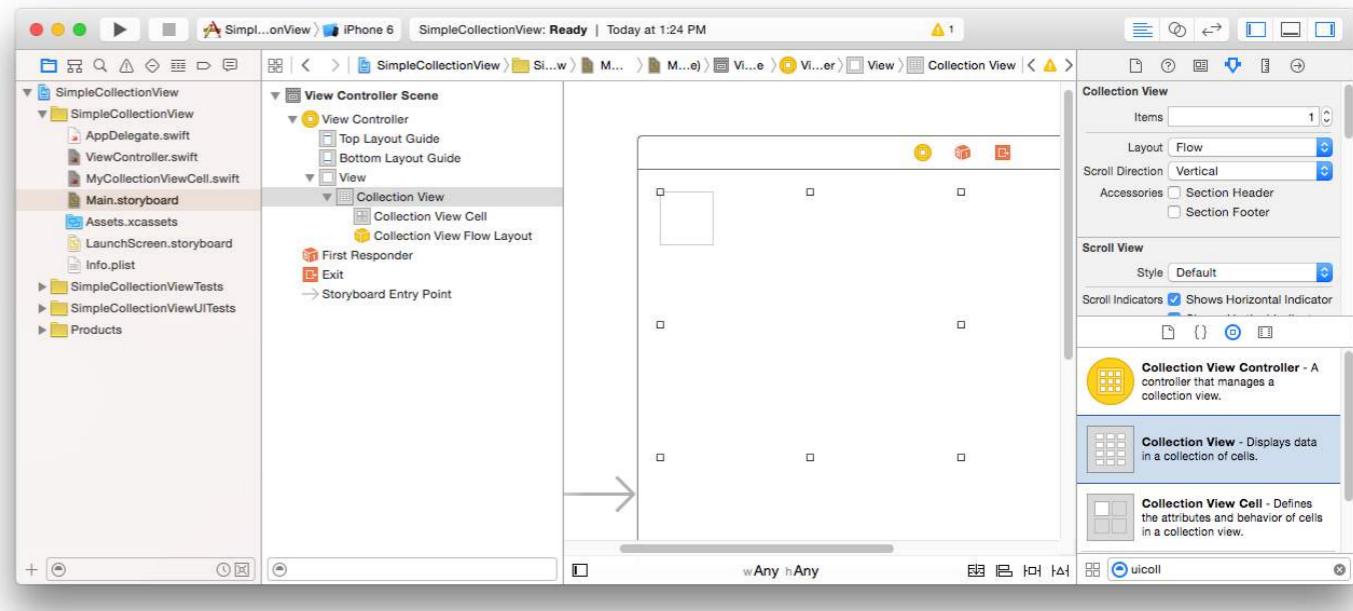
集合视图左上角的小框是一个集合视图单元格。我们将其用作原型单元格。将一个标签拖入单元格并居中。如果需要，可以调整单元格边界大小并添加约束以使标签居中。



在集合视图单元格的属性检查器的标识符框中输入“cell”（不带引号）。注意，这与 ViewController.swift 中的 let reuseIdentifier = "cell" 的值相同。



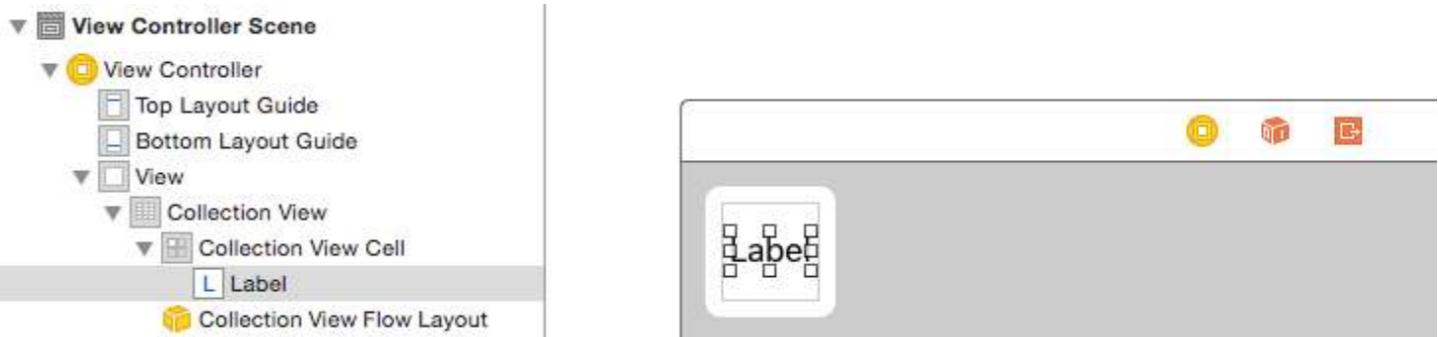
在单元格的身份检查器中，将类名设置为 MyCollectionViewCell，这是我们自定义的类。



Make sure that your defaults in the Attribute Inspector are also

- Items: 1
- Layout: Flow

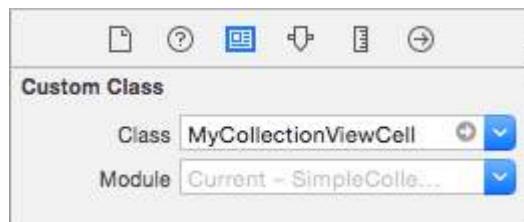
The little box in the top left of the Collection View is a Collection View Cell. We will use it as our prototype cell. Drag a Label into the cell and center it. You can resize the cell borders and add constraints to center the Label if you like.



Write "cell" (without quotes) in the Identifier box of the Attributes Inspector for the Collection View Cell. Note that this is the same value as `let reuseIdentifier = "cell"` in ViewController.swift.

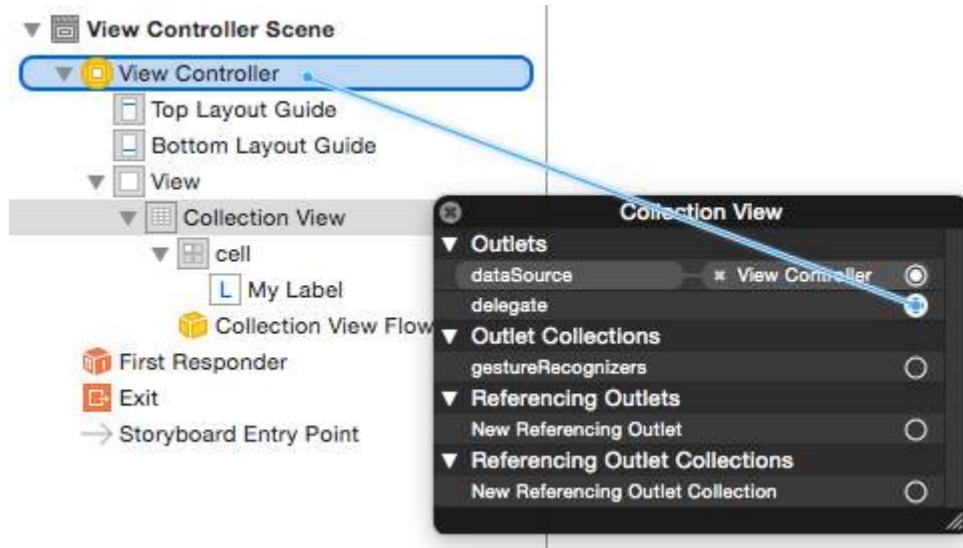


And in the Identity Inspector for the cell, set the class name to MyCollectionViewCell, our custom class that we made.



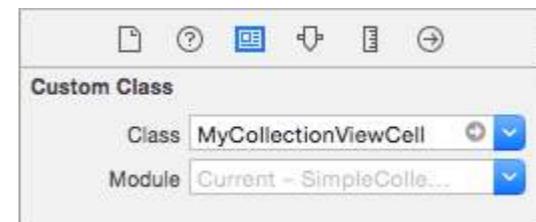
### 连接出站口

- 将集合单元格中的标签连接到MyCollectionViewCell类中的myLabel。（你可以Control-drag。）将集合视图的
- delegate和dataSource连接到视图控制器。（在文档大纲中右键点击集合视图，然后点击并拖动加号箭头到视图控制器。）



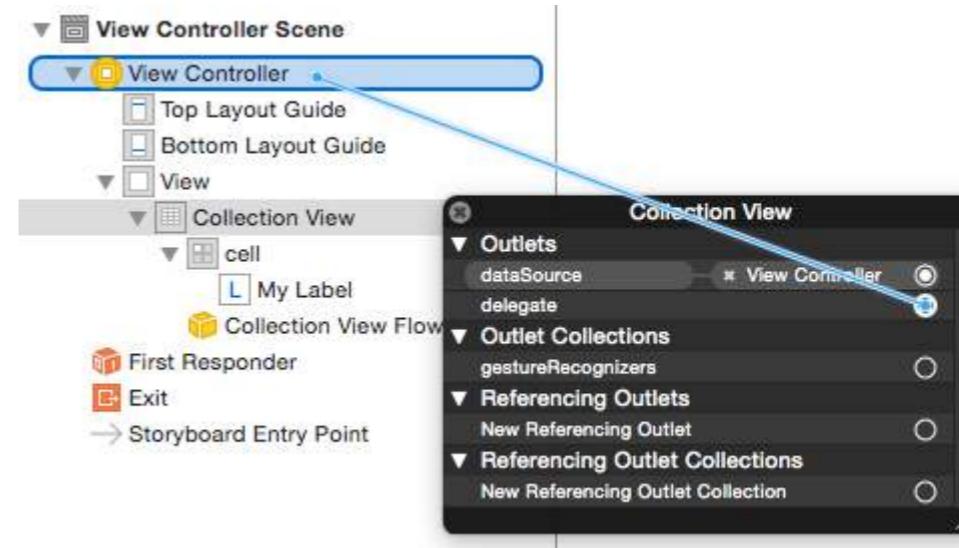
### 完成

这是在为单元格中的标签添加居中约束并将集合视图固定到父视图边缘后显示的效果。



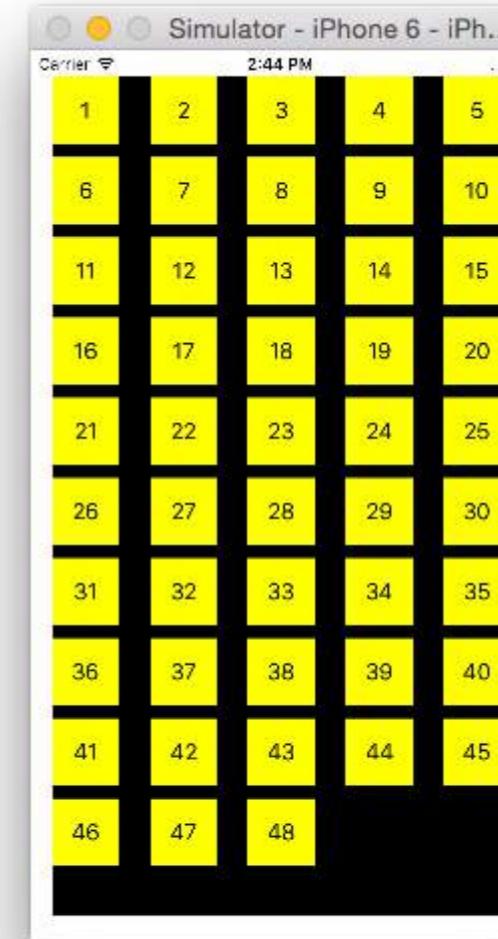
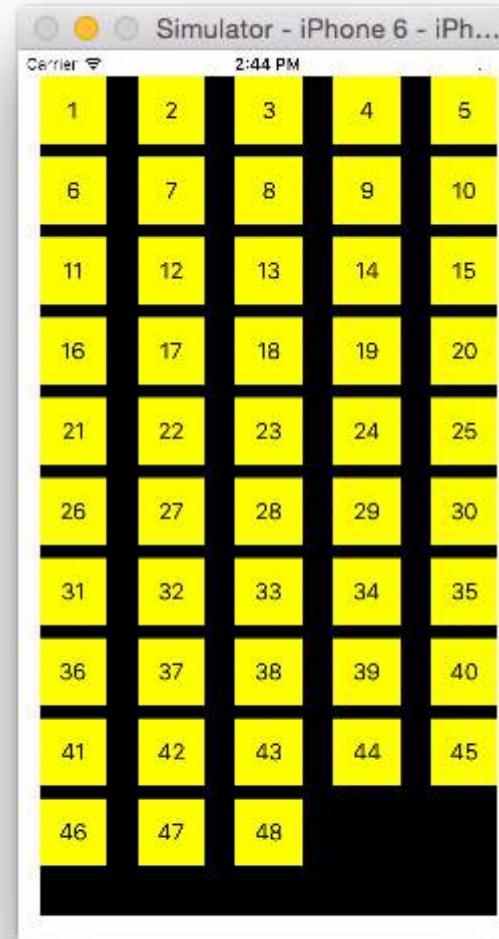
### Hook up the outlets

- Hook the Label in the collection cell to myLabel in the MyCollectionViewCell class. (You can Control-drag.)
- Hook the Collection View delegate and dataSource to the View Controller. (Right click Collection View in the Document Outline. Then click and drag the plus arrow up to the View Controller.)



### Finished

Here is what it looks like after adding constraints to center the Label in the cell and pinning the Collection View to the walls of the parent.

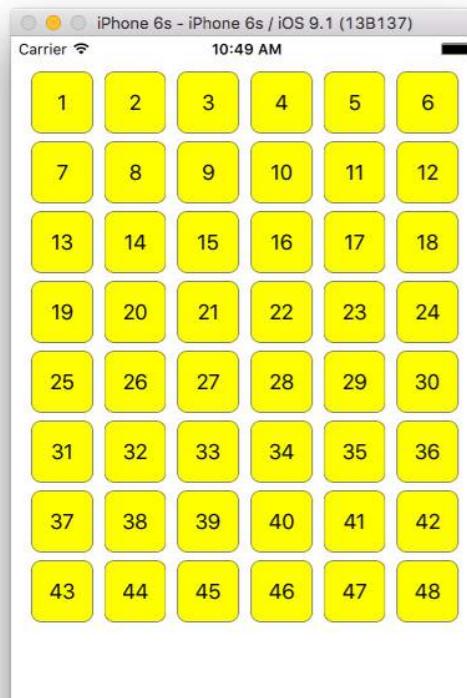


## 改进

如果您想改进外观，请参阅此示例来源的原始帖子。

## Making Improvements

If you want to make improvements on the appearance, [see the original post that this example comes from](#).



- [一个简单的UICollectionView教程](#)
- [UICollectionView教程第1部分：入门](#)
- [UICollectionView教程第2部分：可重用视图和单元格选择](#)

## 第38.4节：使用

DataSource和FlowLayout管理多个Collection视图

这里我们管理多个集合视图的代理方法及其didselect事件。

```
extension ProductsVC: UICollectionViewDelegate, UICollectionViewDataSource{

    // 标记: - UICollectionViewDataSource
    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
        guard collectionView == collectionCategory else {
            return arrOfProducts.count
        }
        return arrOfCategory.count
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {
        guard collectionView == collectionProduct else {
            let cell = collectionView.dequeueReusableCell(withIdentifier: "ProductCategoryCell", for: indexPath) as! ProductCategoryCell
            cell.viewBackground.layer.borderWidth = 0.5
            //Do some thing as per use
            return cell
        }

        let cell = collectionView.dequeueReusableCell(withIdentifier: cellIdentifier, for: indexPath) as! ProductCell
        cell.contentView.layer.borderWidth = 0.5
        cell.contentView.layer.borderColor = UIColor.black.cgColor
        let json = arrOfProducts[indexPath.row]
        //Do something as per use

        return cell
    }

    func collectionView(_ collectionView: UICollectionView, didSelectItemAt indexPath: IndexPath) {
        guard collectionView == collectionCategory else {
            let json = arrOfProducts[indexPath.row]
            // 在这里对 collectionProduct 进行操作
            return
        }
        let json = arrOfCategory[indexPath.row] as [String: AnyObject]
        let id = json["cId"] as? String ?? ""
        // 进行一些操作
    }
}

extension ProductsVC: UICollectionViewDelegateFlowLayout{
    // MARK: - UICollectionViewDelegateFlowLayout
    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout:
```

## Further study

- [A Simple UICollectionView Tutorial](#)
- [UICollectionView Tutorial Part 1: Getting Started](#)
- [UICollectionView Tutorial Part 2: Reusable Views and Cell Selection](#)

## Section 38.4: Manage Multiple Collection view with DataSource and Flowlayout

Here we are managing multiple collection there delegate methods with didselect events.

```
extension ProductsVC: UICollectionViewDelegate, UICollectionViewDataSource{

    // MARK: - UICollectionViewDataSource
    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
        guard collectionView == collectionCategory else {
            return arrOfProducts.count
        }
        return arrOfCategory.count
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {
        guard collectionView == collectionProduct else {
            let cell = collectionView.dequeueReusableCell(withIdentifier: "ProductCategoryCell", for: indexPath) as! ProductCategoryCell
            cell.viewBackground.layer.borderWidth = 0.5
            //Do some thing as per use
            return cell
        }

        let cell = collectionView.dequeueReusableCell(withIdentifier: cellIdentifier, for: indexPath) as! ProductCell
        cell.contentView.layer.borderWidth = 0.5
        cell.contentView.layer.borderColor = UIColor.black.cgColor
        let json = arrOfProducts[indexPath.row]
        //Do something as per use

        return cell
    }

    func collectionView(_ collectionView: UICollectionView, didSelectItemAt indexPath: IndexPath) {
        guard collectionView == collectionCategory else {
            let json = arrOfProducts[indexPath.row]
            // Do something for collectionProduct here
            return
        }
        let json = arrOfCategory[indexPath.row] as [String: AnyObject]
        let id = json["cId"] as? String ?? ""
        // Do something
    }
}

extension ProductsVC: UICollectionViewDelegateFlowLayout{

    // MARK: - UICollectionViewDelegateFlowLayout
    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout:
```

```

UICollectionViewLayout, sizeForItemAt indexPath: IndexPath) -> CGSize {
    let collectionWidth = collectionView.bounds.width
    guard collectionView == collectionProduct else {
        var itemWidth = collectionWidth / 4 - 1;
        if(UI_USER_INTERFACE_IDIOM() == .pad) {
            itemWidth = collectionWidth / 4 - 1;
        }
        return CGSize(width: itemWidth, height: 50)
    }

    var itemWidth = collectionWidth / 2 - 1;
    if(UI_USER_INTERFACE_IDIOM() == .pad) {
        itemWidth = collectionWidth / 4 - 1;
    }
    return CGSize(width: itemWidth, height: 250);
}

func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, minimumInteritemSpacingForSectionAt section: Int) -> CGFloat {
    return 1
}

func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, minimumLineSpacingForSectionAt section: Int) -> CGFloat {
    return 1
}
}

```

```

UICollectionViewLayout, sizeForItemAt indexPath: IndexPath) -> CGSize {
    let collectionWidth = collectionView.bounds.width
    guard collectionView == collectionProduct else {
        var itemWidth = collectionWidth / 4 - 1;
        if(UI_USER_INTERFACE_IDIOM() == .pad) {
            itemWidth = collectionWidth / 4 - 1;
        }
        return CGSize(width: itemWidth, height: 50)
    }

    var itemWidth = collectionWidth / 2 - 1;
    if(UI_USER_INTERFACE_IDIOM() == .pad) {
        itemWidth = collectionWidth / 4 - 1;
    }
    return CGSize(width: itemWidth, height: 250);
}

func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, minimumInteritemSpacingForSectionAt section: Int) -> CGFloat {
    return 1
}

func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, minimumLineSpacingForSectionAt section: Int) -> CGFloat {
    return 1
}
}

```

23-01-2017	24-01-2017	25-01-2017
------------	------------	------------

11:00	11:15	11:30	11:45
12:00	12:15	12:30	12:45
13:00	13:15	13:30	13:45
14:00	14:15	14:30	14:45
15:00	15:15	15:30	15:45
16:00	16:15	16:30	16:45

## 第38.5节：UICollectionViewDelegate 设置和项目选择

有时，如果需要将某个操作绑定到集合视图的单元格选择上，您必须实现 `UICollectionViewDelegate` 协议。

假设集合视图位于一个 `UIViewController` `MyViewController` 内。

### Objective-C

在你的 `MyViewController.h` 中声明它实现了 `UICollectionViewDelegate` 协议，如下所示

```
@interface MyViewController : UIViewController <UICollectionViewDelegate, /*之前已有的  
delegate, 作为 UICollectionViewDataSource */>
```

### Swift

在你的 `MyViewController.swift` 中添加以下内容

```
class MyViewController : UICollectionViewDelegate {
```

23-01-2017	24-01-2017	25-01-2017
------------	------------	------------

11:00	11:15	11:30	11:45
12:00	12:15	12:30	12:45
13:00	13:15	13:30	13:45
14:00	14:15	14:30	14:45
15:00	15:15	15:30	15:45
16:00	16:15	16:30	16:45

## Section 38.5: UICollectionViewDelegate setup and item selection

Sometimes, if an action should be bind to a collection view's cell selection, you have to implement the `UICollectionViewDelegate` protocol.

Let's say the collection view is inside a `UIViewController` `MyViewController`.

### Objective-C

In your `MyViewController.h` declares that it implements the `UICollectionViewDelegate` protocol, as below

```
@interface MyViewController : UIViewController <UICollectionViewDelegate, /* previous existing  
delegate, as UICollectionViewDataSource */>
```

### Swift

In your `MyViewController.swift` add the following

```
class MyViewController : UICollectionViewDelegate {
```

```
}
```

必须实现的方法是

#### Objective-C

```
- (void)collectionView:(UICollectionView *)collectionView didSelectItemAtIndexPath:(NSIndexPath *)indexPath
```

```
    UICollectionViewCell *cell = [collectionView cellForItemAtIndexPath:indexPath];
    cell.backgroundColor = [UIColor greenColor];
```

举个例子，我们可以将选中单元格的背景色设置为绿色。

#### Objective-C

```
- (void)collectionView:(UICollectionView *)collectionView didSelectItemAtIndexPath:(NSIndexPath *)indexPath
```

```
    UICollectionViewCell *cell = [collectionView cellForItemAtIndexPath:indexPath];
    cell.backgroundColor = [UIColor greenColor];
```

#### Swift

```
class MyViewController : UICollectionViewDelegate {
    func collectionView(collectionView: UICollectionView, didSelectItemAtIndexPath indexPath: NSIndexPath)
    {
        var cell : UICollectionViewCell = collectionView.cellForItemAtIndexPath(indexPath)!
        cell.backgroundColor = UIColor.greenColor()
    }
}
```

## 第38.6节：通过代码创建集合视图

#### Swift

```
func createCollectionView() {
    let layout: UICollectionViewFlowLayout = UICollectionViewFlowLayout()
    let collectionView = UICollectionView(frame: CGRectMake(x: 0, y: 0, width: view.frame.width,
height: view.frame.height), collectionViewLayout: layout)
    collectionView.dataSource = self
    collectionView.delegate = self
    view.addSubview(collectionView)
}
```

#### Objective-C

```
}
```

The method that must be implemented is

#### Objective-C

```
- (void)collectionView:(UICollectionView *)collectionView didSelectItemAtIndexPath:(NSIndexPath *)indexPath
```

```
{}
```

#### Swift

```
func collectionView(collectionView: UICollectionView, didSelectItemAtIndexPath indexPath: NSIndexPath)
```

```
{}
```

As just an example we can set the background color of selected cell to green.

#### Objective-C

```
- (void)collectionView:(UICollectionView *)collectionView didSelectItemAtIndexPath:(NSIndexPath *)indexPath
```

```
{
    UICollectionViewCell *cell = [collectionView cellForItemAtIndexPath:indexPath];
    cell.backgroundColor = [UIColor greenColor];
}
```

#### Swift

```
class MyViewController : UICollectionViewDelegate {
    func collectionView(collectionView: UICollectionView, didSelectItemAtIndexPath indexPath: NSIndexPath)
    {
        var cell : UICollectionViewCell = collectionView.cellForItemAtIndexPath(indexPath)!
        cell.backgroundColor = UIColor.greenColor()
    }
}
```

## Section 38.6: Create a Collection View Programmatically

#### Swift

```
func createCollectionView() {
    let layout: UICollectionViewFlowLayout = UICollectionViewFlowLayout()
    let collectionView = UICollectionView(frame: CGRectMake(x: 0, y: 0, width: view.frame.width,
height: view.frame.height), collectionViewLayout: layout)
    collectionView.dataSource = self
    collectionView.delegate = self
    view.addSubview(collectionView)
}
```

#### Objective-C

```

- (void)createCollectionView {
    UICollectionViewFlowLayout *layout = [[UICollectionViewFlowLayout alloc] init];
    UICollectionView *collectionView = [[UICollectionView alloc] initWithFrame:CGRectMake(0, 0,
self.view.frame.size.width, self.view.frame.size.height) collectionViewLayout:layout];
    [collectionView setDataSource:self];
    [collectionView setDelegate:self];
    [self.view addSubview:collectionView];
}

```

## 第38.7节：Swift - UICollectionViewDelegateFlowLayout

```

// MARK: - UICollectionViewDelegateFlowLayout
扩展 ViewController: UICollectionViewDelegateFlowLayout {
    func collectionView(collectionView: UICollectionView, layout collectionViewLayout:
    UICollectionViewLayout, sizeForItemAtIndexPath indexPath: NSIndexPath) -> CGSize {
        返回 CGSize(宽度: 50, 高度: 50)
    }
    func collectionView(collectionView: UICollectionView, layout collectionViewLayout:
    UICollectionViewLayout, insetForSectionAtIndex section: Int) -> UIEdgeInsets {
        返回 UIEdgeInsets(上: 5, 左: 5, 下: 5, 右: 5)
    }
    func collectionView(collectionView: UICollectionView, layout collectionViewLayout:
    UICollectionViewLayout, minimumLineSpacingForSectionAtIndex section: Int) -> CGFloat {
        返回 5.0
    }
    func collectionView(collectionView: UICollectionView, layout collectionViewLayout:
    UICollectionViewLayout, minimumInteritemSpacingForSectionAtIndex section: Int) -> CGFloat {
        返回 5.0
    }
}

```

## 第38.8节：执行批量更新

你可以使用performBatchUpdates方法为你的集合视图执行复杂的动画更改。在更新块内，你可以指定多个修改，使它们同时动画执行。

```

collectionView.performBatchUpdates({
    // 执行更新
}, nil)

```

在更新块内，您可以执行插入、删除、移动和重新加载操作。以下是如何确定使用哪个indexPath：

类型	NSIndexPath
新数组中的插入索引	
旧数组中的删除索引	
移动	从：旧数组，至：新数组
重新加载	新数组或旧数组（无关紧要）

您只应对未移动但内容已更改的单元格调用重新加载。需要注意的是，移动不会刷新单元格的内容，只会移动其位置。

为了验证批量更新能正确执行，请确保用于删除、移动-从和重新加载的indexPath集合是唯一的，且用于插入、移动-至和重新加载的indexPath集合也是唯一的。

以下是一个正确的批量更新示例：

```

- (void)createCollectionView {
    UICollectionViewFlowLayout *layout = [[UICollectionViewFlowLayout alloc] init];
    UICollectionView *collectionView = [[UICollectionView alloc] initWithFrame:CGRectMake(0, 0,
self.view.frame.size.width, self.view.frame.size.height) collectionViewLayout:layout];
    [collectionView setDataSource:self];
    [collectionView setDelegate:self];
    [self.view addSubview:collectionView];
}

```

## Section 38.7: Swift - UICollectionViewDelegateFlowLayout

```

// MARK: - UICollectionViewDelegateFlowLayout
extension ViewController: UICollectionViewDelegateFlowLayout {
    func collectionView(collectionView: UICollectionView, layout collectionViewLayout:
    UICollectionViewLayout, sizeForItemAtIndexPath indexPath: NSIndexPath) -> CGSize {
        return CGSize(width: 50, height: 50)
    }
    func collectionView(collectionView: UICollectionView, layout collectionViewLayout:
    UICollectionViewLayout, insetForSectionAtIndex section: Int) -> UIEdgeInsets {
        return UIEdgeInsets(top: 5, left: 5, bottom: 5, right: 5)
    }
    func collectionView(collectionView: UICollectionView, layout collectionViewLayout:
    UICollectionViewLayout, minimumLineSpacingForSectionAtIndex section: Int) -> CGFloat {
        return 5.0
    }
    func collectionView(collectionView: UICollectionView, layout collectionViewLayout:
    UICollectionViewLayout, minimumInteritemSpacingForSectionAtIndex section: Int) -> CGFloat {
        return 5.0
    }
}

```

## Section 38.8: Performing batch updates

You can animate complex changes to your collection view using the performBatchUpdates method. Inside the update block, you can specify several modifications to have them animate all at once.

```

collectionview.performBatchUpdates({
    // Perform updates
}, nil)

```

Inside the update block, you can perform insertions, deletions, moves, and reloads. Here is how to determine which indexPath to use:

Type	NSIndexPath
Insertion	Index in new array
Deletion	Index in old array
Move	from: old array, to: new array
Reload	either new or old array (it shouldn't matter)

You should only call reload on cells that have not moved, but their content has changed. It is important to note that a move will not refresh the content of a cell, but only move its location.

To verify that your batch update will be performed correctly, make sure the set of indexPaths for deletion, move-from, and reload are unique, and the set of indexPaths for insertion, move-to, and reload are unique.

Here's an example of a proper batch update:

```
let from = [1, 2, 3, 4, 5]
let to = [1, 3, 6, 4, 5]

collectionView.performBatchUpdates({
    collectionView.insertItemsAtIndexPaths([NSIndexPath(forItem: 2, inSection: 0)])
    collectionView.deleteItemsAtIndexPaths([NSIndexPath(forItem: 1, inSection: 0)])
    collectionView.moveItemAtIndexPath(NSIndexPath(forItem: 2, inSection: 0),
                                    toIndexPath: NSIndexPath(forItem: 1, inSection: 0))
}, nil)
```

```
let from = [1, 2, 3, 4, 5]
let to = [1, 3, 6, 4, 5]

collectionView.performBatchUpdates({
    collectionView.insertItemsAtIndexPaths([NSIndexPath(forItem: 2, inSection: 0)])
    collectionView.deleteItemsAtIndexPaths([NSIndexPath(forItem: 1, inSection: 0)])
    collectionView.moveItemAtIndexPath(NSIndexPath(forItem: 2, inSection: 0),
                                    toIndexPath: NSIndexPath(forItem: 1, inSection: 0))
}, nil)
```

# 第39章：UISearchController

参数	详情
UISearchController.searchBar	要安装在界面中的搜索栏。 (只读)
UISearchController.searchResultsUpdater	负责更新搜索结果控制器内容的对象。
UISearchController.isActive	搜索界面的呈现状态。
UISearchController.obscreensBackgroundDuringPresentation	一个布尔值，指示搜索时底层内容是否被遮挡。
UISearchController.dimsBackgroundDuringPresentation	一个布尔值，指示搜索时底层内容是否被变暗。
UISearchController.hidesNavigationBarDuringPresentation	一个布尔值，指示搜索时导航栏是否应被隐藏。
UIViewController.definesPresentationContext	一个布尔值，指示当视图控制器或其子孙视图控制器呈现视图控制器时，该视图控制器的视图是否被覆盖。
UIViewController.navigationItem.titleView	当接收者是顶部项目时，导航栏中央显示的自定义视图，可以放置搜索栏。
UITableViewController.tableView.tableHeaderView	返回一个显示在表格上方的辅助视图，可以放置搜索栏。

## 第39.1节：导航栏标题中的搜索栏

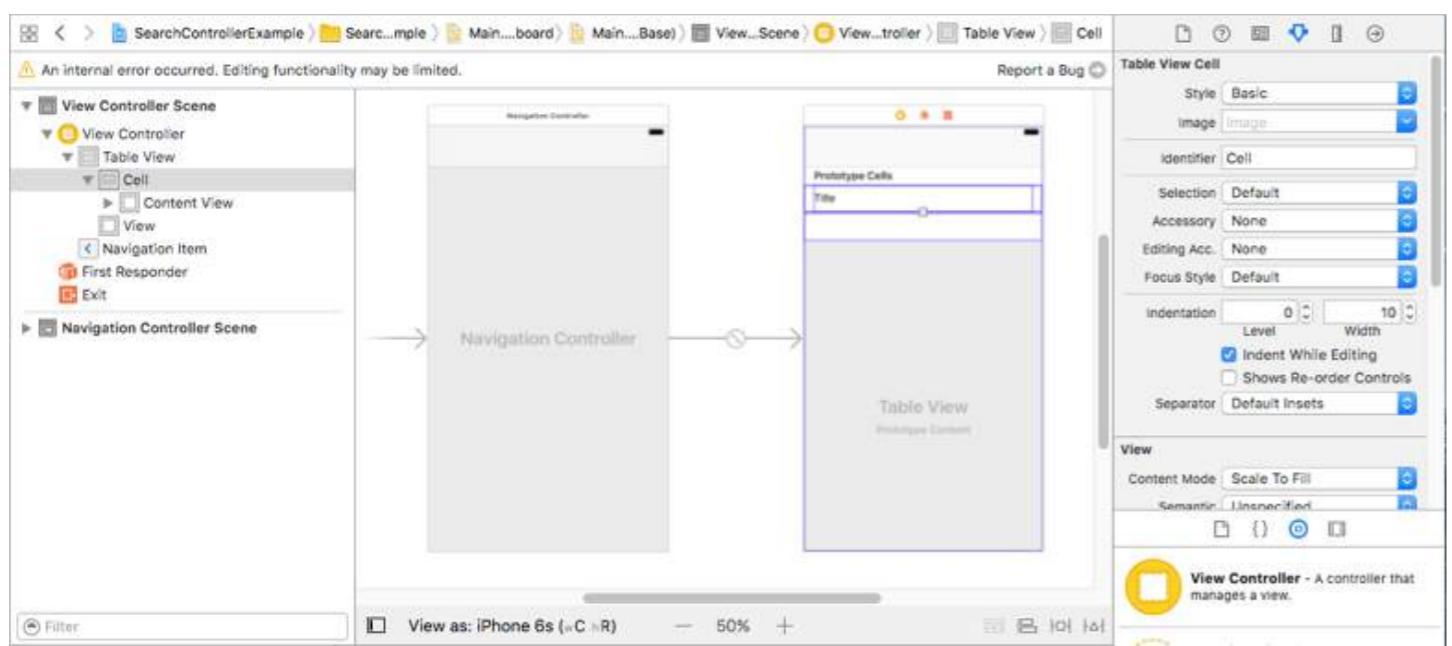
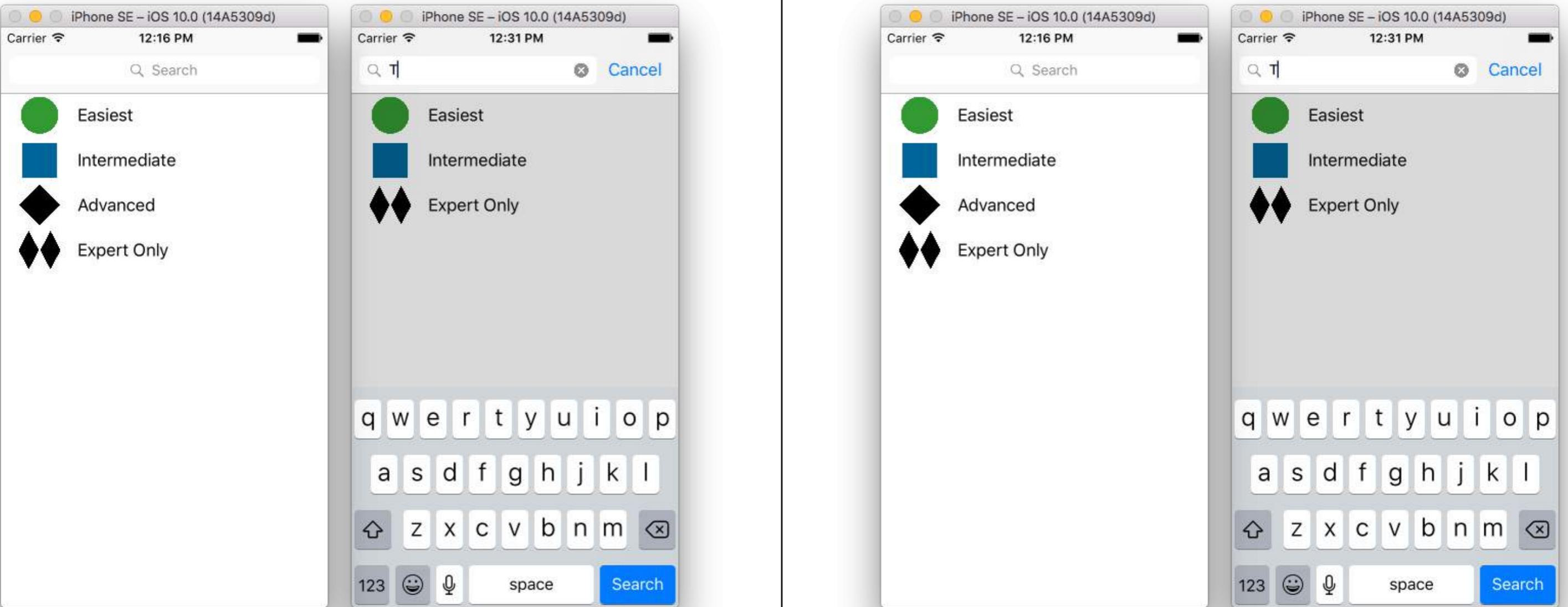
此示例使用搜索控制器来过滤表视图控制器中的数据。搜索栏放置在表视图所嵌入的导航栏中。

# Chapter 39: UISearchController

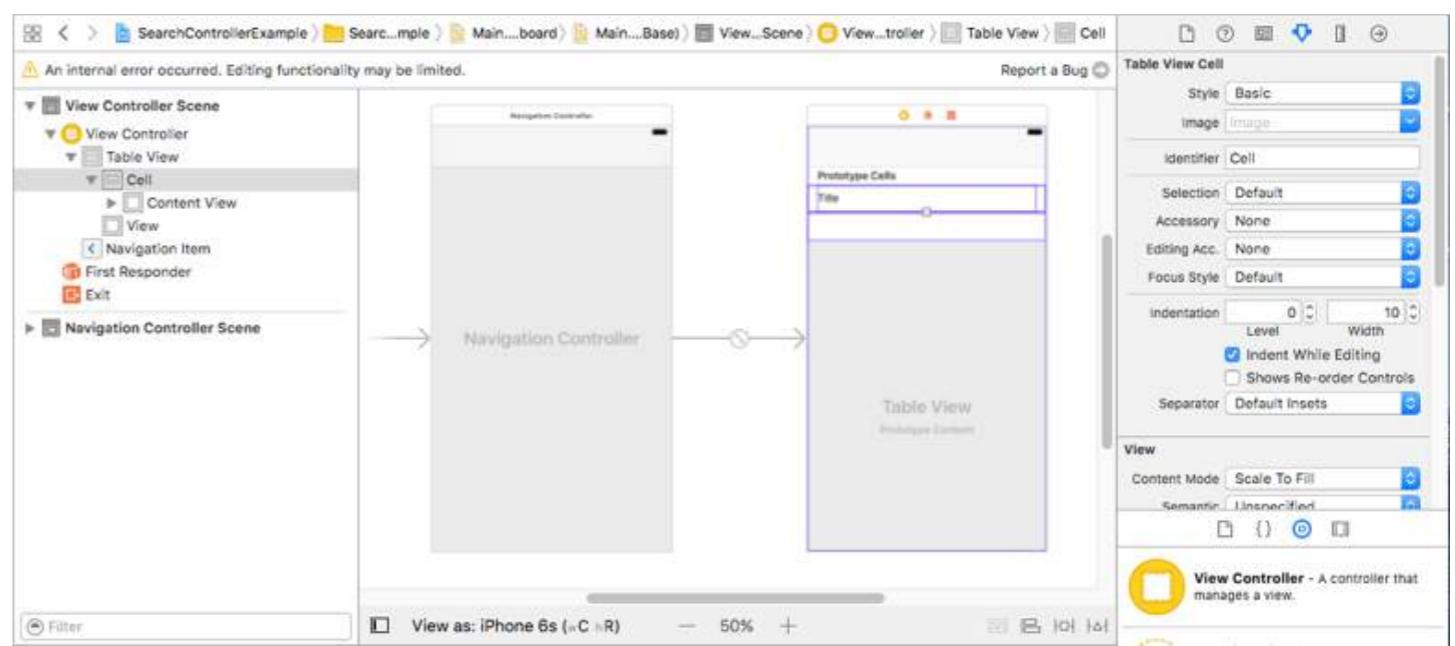
Parameter	Details
UISearchController.searchBar	The search bar to install in your interface. ( <i>read-only</i> )
UISearchController.searchResultsUpdater	The object responsible for updating the contents of the search results controller.
UISearchController.isActive	The presented state of the search interface.
UISearchController.obscreensBackgroundDuringPresentation	A Boolean indicating whether the underlying content is obscured during a search.
UISearchController.dimsBackgroundDuringPresentation	A Boolean indicating whether the underlying content is dimmed during a search.
UISearchController.hidesNavigationBarDuringPresentation	A Boolean indicating whether the navigation bar should be hidden when searching.
UIViewController.definesPresentationContext	A Boolean value that indicates whether this view controller's view is covered when the view controller or one of its descendants presents a view controller.
UIViewController.navigationItem.titleView	A custom view displayed in the center of the navigation bar when the receiver is the top item in which a search bar can be placed.
UITableViewController.tableView.tableHeaderView	Returns an accessory view that is displayed above the table in which a search bar can be placed.

## Section 39.1: Search Bar in Navigation Bar Title

This example uses a search controller to filter the data inside a table view controller. The search bar is placed inside the navigation bar that the table view is embedded into.



将UITableViewController嵌入到UINavigationController中以获取UINavigationItem（包含导航栏）。然后设置我们的自定义视图控制器类继承自UITableViewController并采用



Embed a [UITableViewController](#) into a [UINavigationController](#) to get the [UINavigationItem](#) (which contains the navigation bar). Then set our custom ViewController class to inherit from [UITableViewController](#) and adopt the

```

class ViewController: UITableViewController, UISearchResultsUpdating {

    let entries = [(title: "最简单", image: "green_circle"),
                  (title: "中级", image: "blue_square"),
                  (title: "高级", image: "black_diamond"),
                  (title: "仅限专家", image: "double_black_diamond")]

    // 一个空元组，将用搜索结果更新。
    var searchResults : [(title: String, image: String)] = []

    let searchController = UISearchController(searchResultsController: nil)

    override func viewDidLoad() {
        super.viewDidLoad()

        searchController.searchResultsUpdater = self
        self.definesPresentationContext = true

        // 将搜索栏放置在导航项的标题视图中。
        self.navigationItem.titleView = searchController.searchBar

        // 不隐藏导航栏，因为搜索栏在其中。
        searchController.hidesNavigationBarDuringPresentation = false
    }

    func filterContent(for searchText: String) {
        // 根据标题值更新searchResults数组，匹配我们的entries中的内容。
        searchResults = entries.filter({ (title: String, image: String) -> Bool in
            let match = title.range(of: searchText, options: .caseInsensitive)
            // 如果范围包含匹配，则返回该元组。
            return match != nil
        })
    }

    // MARK: - UISearchResultsUpdating 方法

    func updateSearchResults(for searchController: UISearchController) {
        // 如果搜索栏中有文本，则用该字符串过滤数据
        filterContent(for: searchController.searchBar.text!)
        // 使用搜索结果数据重新加载表视图。
        tableView.reloadData()
    }

    // MARK: - UITableViewController 方法

    override func numberOfSections(in tableView: UITableView) -> Int { return 1 }

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        // 如果搜索栏处于激活状态，则使用 searchResults 数据。
        return searchController.isActive ? searchResults.count : entries.count
    }

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        // 如果搜索栏处于激活状态，则使用 searchResults 数据。
        let entry = searchController.isActive ?
            searchResults[indexPath.row] : entries[indexPath.row]

```

```

class ViewController: UITableViewController, UISearchResultsUpdating {

    let entries = [(title: "Easiest", image: "green_circle"),
                  (title: "Intermediate", image: "blue_square"),
                  (title: "Advanced", image: "black_diamond"),
                  (title: "Expert Only", image: "double_black_diamond")]

    // An empty tuple that will be updated with search results.
    var searchResults : [(title: String, image: String)] = []

    let searchController = UISearchController(searchResultsController: nil)

    override func viewDidLoad() {
        super.viewDidLoad()

        searchController.searchResultsUpdater = self
        self.definesPresentationContext = true

        // Place the search bar in the navigation item's title view.
        self.navigationItem.titleView = searchController.searchBar

        // Don't hide the navigation bar because the search bar is in it.
        searchController.hidesNavigationBarDuringPresentation = false
    }

    func filterContent(for searchText: String) {
        // Update the searchResults array with matches
        // in our entries based on the title value.
        searchResults = entries.filter({ (title: String, image: String) -> Bool in
            let match = title.range(of: searchText, options: .caseInsensitive)
            // Return the tuple if the range contains a match.
            return match != nil
        })
    }

    // MARK: - UISearchResultsUpdating method

    func updateSearchResults(for searchController: UISearchController) {
        // If the search bar contains text, filter our data with the string
        if let searchText = searchController.searchBar.text {
            filterContent(for: searchText)
            // Reload the table view with the search result data.
            tableView.reloadData()
        }
    }

    // MARK: - UITableViewController methods

    override func numberOfSections(in tableView: UITableView) -> Int { return 1 }

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        // If the search bar is active, use the searchResults data.
        return searchController.isActive ? searchResults.count : entries.count
    }

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        // If the search bar is active, use the searchResults data.
        let entry = searchController.isActive ?
            searchResults[indexPath.row] : entries[indexPath.row]

```

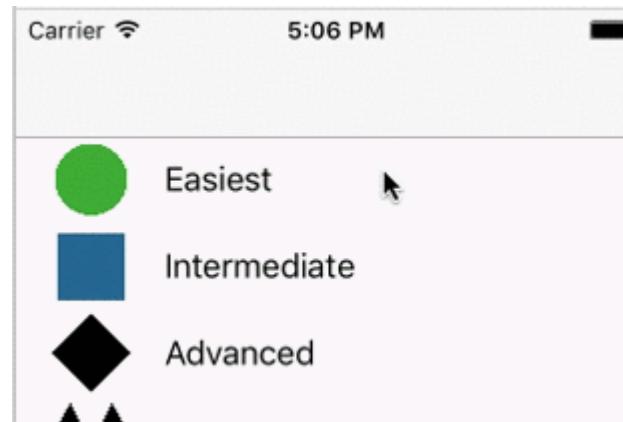
```

let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath)
cell.textLabel?.text = entry.title
cell.imageView?.image = UIImage(named: entry.image)
return cell
}
}

```

## 第39.2节：表视图头部的搜索栏

此示例使用搜索控制器来过滤表视图控制器中的单元格。搜索栏放置在表视图的头部视图中。表视图内容向下偏移与搜索栏高度相同，因此搜索栏初始时被隐藏。当向上滚动超过表视图顶部边缘时，搜索栏会显示出来。然后当搜索栏变为激活状态时，它会隐藏导航栏。



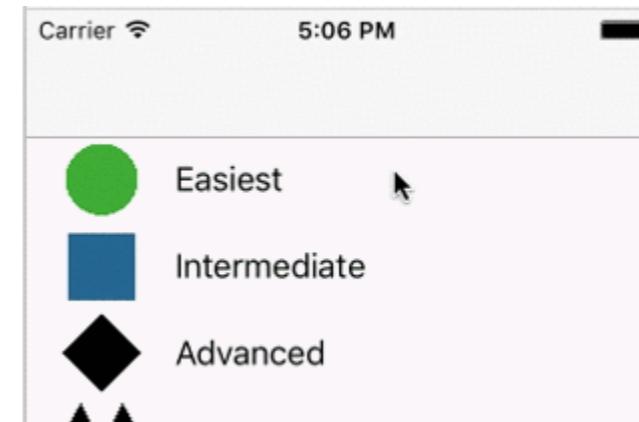
```

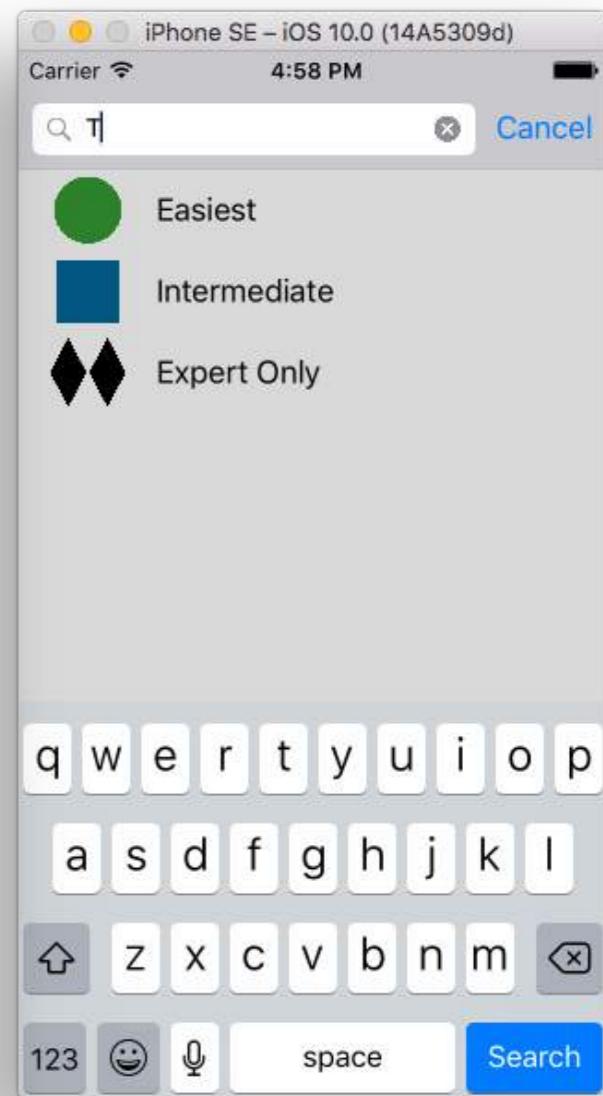
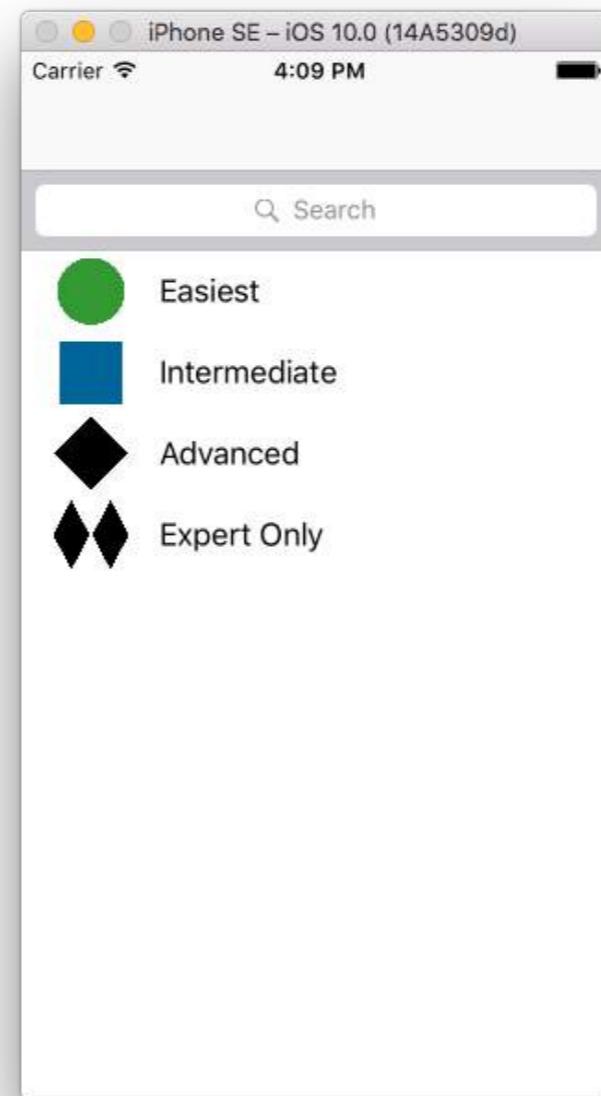
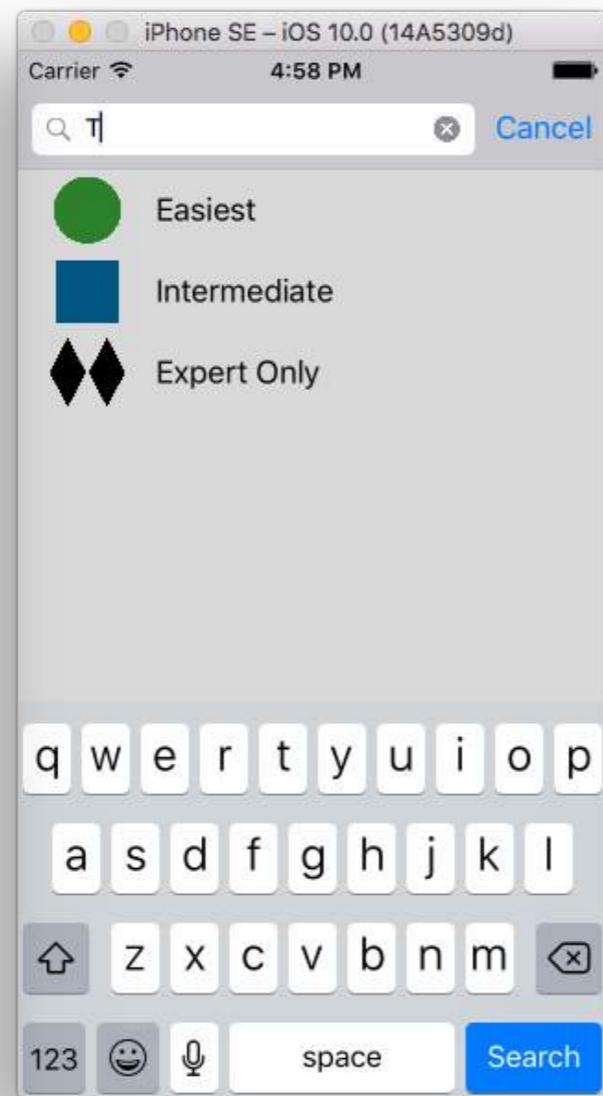
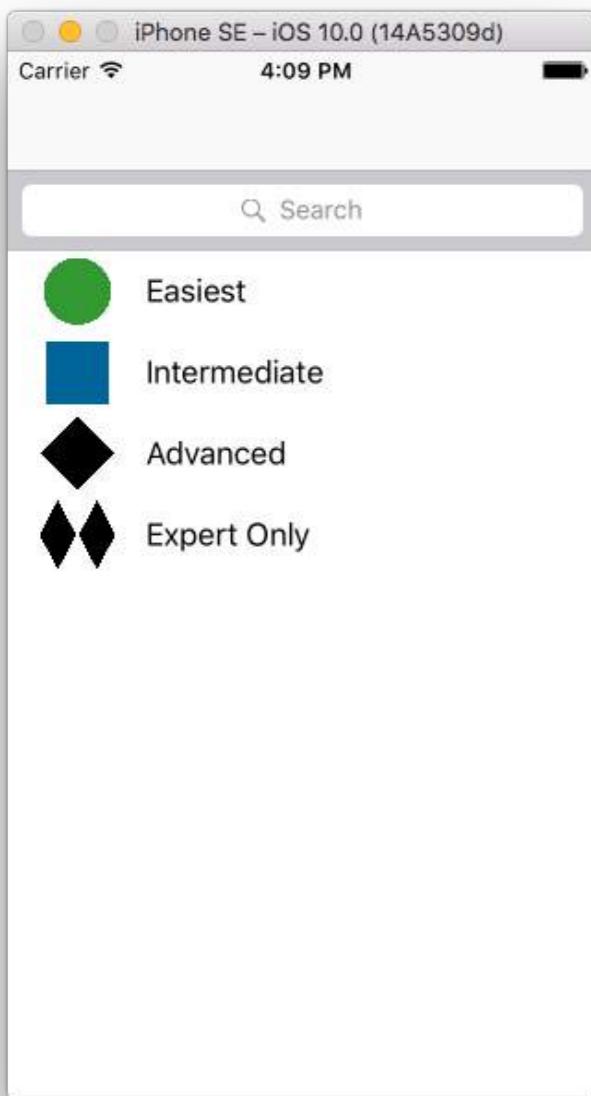
let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath)
cell.textLabel?.text = entry.title
cell.imageView?.image = UIImage(named: entry.image)
return cell
}
}

```

## Section 39.2: Search Bar in Table View Header

This example uses a search controller to filter the cells in a table view controller. The search bar is placed inside the header view of the table view. The table view content is offset with the same height as the search bar so that the search bar is hidden at first. Upon scrolling up past the top edge of the table view, the search bar is revealed. Then when the search bar becomes active, it hides the navigation bar.





将UITableViewController嵌入UINavigationController以获取UINavigationItem（其中包含导航栏）。然后设置自定义的ViewController类继承自UITableViewController并采用



Embed a UITableViewController into a UINavigationController to get the UINavigationItem (which contains the navigation bar). Then set our custom ViewController class to inherit from UITableViewController and adopt the

## UISearchResultsUpdating协议。

```
class ViewController: UITableViewController, UISearchResultsUpdating {

    let entries = [(title: "最简单", image: "green_circle"),
                  (title: "中级", image: "blue_square"),
                  (title: "高级", image: "black_diamond"),
                  (title: "仅限专家", image: "double_black_diamond")]

    // 一个空元组，将用搜索结果更新。
    var searchResults : [(title: String, image: String)] = []

    let searchController = UISearchController(searchResultsController: nil)

    override func viewDidLoad() {
        super.viewDidLoad()

        searchController.searchResultsUpdater = self
        self.definesPresentationContext = true

        // 将搜索栏放置在表视图的头部。
        self.tableView.tableHeaderView = searchController.searchBar

        // 设置内容偏移为搜索栏的高度
        // 以便视图首次显示时隐藏搜索栏。
        self.tableView.contentOffset = CGPoint(x: 0, y: searchController.searchBar.frame.height)
    }

    func filterContent(for searchText: String) {
        // 使用标题值更新searchResults数组中的匹配项
        // 在我们的条目中进行匹配
        searchResults = entries.filter({ (title: String, image: String) -> Bool in
            let match = title.range(of: searchText, options: .caseInsensitive)
            // 如果范围包含匹配，则返回该元组。
            return match != nil
        })
    }

    // MARK: - UISearchResultsUpdating 方法

    func updateSearchResults(for searchController: UISearchController) {
        // 如果搜索栏中有文本，则用该字符串过滤数据
        filterContent(for: searchController.searchBar.text!)
        tableView.reloadData()
    }

    // MARK: - UITableViewController 方法

    override func numberOfSections(in tableView: UITableView) -> Int { return 1 }

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        // 如果搜索栏处于激活状态，则使用 searchResults 数据。
        return searchController.isActive ? searchResults.count : entries.count
    }

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        // 如果搜索栏处于激活状态，则使用 searchResults 数据。
        let entry = searchController.isActive ?

```

## UISearchResultsUpdating protocol.

```
class ViewController: UITableViewController, UISearchResultsUpdating {

    let entries = [(title: "Easiest", image: "green_circle"),
                  (title: "Intermediate", image: "blue_square"),
                  (title: "Advanced", image: "black_diamond"),
                  (title: "Expert Only", image: "double_black_diamond")]

    // An empty tuple that will be updated with search results.
    var searchResults : [(title: String, image: String)] = []

    let searchController = UISearchController(searchResultsController: nil)

    override func viewDidLoad() {
        super.viewDidLoad()

        searchController.searchResultsUpdater = self
        self.definesPresentationContext = true

        // Place the search bar in the table view's header.
        self.tableView.tableHeaderView = searchController.searchBar

        // Set the content offset to the height of the search bar's height
        // to hide it when the view is first presented.
        self.tableView.contentOffset = CGPoint(x: 0, y: searchController.searchBar.frame.height)
    }

    func filterContent(for searchText: String) {
        // Update the searchResults array with matches
        // in our entries based on the title value.
        searchResults = entries.filter({ (title: String, image: String) -> Bool in
            let match = title.range(of: searchText, options: .caseInsensitive)
            // Return the tuple if the range contains a match.
            return match != nil
        })
    }

    // MARK: - UISearchResultsUpdating method

    func updateSearchResults(for searchController: UISearchController) {
        // If the search bar contains text, filter our data with the string
        if let searchText = searchController.searchBar.text {
            filterContent(for: searchText)
            // Reload the table view with the search result data.
            tableView.reloadData()
        }
    }

    // MARK: - UITableViewController methods

    override func numberOfSections(in tableView: UITableView) -> Int { return 1 }

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        // If the search bar is active, use the searchResults data.
        return searchController.isActive ? searchResults.count : entries.count
    }

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        // If the search bar is active, use the searchResults data.
        let entry = searchController.isActive ?

```

```
searchResults[indexPath.row] : entries[indexPath.row]
```

```
    let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath)
    cell.textLabel?.text = entry.title
    cell.imageView?.image = UIImage(named: entry.image)
    return cell
}
```

### 第39.3节：实现

首先，让你的类遵守UISearchResultsUpdating协议。

```
class MyTableViewController: UITableViewController, UISearchResultsUpdating {}
```

添加搜索控制器属性：

```
class MyTableViewController: UITableViewController, UISearchResultsUpdating {
    let searchController = UISearchController(searchResultsController: nil)
}
```

添加搜索栏：

```
override func viewDidLoad() {
    super.viewDidLoad()

    searchController.searchResultsUpdater = self
    searchController.hidesNavigationBarDuringPresentation = false
    searchControllerdimsBackgroundDuringPresentation = false
    searchController.searchBar.sizeToFit()
    self.tableView.tableHeaderView = searchController.searchBar
}
```

最后，实现来自

UISearchResultsUpdating协议的updateSearchResultsForSearchController方法：

```
func updateSearchResultsForSearchController(searchController: UISearchController) {
}
```

### 第39.4节：Objective-C中的UISearchController

代理：UISearchBarDelegate, UISearchControllerDelegate, UISearchBarDelegate

```
@property (strong, nonatomic) UISearchController *searchController;

- (void)searchBarConfiguration
{
    self.searchController = [[UISearchController alloc] initWithSearchResultsController:nil];
    self.searchController.searchBar.delegate = self;
    self.searchController.hidesNavigationBarDuringPresentation = NO;

    // 初始时隐藏导航栏。当用户向下拉列表时，搜索栏会被显示。
    [self.tableView setContentOffset:CGPointMake(0,
self.searchController.searchBar.frame.size.height)];

    self.searchController.searchBar.backgroundColor = [UIColor DarkBlue];
}
```

```
searchResults[indexPath.row] : entries[indexPath.row]
```

```
    let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath)
    cell.textLabel?.text = entry.title
    cell.imageView?.image = UIImage(named: entry.image)
    return cell
}
```

### Section 39.3: Implementation

First, make your class comply with the `UISearchResultsUpdating` protocol.

```
class MyTableViewController: UITableViewController, UISearchResultsUpdating {}
```

Add the search controller property:

```
class MyTableViewController: UITableViewController, UISearchResultsUpdating {
    let searchController = UISearchController(searchResultsController: nil)
}
```

Add the search bar:

```
override func viewDidLoad() {
    super.viewDidLoad()

    searchController.searchResultsUpdater = self
    searchController.hidesNavigationBarDuringPresentation = false
    searchControllerdimsBackgroundDuringPresentation = false
    searchController.searchBar.sizeToFit()
    self.tableView.tableHeaderView = searchController.searchBar
}
```

And finally, implement the `updateSearchResultsForSearchController` method that comes from the `UISearchResultsUpdating` protocol:

```
func updateSearchResultsForSearchController(searchController: UISearchController) {
}
```

### Section 39.4: UISerachController in Objective-C

Delegate: UISearchBarDelegate, UISearchControllerDelegate, UISearchBarDelegate

```
@property (strong, nonatomic) UISearchController *searchController;

- (void)searchBarConfiguration
{
    self.searchController = [[UISearchController alloc] initWithSearchResultsController:nil];
    self.searchController.searchBar.delegate = self;
    self.searchController.hidesNavigationBarDuringPresentation = NO;

    // Hides search bar initially. When the user pulls down on the list, the search bar is revealed.
    [self.tableView setContentOffset:CGPointMake(0,
self.searchController.searchBar.frame.size.height)];

    self.searchController.searchBar.backgroundColor = [UIColor DarkBlue];
}
```

```
self.searchController.searchBar.tintColor = [UIColor DarkBlue];

self.tableView.contentOffset = CGPointMake(0,
CGRectGetHeight(_searchController.searchBar.frame));
self.tableView.tableHeaderView = _searchController.searchBar;
_searchController.searchBar.delegate = self;
_searchController.searchBar.showsCancelButton = YES;
self.tapGestureRecognizer = [[UITapGestureRecognizer alloc] initWithTarget:self
action:@selector(resetSearchbarAndTableView)];
[self.view addGestureRecognizer:self.tapGestureRecognizer];

}

- (void)resetSearchbarAndTableView{
// 重新加载你的表视图并收起键盘。
}

- (void)searchBarCancelButtonClicked:(UISearchBar *)searchBar{
// 搜索已取消
...
    (UISearchBar *)searchBar{
// 根据需要使用NSPredicate或其他方式实现数据过滤。
// 然后重新加载你的数据控件，如表视图。
}
}
```

```
self.searchController.searchBar.tintColor = [UIColor DarkBlue];

self.tableView.contentOffset = CGPointMake(0,
CGRectGetHeight(_searchController.searchBar.frame));
self.tableView.tableHeaderView = _searchController.searchBar;
_searchController.searchBar.delegate = self;
_searchController.searchBar.showsCancelButton = YES;
self.tapGestureRecognizer = [[UITapGestureRecognizer alloc] initWithTarget:self
action:@selector(resetSearchbarAndTableView)];
[self.view addGestureRecognizer:self.tapGestureRecognizer];

}

- (void)resetSearchbarAndTableView{
// Reload your tableview and resign keyboard.
}

- (void)searchBarCancelButtonClicked:(UISearchBar *)searchBar{
// Search cancelled
}
- (void)searchBarSearchButtonClicked:(UISearchBar *)searchBar{
// Implement filtration of your data as per your need using NSPredicate or else.
// then reload your data control like Tableview.
}
```

# 第40章：UITabBarController

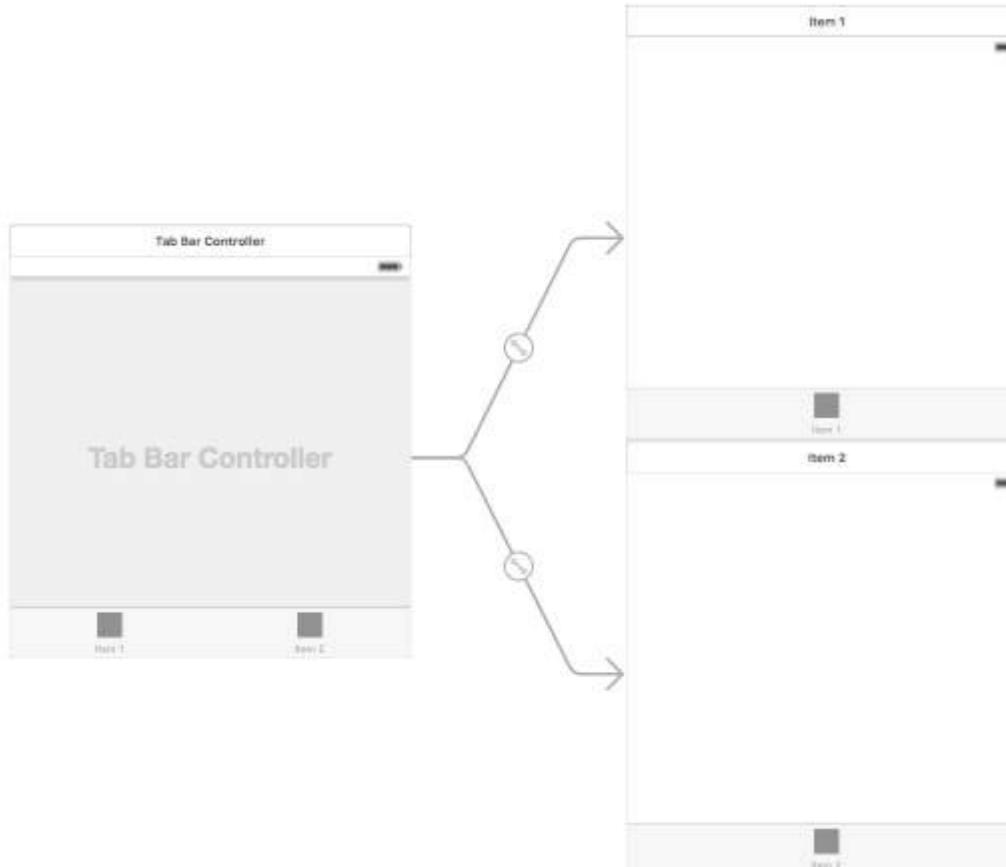
## 第40.1节：创建实例

“标签栏”通常出现在大多数iOS应用中，用于在每个标签中展示不同的视图。

要使用界面构建器创建标签栏控制器，请从对象库中将标签栏控制器拖入画布。



默认情况下，标签栏控制器带有两个视图。要添加更多视图，请按住控制键从标签栏控制器拖动到新视图，并在segue下拉菜单中选择“视图控制器”。



## 第40.2节：带标签栏的导航控制器

导航控制器可以直接在故事板中嵌入到每个标签中。它可以像截图中添加的那样。

要向从标签栏控制器连接的视图控制器添加导航控制器，流程如下

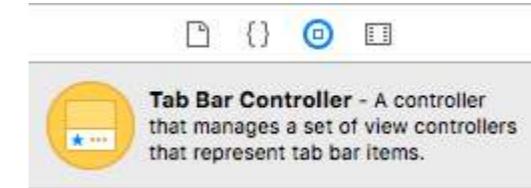
- 选择需要添加导航控制器的视图控制器。这里以搜索视图控制器作为选择示例。
- 在Xcode的编辑器菜单中，选择嵌入到 -> 导航控制器选项

# Chapter 40: UITabBarController

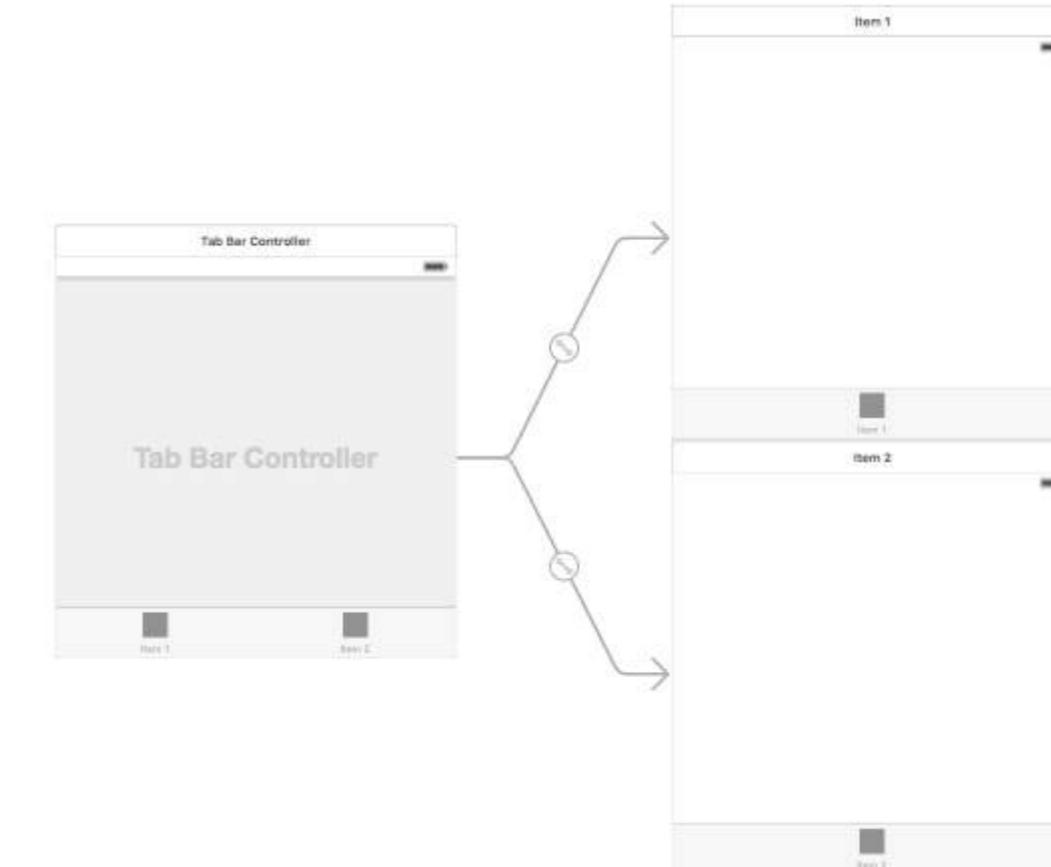
## Section 40.1: Create an instance

A 'tab bar' is commonly found in most iOS apps and is used to present distinct views in each tab.

To create a tab bar controller using the interface builder, drag a tab bar Controller from the Object Library into the canvas.



By default a tab bar controller comes with two views. To add additional views, control drag from the tab bar controller to the new view and select 'view controllers' in the segue-drop down.

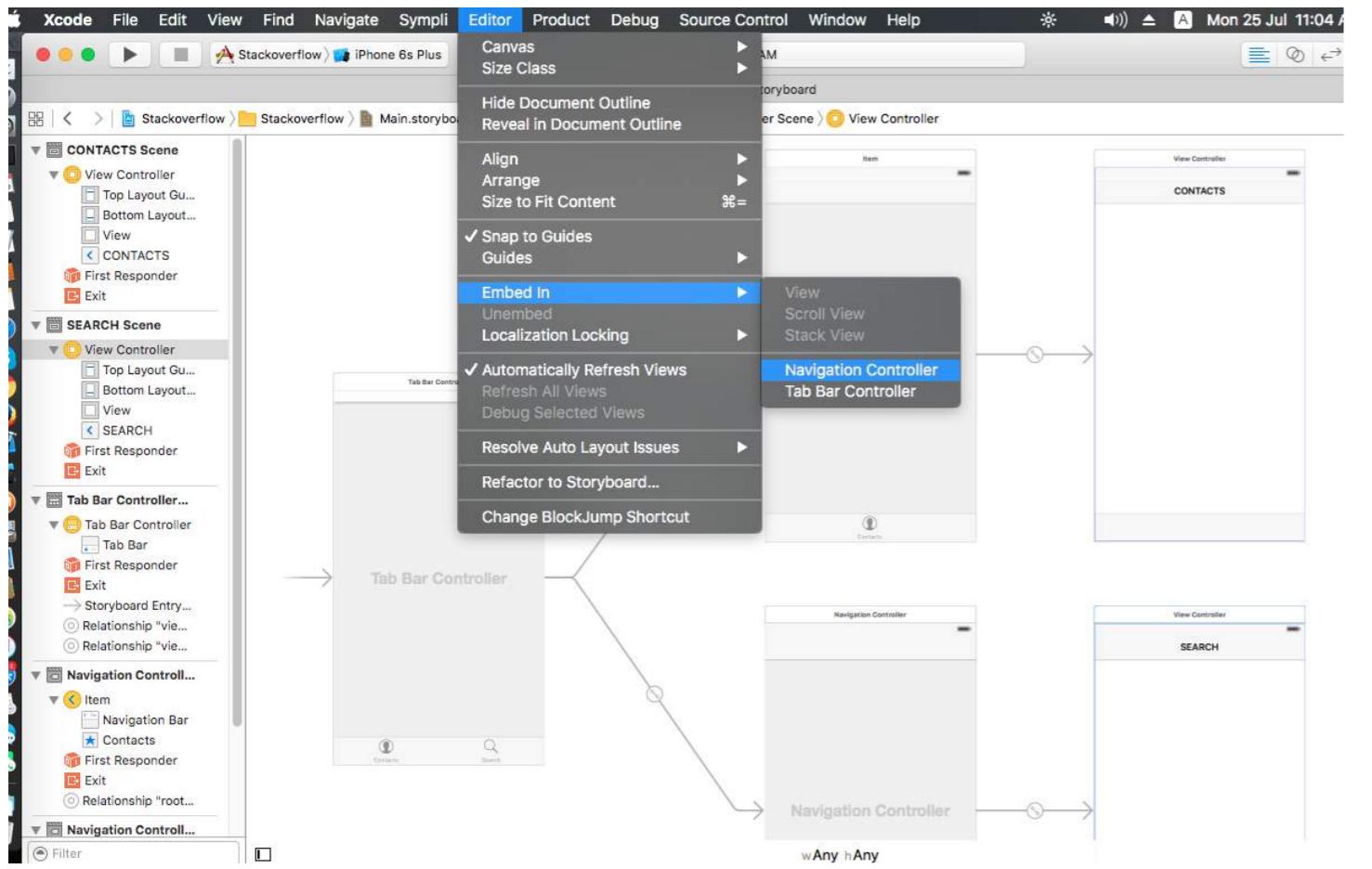


## Section 40.2: Navigation Controller with TabBar

Navigation controller can be embed in each tabs using storyboard it self. It can be like in the screenshot added.

To add a Navigation Controller to a View Controller connecting from Tab Bar Controller, here are the flow

- Select the view controller for which we need to add navigation controller. Here let it be the Search View Controller as the selection display.
- From the **Editor** menu of the Xcode, select **Embed In -> Navigation Controller** option



## 第40.3节：标签栏颜色自定义

```
[[UITabBar appearance] setTintColor:[UIColor whiteColor]];
[[UITabBar appearance] setBarTintColor:[UIColor tabBarBackgroundColor]];
[[UITabBar appearance] setBackgroundColor:[UIColor tabBarInactiveColor]];
[[UINavigationBar appearance] setBarTintColor:[UIColor appBlueColor]];
[[UINavigationBar appearance] setTintColor:[UIColor whiteColor]];
[[UINavigationBar appearance] setBarStyle: UIBarStyleBlack];
```

## 第40.4节：更改标签栏项目标题和图标

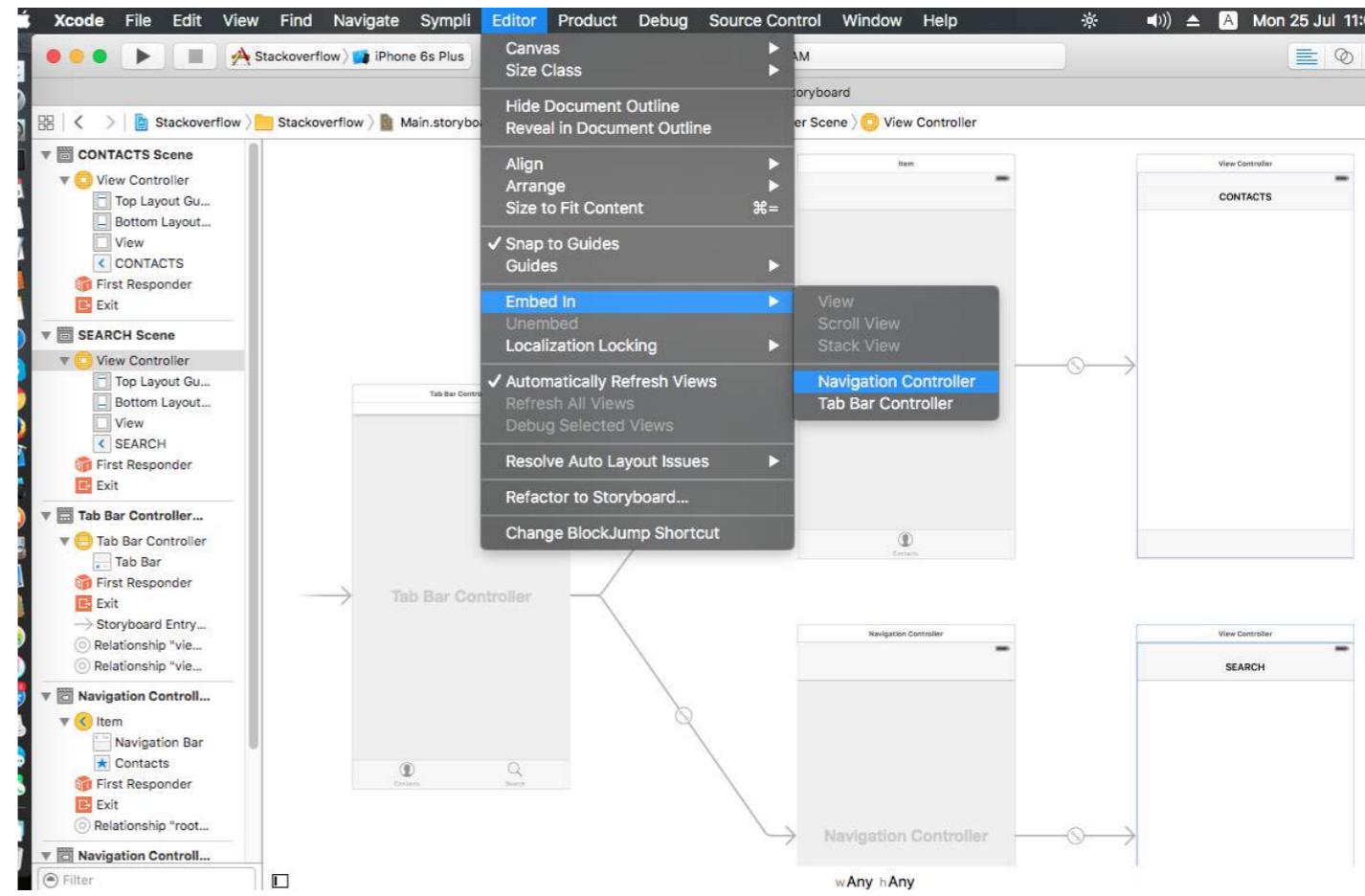
### 使用故事板：

从对应的视图控制器中选择标签栏项目，然后进入属性检查器

如果想使用内置图标和标题，将“系统项目”设置为相应的值。

对于自定义图标，将所需的图片添加到资源文件夹中，并将之前的“系统项目”设置为“自定义”。

现在，从“选中图像”下拉菜单中设置选中标签时显示的图标，从“图像”下拉菜单中设置默认标签图标。在“标题”字段中添加相应的标题。



## Section 40.3: Tab Bar color customizing

```
[[UITabBar appearance] setTintColor:[UIColor whiteColor]];
[[UITabBar appearance] setBarTintColor:[UIColor tabBarBackgroundColor]];
[[UITabBar appearance] setBackgroundColor:[UIColor tabBarInactiveColor]];
[[UINavigationBar appearance] setBarTintColor:[UIColor appBlueColor]];
[[UINavigationBar appearance] setTintColor:[UIColor whiteColor]];
[[UINavigationBar appearance] setBarStyle: UIBarStyleBlack];
```

## Section 40.4: Changing Tab Bar Item Title and Icon

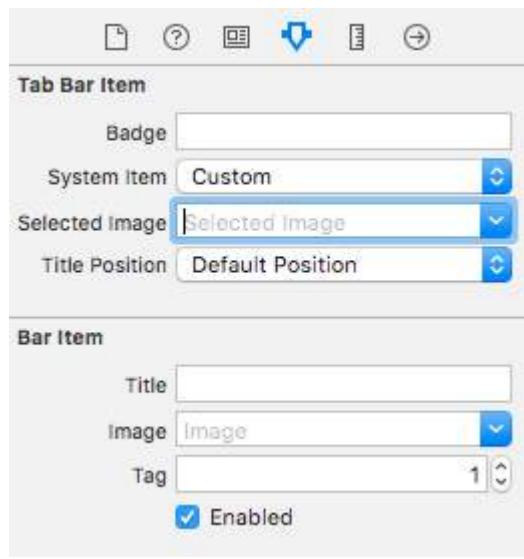
### Using the Story Board:

Select the tab bar item from the corresponding view controller and go to the attributes inspector

If you want a built-in icon and title, set the 'System Item' to the corresponding value.

For a custom icon, add the required images to the assets folder and set the 'System Item' from earlier to 'custom'.

Now, set the icon to be shown when the tab is selected from the 'selected image' drop down and the default tab icon from the 'image' drop down. Add the corresponding title in the 'title' field.



通过代码实现：

在视图控制器的viewDidLoad()方法中，添加以下代码：

**Objective-C :**

```
self.title = @"item";

self.tabBarItem.image = [UIImage imageNamed:@"item"];
self.tabBarItem.selectedImage = [UIImage imageNamed:@"item_selected"];
```

**Swift :**

```
self.title = "item"
self.tabBarItem.image = UIImage(named: "item")
self.tabBarItem.selectedImage = UIImage(named: "item_selected")
```

## 第40.5节：不使用Storyboard，程序化创建标签栏控制器

```
class AppDelegate: UIResponder, UIApplicationDelegate {

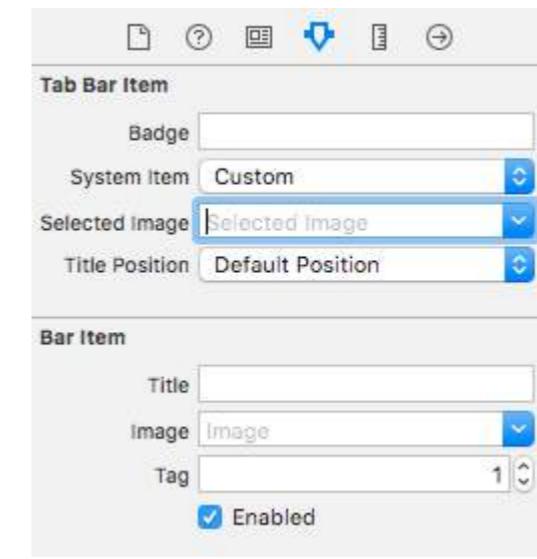
    var window: UIWindow?

    var firstTabNavigationController: UINavigationController!
    var secondTabNavigationController: UINavigationController!
    var thirdTabNavigationController: UINavigationController!
    var fourthTabNavigationController: UINavigationController!
    var fifthTabNavigationController: UINavigationController!

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application launch.
        Fabric.with([Crashlytics.self])

        window = UIWindow(frame: UIScreen.main.bounds)

        window?.backgroundColor = UIColor.black
    }
}
```



**Programmatically:**

In the viewDidLoad() method of the view controller, add the following code:

**Objective-C :**

```
self.title = @"item"

self.tabBarItem.image = [UIImage imageNamed:@"item"];
self.tabBarItem.selectedImage = [UIImage imageNamed:@"item_selected"];
```

**Swift:**

```
self.title = "item"
self.tabBarItem.image = UIImage(named: "item")
self.tabBarItem.selectedImage = UIImage(named: "item_selected")
```

## Section 40.5: Create Tab Bar controller programmatically without Storyboard

```
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    var firstTabNavigationController: UINavigationController!
    var secondTabNavigationController: UINavigationController!
    var thirdTabNavigationController: UINavigationController!
    var fourthTabNavigationController: UINavigationController!
    var fifthTabNavigationController: UINavigationController!

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application launch.
        Fabric.with([Crashlytics.self])

        window = UIWindow(frame: UIScreen.main.bounds)

        window?.backgroundColor = UIColor.black
    }
}
```

```

let tabBarController = UITabBarController()

firstTabNavigationController = UINavigationController.init(rootViewController:
FirstViewController())
secondTabNavigationController = UINavigationController.init(rootViewController:
SecondViewController())
thirdTabNavigationController = UINavigationController.init(rootViewController:
ThirdViewController())
fourthTabNavigationController = UINavigationController.init(rootViewController:
FourthViewController())
fifthTabNavigationController = UINavigationController.init(rootViewController:
FifthViewController())

tabBarController.viewControllers = [firstTabNavigationController,
secondTabNavigationController, thirdTabNavigationController, fourthTabNavigationController,
fifthTabNavigationController]

2)
let item1 = UITabBarItem(title: "首页", image: UIImage(named: "ico-home"), tag: 0)
let item2 = UITabBarItem(title: "竞赛", image: UIImage(named: "ico-contest"), tag: 1)
let item3 = UITabBarItem(title: "发布图片", image: UIImage(named: "ico-photo"), tag:
2)
let item4 = UITabBarItem(title: "奖品", image: UIImage(named: "ico-prizes"), tag: 3)
let item5 = UITabBarItem(title: "个人资料", image: UIImage(named: "ico-profile"), tag: 4)

firstTabNavigationController.tabBarItem = item1
secondTabNavigationController.tabBarItem = item2
thirdTabNavigationController.tabBarItem = item3
fourthTabNavigationController.tabBarItem = item4
fifthTabNavigationController.tabBarItem = item5

UITabBar.appearance().tintColor = UIColor(red: 0/255.0, green: 146/255.0, blue: 248/255.0,
alpha: 1.0)

self.window?.rootViewController = tabBarController

window?.makeKeyAndVisible()

return true
}

```

## 第40.6节：带自定义颜色选择的UITabBarController

在Swift3中构建UITabBarController，根据选择更改选中标签颜色，同时更改图像颜色和标题。

```

import UIKit

class TabbarController: UITabBarController {

override func viewDidLoad() {
super.viewDidLoad()

self.navigationController?.isNavigationBarHidden = true

UITabBar.appearance().tintColor = UIColor.purple

// 设置红色为选中背景色
let numberofItems = CGFloat(tabBar.items!.count)
let tabBarItemSize = CGSize(width: tabBar.frame.width / numberofItems, height:
tabBar.frame.height)

```

```

let tabBarController = UITabBarController()

firstTabNavigationController = UINavigationController.init(rootViewController:
FirstViewController())
secondTabNavigationController = UINavigationController.init(rootViewController:
SecondViewController())
thirdTabNavigationController = UINavigationController.init(rootViewController:
ThirdViewController())
fourthTabNavigationController = UINavigationController.init(rootViewController:
FourthViewController())
fifthTabNavigationController = UINavigationController.init(rootViewController:
FifthViewController())

tabBarController.viewControllers = [firstTabNavigationController,
secondTabNavigationController, thirdTabNavigationController, fourthTabNavigationController,
fifthTabNavigationController]

2)
let item1 = UITabBarItem(title: "Home", image: UIImage(named: "ico-home"), tag: 0)
let item2 = UITabBarItem(title: "Contest", image: UIImage(named: "ico-contest"), tag: 1)
let item3 = UITabBarItem(title: "Post a Picture", image: UIImage(named: "ico-photo"), tag:
2)
let item4 = UITabBarItem(title: "Prizes", image: UIImage(named: "ico-prizes"), tag: 3)
let item5 = UITabBarItem(title: "Profile", image: UIImage(named: "ico-profile"), tag: 4)

firstTabNavigationController.tabBarItem = item1
secondTabNavigationController.tabBarItem = item2
thirdTabNavigationController.tabBarItem = item3
fourthTabNavigationController.tabBarItem = item4
fifthTabNavigationController.tabBarItem = item5

UITabBar.appearance().tintColor = UIColor(red: 0/255.0, green: 146/255.0, blue: 248/255.0,
alpha: 1.0)

self.window?.rootViewController = tabBarController

window?.makeKeyAndVisible()

return true
}

```

## Section 40.6: UITabBarController with custom color selection

UITabBarController building in Swift 3 Change image color and title according to selection with changing selected tab color.

```

import UIKit

class TabbarController: UITabBarController {

override func viewDidLoad() {
super.viewDidLoad()

self.navigationController?.isNavigationBarHidden = true

UITabBar.appearance().tintColor = UIColor.purple

// set red as selected background color
let numberofItems = CGFloat(tabBar.items!.count)
let tabBarItemSize = CGSize(width: tabBar.frame.width / numberofItems, height:
tabBar.frame.height)

```

```

tabBar.selectionIndicatorImage =
UIImage.imageWithColor(UIColor.lightText.withAlphaComponent(0.5), size:
tabBarItemSize).resizableImage(withCapInsets: UIEdgeInsets.zero)

    // 移除默认边框
tabBar.frame.size.width = self.view.frame.width + 4
    tabBar.frame.origin.x = -2

}

override func viewDidAppear(_ animated: Bool) {
    // 用于图片
    let firstViewController:UIViewController = NotificationVC()
    // 以下语句是你需要的
    let customTabBarItem:UITabBarItem = UITabBarItem(title: nil, image: UIImage(named:
"notification@2x")?.withRenderingMode(UIImageRenderingMode.alwaysOriginal), selectedImage:
UIImage(named: "notification_sel@2x"))
    firstViewController.tabBarItem = customTabBarItem

    for item in self.tabBar.items! {
        let unselectedItem = [NSForegroundColorAttributeName: UIColor.white]
        let selectedItem = [NSForegroundColorAttributeName: UIColor.purple]

item.setTitleTextAttributes(unselectedItem, for: .normal)
        item.setTitleTextAttributes(selectedItem, for: .selected)
    }
}

extension UIImage {
    class func imageWithColor(_ color: UIColor, size: CGSize) -> UIImage {
        let rect: CGRect = CGRect(origin: CGPoint(x: 0,y :0), size: CGSize(width: size.width,
height: size.height))
        UIGraphicsBeginImageContextWithOptions(size, false, 0)
        color.setFill()
        UIRectFill(rect)
        let image: UIImage = UIGraphicsGetImageFromCurrentImageContext()!
        UIGraphicsEndImageContext()
        return image
    }
}

```

为标签栏选择图片并设置标签标题

```

tabBar.selectionIndicatorImage =
UIImage.imageWithColor(UIColor.lightText.withAlphaComponent(0.5), size:
tabBarItemSize).resizableImage(withCapInsets: UIEdgeInsets.zero)

    // remove default border
tabBar.frame.size.width = self.view.frame.width + 4
    tabBar.frame.origin.x = -2

}

override func viewDidAppear(_ animated: Bool) {
    // For Images
    let firstViewController:UIViewController = NotificationVC()
    // The following statement is what you need
    let customTabBarItem:UITabBarItem = UITabBarItem(title: nil, image: UIImage(named:
"notification@2x")?.withRenderingMode(UIImageRenderingMode.alwaysOriginal), selectedImage:
UIImage(named: "notification_sel@2x"))
    firstViewController.tabBarItem = customTabBarItem

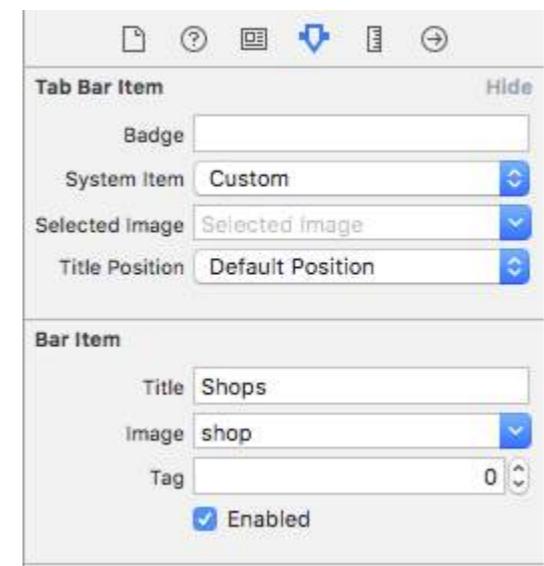
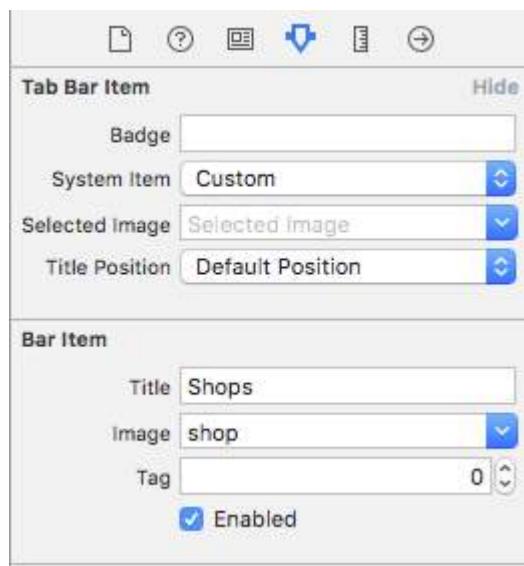
    for item in self.tabBar.items! {
        let unselectedItem = [NSForegroundColorAttributeName: UIColor.white]
        let selectedItem = [NSForegroundColorAttributeName: UIColor.purple]

item.setTitleTextAttributes(unselectedItem, for: .normal)
        item.setTitleTextAttributes(selectedItem, for: .selected)
    }
}

extension UIImage {
    class func imageWithColor(_ color: UIColor, size: CGSize) -> UIImage {
        let rect: CGRect = CGRect(origin: CGPoint(x: 0,y :0), size: CGSize(width: size.width,
height: size.height))
        UIGraphicsBeginImageContextWithOptions(size, false, 0)
        color.setFill()
        UIRectFill(rect)
        let image: UIImage = UIGraphicsGetImageFromCurrentImageContext()!
        UIGraphicsEndImageContext()
        return image
    }
}

```

Choosing image for tab bar and set the tab title here



选择另一个标签



Selection another tab



# 第41章：UIWebView

## 第41.1节：创建UIWebView实例

### Swift

```
let webview = UIWebView(frame: CGRect(x: 0, y: 0, width: 320, height: 480))
```

### Objective-C

```
UIWebView *webview = [[UIWebView alloc] initWithFrame:CGRectMake(0, 0, 320, 480)];  
  
//定义UIWebView框架的另一种方式  
UIWebView *webview = [[UIWebView alloc] init];  
CGRect webviewFrame = webview.frame;  
webviewFrame.size.width = 320;  
webviewFrame.size.height = 480;  
webviewFrame.origin.x = 0;  
webviewFrame.origin.y = 0;  
webview.frame = webviewFrame;
```

## 第41.2节：确定内容大小

在许多情况下，例如在表视图单元格中使用网页视图时，确定渲染后的HTML页面的内容大小非常重要。加载页面后，可以在UIWebViewDelegate代理方法中计算该大小：

```
- (void) webViewDidFinishLoad:(UIWebView *) aWebView {  
    CGRect frame = aWebView.frame;  
    frame.size.height = 1;  
    aWebView.frame = frame;  
    CGSize fittingSize = [aWebView sizeThatFits:CGSizeZero];  
    frame.size = fittingSize;  
    aWebView.frame = frame;  
  
    NSLog(@"size: %f, %f", fittingSize.width, fittingSize.height);  
}
```

该代码采用了一个额外技巧，即在测量适合大小之前，先将网页视图的高度短暂设置为1。否则，它只会报告当前的框架大小。测量后，我们立即将高度设置为实际内容高度。

### 来源

## 第41.3节：加载HTML字符串

Web 视图对于加载本地生成的 HTML 字符串非常有用。

```
NSString *html = @"<!DOCTYPE html><html><body>你好，世界</body></html>";  
[webView loadHTMLString:html baseURL:nil];
```

### Swift

```
let htmlString = "<h1>我的第一个标题</h1><p>我的第一段。</p>"  
webView.loadHTMLString(htmlString, baseURL: nil)
```

# Chapter 41: UIWebView

## Section 41.1: Create a UIWebView instance

### Swift

```
let webview = UIWebView(frame: CGRect(x: 0, y: 0, width: 320, height: 480))
```

### Objective-C

```
UIWebView *webview = [[UIWebView alloc] initWithFrame:CGRectMake(0, 0, 320, 480)];  
  
//Alternative way of defining frame for UIWebView  
UIWebView *webview = [[UIWebView alloc] init];  
CGRect webviewFrame = webview.frame;  
webviewFrame.size.width = 320;  
webviewFrame.size.height = 480;  
webviewFrame.origin.x = 0;  
webviewFrame.origin.y = 0;  
webview.frame = webviewFrame;
```

## Section 41.2: Determining content size

In many cases, for instance when using web views in table view cells, it's important to determine the content size of the rendered HTML page. After loading the page, this can be calculated in the `UIWebViewDelegate` delegate method:

```
- (void) webViewDidFinishLoad:(UIWebView *) aWebView {  
    CGRect frame = aWebView.frame;  
    frame.size.height = 1;  
    aWebView.frame = frame;  
    CGSize fittingSize = [aWebView sizeThatFits:CGSizeZero];  
    frame.size = fittingSize;  
    aWebView.frame = frame;  
  
    NSLog(@"size: %f, %f", fittingSize.width, fittingSize.height);  
}
```

The code employs an additional trick of shortly setting the height of the web view to 1 prior to measuring the fitting size. Otherwise it would simply report the current frame size. After measuring we immediately set the height to the actual content height.

### Source

## Section 41.3: Load HTML string

Web views are useful to load locally generated HTML strings.

```
NSString *html = @"<!DOCTYPE html><html><body>Hello World</body></html>";  
[webView loadHTMLString:html baseURL:nil];
```

### Swift

```
let htmlString = "<h1>My First Heading</h1><p>My first paragraph.</p>"  
webView.loadHTMLString(htmlString, baseURL: nil)
```

可以指定本地基础 URL。这对于引用应用程序包中的图片、样式表或脚本非常有用：

```
NSString *html = @"<!DOCTYPE html><html><head><link href='style.css' rel='stylesheet' type='text/css'></head><body>你好，世界</body></html>";  
[self loadHTMLString:html baseURL:[NSURL fileURLWithPath:[[NSBundle mainBundle] resourcePath]]];
```

在这种情况下，style.css 是从应用程序的资源目录本地加载的。当然，也可以指定远程 URL。

## 第41.4节：发起 URL 请求

从 url 在 webview 中加载内容

### Swift

```
webview.loadRequest(NSURLRequest(URL: NSURL(string: "http://www.google.com")!))
```

### Objective-C

```
[webview loadRequest:[NSURLRequest requestWithURL:[NSURL URLWithString:@"http://www.google.com"]]];
```

## 第41.5节：加载JavaScript

我们可以使用方法 `stringByEvaluatingJavaScriptFromString()` 在 `UIWebView` 上运行自定义 JavaScript。该方法返回传入脚本参数中 JavaScript 脚本运行的结果，如果脚本执行失败则返回 nil。

### Swift

从字符串加载脚本

```
webview.stringByEvaluatingJavaScriptFromString("alert('这是JavaScript！')");
```

从本地文件加载脚本

```
//假设项目中有一个名为"JavaScript.js"的JavaScript文件。  
let filePath = NSBundle.mainBundle().pathForResource("JavaScript", ofType: "js")  
do {  
    let jsContent = try String.init(contentsOfFile: filePath!, encoding:  
NSUTF8StringEncoding)  
    webview.stringByEvaluatingJavaScriptFromString(jsContent)  
}  
catch let error as NSError{  
    print(error.debugDescription)  
}
```

### Objective-C

从字符串加载脚本

```
[webview stringByEvaluatingJavaScriptFromString:@"alert('这是JavaScript！')";];
```

从本地文件加载脚本

```
//假设项目中有一个名为"JavaScript.js"的JavaScript文件。  
NSString *filePath = [[NSBundle mainBundle] pathForResource:@"JavaScript" ofType:@"js"];
```

A local base URL may be specified. This is useful to reference images, stylesheets or scripts from the app bundle:

```
NSString *html = @"<!DOCTYPE html><html><head><link href='style.css' rel='stylesheet' type='text/css'></head><body>Hello World</body></html>";  
[self loadHTMLString:html baseURL:[NSURL fileURLWithPath:[[NSBundle mainBundle] resourcePath]]];
```

In this case, `style.css` is loaded locally from the app's resource directory. Of course it's also possible to specify a remote URL.

## Section 41.4: Making a URL request

Load content in webview from the url

### Swift

```
webview.loadRequest(NSURLRequest(URL: NSURL(string: "http://www.google.com")!))
```

### Objective-C

```
[webview loadRequest:[NSURLRequest requestWithURL:[NSURL URLWithString:@"http://www.google.com"]]];
```

## Section 41.5: Load JavaScript

We can run custom JavaScript on a `UIWebView` using the method `stringByEvaluatingJavaScriptFromString()`. This method returns the result of running the JavaScript script passed in the `script` parameter, or nil if the script fails.

### Swift

Load script from String

```
webview.stringByEvaluatingJavaScriptFromString("alert('This is JavaScript!')");
```

Load script from Local file

```
//Suppose you have javascript file named "JavaScript.js" in project.  
let filePath = NSBundle.mainBundle().pathForResource("JavaScript", ofType: "js")  
do {  
    let jsContent = try String.init(contentsOfFile: filePath!, encoding:  
NSUTF8StringEncoding)  
    webview.stringByEvaluatingJavaScriptFromString(jsContent)  
}  
catch let error as NSError{  
    print(error.debugDescription)  
}
```

### Objective-C

Load script from String

```
[webview stringByEvaluatingJavaScriptFromString:@"alert('This is JavaScript!')";];
```

Load script from Local file

```
//Suppose you have javascript file named "JavaScript.js" in project.  
NSString *filePath = [[NSBundle mainBundle] pathForResource:@"JavaScript" ofType:@"js"];
```

```
NSString *jsContent = [NSString stringWithContentsOfFile:filePath encoding:NSUTF8StringEncoding error:&nil];
[webView stringByEvaluatingJavaScriptFromString:jsContent];
```

注意 `stringByEvaluatingJavaScriptFromString:` 方法会同步等待JavaScript执行完成。如果加载的网页内容中包含未经审核的JavaScript代码，调用此方法可能会导致应用程序卡死。最佳做法是采用WKWebView类并使用其 `evaluateJavaScript:completionHandler:` 方法代替。但WKWebView仅在iOS 8.0及更高版本可用。

## 第41.6节：停止加载网页内容

方法`stopLoading()`用于停止webView当前的加载过程。

### Swift

```
webView.stopLoading()
```

### Objective-C

```
[webView stopLoading];
```

## 第41.7节：重新加载当前网页内容

### Swift

```
webView.reload()
```

### Objective-C

```
[webView reload];
```

## 第41.8节：加载文档文件，如.pdf、.txt、.doc等

除了网页，我们还可以将文档文件加载到iOS WebView中，如.pdf、.txt、.doc等。`loadData`方法用于将`NSData`加载到webView中。

### Swift

```
//假设项目中有一个名为"home.txt"的文本文件。
let localFilePath = NSBundle.mainBundle().pathForResource("home", ofType:"txt");
let data = NSFileManager.defaultManager().contentsAtPath(localFilePath!);
webView.loadData(data!, MIMEType: "application/txt", textEncodingName:"UTF-8" , baseURL: NSURL())
```

### Objective-C

```
//假设项目中有一个名为"home.txt"的文本文件。
NSString *localFilePath = [[NSBundle mainBundle] pathForResource:@"home" ofType:@"txt"];
NSData *data = [[NSFileManager defaultManager] contentsAtPath:localFilePath];
[webView loadData:data MIMEType:@"application/txt" textEncodingName:@"UTF-8" baseURL:[NSURL new]];
```

## 第41.9节：在webView中加载本地HTML文件

首先，将HTML文件添加到您的项目中（如果系统提示选择添加文件的选项，请选择如果需要则复制项目）

```
NSString *jsContent = [NSString stringWithContentsOfFile:filePath encoding:NSUTF8StringEncoding error:&nil];
[webView stringByEvaluatingJavaScriptFromString:jsContent];
```

**Note** The `stringByEvaluatingJavaScriptFromString:` method waits synchronously for JavaScript evaluation to complete. If you load web content whose JavaScript code you have not vetted, invoking this method could hang your app. Best practice is to adopt the WKWebView class and use its `evaluateJavaScript:completionHandler:` method instead. But WKWebView is available from iOS 8.0 and later.

## Section 41.6: Stop Loading Web Content

Method `stopLoading()` stops the current loading process of the webView.

### Swift

```
webView.stopLoading()
```

### Objective-C

```
[webView stopLoading];
```

## Section 41.7: Reload Current Web Content

### Swift

```
webView.reload()
```

### Objective-C

```
[webView reload];
```

## Section 41.8: Load Document files like .pdf, .txt, .doc etc

Instead of web pages, we can also load the document files into iOS WebView like .pdf, .txt, .doc etc.. `loadData` method is used to load `NSData` into webView.

### Swift

```
//Assuming there is a text file in the project named "home.txt".
let localFilePath = NSBundle.mainBundle().pathForResource("home", ofType:"txt");
let data = NSFileManager.defaultManager().contentsAtPath(localFilePath!);
webView.loadData(data!, MIMEType: "application/txt", textEncodingName:"UTF-8" , baseURL: NSURL())
```

### Objective-C

```
//Assuming there is a text file in the project named "home.txt".
NSString *localFilePath = [[NSBundle mainBundle] pathForResource:@"home" ofType:@"txt"];
NSData *data = [[NSFileManager defaultManager] contentsAtPath:localFilePath];
[webView loadData:data MIMEType:@"application/txt" textEncodingName:@"UTF-8" baseURL:[NSURL new]];
```

## Section 41.9: Load local HTML file in webView

First, add the HTML File to your Project (If you are asked to choose options for adding the file, select *Copy items if necessary*)

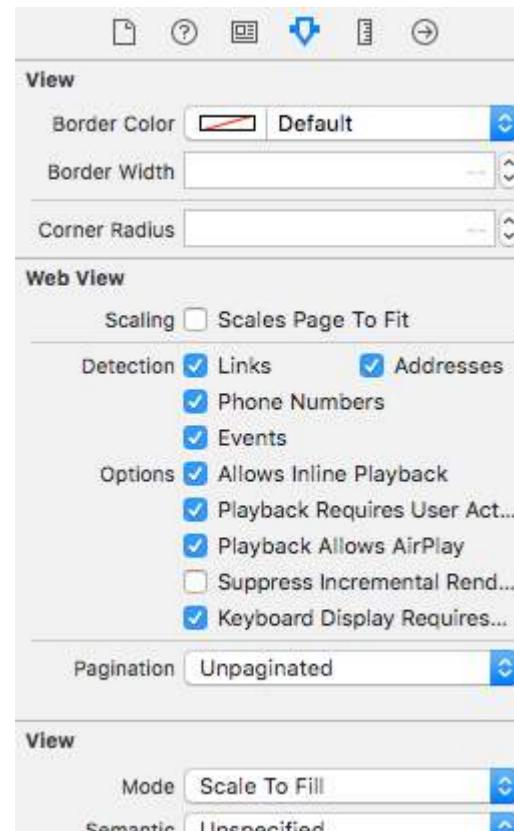
需要)

以下代码行将HTML文件的内容加载到webView中

```
webView.loadRequest(NSURLRequest(URL: NSURL(fileURLWithPath:  
NSBundle mainBundle().pathForResource("YOUR HTML FILE", ofType: "html")!))
```

- 如果您的HTML文件名为index.html, 请将YOUR HTML FILE替换为index
- 您可以在viewDidLoad()、viewDidAppear()或任何其他函数中使用此代码

## 第41.10节：使UIWebView内的链接可点击



在vc.h中

```
@interface vc : UIViewController<UIWebViewDelegate>
```

在vc.m中

```
- (BOOL)webView:(UIWebView *)webView shouldStartLoadWithRequest:(NSURLRequest *)request  
navigationType:(UIWebViewNavigationType)navigationType{  
  
    if (navigationType == UIWebViewNavigationTypeLinkClicked){  
        //如果你想在浏览器中打开它, 保留此代码; 如果想在同一网页视图中打开, 去掉 return NO;  
        NSURL *url = request.URL;  
        if ([[UIApplication sharedApplication] canOpenURL:url]) {  
            [[UIApplication sharedApplication] openURL:url];  
        }  
        return NO;  
    }  
  
    return YES;  
}
```

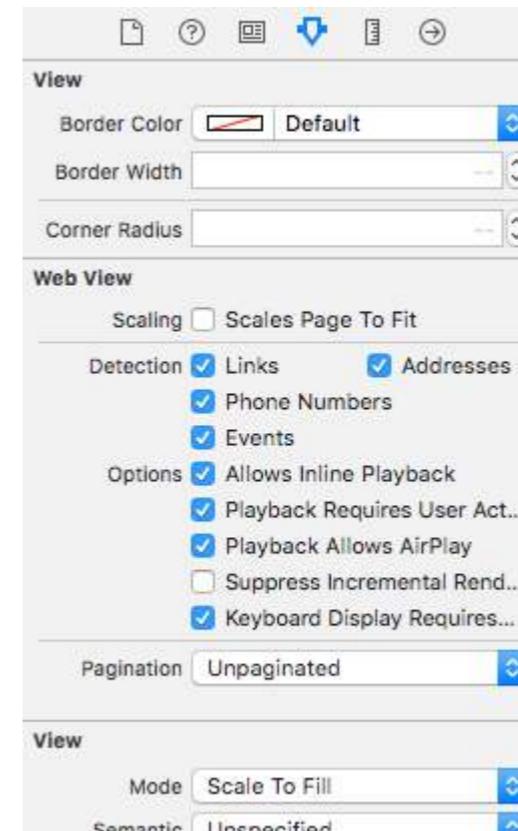
needed)

The following line of code loads the content of the HTML file into the webView

```
webView.loadRequest(NSURLRequest(URL: NSURL(fileURLWithPath:  
NSBundle mainBundle().pathForResource("YOUR HTML FILE", ofType: "html")!))
```

- If your HTML file is called index.html replace YOUR HTML FILE with index
- You can use this code either in viewDidLoad() or viewDidAppear() or any other function

## Section 41.10: Make links That inside UIWebview clickable



在 vc.h

```
@interface vc : UIViewController<UIWebViewDelegate>
```

in vc.m

```
- (BOOL)webView:(UIWebView *)webView shouldStartLoadWithRequest:(NSURLRequest *)request  
navigationType:(UIWebViewNavigationType)navigationType{  
  
    if (navigationType == UIWebViewNavigationTypeLinkClicked){  
        //open it on browser if you want to open it in same web view remove return NO;  
        NSURL *url = request.URL;  
        if ([[UIApplication sharedApplication] canOpenURL:url]) {  
            [[UIApplication sharedApplication] openURL:url];  
        }  
        return NO;  
    }  
  
    return YES;  
}
```

}

}

# 第42章：UIActivityViewController

## 参数名称

activityItems 包含执行活动的对象数组。该数组不能为nil，且必须至少包含一个对象。

applicationActivities 是一个 UIActivity 对象数组，表示您的应用程序支持的自定义服务。  
此参数可以为nil。

## 第42.1节：初始化活动视图控制器

### Objective-C

```
NSString *textToShare = @"StackOverflow文档！！我们可以为文档做的，  
就像我们为问答所做的一样。";  
NSURL *documentationURL = [NSURL URLWithString:@"http://stackoverflow.com/tour/documentation"];  
  
NSArray *objectsToShare = @[textToShare, documentationURL];  
  
UIActivityViewController *activityVC = [[UIActivityViewController alloc]  
initWithActivityItems:objectsToShare applicationActivities:nil];  
  
[self presentViewController:activityVC animated:YES completion:nil];
```

### Swift

```
let textToShare = "StackOverflow文档！！我们可以为文档做的，  
就像我们为问答所做的一样."  
let documentationURL = NSURL(string:"http://stackoverflow.com/tour/documentation")  
  
let objToShare : [AnyObject] = [textToShare, documentationURL!]  
  
let activityVC = UIActivityViewController(activityItems: objToShare, applicationActivities: nil)  
self.presentViewController(activityVC, animated: true, completion: nil)
```

# Chapter 42: UIActivityViewController

## Parameter Name

activityItems Contains array of object to perform the activity. This array must not be nil and must contain at least one object.

applicationActivities An array of UIActivity objects representing the custom services that your application supports. This parameter can be nil.

## Section 42.1: Initializing the Activity View Controller

### Objective-C

```
NSString *textToShare = @"StackOverflow Documentation!! Together, we can do for Documentation what  
we did for Q&A.";  
NSURL *documentationURL = [NSURL URLWithString:@"http://stackoverflow.com/tour/documentation"];  
  
NSArray *objectsToShare = @[textToShare, documentationURL];  
  
UIActivityViewController *activityVC = [[UIActivityViewController alloc]  
initWithActivityItems:objectsToShare applicationActivities:nil];  
  
[self presentViewController:activityVC animated:YES completion:nil];
```

### Swift

```
let textToShare = "StackOverflow Documentation!! Together, we can do for Documentation what we did  
for Q&A."  
let documentationURL = NSURL(string:"http://stackoverflow.com/tour/documentation")  
  
let objToShare : [AnyObject] = [textToShare, documentationURL!]  
  
let activityVC = UIActivityViewController(activityItems: objToShare, applicationActivities: nil)  
self.presentViewController(activityVC, animated: true, completion: nil)
```

# 第43章：UIControl - 使用块进行事件处理

## 第43.1节：介绍

通常，当使用UIControl或UIButton时，我们会添加一个selector作为回调动作，用于响应按钮或控件上的事件，例如用户按下按钮或触摸控件。

例如，我们会这样做：

```
import UIKit

class ViewController: UIViewController {
    @IBOutlet weak var button: UIButton!

    override func viewDidLoad() {
        super.viewDidLoad()

        let button = UIButton(frame: CGRect(x: 0, y: 0, width: 100, height: 44))
        button.addTarget(self, action: #selector(self.onButtonPress(_:)), for: .touchUpInside)
        self.view.addSubview(button)
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }

    func onButtonPress(_ button: UIButton!) {
        print("按下")
    }
}
```

关于selector，编译器只需要知道它的存在即可。这可以通过protocol实现，而不必具体实现。

例如，以下代码会导致应用程序崩溃：

```
import UIKit

@protocol ButtonEvent {
    @objc 可选函数 onButtonPress(_ button: UIButton)
}

class ViewController: UIViewController, ButtonEvent {
    @IBOutlet weak var button: UIButton!

    override func viewDidLoad() {
        super.viewDidLoad()

        let button = UIButton(frame: CGRect(x: 0, y: 0, width: 100, height: 44))
        button.addTarget(self, action: #selector(ButtonEvent.onButtonPress(_:)), for:
.touchUpInside)
        self.view.addSubview(button)
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }
}
```

# Chapter 43: UIControl - Event Handling with Blocks

## Section 43.1: Introduction

Typically, when using UIControl or UIButton, we add a selector as a callback action for when an event occurs on a button or control, such as the user pressing the button or touching the control.

For example, we would do the following:

```
import UIKit

class ViewController: UIViewController {
    @IBOutlet weak var button: UIButton!

    override func viewDidLoad() {
        super.viewDidLoad()

        let button = UIButton(frame: CGRect(x: 0, y: 0, width: 100, height: 44))
        button.addTarget(self, action: #selector(self.onButtonPress(_:)), for: .touchUpInside)
        self.view.addSubview(button)
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }

    func onButtonPress(_ button: UIButton!) {
        print("PRESSED")
    }
}
```

When it comes to selector, the compiler only needs to know that it exists.. This can be done through a protocol and not be implemented.

For example, the following would crash your application:

```
import UIKit

@protocol ButtonEvent {
    @objc 可选函数 onButtonPress(_ button: UIButton)
}

class ViewController: UIViewController, ButtonEvent {
    @IBOutlet weak var button: UIButton!

    override func viewDidLoad() {
        super.viewDidLoad()

        let button = UIButton(frame: CGRect(x: 0, y: 0, width: 100, height: 44))
        button.addTarget(self, action: #selector(ButtonEvent.onButtonPress(_:)), for:
.touchUpInside)
        self.view.addSubview(button)
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }
}
```

```
}
```

```
}
```

这是因为您的应用程序没有实现 `onButtonPress` 函数。

如果您可以在按钮初始化的同时完成所有这些操作呢？如果您不必指定回调函数，而是可以指定随时添加和移除的代码块呢？为什么还要担心实现选择器呢？

## 解决方案

```
import Foundation
import UIKit

protocol RemovableTarget {
    func enable();
    func disable();
}

extension UIControl {
    func addEventHandler(event: UIControlEvents, runnable: (control: UIControl) -> Void) -> RemovableTarget {
        class Target : RemovableTarget {
            private var event: UIControlEvents
            private weak var control: UIControl?
            private var runnable: (control: UIControl) -> Void

            private init(event: UIControlEvents, control: UIControl, runnable: (control: UIControl) -> Void) {
                self.event = event
                self.control = control
                self.runnable = runnable
            }

            @objc
            private func run(_ control: UIControl) {
                runnable(control: control)
            }

            private func enable() {
                control?.addTarget(self, action: #selector(Target.run(_:)), for: event)
                objc_setAssociatedObject(self, unsafeAddress(of: self), self, .OBJC_ASSOCIATION_RETAIN)
            }

            private func disable() {
                control?.removeTarget(self, action: #selector(Target.run(_:)), for: self.event)
                objc_setAssociatedObject(self, unsafeAddress(of: self), nil, .OBJC_ASSOCIATION_ASSIGN)
            }
        }

        let target = Target(event: event, control: self, runnable: runnable)
        target.enable()
        return target
    }
}
```

以上是对 `UIControl` 的一个简单扩展。它添加了一个内部私有类，该类有一个回调 `func run(_`

```
}
```

```
}
```

This is because your application does NOT implement the `onButtonPress` function.

Now what if you could do all of this alongside the initialization of the button? What if you didn't have to specify callbacks and could instead specify blocks that can be added and removed at any time? Why worry about implementing selectors?

## Solution

```
import Foundation
import UIKit

protocol RemovableTarget {
    func enable();
    func disable();
}

extension UIControl {
    func addEventHandler(event: UIControlEvents, runnable: (control: UIControl) -> Void) -> RemovableTarget {
        class Target : RemovableTarget {
            private var event: UIControlEvents
            private weak var control: UIControl?
            private var runnable: (control: UIControl) -> Void

            private init(event: UIControlEvents, control: UIControl, runnable: (control: UIControl) -> Void) {
                self.event = event
                self.control = control
                self.runnable = runnable
            }

            @objc
            private func run(_ control: UIControl) {
                runnable(control: control)
            }

            private func enable() {
                control?.addTarget(self, action: #selector(Target.run(_:)), for: event)
                objc_setAssociatedObject(self, unsafeAddress(of: self), self, .OBJC_ASSOCIATION_RETAIN)
            }

            private func disable() {
                control?.removeTarget(self, action: #selector(Target.run(_:)), for: self.event)
                objc_setAssociatedObject(self, unsafeAddress(of: self), nil, .OBJC_ASSOCIATION_ASSIGN)
            }
        }

        let target = Target(event: event, control: self, runnable: runnable)
        target.enable()
        return target
    }
}
```

The above is a simple extension on `UIControl`. It adds an inner private class that has a callback `func run(_`

control: UIControl), 作为事件的动作使用。

接下来我们使用对象关联来添加和移除目标，因为UIControl不会保留该目标。

事件处理函数返回一个Protocol，以隐藏Target类的内部实现，同时允许你在任何时候enable和disable该目标。

#### 使用示例：

```
import Foundation
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        //创建一个按钮。
        let button = UIButton(frame: CGRect(x: 0, y: 0, width: 100, height: 44))

        //添加事件动作块/监听器 -- 处理按钮按下事件。
        let target = button.addEventHandler(event: .touchUpInside) { (control) in
            print("Pressed")
        }

        self.view.addSubview(button)

        //启用/禁用监听器/事件动作块的示例。
        DispatchQueue.main.after(when: DispatchTime.now() + 5) {
            target.disable() //禁用监听器。

            DispatchQueue.main.after(when: DispatchTime.now() + 5) {
                target.enable() //启用监听器。
            }
        }

        override func didReceiveMemoryWarning() {
            super.didReceiveMemoryWarning()
        }
    }
}
```

control: UIControl) that is used as the events' action.

Next we use object association to add and remove the target because it will not be retained by the UIControl.

The event handler function returns a Protocol in order to hide the inner workings of the Target class but also to allow you to enable and disable the target at any given time.

#### Usage Example:

```
import Foundation
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        //Create a button.
        let button = UIButton(frame: CGRect(x: 0, y: 0, width: 100, height: 44))

        //Add an event action block/listener -- Handles Button Press.
        let target = button.addEventHandler(event: .touchUpInside) { (control) in
            print("Pressed")
        }

        self.view.addSubview(button)

        //Example of enabling/disabling the listener/event-action-block.
        DispatchQueue.main.after(when: DispatchTime.now() + 5) {
            target.disable() //Disable the listener.

            DispatchQueue.main.after(when: DispatchTime.now() + 5) {
                target.enable() //Enable the listener.
            }
        }

        override func didReceiveMemoryWarning() {
            super.didReceiveMemoryWarning()
        }
    }
}
```

# 第44章：UISplitViewController

## 第44.1节：使用Objective C中的代理实现主视图和详细视图的交互

UISplitViewController 必须是您的应用程序的 rootViewController。

### AppDelegate.m

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // 应用启动后自定义的重写点。
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor blackColor];
    [self.window makeKeyAndVisible];
    self.window.clipsToBounds = YES;
    SplitViewController *spView = [[SplitViewController alloc] init];
    self.window.rootViewController = spView;
    [self.window makeKeyAndVisible];
    return YES;
}
```

只需为 UISplitViewController 创建一个对象，并将该视图控制器设置为您的

应用程序的 rootViewController。

### SplitViewController.h

```
#import <UIKit/UIKit.h>
#import "MasterViewController.h"
#import "DetailViewController.h"
@interface ViewController : UISplitViewController
{
    DetailViewController *detailVC;
    MasterViewController *masterVC;
    NSMutableArray *array;
}
@end
```

MasterViewController 始终位于设备的左侧，您可以在 UISplitViewController 的代理方法中设置宽度，DetailViewController 位于应用程序的右侧

### SplitViewController.m

```
#import "ViewController.h"
#define ANIMATION_LENGTH 0.3
@interface ViewController ()
@end

@implementation ViewController
- (void)viewDidLoad
{
    [super viewDidLoad];
    masterVC = [[MasterViewController alloc] init];
    detailVC = [[DetailViewController alloc] init];
    [masterVC setDetailDelegate:(id)detailVC];
    NSArray *vcArray = [NSArray arrayWithObjects:masterVC, detailVC, nil];
}
```

# Chapter 44: UISplitViewController

## Section 44.1: Master and Detail View interaction using Delegates in Objective C

UISplitViewController must be the rootViewController of your application.

### AppDelegate.m

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // Override point for customization after application launch.
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor blackColor];
    [self.window makeKeyAndVisible];
    self.window.clipsToBounds = YES;
    SplitViewController *spView = [[SplitViewController alloc] init];
    self.window.rootViewController = spView;
    [self.window makeKeyAndVisible];
    return YES;
}
```

Just create an object for the UISplitViewController and set that viewcontroller as the rootviewcontroller for your application.

### SplitViewController.h

```
#import <UIKit/UIKit.h>
#import "MasterViewController.h"
#import "DetailViewController.h"
@interface ViewController : UISplitViewController
{
    DetailViewController *detailVC;
    MasterViewController *masterVC;
    NSMutableArray *array;
}
@end
```

MasterViewController is always on the left side of the device you can set the width in UISplitViewController delegate methods and DetailViewController is on the Right side of the application

### SplitViewController.m

```
#import "ViewController.h"
#define ANIMATION_LENGTH 0.3
@interface ViewController ()
@end

@implementation ViewController
- (void)viewDidLoad
{
    [super viewDidLoad];
    masterVC = [[MasterViewController alloc] init];
    detailVC = [[DetailViewController alloc] init];
    [masterVC setDetailDelegate:(id)detailVC];
    NSArray *vcArray = [NSArray arrayWithObjects:masterVC, detailVC, nil];
}
```

```

self.preferredDisplayMode = UISplitViewControllerDisplayModeAutomatic;
self.viewControllers = vcArray;
self.delegate = (id)self;
self.presentsWithGesture = YES;
}

```

创建的主视图控制器和详细视图控制器被添加到一个数组中，该数组被设置为 UISplitViewController 中的 self.viewControllers。self.preferredDisplayMode 是用于显示主视图控制器和详细视图控制器的模式，详见 Apple 文档中的 DisplayMode。self.presentsWithGesture 启用滑动手势以显示主视图控制器。

### MasterViewController.h

```

#import <UIKit/UIKit.h>

@protocol DetailViewDelegate <NSObject>
@required
- (void)sendSelectedNavController:(UIViewController *)viewController;
@end

@interface MasterViewController : UIViewController
{
    UITableView *mainTableView;
    NSMutableArray *viewControllerArray;
}
@property (nonatomic, retain) id<DetailViewDelegate> detailDelegate;
@end

```

创建一个带有 sendSelectedNavController:(UIViewController \*)viewController 方法的 DetailViewDelegate 委托，用于向 DetailViewController 发送 UIViewController。然后在 MasterViewController 中，mainTableView 是左侧的表视图。viewControllerArray 包含所有需要在 DetailViewController 中显示的 UIViewController。

### MasterViewController.m

```

#import "MasterViewController.h"

@implementation MasterViewController
@synthesize detailDelegate;

-(void)viewDidLoad
{
[super viewDidLoad];

UIViewController *dashBoardVC = [[UIViewController alloc] init];
[dashBoardVC.view setBackgroundColor:[UIColor redColor]];
UIViewController *inventVC = [[UIViewController alloc] init];
[inventVC.view setBackgroundColor:[UIColor whiteColor]];
UIViewController *alarmVC = [[UIViewController alloc] init];
[alarmVC.view setBackgroundColor: [UIColor purpleColor]];
UIViewController *scanDeviceVC = [[UIViewController alloc] init];
[scanDeviceVC.view setBackgroundColor:[UIColor cyanColor]];
UIViewController *serverDetailVC = [[UIViewController alloc] init];
[serverDetailVC.view setBackgroundColor: [UIColor whiteColor]];
viewControllerArray = [[NSMutableArray alloc] initWithObjects:dashBoardVC,inventVC,alarmVC,scanDeviceVC,serverDetailVC,nil];
mainTableView = [[UITableView alloc] initWithFrame:CGRectMake(0, 50,self.view.frame.size.width, self.view.frame.size.height-50) style:UITableViewStylePlain];
[mainTableView setDelegate:(id)self];

```

```

self.preferredDisplayMode = UISplitViewControllerDisplayModeAutomatic;
self.viewControllers = vcArray;
self.delegate = (id)self;
self.presentsWithGesture = YES;
}

```

Created master and detail ViewControllers are added to an array which is set to `self.viewControllers` in `UISplitViewController`. `self.preferredDisplayMode` is the mode set for displaying of master and DetailViewController [Apple Documentation for DisplayMode](#). `self.presentsWithGesture` enables swipe gesture for displaying MasterViewController

### MasterViewController.h

```

#import <UIKit/UIKit.h>

@protocol DetailViewDelegate <NSObject>
@required
- (void)sendSelectedNavController:(UIViewController *)viewController;
@end

@interface MasterViewController : UIViewController
{
    UITableView *mainTableView;
    NSMutableArray *viewControllerArray;
}
@property (nonatomic, retain) id<DetailViewDelegate> detailDelegate;
@end

```

Create a DetailViewDelegate delegate with `sendSelectedNavController:(UIViewController *)viewController` method for sending the `UIViewController` to the DetailViewController. Then in MasterViewController the mainTableView is the tableview in the leftside. The viewControllerArray contains all the UIViewController that needs to be displayed in DetailViewController

### MasterViewController.m

```

#import "MasterViewController.h"

@implementation MasterViewController
@synthesize detailDelegate;

-(void)viewDidLoad
{
[super viewDidLoad];

UIViewController *dashBoardVC = [[UIViewController alloc] init];
[dashBoardVC.view setBackgroundColor:[UIColor redColor]];
UIViewController *inventVC = [[UIViewController alloc] init];
[inventVC.view setBackgroundColor:[UIColor whiteColor]];
UIViewController *alarmVC = [[UIViewController alloc] init];
[alarmVC.view setBackgroundColor: [UIColor purpleColor]];
UIViewController *scanDeviceVC = [[UIViewController alloc] init];
[scanDeviceVC.view setBackgroundColor:[UIColor cyanColor]];
UIViewController *serverDetailVC = [[UIViewController alloc] init];
[serverDetailVC.view setBackgroundColor: [UIColor whiteColor]];
viewControllerArray = [[NSMutableArray alloc] initWithObjects:dashBoardVC,inventVC,alarmVC,scanDeviceVC,serverDetailVC,nil];
mainTableView = [[UITableView alloc] initWithFrame:CGRectMake(0, 50,self.view.frame.size.width, self.view.frame.size.height-50) style:UITableViewStylePlain];
[mainTableView setDelegate:(id)self];

```

```

[mainTableView setDataSource:(id)self];
[mainTableView setSeparatorStyle:UITableViewCellSeparatorStyleNone];
[mainTableView setScrollsToTop:NO];
[self.view addSubview:mainTableView];
}

- (CGFloat)tableView:(UITableView *)tableView
heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
    return 100;
}
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    return [viewControllerArray count];
}

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1; //节的数量
}

- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSString *cellId = [NSString
stringWithFormat:@"Cell%li%ld",(long)indexPath.section,(long)indexPath.row];
UITableViewCell *cell =[tableView dequeueReusableCellWithIdentifier:cellId];

if (cell == nil)
{
    cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:cellId];
}
else
{
    contentView setBackgroundColor:[UIColor redColor]];
    cell.textLabel.text =[NSString stringWithFormat:@"我的视图控制器索引 %ld",(long)indexPath.row];
    return cell;
}

- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    [detailDelegate sendSelectedNavController:[viewControllerArray objectAtIndex:indexPath.row]];
}

@end

```

创建了一些UIViewController并将其添加到一个数组中。然后初始化表视图，在didSelectRowAtIndexPath方法中，我使用detailDelegate将数组中对应的UIViewController作为参数发送给DetailViewController

### DetailViewController.h

```

#import <UIKit/UIKit.h>

@interface DetailViewController : UIViewController<UICollectionViewDelegate>
{
    UIViewController *tempNav;
}
@end

```

```

[mainTableView setDataSource:(id)self];
[mainTableView setSeparatorStyle:UITableViewCellSeparatorStyleNone];
[mainTableView setScrollsToTop:NO];
[self.view addSubview:mainTableView];
}

- (CGFloat)tableView:(UITableView *)tableView
heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
    return 100;
}
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    return [viewControllerArray count];
}

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1; //count of section
}

- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSString *cellId = [NSString
stringWithFormat:@"Cell%li%ld",(long)indexPath.section,(long)indexPath.row];
UITableViewCell *cell =[tableView dequeueReusableCellWithIdentifier:cellId];

if (cell == nil)
{
    cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:cellId];
}
[cell.contentView setBackgroundColor:[UIColor redColor]];
cell.textLabel.text =[NSString stringWithFormat:@"My VC at index %ld",(long)indexPath.row];
return cell;
}

- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    [detailDelegate sendSelectedNavController:[viewControllerArray objectAtIndex:indexPath.row]];
}

@end

```

Create some UIViewController and added it to an array. The Table view is initialized then on didSelectRowAtIndexPath method I send a UIViewController to the DetailViewController using detailDelegate with the corresponding UIViewController in array as parameter

### DetailViewController.h

```

#import <UIKit/UIKit.h>

@interface DetailViewController : UIViewController<UICollectionViewDelegate>
{
    UIViewController *tempNav;
}
@end

```

### DetailViewController.m

```
#import "DetailViewController.h"

@implementation DetailViewController
-(void)viewDidLoad
{
    [super viewDidLoad];
    [self.view setBackgroundColor:[UIColor whiteColor]];
}
-(void)sendSelectedNavController:(UIViewController *)navController
{
    NSArray *viewsToRemove = [self.view subviews];
    for (UIView *v in viewsToRemove) {
        [v removeFromSuperview];
    }
    tempNav = navController;
    [self.view addSubview:tempNav.view];
}
@end
```

sendSelectedNavController 在此声明，功能是移除DetailViewController中的所有视图，并添加从MasterViewController传入的UIViewController

添加一些应用程序的截图

### DetailViewController.m

```
#import "DetailViewController.h"

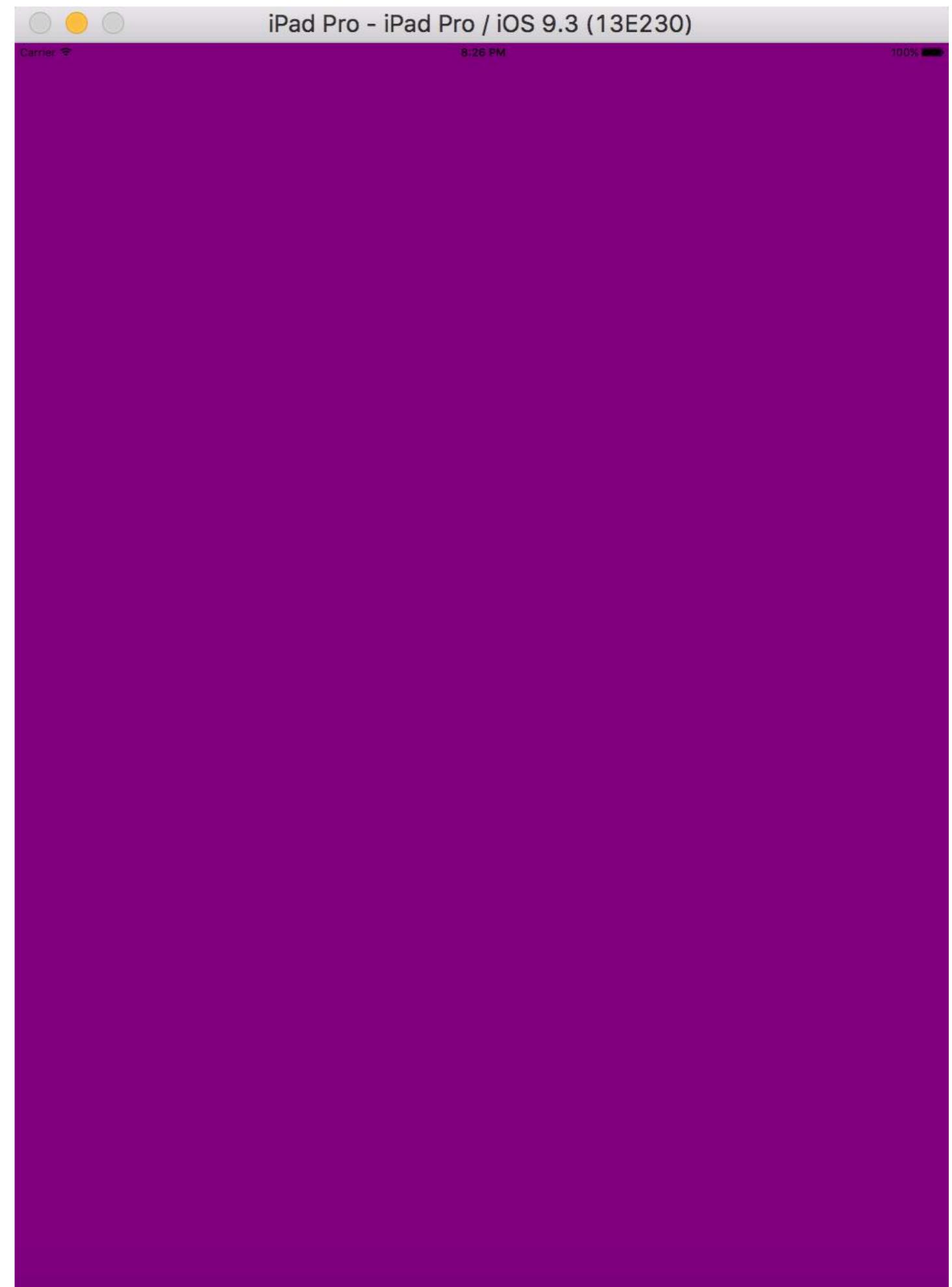
@implementation DetailViewController
-(void)viewDidLoad
{
    [super viewDidLoad];
    [self.view setBackgroundColor:[UIColor whiteColor]];
}
-(void)sendSelectedNavController:(UIViewController *)navController
{
    NSArray *viewsToRemove = [self.view subviews];
    for (UIView *v in viewsToRemove) {
        [v removeFromSuperview];
    }
    tempNav = navController;
    [self.view addSubview:tempNav.view];
}
@end
```

sendSelectedNavController is declared here with removing all the views in the DetailViewController and adding the passed UIViewController from the MasterViewController

Adding some screen shots of the application



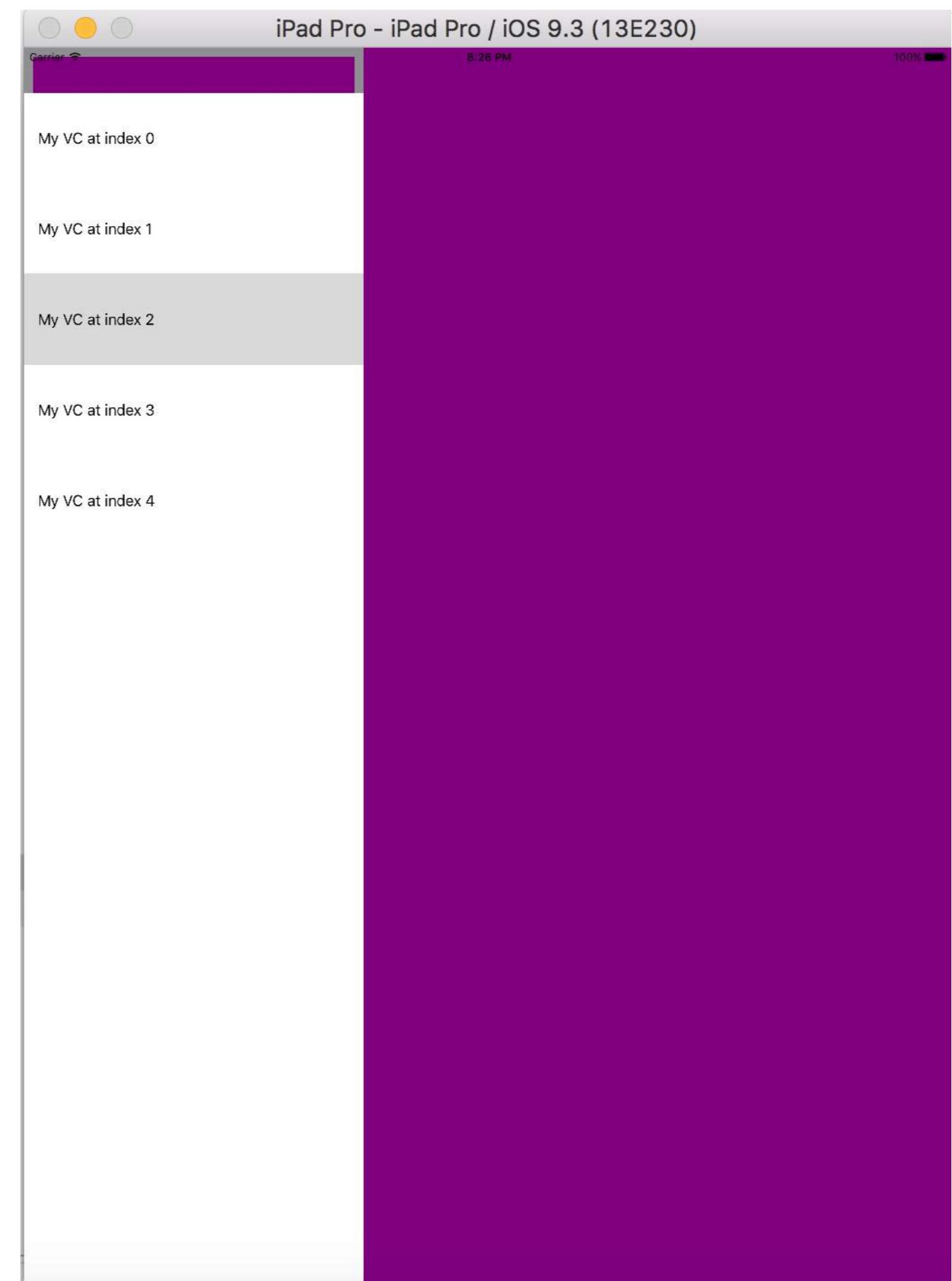
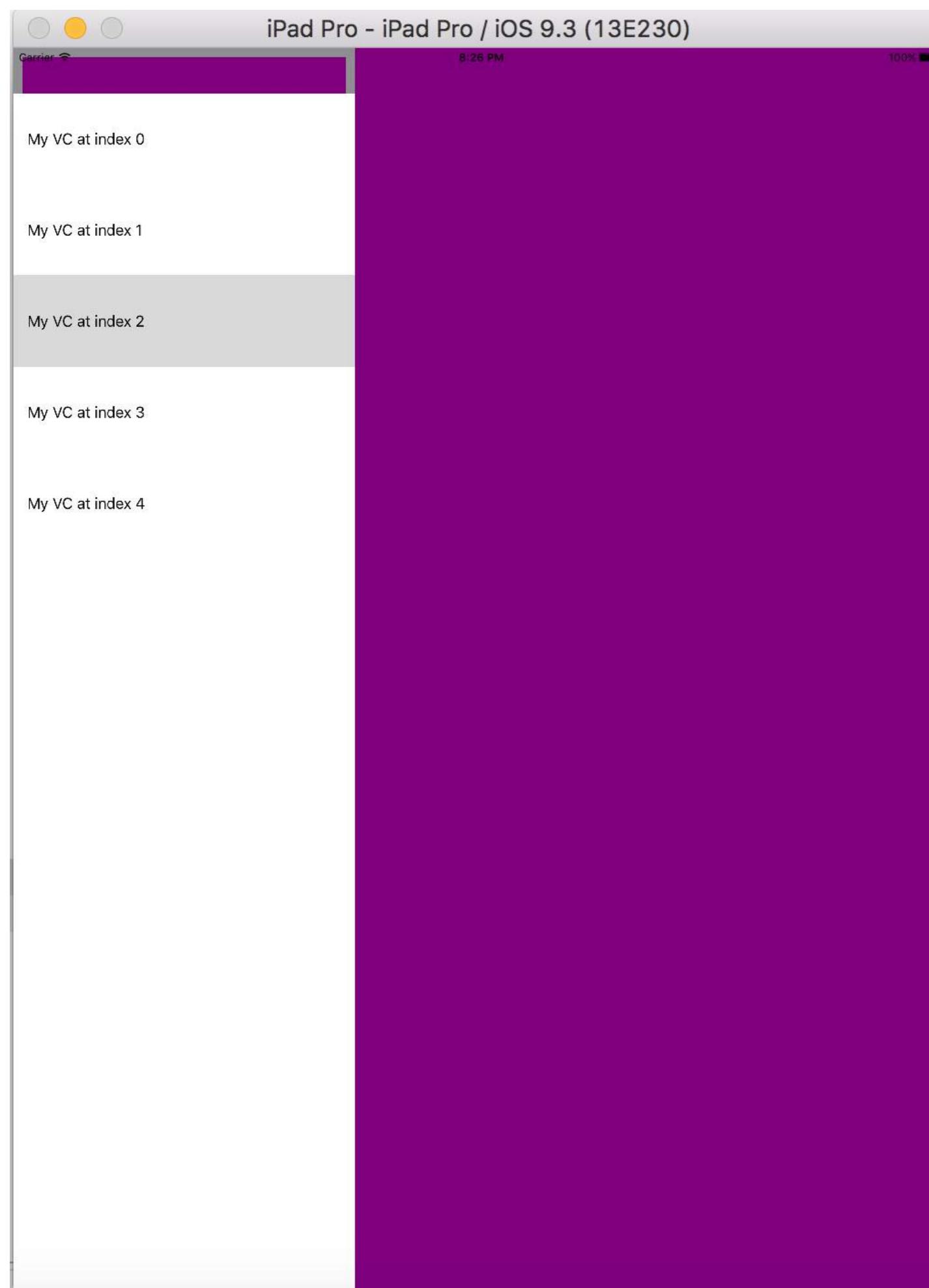
启动应用程序时，我们没有得到MasterViewController，因为我们将preferredDisplayMode设置为

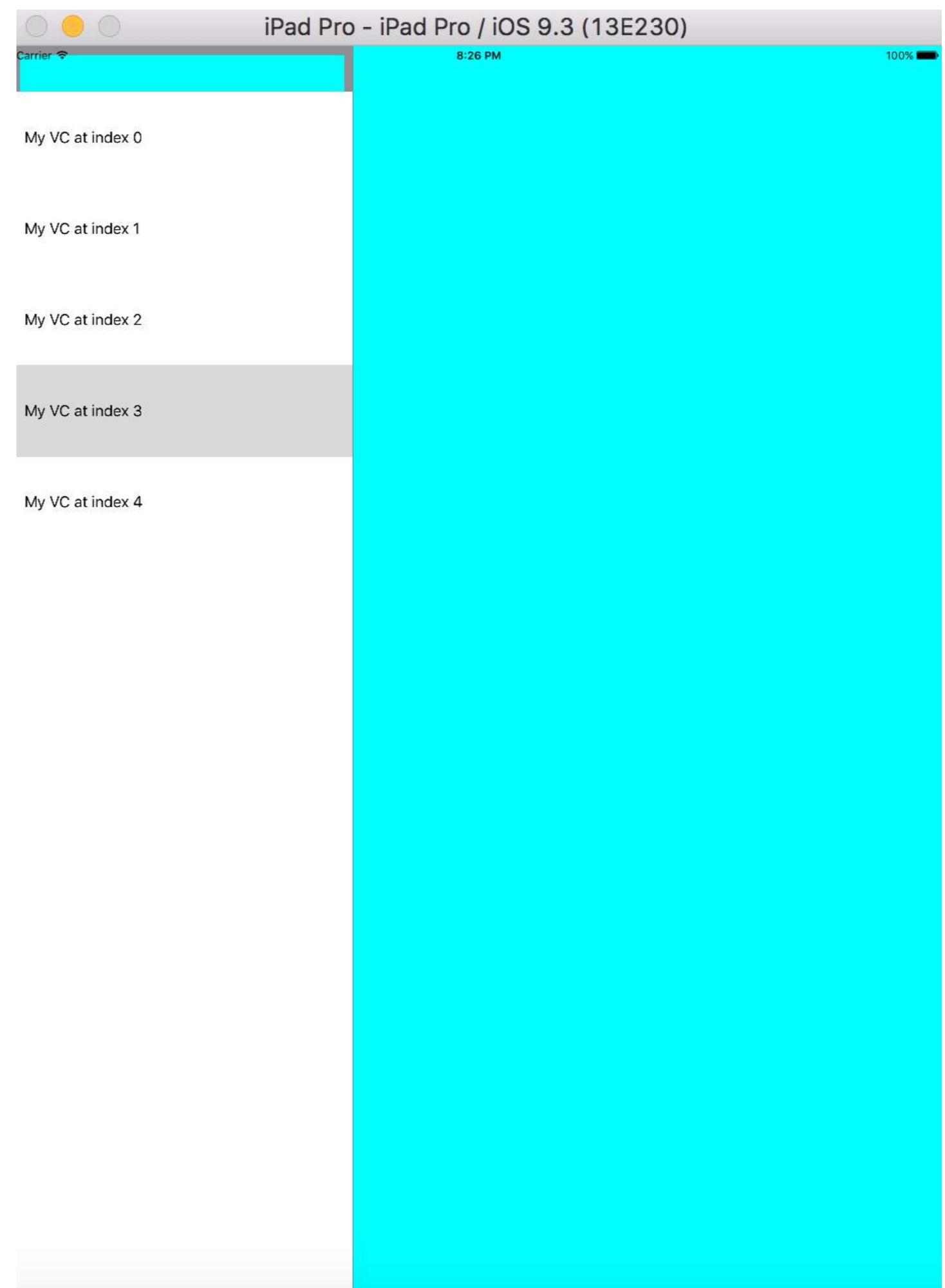
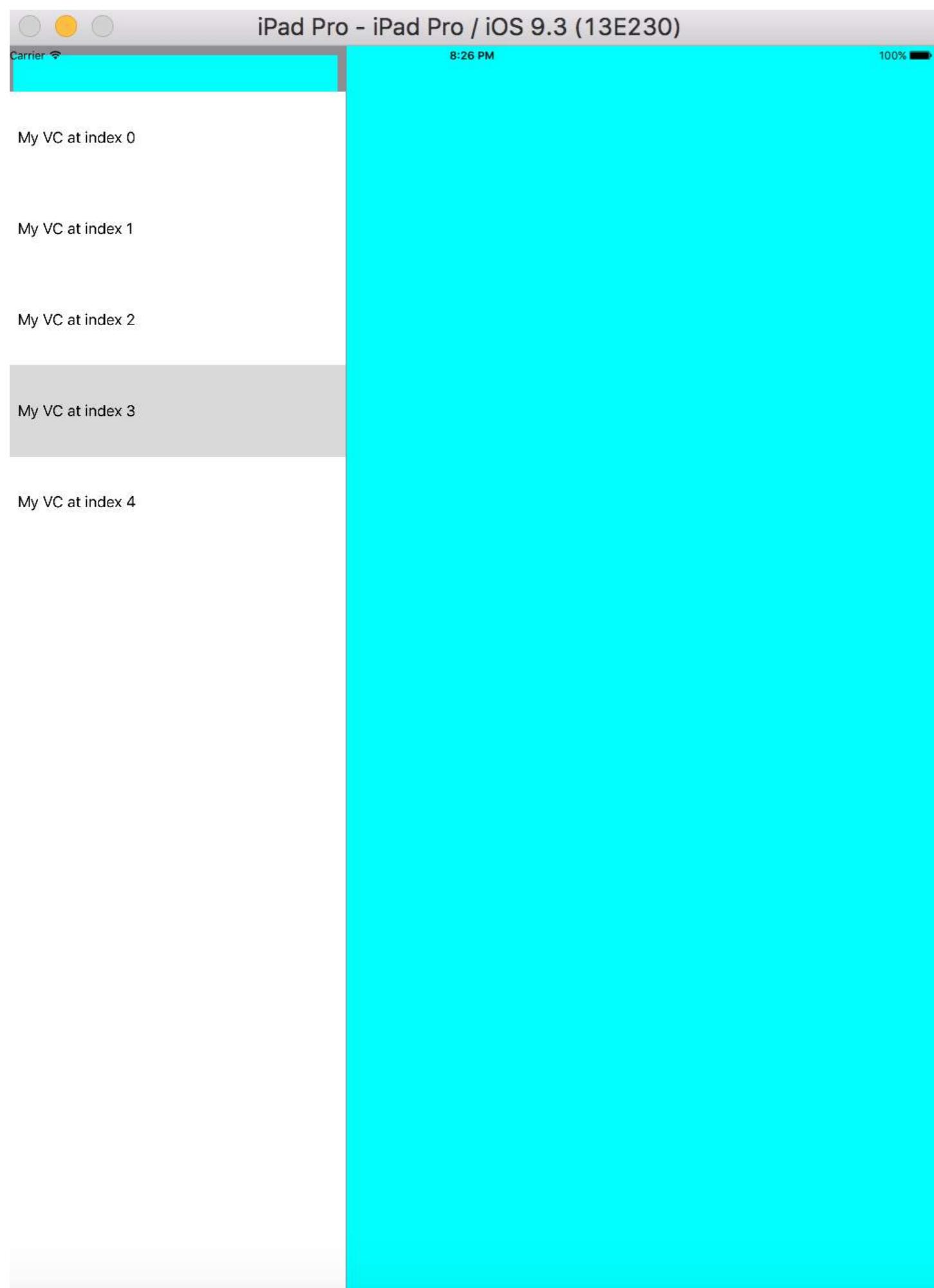


On launching the application we don't get MasterViewController since we gave the preferredDisplayMode as

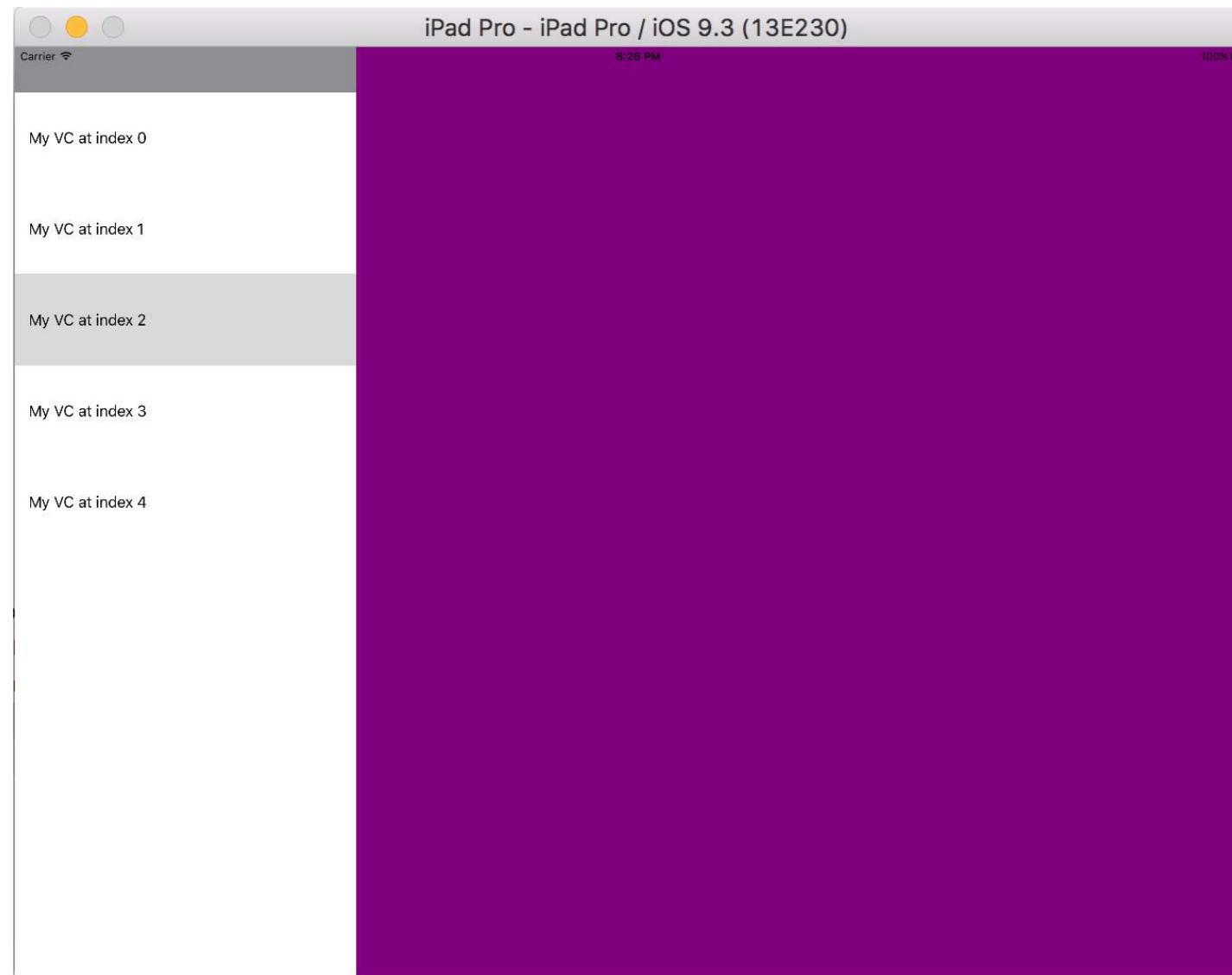
自动，滑动屏幕时我们会得到如下面图片所示的MasterViewController，但在横屏模式下，我们会同时看到MasterViewController和DetailViewController

automatic on swiping the screen we get the MasterViewController as attached in the below image but in Landscape mode we get both the MasterViewController and DetailViewController

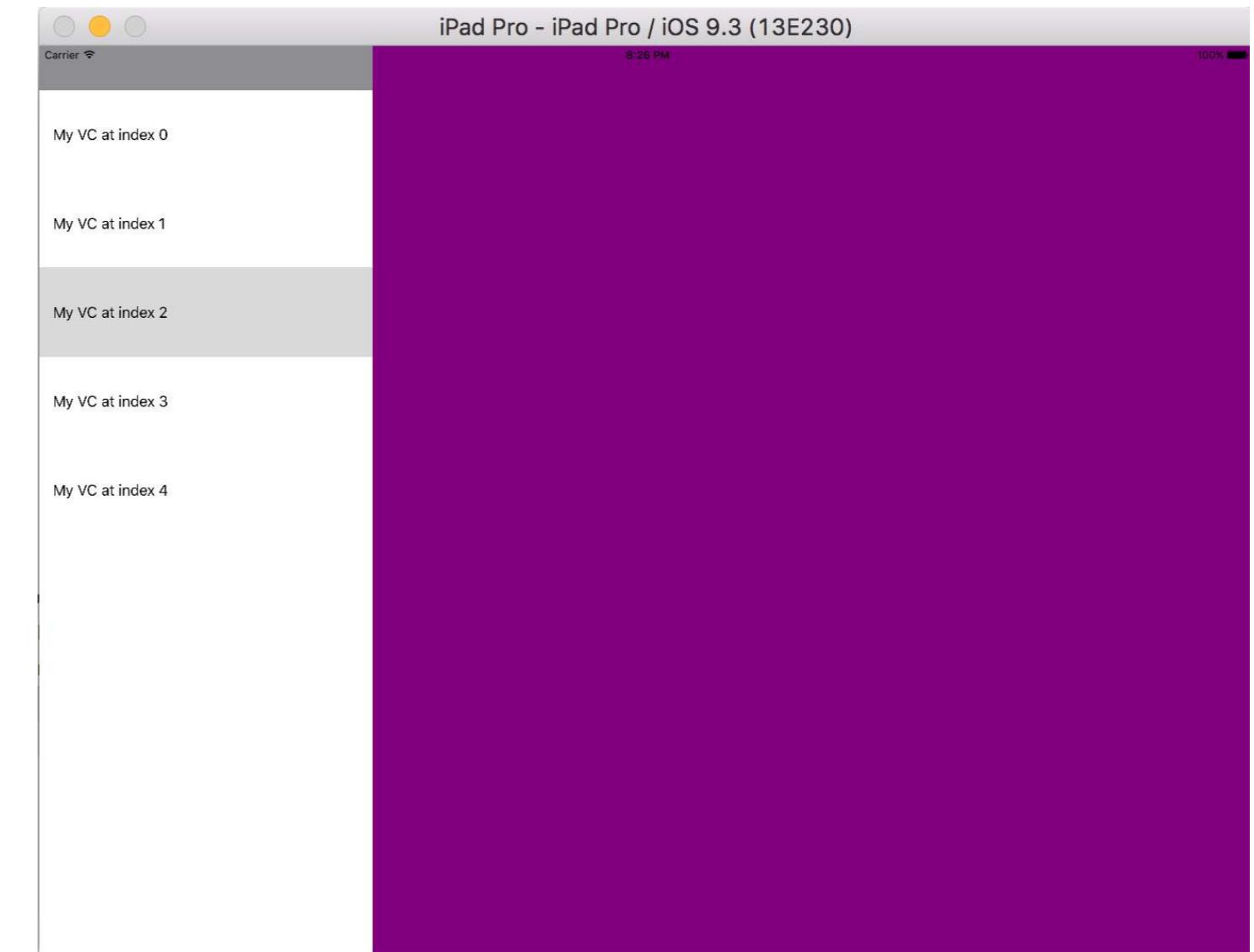


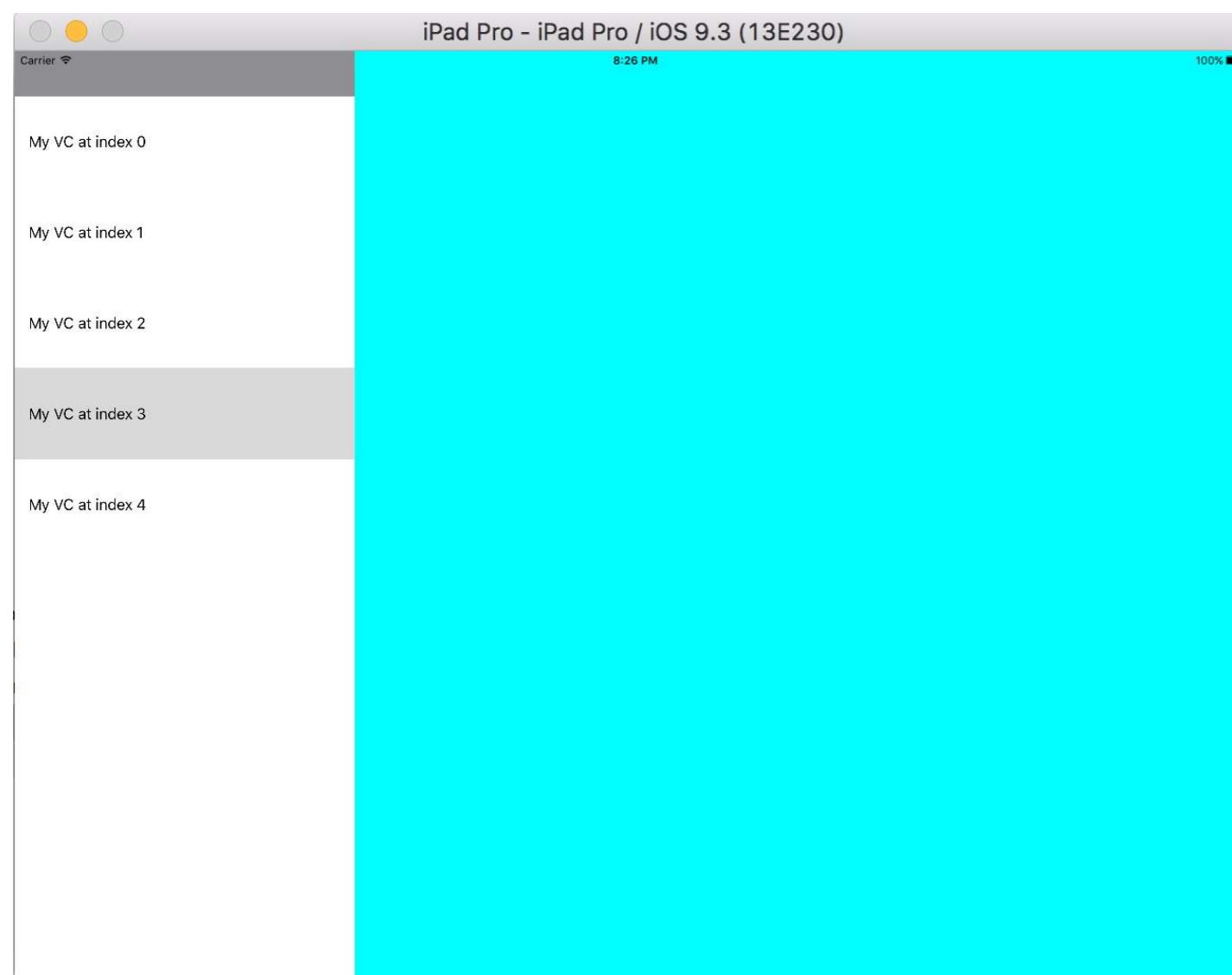


## 关于横向布局



on Landscape Orientation





# 第45章：UISlider

## 第45.1节：UISlider

### Objective-C

在ViewController.h或ViewController.m的接口中声明一个滑块属性

```
@property (strong, nonatomic) UISlider *slider;

//定义滑块的框架并添加到视图中
CGRect frame = CGRectMake(0.0, 100.0, 320.0, 10.0);
UISlider *slider = [[UISlider alloc] initWithFrame:frame];
[slider addTarget:self action:@selector(sliderAction:) forControlEvents:UIControlEventValueChanged];
[slider setBackgroundColor:[UIColor clearColor]];
self.slider.minimumValue = 0.0;
self.slider.maximumValue = 50.0;
//发送NO/False将只在用户不再触摸屏幕时更新滑块的值。因此只发送最终值
self.slider.continuous = YES;
self.slider.value = 25.0;
[self.view addSubview slider];
```

处理滑块变化事件

```
- (IBAction)sliderAction:(id)sender {
    NSLog(@"滑块值 %f", sender.value);
}
```

## 第45.2节：SWIFT示例

```
let frame = CGRect(x: 0, y: 100, width: 320, height: 10)
let slider = UISlider(frame: frame)
slider.addTarget(self, action: #selector(sliderAction), for: .valueChanged)
slider.backgroundColor = .clear
slider.minimumValue = 0.0
slider.maximumValue = 50.0
//发送NO/False将只在用户不再触摸屏幕时更新滑块的值。因此只发送最终值
slider.isContinuous = true
slider.value = 25.0
view.addSubview(slider)
```

处理滑块变化事件

```
func sliderAction(sender:UISlider!)
{
    print("value--\((sender.value))")
}
```

## 第45.3节：添加自定义滑块拇指图像

要为滑块的拇指添加自定义图像，只需调用[setThumbImage](#)方法并传入你的自定义图像：

# Chapter 45: UISlider

## Section 45.1: UISlider

### Objective-C

Declare a slider property in the ViewController.h or in the interface of ViewController.m

```
@property (strong, nonatomic) UISlider *slider;

//Define frame of slider and add to view
CGRect frame = CGRectMake(0.0, 100.0, 320.0, 10.0);
UISlider *slider = [[UISlider alloc] initWithFrame:frame];
[slider addTarget:self action:@selector(sliderAction:) forControlEvents:UIControlEventValueChanged];
[slider setBackgroundColor:[UIColor clearColor]];
self.slider.minimumValue = 0.0;
self.slider.maximumValue = 50.0;
//sending a NO/False would update the value of slider only when the user is no longer touching the screen. Hence sending only the final value
self.slider.continuous = YES;
self.slider.value = 25.0;
[self.view addSubview slider];
```

Handle the slider change event

```
- (IBAction)sliderAction:(id)sender {
    NSLog(@"Slider Value %f", sender.value);
}
```

## Section 45.2: SWIFT Example

```
let frame = CGRect(x: 0, y: 100, width: 320, height: 10)
let slider = UISlider(frame: frame)
slider.addTarget(self, action: #selector(sliderAction), for: .valueChanged)
slider.backgroundColor = .clear
slider.minimumValue = 0.0
slider.maximumValue = 50.0
//sending a NO/False would update the value of slider only when the user is no longer touching the screen. Hence sending only the final value
slider.isContinuous = true
slider.value = 25.0
view.addSubview(slider)
```

Handling the slider change event

```
func sliderAction(sender:UISlider!)
{
    print("value--\((sender.value))")
}
```

## Section 45.3: Adding a custom thumb image

To add a custom image for the thumb of the slider, simply call the [setThumbImage](#) method with your custom image:

Swift 3.1 :

```
let slider = UISlider()  
let thumbImage = UIImage  
slider.setThumbImage(thumbImage, for: .normal)
```

Swift 3.1:

```
let slider = UISlider()  
let thumbImage = UIImage  
slider.setThumbImage(thumbImage, for: .normal)
```

# 第46章：UIStoryboard

UIStoryboard对象封装了存储在Interface Builder故事板资源文件中的视图控制器图。该视图控制器图表示应用程序用户界面全部或部分的视图控制器。

## 第46.1节：通过编程方式获取UIStoryboard实例

SWIFT：

通过代码获取UIStoryboard实例可以按如下方式进行：

```
let storyboard = UIStoryboard(name: "Main", bundle: nil)
```

其中：

- **name** => 不带扩展名的故事板名称
- **bundle** => 包含故事板文件及其相关资源的包。如果指定为nil，该方法将在当前应用的主包中查找。

例如，你可以使用上述创建的实例访问该

故事板中实例化的某个UIViewController：

```
let viewController = storyboard.instantiateViewController(withIdentifier: "yourIdentifier")
```

OBJECTIVE-C：

在Objective-C中获取UIStoryboard实例可以按如下方式进行：

```
UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"MainStoryboard" bundle:nil];
```

访问该故事板中实例化的UIViewController示例：

```
MyViewController *myViewController = [storyboard instantiateViewControllerWithIdentifier:@"MyViewControllerIdentifier"];
```

## 第46.2节：打开另一个故事板

```
let storyboard = UIStoryboard(name: "StoryboardName", bundle: nil)
let vc = storyboard.instantiateViewController(withIdentifier: "ViewControllerID") as YourViewController
self.present(vc, animated: true, completion: nil)
```

# Chapter 46: UIStoryboard

A UIStoryboard object encapsulates the view controller graph stored in an Interface Builder storyboard resource file. This view controller graph represents the view controllers for all or part of your application's user interface.

## Section 46.1: Getting an instance of UIStoryboard programmatically

SWIFT:

Getting an instance of **UIStoryboard** programmatically can be done as follows:

```
let storyboard = UIStoryboard(name: "Main", bundle: nil)
```

where:

- **name** => the name of the storyboard without the extension
- **bundle** => the bundle containing the storyboard file and its related resources. If you specify nil, this method looks in the main bundle of the current application.

For example, you can use the instance created above to access a certain **UIViewController** instantiated within that storyboard:

```
let viewController = storyboard.instantiateViewController(withIdentifier: "yourIdentifier")
```

OBJECTIVE-C:

Getting an instance of **UIStoryboard** in Objective-C can be done as follows:

```
UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"MainStoryboard" bundle:nil];
```

Example of accessing **UIViewController** instantiated within that storyboard:

```
MyViewController *myViewController = [storyboard instantiateViewControllerWithIdentifier:@"MyViewControllerIdentifier"];
```

## Section 46.2: Open another storyboard

```
let storyboard = UIStoryboard(name: "StoryboardName", bundle: nil)
let vc = storyboard.instantiateViewController(withIdentifier: "ViewControllerID") as YourViewController
self.present(vc, animated: true, completion: nil)
```

# 第47章：UIPageViewController

UIPageViewController 使用用户能够通过滑动手势轻松地在多个视图之间切换。为了创建 UIPageViewController，您必须实现 UIPageViewControllerDataSource 方法。这些方法包括返回当前 UIPageViewController 之前和之后的 UIPageViewController 的方法以及 presentationCount 和 presentationIndex 方法。

## 第47.1节：通过编程方式创建水平分页的UI PageViewController

1. 初始化将由 UIPageViewController 管理的视图控制器数组。添加一个基础视图控制器类，该类具有属性 identifier，用于在使用 UIPageViewController 数据源方法时识别视图控制器。让视图控制器继承该基础类。

```
UIViewController *firstVC = [[UIViewController alloc] init];
firstVC.identifier = 0
UIViewController *secondVC = [[UIViewController alloc] init];
secondVC.identifier = 1
NSArray *viewControllers = [[NSArray alloc] initWithObjects: firstVC, secondVC, nil];
```

2. 创建 UIPageViewController 实例。

```
UIPageViewController *pageViewController = [[UIPageViewController alloc]
initWithTransitionStyle:UIPageViewControllerTransitionStyleScroll
navigationOrientation:UIPageViewControllerNavigationOrientationHorizontal
options:nil];
```

3. 数据源是当前类，必须实现 UIPageViewControllerDataSource 协议。

```
pageViewController.dataSource = self;
```

4. setViewControllers 将只添加第一个视图控制器，后续视图控制器将通过数据源添加到堆栈中方法

```
if (viewControllers.count) {
    [pageViewController setViewControllers:@[[viewControllers objectAtIndex:0]]
        direction:UIPageViewControllerNavigationDirectionForward
        animated:NO
completion:nil];
}
```

5. 将 UIPageViewController 添加为子视图控制器，以便它能接收来自父视图控制器的外观 和 旋转 事件。

```
[self addChildViewController:pageViewController];
pageViewController.view.frame = self.view.frame;
[self.view addSubview:pageViewController.view];
[pageViewController didMoveToParentViewController:self];
```

6. 实现 UIPageViewControllerDataSource 方法

```
- (UIViewController *)pageViewController:(UIPageViewController *)pageViewController
    viewControllerBeforeViewController:(UIViewController *)viewController
{
    index = [(Your View Controller Base Class *)viewController identifier];
```

# Chapter 47: UIPageViewController

UIPageViewController provides users the ability to easily transition between several views by using a swipe gesture. In order to create a UIPageViewController, you must implement the UIPageViewControllerDataSource methods. These include methods to return both the UIPageViewController before and after the current UIPageViewController along with the presentationCount and presentationIndex methods.

## Section 47.1: Create a horizontal paging UIPageViewController programmatically

1. Init array of view controllers which will be managed by UIPageViewController. Add a base view controller class which has property identifier which will be used to identify view controllers when working with UIPageViewController data source methods. Let the view controllers to inherit from that base class.

```
UIViewController *firstVC = [[UIViewController alloc] init];
firstVC.identifier = 0
UIViewController *secondVC = [[UIViewController alloc] init];
secondVC.identifier = 1
NSArray *viewControllers = [[NSArray alloc] initWithObjects: firstVC, secondVC, nil];
```

2. Create UIPageViewController instance.

```
UIPageViewController *pageViewController = [[UIPageViewController alloc]
initWithTransitionStyle:UIPageViewControllerTransitionStyleScroll
navigationOrientation:UIPageViewControllerNavigationOrientationHorizontal
options:nil];
```

3. Data source is current class which must implement UIPageViewControllerDataSource protocol.

```
pageViewController.dataSource = self;
```

4. setViewControllers will add only first view controller, next will be added to the stack using data source methods

```
if (viewControllers.count) {
    [pageViewController setViewControllers:@[[viewControllers objectAtIndex:0]]
        direction:UIPageViewControllerNavigationDirectionForward
        animated:NO
completion:nil];
}
```

5. Add UIPageViewController as a child view controller so it will receive from its parent view controller appearance and rotation events.

```
[self addChildViewController:pageViewController];
pageViewController.view.frame = self.view.frame;
[self.view addSubview:pageViewController.view];
[pageViewController didMoveToParentViewController:self];
```

6. Implementing UIPageViewControllerDataSource methods

```
- (UIViewController *)pageViewController:(UIPageViewController *)pageViewController
    viewControllerBeforeViewController:(UIViewController *)viewController
{
    index = [(Your View Controller Base Class *)viewController identifier];
```

```

index--;
return [self childViewControllerAtIndex:index];
}

- (UIViewController *)pageViewController:(UIPageViewController *)pageViewController
    viewControllerAfterViewController:(UIViewController *)viewController
{
index = [(你的视图控制器基类 *)viewController identifier];
index++;
return [self childViewControllerAtIndex:index];
}

- (NSInteger)presentationCountForPageViewController:(UIPageViewController *)pageViewController
{
    return [viewControllers count];
}

- (NSInteger)presentationIndexForPageViewController:(UIPageViewController *)pageViewController
{
    return index;
}

```

7. 实用方法，通过索引返回视图控制器，如果索引越界则返回nil。

```

- (UIViewController *)childViewControllerAtIndex:(NSInteger)index
{
    if (index <= ([viewControllers count] - 1)) {
        return [viewControllers objectAtIndex:index];
    } else {
        return nil;
    }
}

```

## 第47.2节：创建水平分页视图控制器（无限分页）的简单方法

1. 让我们创建一个新项目，我选择单视图应用程序以便更好地演示

```

index--;
return [self childViewControllerAtIndex:index];
}

- (UIViewController *)pageViewController:(UIPageViewController *)pageViewController
    viewControllerAfterViewController:(UIViewController *)viewController
{
index = [(Your View Controller Base Class *)viewController identifier];
index++;
return [self childViewControllerAtIndex:index];
}

- (NSInteger)presentationCountForPageViewController:(UIPageViewController *)pageViewController
{
    return [viewControllers count];
}

- (NSInteger)presentationIndexForPageViewController:(UIPageViewController *)pageViewController
{
    return index;
}

```

7. Utility method which returns a view controller using an index, if index is out of bounds it returns nil.

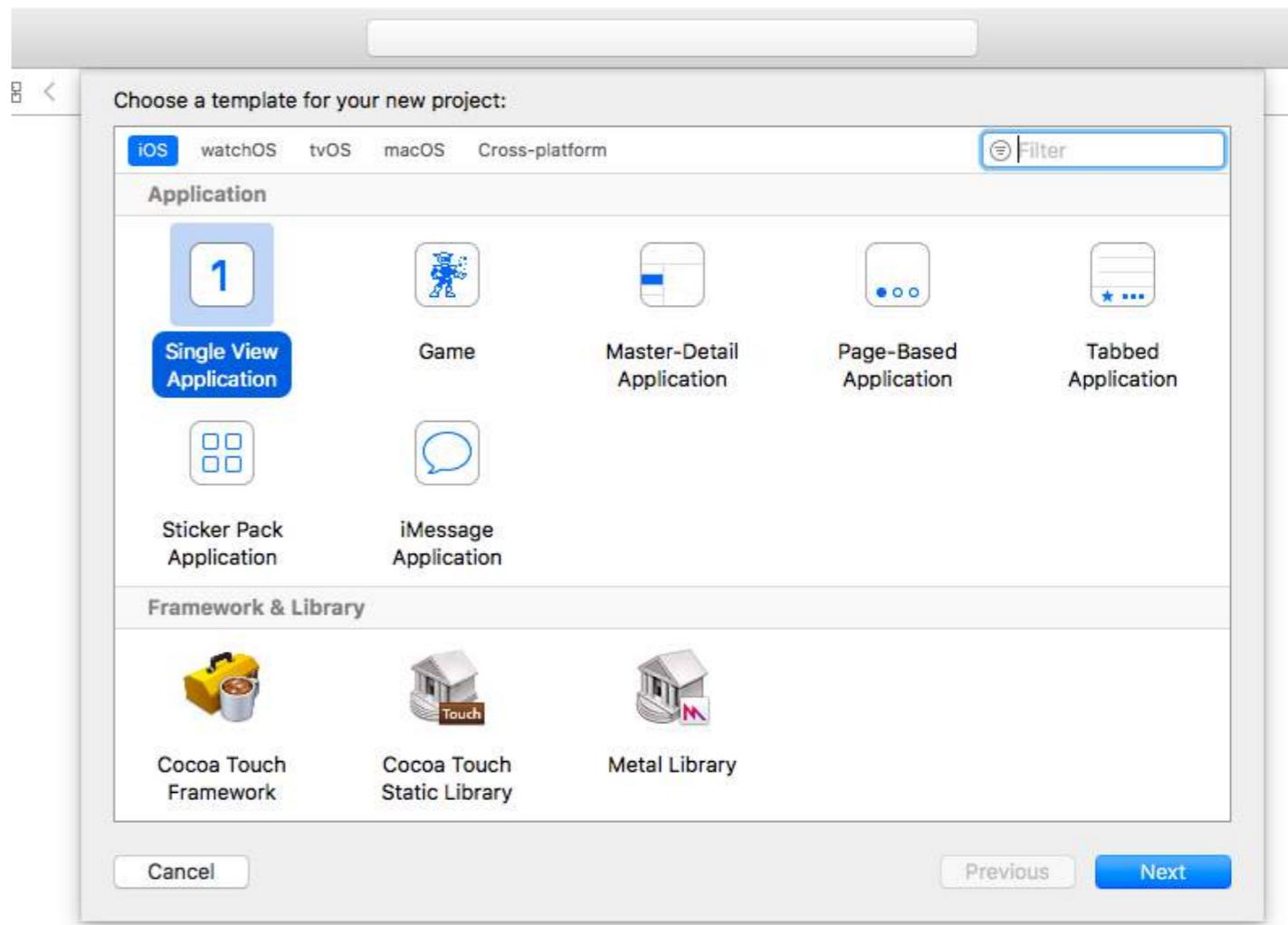
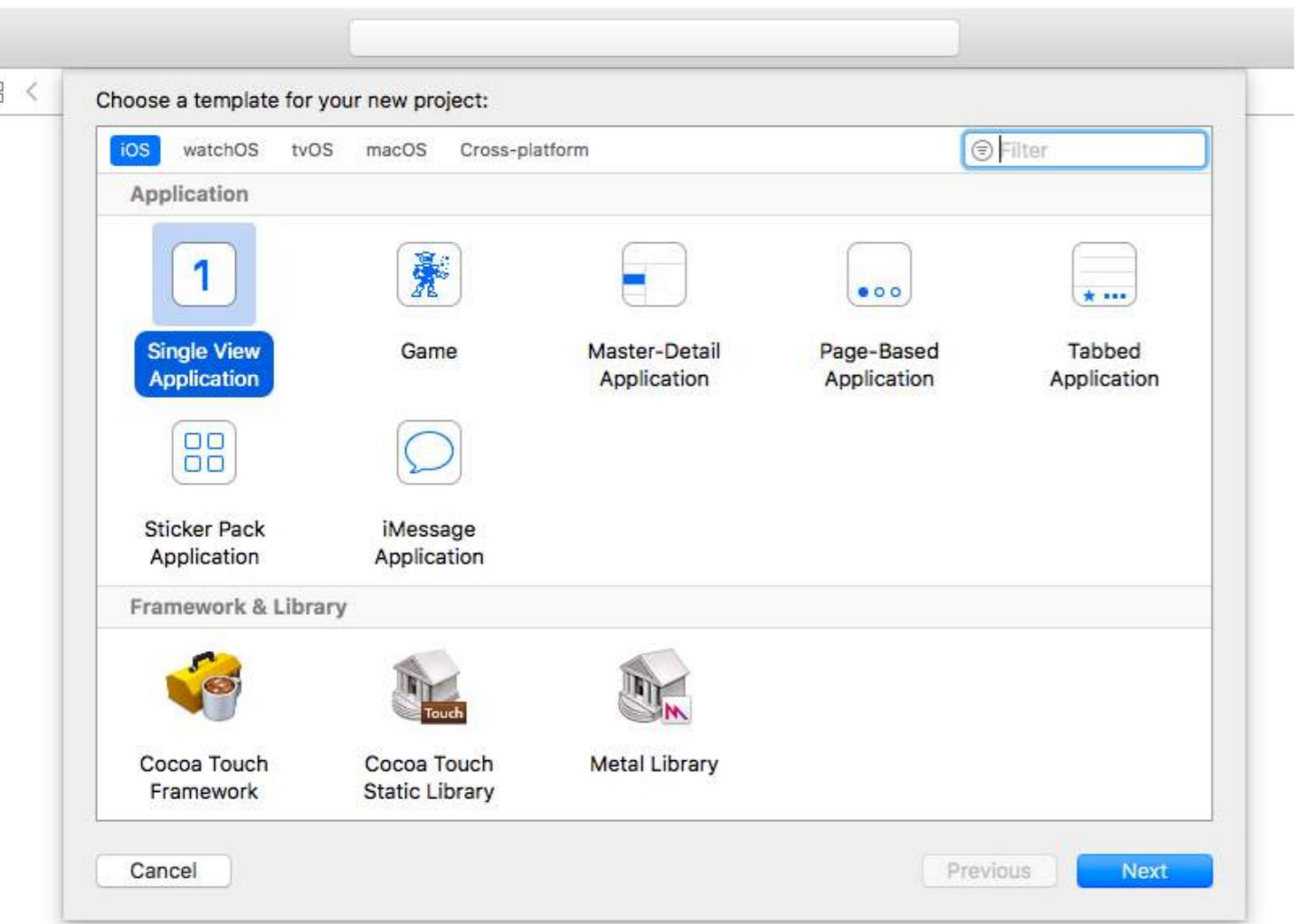
```

- (UIViewController *)childViewControllerAtIndex:(NSInteger)index
{
    if (index <= ([viewControllers count] - 1)) {
        return [viewControllers objectAtIndex:index];
    } else {
        return nil;
    }
}

```

## Section 47.2: A simple way to create horizontal page view controllers ( infinite pages )

1. Let's create a new project, I'm choosing Single View Application for better demonstration

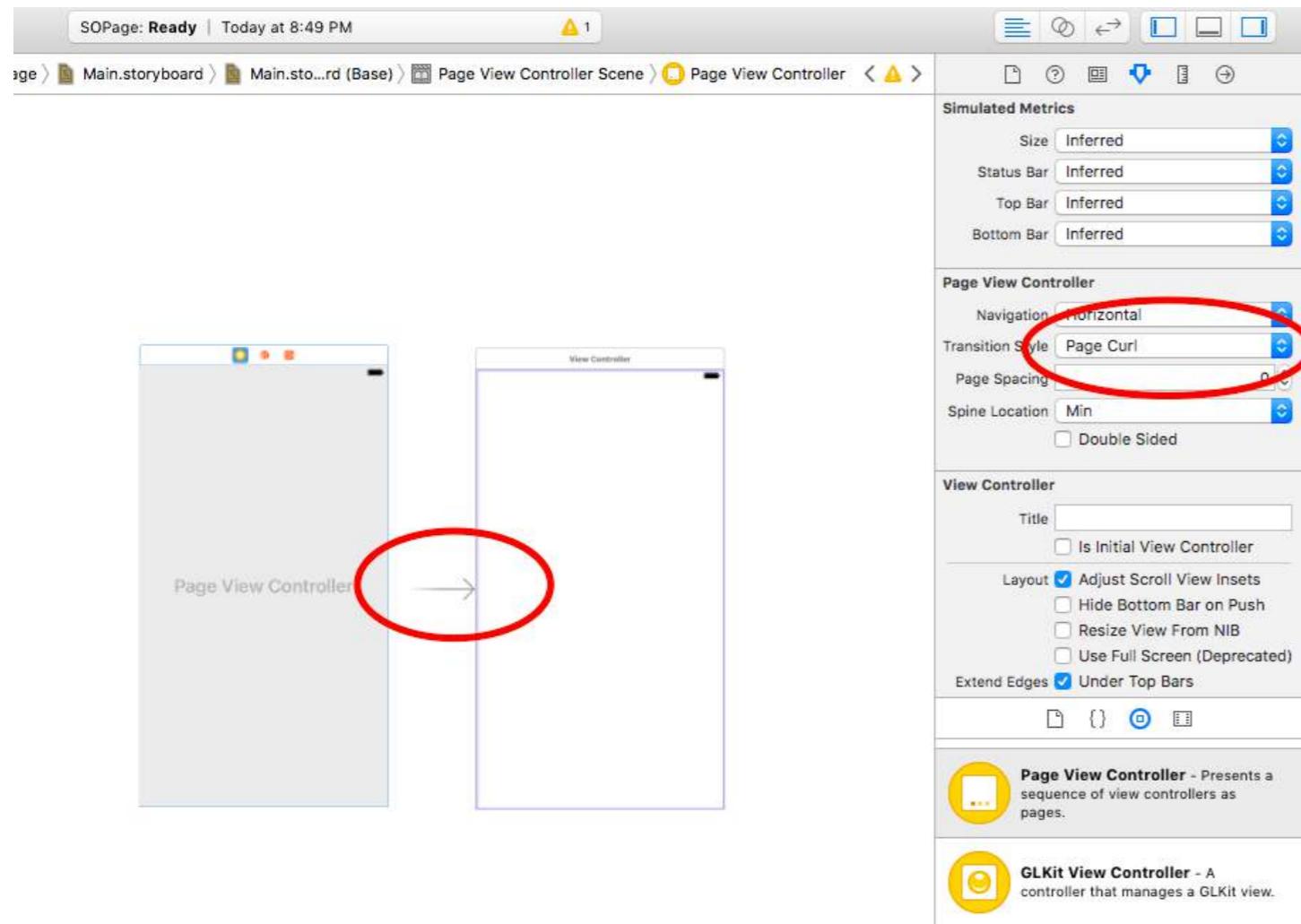


2. 将分页视图控制器拖到故事板上，之后你需要更改两件事：

1. 将分页视图控制器设置为初始视图控制器
2. 将过渡样式更改为滚动

2. Drag a page view controller to the storyboard, there are 2 things you should change after that:

1. Set the page view controller as initial view controller
2. Change the transition style to scroll



3. 你需要创建一个UIPageViewController类，然后将其设置为故事板上分页视图控制器的自定义类

4. 将这段代码粘贴到你的UIPageViewController类中，你将得到一个色彩丰富的无限分页应用程序 :)

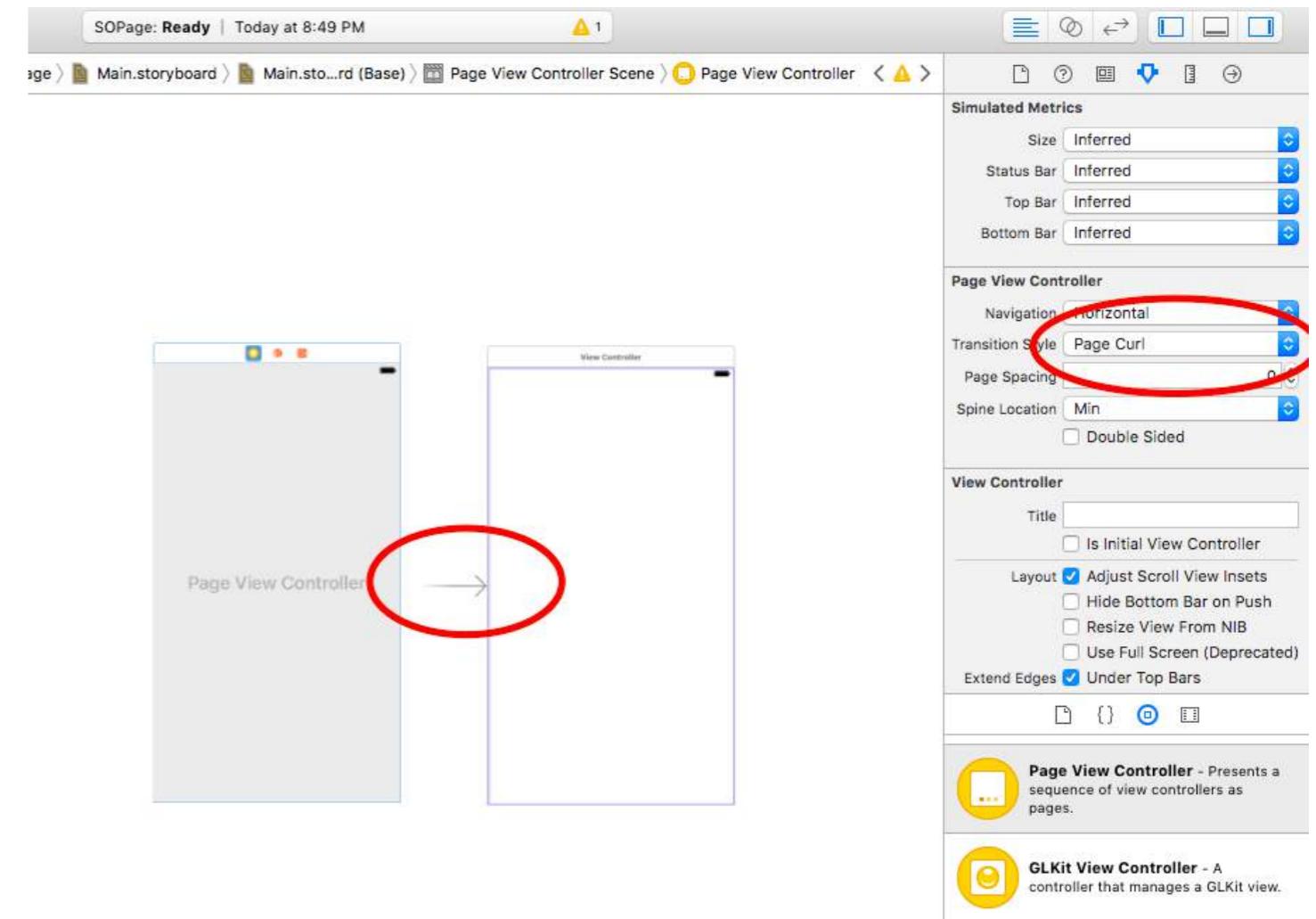
```
class PageViewController: UIPageViewController, UIPageViewControllerDataSource {

    override func viewDidLoad() {
        self.dataSource = self
        let controller = createViewController()
        self.setViewControllers([controller], direction: .forward, animated: false,
completion: nil)
    }

    func pageViewController(_ pageViewController: UIPageViewController, viewControllerBefore viewController: UIViewController) -> UIViewController? {
        let controller = createViewController()
        return controller
    }

    func pageViewController(_ pageViewController: UIPageViewController, viewControllerAfter viewController: UIViewController) -> UIViewController? {
        let controller = createViewController()
        return controller
    }

    func createViewController() -> UIViewController {
        var randomColor: UIColor {
            return UIColor(hue: CGFloat(Float(arc4random_uniform(360))/360, saturation: 0.5,
```



3. And you need to create a UIPageViewController class, then set it as custom class of the page view controller on the storyboard

4. Paste this code into your UIPageViewController class, you should get a colorful infinite paged app :)

```
class PageViewController: UIPageViewController, UIPageViewControllerDataSource {

    override func viewDidLoad() {
        self.dataSource = self
        let controller = createViewController()
        self.setViewControllers([controller], direction: .forward, animated: false,
completion: nil)
    }

    func pageViewController(_ pageViewController: UIPageViewController, viewControllerBefore viewController: UIViewController) -> UIViewController? {
        let controller = createViewController()
        return controller
    }

    func pageViewController(_ pageViewController: UIPageViewController, viewControllerAfter viewController: UIViewController) -> UIViewController? {
        let controller = createViewController()
        return controller
    }

    func createViewController() -> UIViewController {
        var randomColor: UIColor {
            return UIColor(hue: CGFloat(Float(arc4random_uniform(360))/360, saturation: 0.5,
```

```
brightness: 0.8, alpha: 1)
    }
    let storyboard = UIStoryboard(name: "Main", bundle: nil)
    let controller = storyboard.instantiateViewController(withIdentifier: "ViewController")
controller.view.backgroundColor = randomColor
    return controller
}
}
```

这是最终项目的效果，每次滚动时你都会得到一个颜色不同的视图控制器：



```
brightness: 0.8, alpha: 1)
    }
    let storyboard = UIStoryboard(name: "Main", bundle: nil)
    let controller = storyboard.instantiateViewController(withIdentifier: "ViewController")
controller.view.backgroundColor = randomColor
    return controller
}
}
```

This is what the final project looks like, you get a view controller with different color with every scroll:



# 第48章：UISplitViewController

## 第48.1节：使用Objective C中的代理在主视图和详细视图之间交互

UISplitViewController 需要作为你的应用程序'窗口的根视图控制器

### AppDelegate.m

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor blackColor];
    [self.window makeKeyAndVisible];
    self.window.clipsToBounds = YES;
    SplitViewController *spView = [[SplitViewController alloc] init];
    self.window.rootViewController = spView;
    [self.window makeKeyAndVisible];
    return YES;
}
```

只需为你的UISplitViewController创建一个对象，并将其设置为应用程序的rootViewController。

### SplitViewController.h

```
#import <UIKit/UIKit.h>
#import "MasterViewController.h"
#import "DetailViewController.h"
@interface ViewController : UISplitViewController
{
    DetailViewController *detailVC;
    MasterViewController *masterVC;
    NSMutableArray *array;
}
@end
```

MasterViewController 是一个UIViewController，设置在设备的左侧，你可以通过 UISplitViewController 的maximumPrimaryColumnWidth 设置宽度，DetailViewController 位于右侧

### SplitViewController.m

```
#import "ViewController.h"
#define ANIMATION_LENGTH 0.3
@interface ViewController ()
@end

@implementation ViewController
- (void)viewDidLoad
{
    [super viewDidLoad];
    masterVC = [[MasterViewController alloc] init];
    detailVC = [[DetailViewController alloc] init];
    [masterVC setDetailDelegate:(id)detailVC];
    NSArray *vcArray = [NSArray arrayWithObjects:masterVC, detailVC, nil];
    self.preferredDisplayMode = UISplitViewControllerDisplayModeAutomatic;
    self.viewControllers = vcArray;
    self.delegate = (id)self;
}
```

# Chapter 48: UISplitViewController

## Section 48.1: Interacting Between Master and Detail View using Delegates in Objective C

UISplitViewController needs to be the root view controller of your app's window

### AppDelegate.m

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor blackColor];
    [self.window makeKeyAndVisible];
    self.window.clipsToBounds = YES;
    SplitViewController *spView = [[SplitViewController alloc] init];
    self.window.rootViewController = spView;
    [self.window makeKeyAndVisible];
    return YES;
}
```

Just create an object for your UISplitViewController and set it as the rootViewController for your application.

### SplitViewController.h

```
#import <UIKit/UIKit.h>
#import "MasterViewController.h"
#import "DetailViewController.h"
@interface ViewController : UISplitViewController
{
    DetailViewController *detailVC;
    MasterViewController *masterVC;
    NSMutableArray *array;
}
@end
```

MasterViewController is a UIViewController that is set on the left side of the device you can set the width in UISplitViewController using maximumPrimaryColumnWidth and DetailViewController is on the Right side

### SplitViewController.m

```
#import "ViewController.h"
#define ANIMATION_LENGTH 0.3
@interface ViewController ()
@end

@implementation ViewController
- (void)viewDidLoad
{
    [super viewDidLoad];
    masterVC = [[MasterViewController alloc] init];
    detailVC = [[DetailViewController alloc] init];
    [masterVC setDetailDelegate:(id)detailVC];
    NSArray *vcArray = [NSArray arrayWithObjects:masterVC, detailVC, nil];
    self.preferredDisplayMode = UISplitViewControllerDisplayModeAutomatic;
    self.viewControllers = vcArray;
    self.delegate = (id)self;
}
```

```
self.presentsWithGesture = YES;
}
```

主视图和详细视图的UIViewController被添加到一个NSArray中，该数组被设置为self.viewControllers。  
self.preferredDisplayMode是设置MasterViewController和DetailViewController显示模式的属性。self.presentsWithGesture启用滑动手势以显示MasterViewController

### MasterViewController.h

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

@protocol DetailViewDelegate <NSObject>
@required
- (void)sendSelectedNavController:(UIViewController *)viewController;
@end

@interface MasterViewController : UIViewController
{
    UITableView *mainTableView;
    NSMutableArray *viewControllerArray;
}
@property (nonatomic, retain) id<DetailViewDelegate> detailDelegate;
@end
```

创建一个带有sendSelectedNavController方法的DetailViewDelegate代理，用于将UIViewController发送到DetailViewController。然后在MasterViewController中创建一个UITableView。ViewControllerArray包含所有需要在DetailViewController中显示的UIViewControllers

### MasterViewController.m

```
#import "MasterViewController.h"

@implementation MasterViewController
@synthesize detailDelegate;

-(void)viewDidLoad
{
[super viewDidLoad];

UIViewController *dashBoardVC = [[UIViewController alloc] init];
[dashBoardVC.view setBackgroundColor:[UIColor redColor]];
UIViewController *inventVC = [[UIViewController alloc] init];
[inventVC.view setBackgroundColor:[UIColor whiteColor]];
UIViewController *alarmVC = [[UIViewController alloc] init];
[alarmVC.view setBackgroundColor: [UIColor purpleColor]];
UIViewController *scanDeviceVC = [[UIViewController alloc] init];
[scanDeviceVC.view setBackgroundColor:[UIColor cyanColor]];
UIViewController *serverDetailVC = [[UIViewController alloc] init];
[serverDetailVC.view setBackgroundColor: [UIColor whiteColor]];
viewControllerArray = [[NSMutableArray
alloc] initWithObjects:dashBoardVC,inventVC,alarmVC,scanDeviceVC,serverDetailVC,nil];
mainTableView = [[UITableView alloc] initWithFrame:CGRectMake(0, 50,self.view.frame.size.width,
self.view.frame.size.height-50) style:UITableViewStylePlain];
[mainTableView setDelegate:(id)self];
[mainTableView setDataSource:(id)self];
[mainTableView setSeparatorStyle:UITableViewCellSeparatorStyleNone];
[mainTableView setScrollsToTop:NO];
[self.view addSubview:mainTableView];
}
```

```
self.presentsWithGesture = YES;
}
```

The master and detail UIViewController's are added to an NSArray which is set to self.viewControllers.  
self.preferredDisplayMode is the mode set for displaying of MasterViewController and DetailViewController  
. self.presentsWithGesture enables swipe gesture for displaying MasterViewController

### MasterViewController.h

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

@protocol DetailViewDelegate <NSObject>
@required
- (void)sendSelectedNavController:(UIViewController *)viewController;
@end

@interface MasterViewController : UIViewController
{
    UITableView *mainTableView;
    NSMutableArray *viewControllerArray;
}
@property (nonatomic, retain) id<DetailViewDelegate> detailDelegate;
@end
```

Create a DetailViewDelegate Delegate with sendSelectedNavController method for sending the UIViewController's to the DetailViewController. Then in MasterViewController an UITableView is created . The ViewControllerArray contains all the UIViewControllers that needs to be displayed in DetailViewController

### MasterViewController.m

```
#import "MasterViewController.h"

@implementation MasterViewController
@synthesize detailDelegate;

-(void)viewDidLoad
{
[super viewDidLoad];

UIViewController *dashBoardVC = [[UIViewController alloc] init];
[dashBoardVC.view setBackgroundColor:[UIColor redColor]];
UIViewController *inventVC = [[UIViewController alloc] init];
[inventVC.view setBackgroundColor:[UIColor whiteColor]];
UIViewController *alarmVC = [[UIViewController alloc] init];
[alarmVC.view setBackgroundColor: [UIColor purpleColor]];
UIViewController *scanDeviceVC = [[UIViewController alloc] init];
[scanDeviceVC.view setBackgroundColor:[UIColor cyanColor]];
UIViewController *serverDetailVC = [[UIViewController alloc] init];
[serverDetailVC.view setBackgroundColor: [UIColor whiteColor]];
viewControllerArray = [[NSMutableArray
alloc] initWithObjects:dashBoardVC,inventVC,alarmVC,scanDeviceVC,serverDetailVC,nil];
mainTableView = [[UITableView alloc] initWithFrame:CGRectMake(0, 50,self.view.frame.size.width,
self.view.frame.size.height-50) style:UITableViewStylePlain];
[mainTableView setDelegate:(id)self];
[mainTableView setDataSource:(id)self];
[mainTableView setSeparatorStyle:UITableViewCellSeparatorStyleNone];
[mainTableView setScrollsToTop:NO];
[self.view addSubview:mainTableView];
}
```

```

- (CGFloat)tableView:(UITableView *)tableView
heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
    return 100;
}
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    return [viewControllerArray count];
}

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1; //节的数量
}

- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSString *cellId = [NSString
stringWithFormat:@"Cell%li%ld",(long)indexPath.section,(long)indexPath.row];
UITableViewCell *cell =[tableView dequeueReusableCellWithIdentifier:cellId];

if (cell == nil)
{
    cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:cellId];
}
    contentView setBackgroundColor:[UIColor redColor]];
cell.textLabel.text =[NSString stringWithFormat:@"我的视图控制器索引 %ld",(long)indexPath.row];
return cell;
}

- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    [detailDelegate sendSelectedNavController:[viewControllerArray objectAtIndex:indexPath.row]];
}

@end

```

创建了一些UIViewController并将其添加到一个NSMutableArray中。然后初始化UITableView，在didselectrowatindexpath方法中，我使用detailDelegate代理将对应的NSMutableArray中的UIViewController作为参数发送给DetailViewController

### DetailViewController.h

```

#import <UIKit/UIKit.h>

@interface DetailViewController : UIViewController<UICollectionViewDelegate>
{
    UIViewController *tempNav;
}
@end

```

### DetailViewController.m

```

#import "DetailViewController.h"

@implementation DetailViewController
-(void)viewDidLoad

```

```

- (CGFloat)tableView:(UITableView *)tableView
heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
    return 100;
}
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    return [viewControllerArray count];
}

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1; //count of section
}

- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSString *cellId = [NSString
stringWithFormat:@"Cell%li%ld",(long)indexPath.section,(long)indexPath.row];
UITableViewCell *cell =[tableView dequeueReusableCellWithIdentifier:cellId];

if (cell == nil)
{
    cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:cellId];
}
    [cell.contentView setBackgroundColor:[UIColor redColor]];
cell.textLabel.text =[NSString stringWithFormat:@"My VC at index %ld",(long)indexPath.row];
return cell;
}

- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    [detailDelegate sendSelectedNavController:[viewControllerArray objectAtIndex:indexPath.row]];
}

@end

```

Created some UIViewController and added it to an NSMutableArray. The UITableView is initialized then on didselectrowatindexpath method I send a UIViewController to the DetailViewController using detailDelegate delegate with the corresponding UIViewController in the NSMutableArray as a parameter

### DetailViewController.h

```

#import <UIKit/UIKit.h>

@interface DetailViewController : UIViewController<UICollectionViewDelegate>
{
    UIViewController *tempNav;
}
@end

```

### DetailViewController.m

```

#import "DetailViewController.h"

@implementation DetailViewController
-(void)viewDidLoad

```

```

{
    [super viewDidLoad];
    [self.view setBackgroundColor:[UIColor whiteColor]];
}
-(void)sendSelectedNavController:(UIViewController *)navController
{
    NSArray *viewsToRemove = [self.view subviews];
    for (UIView *v in viewsToRemove) {
        [v removeFromSuperview];
    }
    tempNav = navController;
    [self.view 添加子视图:tempNav.view];
}
@end

```

这里声明了sendSelectedNavController方法，方法中移除DetailViewController中的所有UIView，并添加从MasterViewController传递过来的UIViewController。

```

{
    [super viewDidLoad];
    [self.view setBackgroundColor:[UIColor whiteColor]];
}
-(void)sendSelectedNavController:(UIViewController *)navController
{
    NSArray *viewsToRemove = [self.view subviews];
    for (UIView *v in viewsToRemove) {
        [v removeFromSuperview];
    }
    tempNav = navController;
    [self.view addSubview:tempNav.view];
}
@end

```

The sendSelectedNavController is declared here with removing all the `UIView`'s in the DetailViewController and adding the passed `UIViewController` from the MasterViewController.

# 第49章：UIFont

`UIFont` 是一个用于获取和设置字体相关信息的类。它继承自 `NSObject` 并且符合 `Hashable`、`Equatable`、`CVarArg` 和 `NSCopying` 协议。

## 第49.1节：声明和初始化 UIFont

你可以按如下方式声明一个 `UIFont`：

```
var font: UIFont!
```

`UIFont` 有更多的 `init()` 方法：

- `UIFont.init(descriptor: UIFontDescriptor, size: CGFloat)`
- `UIFont.init(name: String, size: CGFloat)`

因此，你可以这样初始化一个 `UIFont`：

```
let font = UIFont(name: "Helvetica Neue", size: 15)
```

默认字体是 `System`，大小为 17。

## 第49.2节：更改标签的字体

要更改标签的文本字体，您需要访问其 `font` 属性：

```
label.font = UIFont(name:"Helvetica Neue", size: 15)
```

上面的代码将标签的字体更改为 `Helvetica Neue`，大小为 15。请注意，字体名称必须拼写正确，否则会抛出以下错误，因为上面初始化的值是一个 `Optional`，因此可能为 `nil`：

意外地在解包 `Optional` 值时发现了 `nil`

# Chapter 49: UIFont

`UIFont` is a class that is used for getting and setting font-related information. It inherits from `NSObject` and conforms to `Hashable`, `Equatable`, `CVarArg` and `NSCopying`.

## Section 49.1: Declaring and initializing UIFont

You can declare a `UIFont` as follows:

```
var font: UIFont!
```

`UIFont` has more `init()` methods:

- `UIFont.init(descriptor: UIFontDescriptor, size: CGFloat)`
- `UIFont.init(name: String, size: CGFloat)`

Therefore, you can initialize a `UIFont` like this:

```
let font = UIFont(name: "Helvetica Neue", size: 15)
```

The default font is `System`, size 17.

## Section 49.2: Changing the font of a label

To change a label's text font, you need to access its `font` property:

```
label.font = UIFont(name:"Helvetica Neue", size: 15)
```

The code above will change the font of the label to `Helvetica Neue`, size 15. Beware that you must spell the font name correctly, otherwise it will throw this error, because the initialized value above is an `Optional`, and thus can be `nil`:

Unexpectedly found nil while unwrapping an Optional value

# 第50章：UIDevice

属性	描述
name	标识设备的名称。
systemName: String	运行在接收器所代表设备上的操作系统名称。
model: String	设备的型号。
systemVersion: String	当前操作系统的版本。

## 第50.1节：获取iOS设备型号名称

Swift 2

```
import UIKit

extension UIDevice {

    var modelName: String {
        var systemInfo = utsname()
        uname(&systemInfo)
        let machineMirror = Mirror(reflecting: systemInfo.machine)
        let identifier = machineMirror.children.reduce("") { identifier, element in
            guard let value = element.value as? Int8 where value != 0 else { return identifier }
            return identifier + String(UnicodeScalar(UInt8(value)))
        }

        switch identifier {
        case "iPod5,1": return "iPod Touch 5"
        case "iPod7,1": return "iPod Touch 6"
        case "iPhone3,1", "iPhone3,2", "iPhone3,3": return "iPhone 4"
        case "iPhone4,1": return "iPhone 4s"
        case "iPhone5,1", "iPhone5,2": return "iPhone 5"
        case "iPhone5,3", "iPhone5,4": return "iPhone 5c"
        case "iPhone6,1", "iPhone6,2": return "iPhone 5s"
        case "iPhone7,2": return "iPhone 6"
        case "iPhone7,1": return "iPhone 6 Plus"
        case "iPhone8,1": return "iPhone 6s"
        case "iPhone8,2": return "iPhone 6s Plus"
        case "iPhone9,1", "iPhone9,3": return "iPhone 7"
        case "iPhone9,2", "iPhone9,4": return "iPhone 7 Plus"
        case "iPhone8,4": return "iPhone SE"
        case "iPad2,1", "iPad2,2", "iPad2,3", "iPad2,4":return "iPad 2"
        case "iPad3,1", "iPad3,2", "iPad3,3": return "iPad 3"
        case "iPad3,4", "iPad3,5", "iPad3,6": return "iPad 4"
        case "iPad4,1", "iPad4,2", "iPad4,3": return "iPad Air"
        case "iPad5,3", "iPad5,4": return "iPad Air 2"
        case "iPad2,5", "iPad2,6", "iPad2,7": return "iPad Mini"
        case "iPad4,4", "iPad4,5", "iPad4,6": return "iPad Mini 2"
        case "iPad4,7", "iPad4,8", "iPad4,9": return "iPad Mini 3"
        case "iPad5,1", "iPad5,2": return "iPad Mini 4"
        case "iPad6,3", "iPad6,4", "iPad6,7", "iPad6,8":return "iPad Pro"
        case "AppleTV5,3": return "Apple TV"
        case "i386", "x86_64": return "Simulator"
        default:返回 标识符
    }
}

if UIDevice.currentDevice().modelName == "iPhone 6 Plus" {
```

# Chapter 50: UIDevice

Property	Description
name	The name identifying the device.
systemName: String	The name of the operating system running on the device represented by the receiver.
model: String	The model of the device.
systemVersion: String	The current version of the operating system..

## Section 50.1: Get iOS device model name

Swift 2

```
import UIKit

extension UIDevice {

    var modelName: String {
        var systemInfo = utsname()
        uname(&systemInfo)
        let machineMirror = Mirror(reflecting: systemInfo.machine)
        let identifier = machineMirror.children.reduce("") { identifier, element in
            guard let value = element.value as? Int8 where value != 0 else { return identifier }
            return identifier + String(UnicodeScalar(UInt8(value)))
        }

        switch identifier {
        case "iPod5,1": return "iPod Touch 5"
        case "iPod7,1": return "iPod Touch 6"
        case "iPhone3,1", "iPhone3,2", "iPhone3,3": return "iPhone 4"
        case "iPhone4,1": return "iPhone 4s"
        case "iPhone5,1", "iPhone5,2": return "iPhone 5"
        case "iPhone5,3", "iPhone5,4": return "iPhone 5c"
        case "iPhone6,1", "iPhone6,2": return "iPhone 5s"
        case "iPhone7,2": return "iPhone 6"
        case "iPhone7,1": return "iPhone 6 Plus"
        case "iPhone8,1": return "iPhone 6s"
        case "iPhone8,2": return "iPhone 6s Plus"
        case "iPhone9,1", "iPhone9,3": return "iPhone 7"
        case "iPhone9,2", "iPhone9,4": return "iPhone 7 Plus"
        case "iPhone8,4": return "iPhone SE"
        case "iPad2,1", "iPad2,2", "iPad2,3", "iPad2,4":return "iPad 2"
        case "iPad3,1", "iPad3,2", "iPad3,3": return "iPad 3"
        case "iPad3,4", "iPad3,5", "iPad3,6": return "iPad 4"
        case "iPad4,1", "iPad4,2", "iPad4,3": return "iPad Air"
        case "iPad5,3", "iPad5,4": return "iPad Air 2"
        case "iPad2,5", "iPad2,6", "iPad2,7": return "iPad Mini"
        case "iPad4,4", "iPad4,5", "iPad4,6": return "iPad Mini 2"
        case "iPad4,7", "iPad4,8", "iPad4,9": return "iPad Mini 3"
        case "iPad5,1", "iPad5,2": return "iPad Mini 4"
        case "iPad6,3", "iPad6,4", "iPad6,7", "iPad6,8":return "iPad Pro"
        case "AppleTV5,3": return "Apple TV"
        case "i386", "x86_64": return "Simulator"
        default: return identifier
    }
}

if UIDevice.currentDevice().modelName == "iPhone 6 Plus" {
```

```
// 是 iPhone 6 Plus
```

```
}
```

### Swift 3

```
import UIKit

public extension UIDevice {

    var modelName: String {
        var systemInfo = utsname()
        uname(&systemInfo)
        let machineMirror = Mirror(reflecting: systemInfo.machine)
        let identifier = machineMirror.children.reduce("") { identifier, element in
            guard let value = element.value as? Int8, value != 0 else { return identifier }
            return identifier + String(UnicodeScalar(UInt8(value)))
        }

        switch identifier {
        case "iPod5,1": return "iPod Touch 5"
        case "iPod7,1": return "iPod Touch 6"
        case "iPhone3,1", "iPhone3,2", "iPhone3,3": return "iPhone 4"
        case "iPhone4,1": return "iPhone 4s"
        case "iPhone5,1", "iPhone5,2": return "iPhone 5"
        case "iPhone5,3", "iPhone5,4": return "iPhone 5c"
        case "iPhone6,1", "iPhone6,2": return "iPhone 5s"
        case "iPhone7,2": return "iPhone 6"
        case "iPhone7,1": return "iPhone 6 Plus"
        case "iPhone8,1": return "iPhone 6s"
        case "iPhone8,2": return "iPhone 6s Plus"
        case "iPhone9,1", "iPhone9,3": return "iPhone 7"
        case "iPhone9,2", "iPhone9,4": return "iPhone 7 Plus"
        case "iPhone8,4": return "iPhone SE"
        case "iPad2,1", "iPad2,2", "iPad2,3", "iPad2,4": return "iPad 2"
        case "iPad3,1", "iPad3,2", "iPad3,3": return "iPad 3"
        case "iPad3,4", "iPad3,5", "iPad3,6": return "iPad 4"
        case "iPad4,1", "iPad4,2", "iPad4,3": return "iPad Air"
        case "iPad5,3", "iPad5,4": return "iPad Air 2"
        case "iPad2,5", "iPad2,6", "iPad2,7": return "iPad Mini"
        case "iPad4,4", "iPad4,5", "iPad4,6": return "iPad Mini 2"
        case "iPad4,7", "iPad4,8", "iPad4,9": return "iPad Mini 3"
        case "iPad5,1", "iPad5,2": return "iPad Mini 4"
        case "iPad6,3", "iPad6,4", "iPad6,7", "iPad6,8": return "iPad Pro"
        case "AppleTV5,3": return "Apple TV"
        case "i386", "x86_64": return "Simulator"
        default: return "Unknown"
    }
}

if UIDevice.current.modelName == "iPhone 7" {
    // 是 iPhone 7
}
```

```
// 是 an iPhone 6 Plus
```

```
}
```

### Swift 3

```
import UIKit

public extension UIDevice {

    var modelName: String {
        var systemInfo = utsname()
        uname(&systemInfo)
        let machineMirror = Mirror(reflecting: systemInfo.machine)
        let identifier = machineMirror.children.reduce("") { identifier, element in
            guard let value = element.value as? Int8, value != 0 else { return identifier }
            return identifier + String(UnicodeScalar(UInt8(value)))
        }

        switch identifier {
        case "iPod5,1": return "iPod Touch 5"
        case "iPod7,1": return "iPod Touch 6"
        case "iPhone3,1", "iPhone3,2", "iPhone3,3": return "iPhone 4"
        case "iPhone4,1": return "iPhone 4s"
        case "iPhone5,1", "iPhone5,2": return "iPhone 5"
        case "iPhone5,3", "iPhone5,4": return "iPhone 5c"
        case "iPhone6,1", "iPhone6,2": return "iPhone 5s"
        case "iPhone7,2": return "iPhone 6"
        case "iPhone7,1": return "iPhone 6 Plus"
        case "iPhone8,1": return "iPhone 6s"
        case "iPhone8,2": return "iPhone 6s Plus"
        case "iPhone9,1", "iPhone9,3": return "iPhone 7"
        case "iPhone9,2", "iPhone9,4": return "iPhone 7 Plus"
        case "iPhone8,4": return "iPhone SE"
        case "iPad2,1", "iPad2,2", "iPad2,3", "iPad2,4": return "iPad 2"
        case "iPad3,1", "iPad3,2", "iPad3,3": return "iPad 3"
        case "iPad3,4", "iPad3,5", "iPad3,6": return "iPad 4"
        case "iPad4,1", "iPad4,2", "iPad4,3": return "iPad Air"
        case "iPad5,3", "iPad5,4": return "iPad Air 2"
        case "iPad2,5", "iPad2,6", "iPad2,7": return "iPad Mini"
        case "iPad4,4", "iPad4,5", "iPad4,6": return "iPad Mini 2"
        case "iPad4,7", "iPad4,8", "iPad4,9": return "iPad Mini 3"
        case "iPad5,1", "iPad5,2": return "iPad Mini 4"
        case "iPad6,3", "iPad6,4", "iPad6,7", "iPad6,8": return "iPad Pro"
        case "AppleTV5,3": return "Apple TV"
        case "i386", "x86_64": return "Simulator"
        default: return identifier
    }
}

if UIDevice.current.modelName == "iPhone 7" {
    // is an iPhone 7
}
```

## 第 50.2 节：识别设备和操作系统

```
UIDevice *deviceInfo = [UIDevice currentDevice];
NSLog(@"设备名称 %@", deviceInfo.name);
//例如：myIphone6s
NSLog(@"系统名称 %@", deviceInfo.systemName);
```

## Section 50.2: Identifying the Device and Operating

```
UIDevice *deviceInfo = [UIDevice currentDevice];
NSLog(@"Device Name %@", deviceInfo.name);
//Ex: myIphone6s
NSLog(@"System Name %@", deviceInfo.systemName);
```

```

//设备名称 iPhone OS
NSLog(@"系统版本 %@", deviceInfo.systemVersion);
//系统版本 9.3
NSLog(@"型号 %@", deviceInfo.model);
//型号 iPhone
NSLog(@"本地化型号 %@", deviceInfo.localizedModel);
//本地化型号 iPhone
int device=deviceInfo.userInterfaceIdiom;
//UIUserInterfaceIdiomPhone=0
//UIUserInterfaceIdiomPad=1
//UIUserInterfaceIdiomTV=2
//UIUserInterfaceIdiomCarPlay=3
//UIUserInterfaceIdiomUnspecified=-1
NSLog(@"供应商标识符 %@", deviceInfo.identifierForVendor);
//identifierForVendor <__NSConcreteUUID 0x7a10ae20> 556395DC-0EB4-4FD5-BC7E-B16F612ECC6D

```

## 第50.3节：获取设备方向

```

UIDevice *deviceInfo = [UIDevice currentDevice];
int d = deviceInfo.orientation;

```

deviceInfo.orientation 返回一个 UIDeviceOrientation 值，具体如下所示：

```

UIDeviceOrientationUnknown 0
UIDeviceOrientationPortrait 1
UIDeviceOrientationPortraitUpsideDown 2
UIDeviceOrientationLandscapeLeft 3
UIDeviceOrientationLandscapeRight 4
UIDeviceOrientationFaceUp 5
UIDeviceOrientationFaceDown 6

```

在视图控制器中监听设备方向变化：

```

- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];
    [[UIDevice currentDevice] beginGeneratingDeviceOrientationNotifications];
    [[NSNotificationCenter defaultCenter] addObserver:self
    selector:@selector(deviceOrientationDidChange)
        name:UIDeviceOrientationDidChangeNotification
        object:nil];
}

-(void)deviceOrientationDidChange
{
    UIDeviceOrientation orientation = [[UIDevice currentDevice] orientation];
    if (orientation == UIDeviceOrientationPortrait || orientation ==
    UIDeviceOrientationPortraitUpsideDown) {
        [self changedToPortrait];
    } else if (orientation == UIDeviceOrientationLandscapeLeft || orientation ==
    UIDeviceOrientationLandscapeRight) {
        [self changedToLandscape];
    }
}

-(void)changedToPortrait
{
    // 函数体
}

-(void)changedToLandscape

```

```

//Device Name iPhone OS
NSLog(@"System Version %@", deviceInfo.systemVersion);
//System Version 9.3
NSLog(@"Model %@", deviceInfo.model);
//Model iPhone
NSLog(@"Localized Model %@", deviceInfo.localizedModel);
//Localized Model iPhone
int device=deviceInfo.userInterfaceIdiom;
//UIUserInterfaceIdiomPhone=0
//UIUserInterfaceIdiomPad=1
//UIUserInterfaceIdiomTV=2
//UIUserInterfaceIdiomCarPlay=3
//UIUserInterfaceIdiomUnspecified=-1
NSLog(@"identifierForVendor %@", deviceInfo.identifierForVendor);
//identifierForVendor <__NSConcreteUUID 0x7a10ae20> 556395DC-0EB4-4FD5-BC7E-B16F612ECC6D

```

## Section 50.3: Getting the Device Orientation

```

UIDevice *deviceInfo = [UIDevice currentDevice];
int d = deviceInfo.orientation;

```

deviceInfo.orientation returns an UIDeviceOrientation value which is shown as below:

```

UIDeviceOrientationUnknown 0
UIDeviceOrientationPortrait 1
UIDeviceOrientationPortraitUpsideDown 2
UIDeviceOrientationLandscapeLeft 3
UIDeviceOrientationLandscapeRight 4
UIDeviceOrientationFaceUp 5
UIDeviceOrientationFaceDown 6

```

Listening for device orientation changes in a View Controller:

```

- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];
    [[UIDevice currentDevice] beginGeneratingDeviceOrientationNotifications];
    [[NSNotificationCenter defaultCenter] addObserver:self
    selector:@selector(deviceOrientationDidChange)
        name:UIDeviceOrientationDidChangeNotification
        object:nil];
}

-(void)deviceOrientationDidChange
{
    UIDeviceOrientation orientation = [[UIDevice currentDevice] orientation];
    if (orientation == UIDeviceOrientationPortrait || orientation ==
    UIDeviceOrientationPortraitUpsideDown) {
        [self changedToPortrait];
    } else if (orientation == UIDeviceOrientationLandscapeLeft || orientation ==
    UIDeviceOrientationLandscapeRight) {
        [self changedToLandscape];
    }
}

-(void)changedToPortrait
{
    // Function Body
}

-(void)changedToLandscape

```

```
{  
    // 函数体  
}
```

禁用任何方向变化检测：

```
- (void)viewWillDisappear:(BOOL)animated {  
    [super viewWillDisappear:animated];  
    [[UIDevice currentDevice] endGeneratingDeviceOrientationNotifications];  
}
```

## 第50.4节：获取设备电池状态

```
// 获取电池监测权限  
[[UIDevice currentDevice] setBatteryMonitoringEnabled:YES];  
UIDevice *myDevice = [UIDevice currentDevice];  
  
[myDevice setBatteryMonitoringEnabled:YES];  
double batLeft = (float)[myDevice batteryLevel] * 100;  
NSLog(@"%@", batLeft);  
  
int d = myDevice.batteryState;  
// 返回一个整数值  
// UIDeviceBatteryStateUnknown 0  
// UIDeviceBatteryStateUnplugged 1  
// UIDeviceBatteryStateCharging 2  
// UIDeviceBatteryStateFull 3  
  
// 使用通知进行电池监测  
-(void)startMonitoringForBatteryChanges  
{  
    // 启用电池状态监测  
    [[UIDevice currentDevice] setBatteryMonitoringEnabled:YES];  
    // 请求在电池电量或状态变化时接收通知  
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(checkBatteryStatus)  
        name:UIDeviceBatteryLevelDidChangeNotification object:nil];  
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(checkBatteryStatus)  
        name:UIDeviceBatteryStateDidChangeNotification object:nil];  
    ...  
}  
  
NSLog(@"Battery Level is %.f", [[UIDevice currentDevice] batteryLevel]*100);  
int d=[[UIDevice currentDevice] batteryState];  
if (d==0)  
{  
    NSLog(@"Unknown");  
}  
else if (d==1)  
{  
    NSLog(@"未连接");  
}  
else if (d==2)  
{  
    NSLog(@"充电中");  
}  
else if (d==3)  
{  
    NSLog(@"电池已满");  
}
```

```
{  
    // Function Body  
}
```

To disable checking for any orientation change:

```
- (void)viewWillDisappear:(BOOL)animated {  
    [super viewWillDisappear:animated];  
    [[UIDevice currentDevice] endGeneratingDeviceOrientationNotifications];  
}
```

## Section 50.4: Getting the Device Battery State

```
// Get permission for Battery Monitoring  
[[UIDevice currentDevice] setBatteryMonitoringEnabled:YES];  
UIDevice *myDevice = [UIDevice currentDevice];  
  
[myDevice setBatteryMonitoringEnabled:YES];  
double batLeft = (float)[myDevice batteryLevel] * 100;  
NSLog(@"%@", batLeft);  
  
int d = myDevice.batteryState;  
// Returns an Integer Value  
// UIDeviceBatteryStateUnknown 0  
// UIDeviceBatteryStateUnplugged 1  
// UIDeviceBatteryStateCharging 2  
// UIDeviceBatteryStateFull 3  
  
// Using notifications for Battery Monitoring  
-(void)startMonitoringForBatteryChanges  
{  
    // Enable monitoring of battery status  
    [[UIDevice currentDevice] setBatteryMonitoringEnabled:YES];  
    // Request to be notified when battery charge or state changes  
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(checkBatteryStatus)  
        name:UIDeviceBatteryLevelDidChangeNotification object:nil];  
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(checkBatteryStatus)  
        name:UIDeviceBatteryStateDidChangeNotification object:nil];  
}  
-(void) checkBatteryStatus  
{  
    NSLog(@"Battery Level is %.f", [[UIDevice currentDevice] batteryLevel]*100);  
    int d=[[UIDevice currentDevice] batteryState];  
    if (d==0)  
    {  
        NSLog(@"Unknown");  
    }  
    else if (d==1)  
    {  
        NSLog(@"Unplugged");  
    }  
    else if (d==2)  
    {  
        NSLog(@"Charging");  
    }  
    else if (d==3)  
    {  
        NSLog(@"Battery Full");  
    }  
}
```

## 第50.5节：使用接近传感器

```
//启用接近传感器
- (void)viewWillAppear:(BOOL)animated {
    [super viewWillAppear:animated];
    [[UIDevice currentDevice] setProximityMonitoringEnabled:YES];
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(sensorStateMonitor:) name:@"UIDeviceProximityStateDidChangeNotification" object:nil];
}

- (void)sensorStateMonitor:(NSNotification *)notification {
    if ([[UIDevice currentDevice] proximityState] == YES)
    {
        NSLog(@"设备靠近用户。");
    }
    else
    {
        NSLog(@"设备未靠近用户。");
    }
}
```

## Section 50.5: Using the Proximity Sensor

```
//Enabling the proximity Sensor
- (void)viewWillAppear:(BOOL)animated {
    [super viewWillAppear:animated];
    [[UIDevice currentDevice] setProximityMonitoringEnabled:YES];
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(sensorStateMonitor:) name:@"UIDeviceProximityStateDidChangeNotification" object:nil];
}

- (void)sensorStateMonitor:(NSNotification *)notification {
    if ([[UIDevice currentDevice] proximityState] == YES)
    {
        NSLog(@"Device is close to user.");
    }
    else
    {
        NSLog(@"Device is not closer to user.");
    }
}
```

## 第50.6节：获取电池状态和电池电量

```
override func viewDidLoad() {
    super.viewDidLoad()
    NotificationCenter.default.addObserver(self, selector: Selector(("batteryStateDidChange:")),
name: NSNotification.Name.UIDeviceBatteryStateDidChange, object: nil)
    NotificationCenter.default.addObserver(self, selector: Selector(("batteryLevelDidChange:")),
name: NSNotification.Name.UIDeviceBatteryLevelDidChange, object: nil)

    // 相关内容...
}

func batteryStateDidChange(notification: NSNotification){
    // 状态已改变：已插入电源、未插入电源、充满电...
}

func batteryLevelDidChange(notification: NSNotification){

    let batteryLevel = UIDevice.current.batteryLevel
    if batteryLevel < 0.0 {
        print("-1.0 表示电池状态为 UIDeviceBatteryStateUnknown")
        return
    }

    print("电池电量 : \(batteryLevel * 100)%")
    // 电池电量已变化 (98%, 99%, ...)
}
```

## Section 50.6: Getting Battery Status and Battery Level

```
override func viewDidLoad() {
    super.viewDidLoad()
    NotificationCenter.default.addObserver(self, selector: Selector(("batteryStateDidChange:")),
name: NSNotification.Name.UIDeviceBatteryStateDidChange, object: nil)
    NotificationCenter.default.addObserver(self, selector: Selector(("batteryLevelDidChange:")),
name: NSNotification.Name.UIDeviceBatteryLevelDidChange, object: nil)

    // Stuff...
}

func batteryStateDidChange(notification: NSNotification){
    // The stage did change: plugged, unplugged, full charge...
}

func batteryLevelDidChange(notification: NSNotification){

    let batteryLevel = UIDevice.current.batteryLevel
    if batteryLevel < 0.0 {
        print(" -1.0 means battery state is UIDeviceBatteryStateUnknown")
        return
    }

    print("Battery Level : \(batteryLevel * 100)%")
    // The battery's level did change (98%, 99%, ...)
}
```

# 第51章：使UIView的选定角变为圆角

## 第51.1节：Objective C代码实现UIView选定角的圆角处理

首先导入#import <QuartzCore/QuartzCore.h>到你的ViewController类中。以下是在代码中设置视图的方法

```
UIView *view1=[[UIView alloc] init];
view1.backgroundColor=[UIColor colorWithRed:255/255.0 green:193/255.0 blue:72/255.0 alpha:1.0];
CGRect view1Frame = view1.frame;
view1Frame.size.width = SCREEN_WIDTH*0.97;
view1Frame.size.height = SCREEN_HEIGHT*0.2158;
view1Frame.origin.x = 0;
view1Frame.origin.y = 0.1422*SCREEN_HEIGHT-10;
view1.frame = view1Frame;
[self setMaskTo:view1 byRoundingCorners:UIRectCornerBottomRight|UIRectCornerTopRight];
[self.view addSubview:view1];
```

这是执行主要操作并对选定边缘进行圆角处理的函数，在我们的例子中是右下角和右上角边缘

```
- (void)setMaskTo:(UIView*)view byRoundingCorners:(UIRectCorner)corners
{
    UIBezierPath *rounded = [UIBezierPath bezierPathWithRoundedRect:view.bounds
                                                               byRoundingCorners:corners
                                                               cornerRadii:CGSizeMake(20.0, 20.0)];
    CAShapeLayer *shape = [[CAShapeLayer alloc] init];
    [shape setPath:rounded.CGPath];
    view.layer.mask = shape;
}
```

# Chapter 51: Make selective UIView corners rounded

## Section 51.1: Objective C code to make selected corner of a UIView rounded

First **import** #import <QuartzCore/QuartzCore.h> into your ViewController class. Here is how I set my view in code

```
UIView *view1=[[UIView alloc] init];
view1.backgroundColor=[UIColor colorWithRed:255/255.0 green:193/255.0 blue:72/255.0 alpha:1.0];
CGRect view1Frame = view1.frame;
view1Frame.size.width = SCREEN_WIDTH*0.97;
view1Frame.size.height = SCREEN_HEIGHT*0.2158;
view1Frame.origin.x = 0;
view1Frame.origin.y = 0.1422*SCREEN_HEIGHT-10;
view1.frame = view1Frame;
[self setMaskTo:view1 byRoundingCorners:UIRectCornerBottomRight|UIRectCornerTopRight];
[self.view addSubview:view1];
```

Here is the function which does the heavy lifting and rounds off the selected edges which is the Bottom Right and the Top Right edge in our case

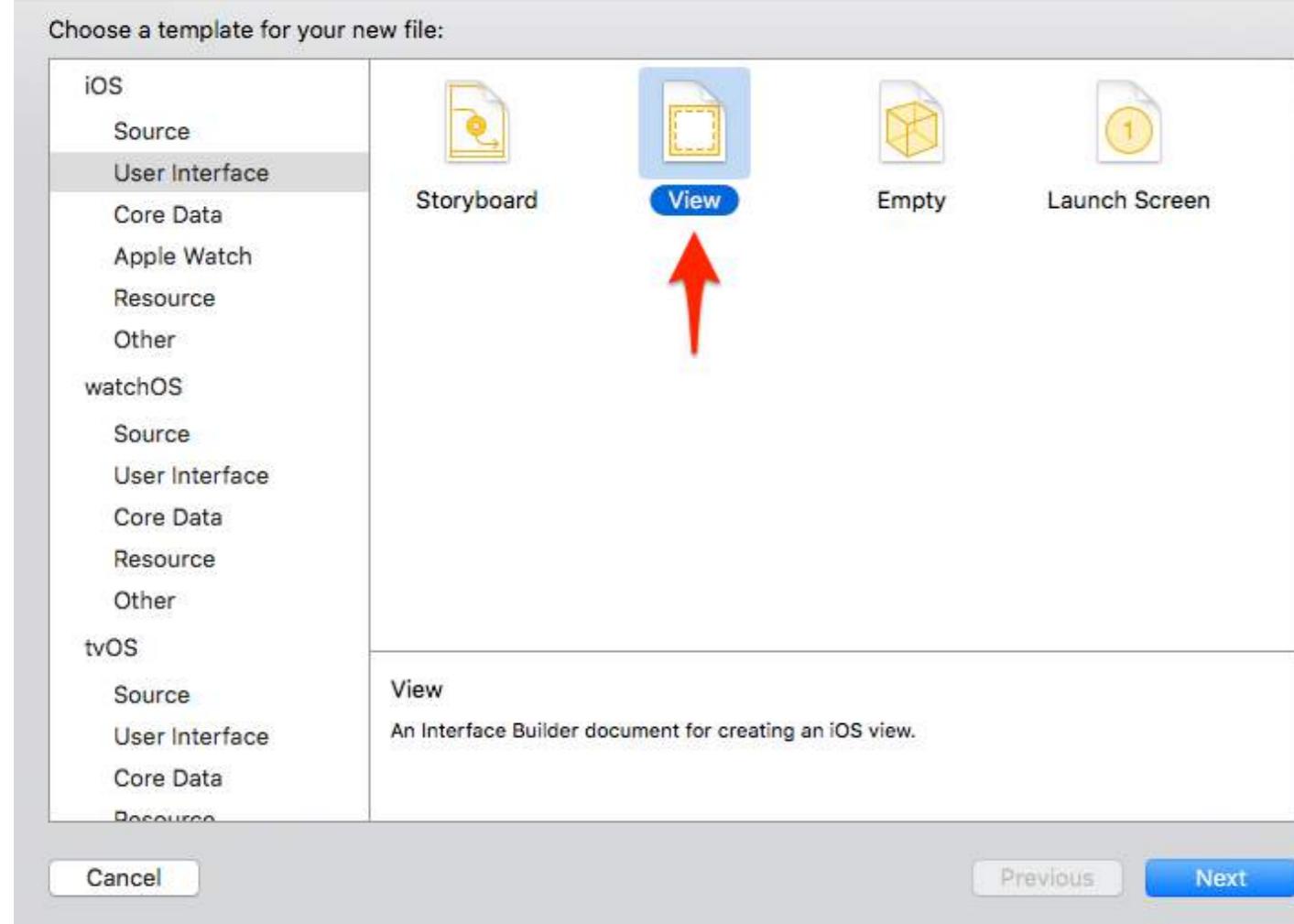
```
- (void)setMaskTo:(UIView*)view byRoundingCorners:(UIRectCorner)corners
{
    UIBezierPath *rounded = [UIBezierPath bezierPathWithRoundedRect:view.bounds
                                                               byRoundingCorners:corners
                                                               cornerRadii:CGSizeMake(20.0, 20.0)];
    CAShapeLayer *shape = [[CAShapeLayer alloc] init];
    [shape setPath:rounded.CGPath];
    view.layer.mask = shape;
}
```

# 第52章：来自XIB文件的自定义UIView

## 第52.1节：连接元素

创建一个XIB文件

Xcode菜单栏 > 文件 > 新建 > 文件。  
选择iOS，用户界面，然后选择“视图”：



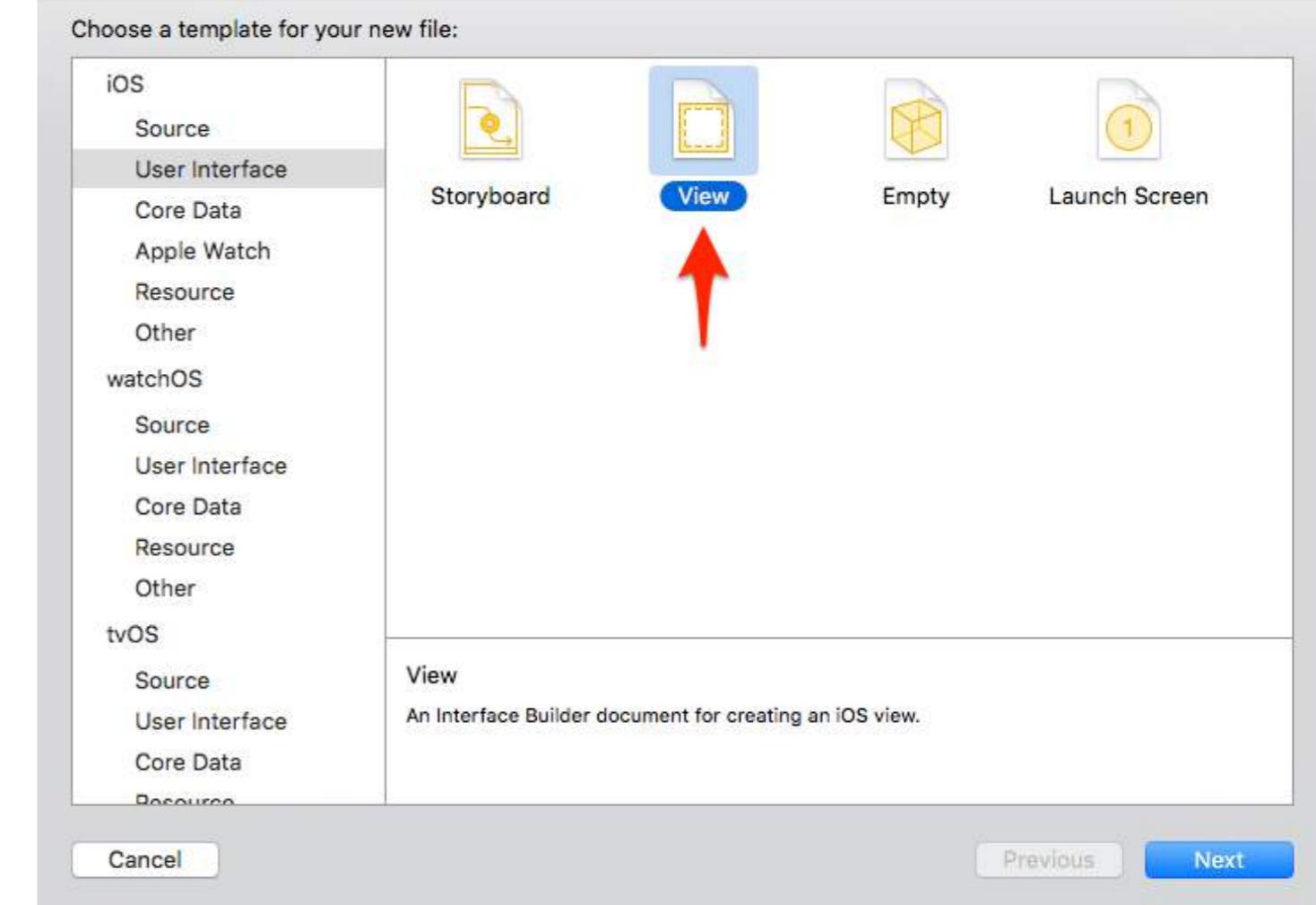
给你的XIB命名（是的，我们正在做一个宝可梦示例????）。  
记得检查你的目标并点击“创建”。

# Chapter 52: Custom UIViews from XIB files

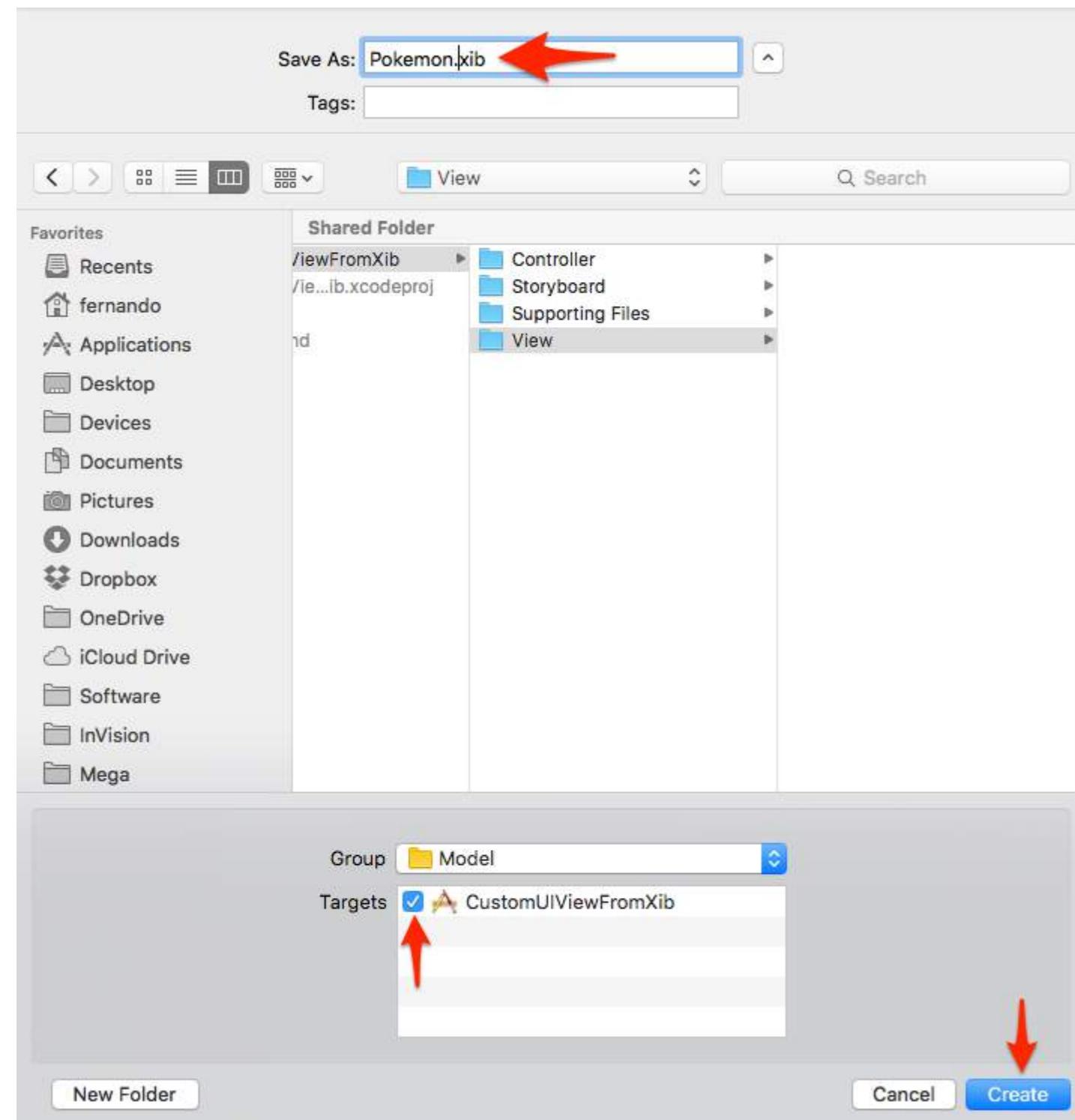
## Section 52.1: Wiring elements

Create a XIB file

Xcode Menu Bar > File > New > File.  
Select iOS, User Interface and then "View":



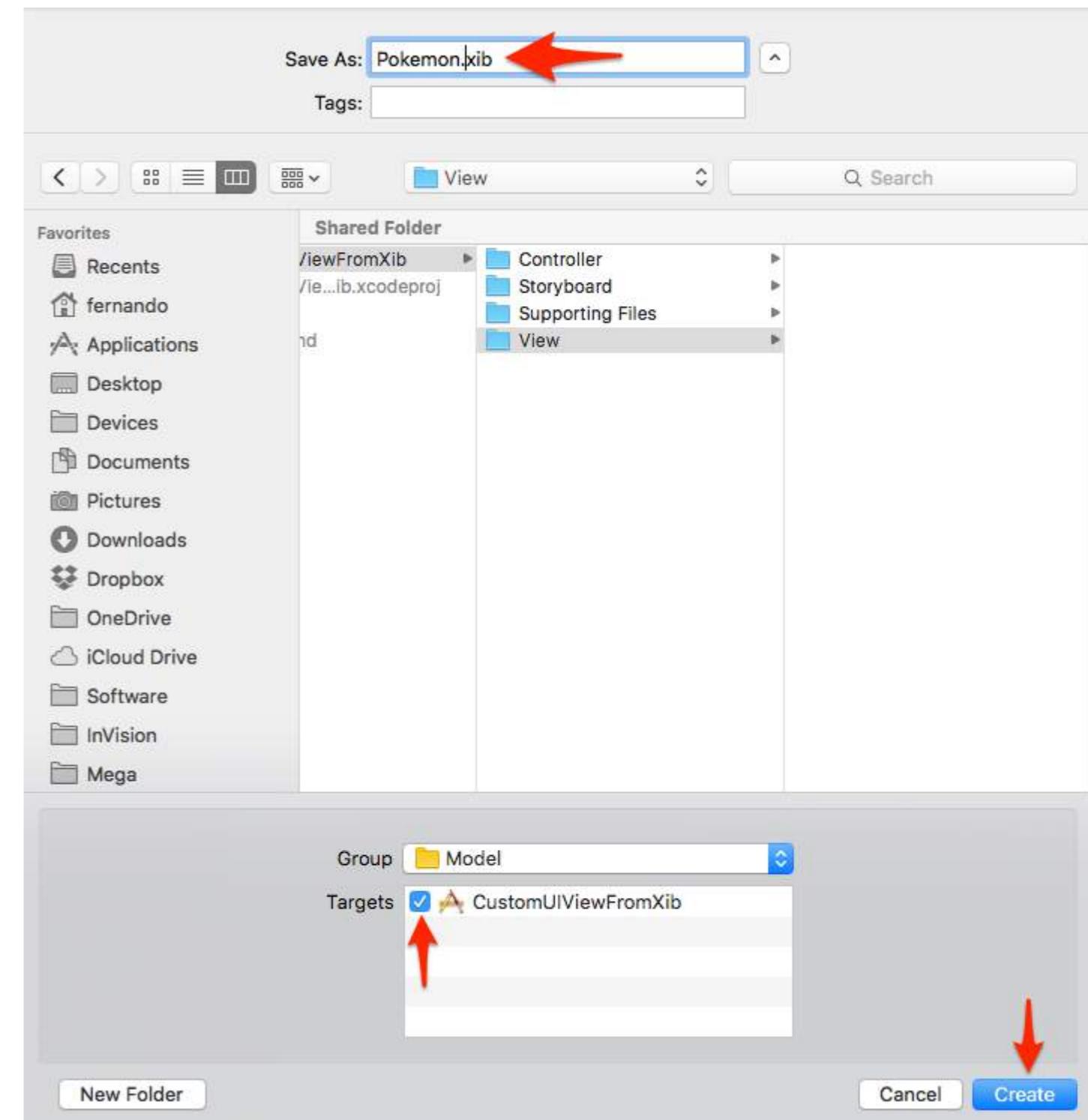
Give your XIB a name (yes, we are doing a Pokemon example????).  
**Remember to check your target** and hit "Create".



设计你的视图

为了简化操作，设置：

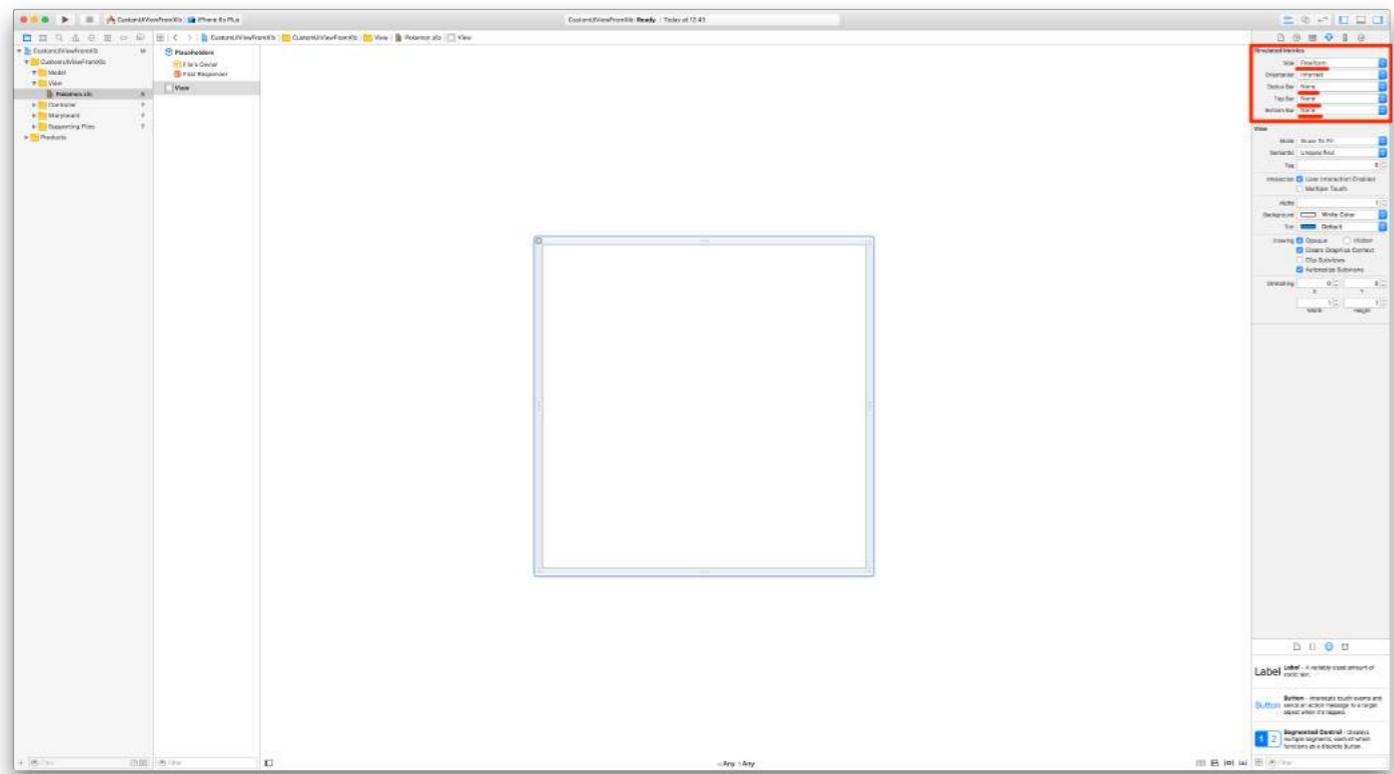
- 大小：自由形状
- 状态栏：无
- 顶部栏：无
- 底部栏：无



Design your view

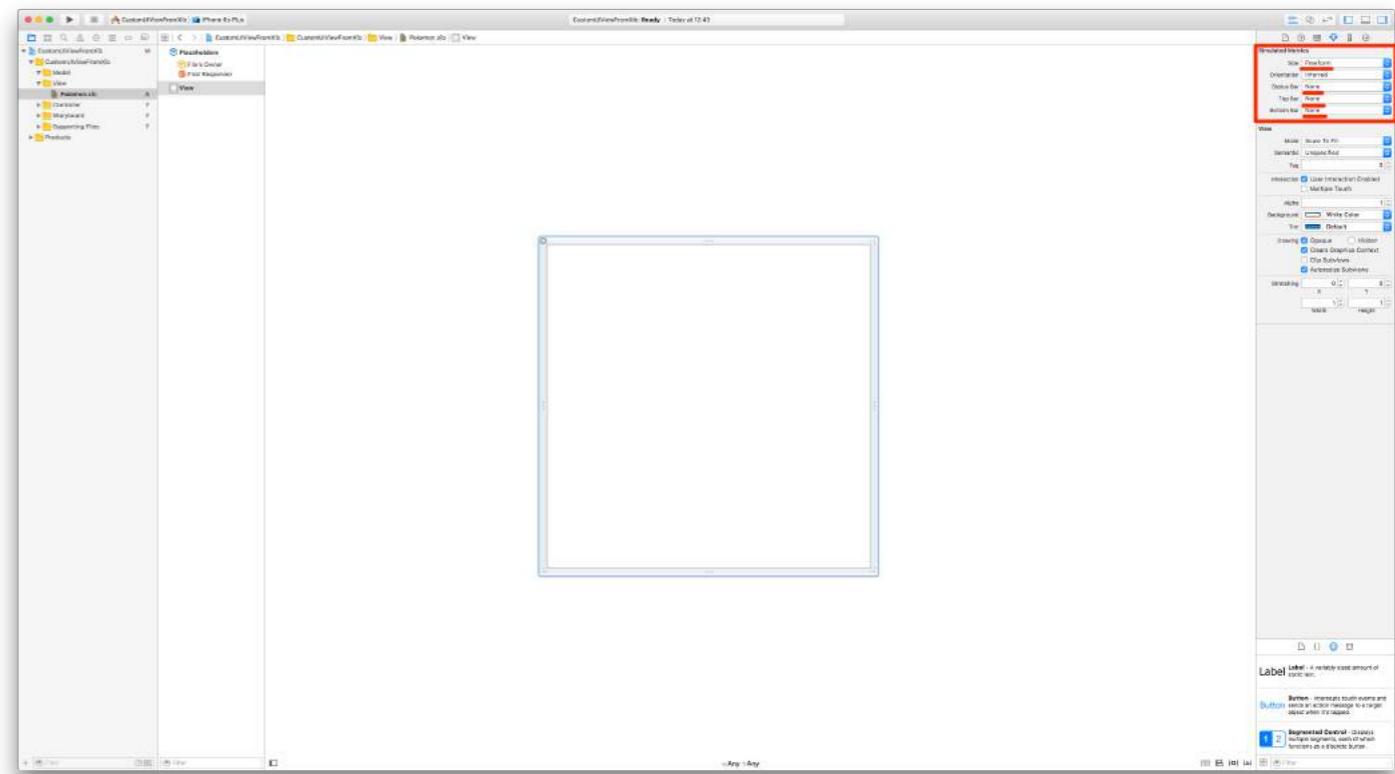
To make things easier, set:

- Size: Freeform
- Status Bar: None
- Top Bar: None
- Bottom Bar: None



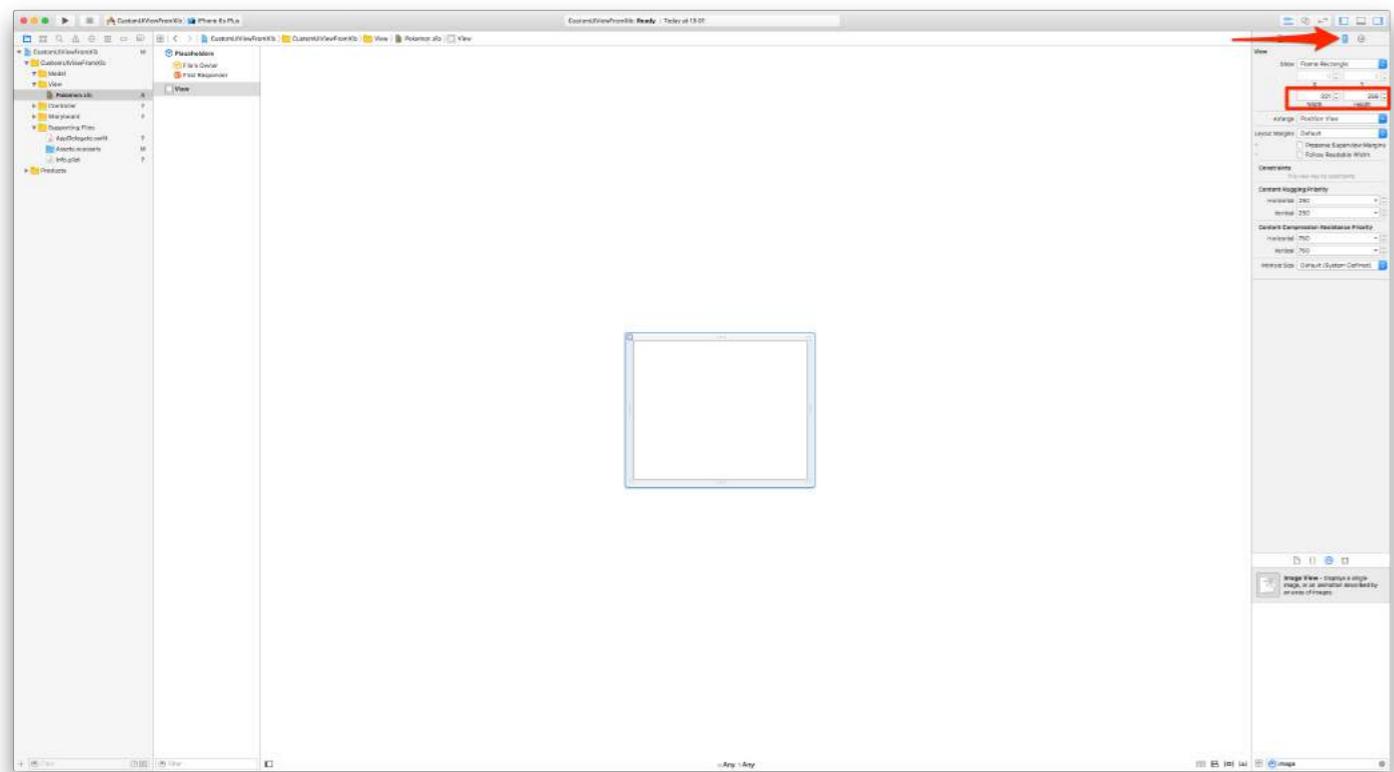
点击大小检查器并调整视图大小。

在本例中，我们将使用宽度321和高度256。



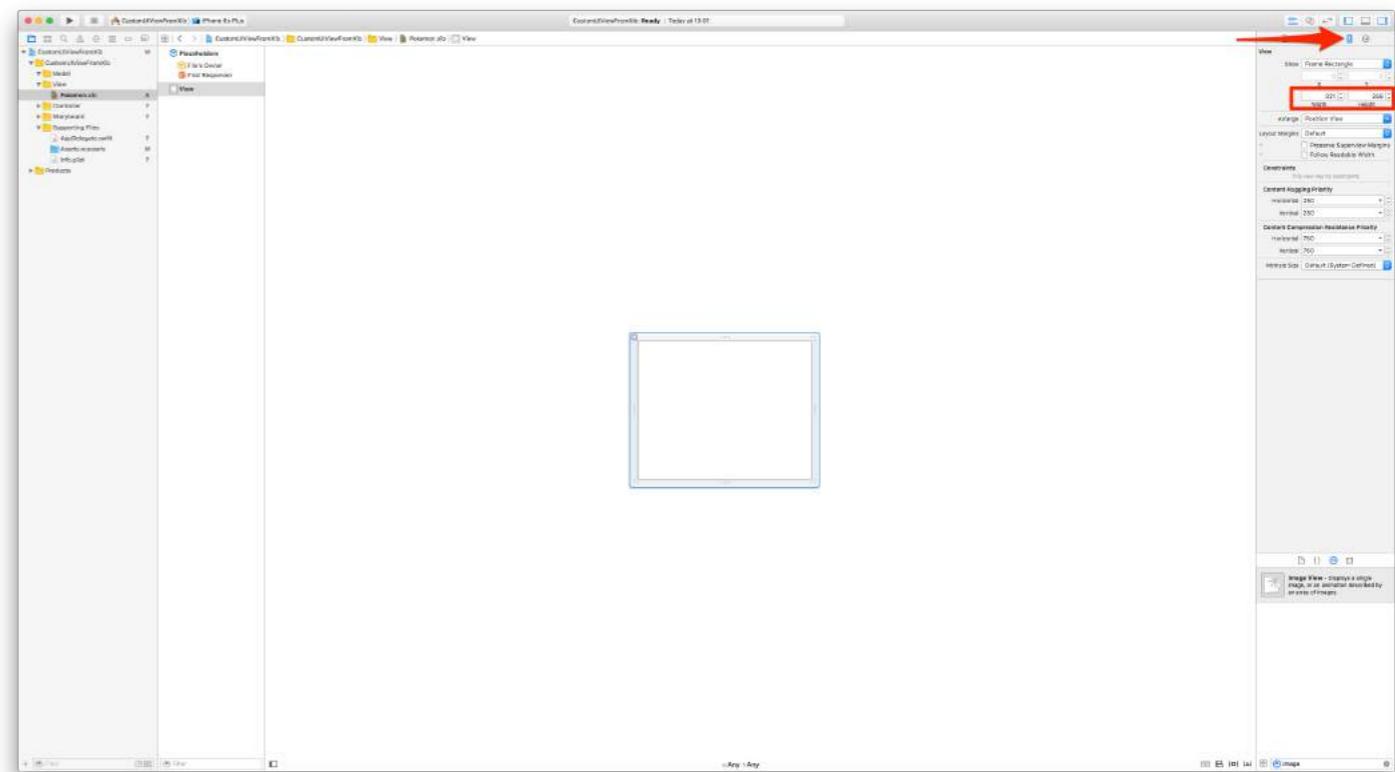
Click on the Size Inspector and resize the view.

For this example we'll be using width 321 and height 256.



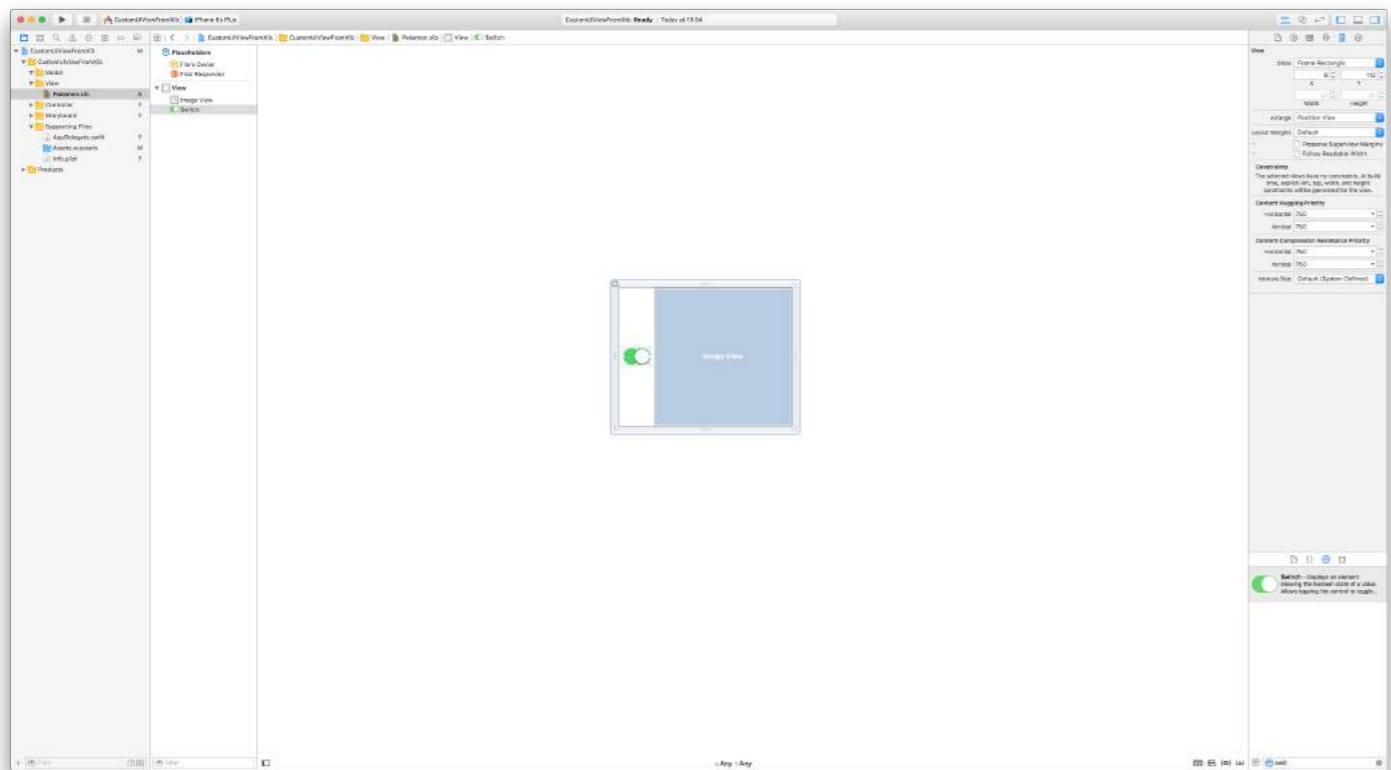
将一些元素拖入你的XIB文件，如下所示。

这里我们将添加一个图像视图（256x256）和一个开关。

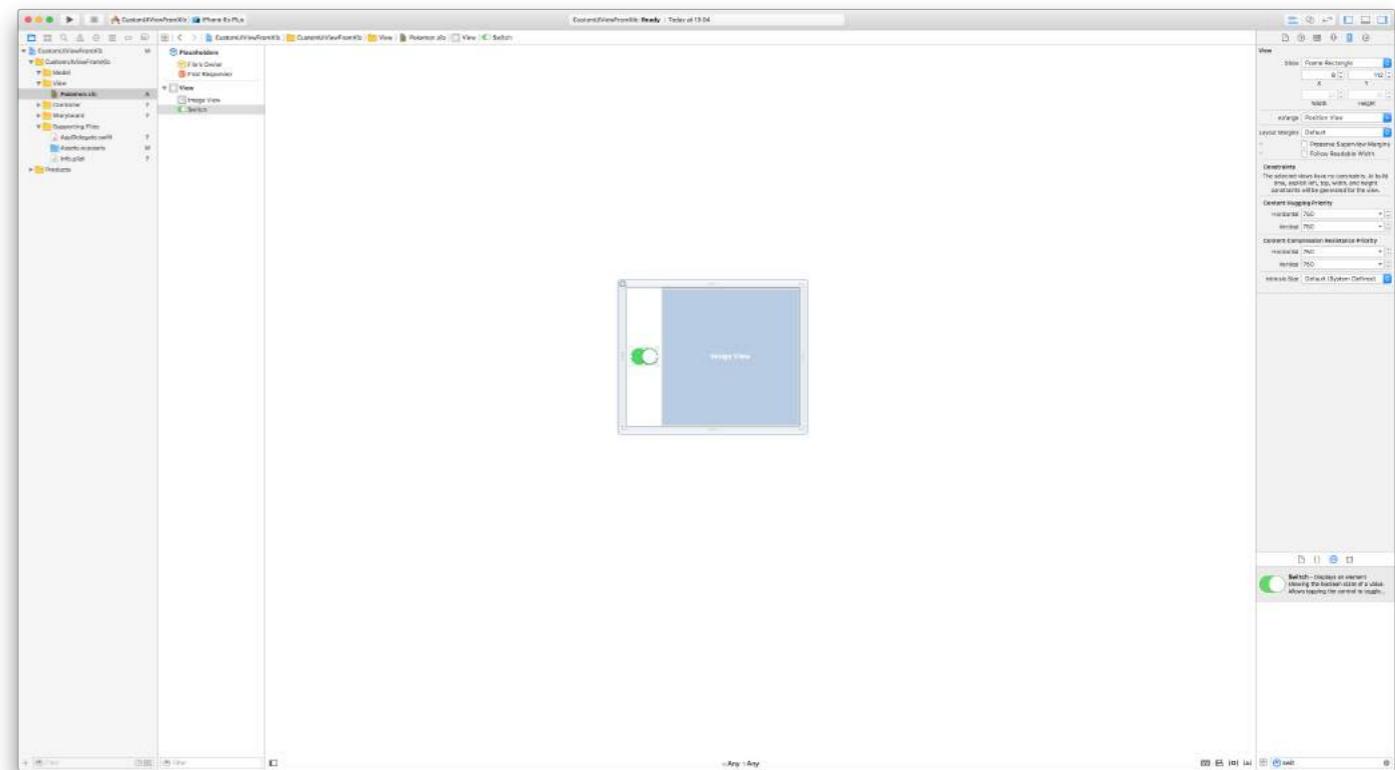


Drop some elements into your XIB file like shown below.

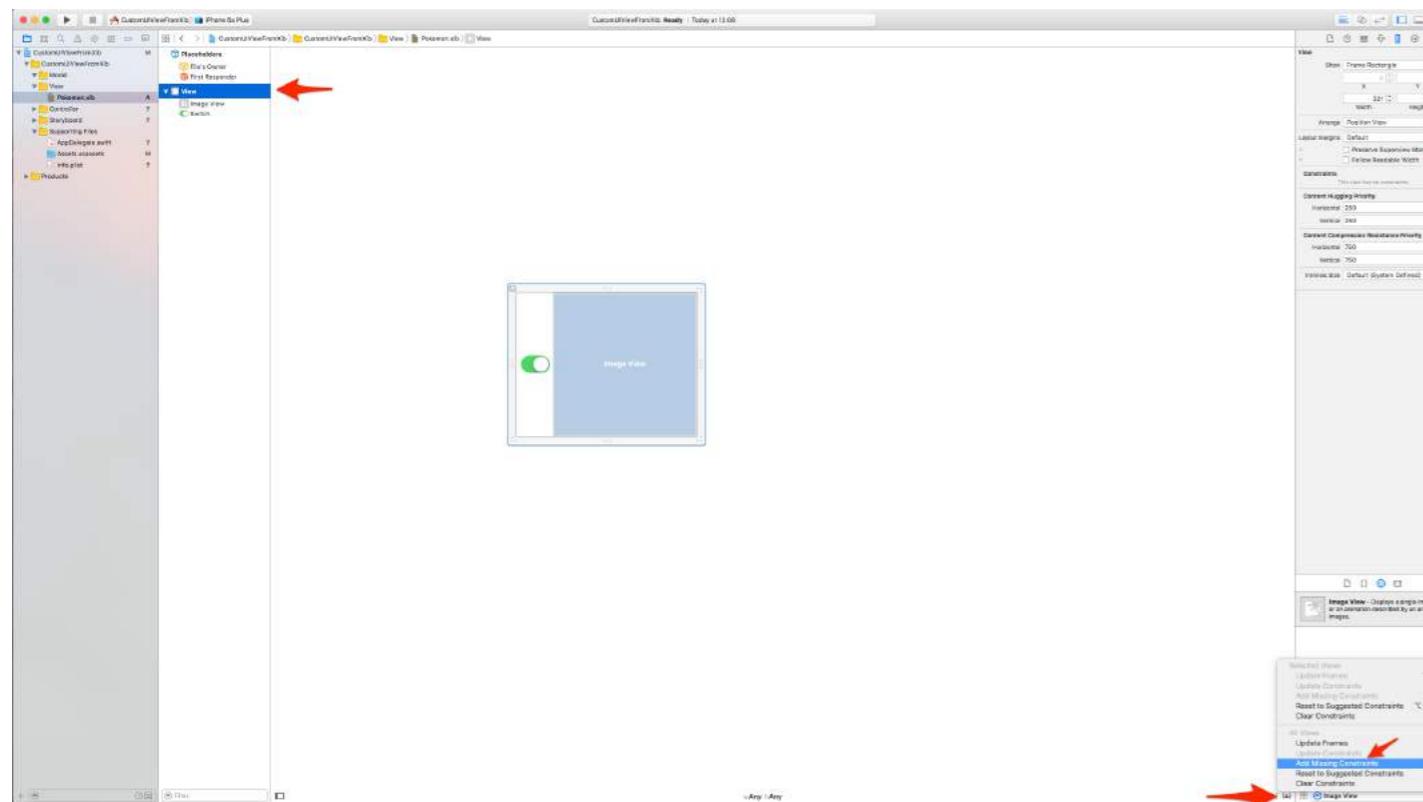
Here we'll be adding an **Image View** (256x256) and a **Switch**.



通过点击“解决自动布局问题”（右下角），然后在“所有视图”下选择“添加缺失的约束”来添加自动布局约束。



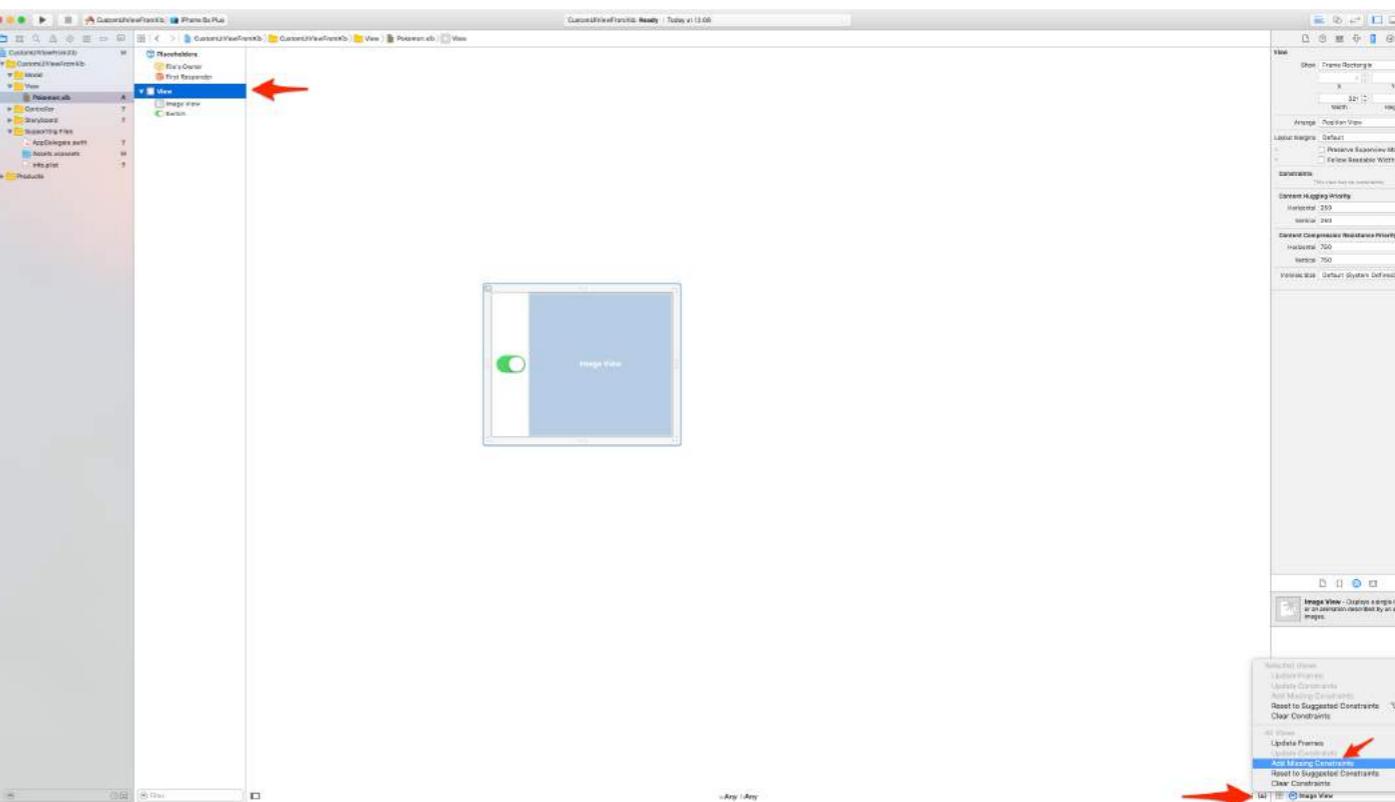
Add Auto-Layout constraints by clicking on "Resolve Auto Layout Issues" (bottom-right) and selecting "Add Missing Constraints" under "All Views".



通过点击“显示助理编辑器”（右上角），然后选择“预览”来预览你所做的更改。

你可以通过点击“加号”按钮来添加 iPhone 屏幕。

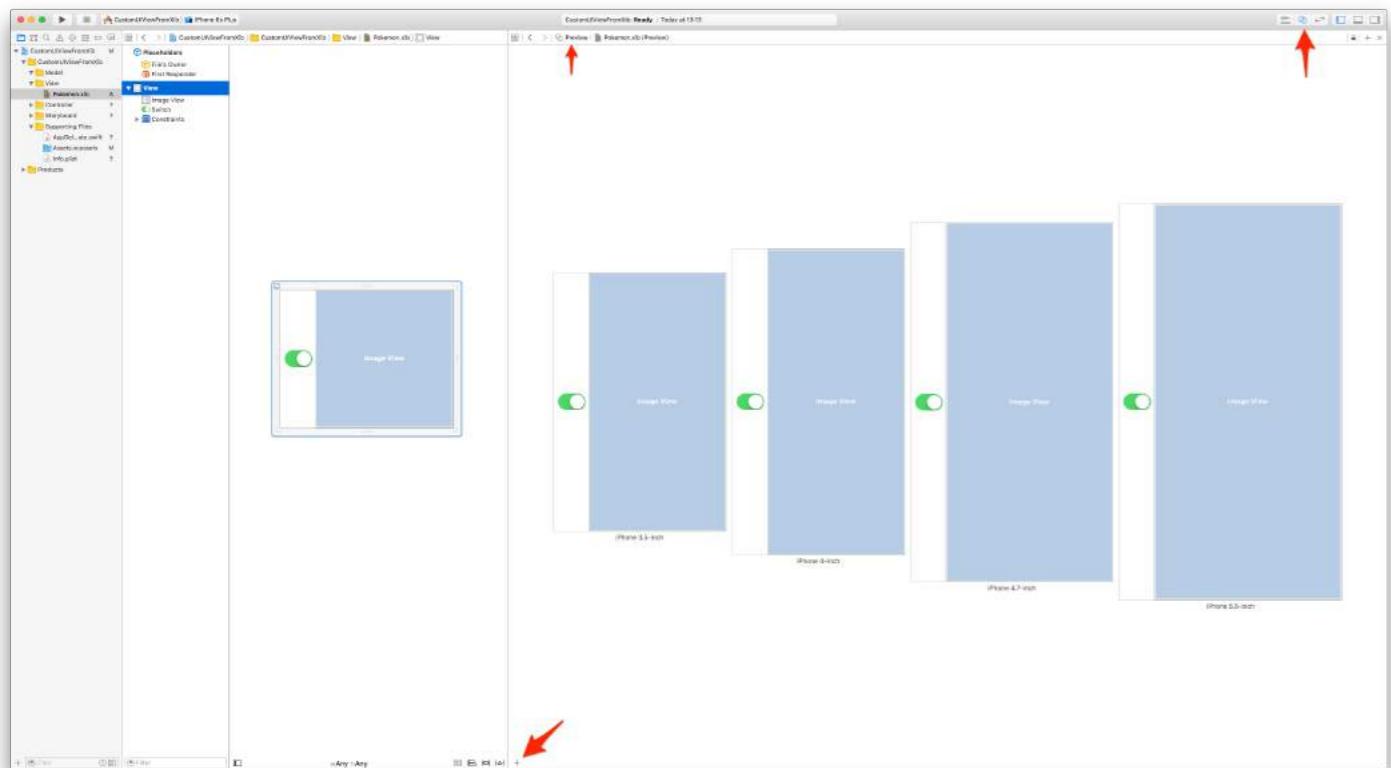
预览应如下所示：



Preview the changes you made by clicking on "Show the Assistant Editor" (top-right), then "Preview".

You can add iPhone screens by clicking on the "Plus" button.

The preview should look like this:

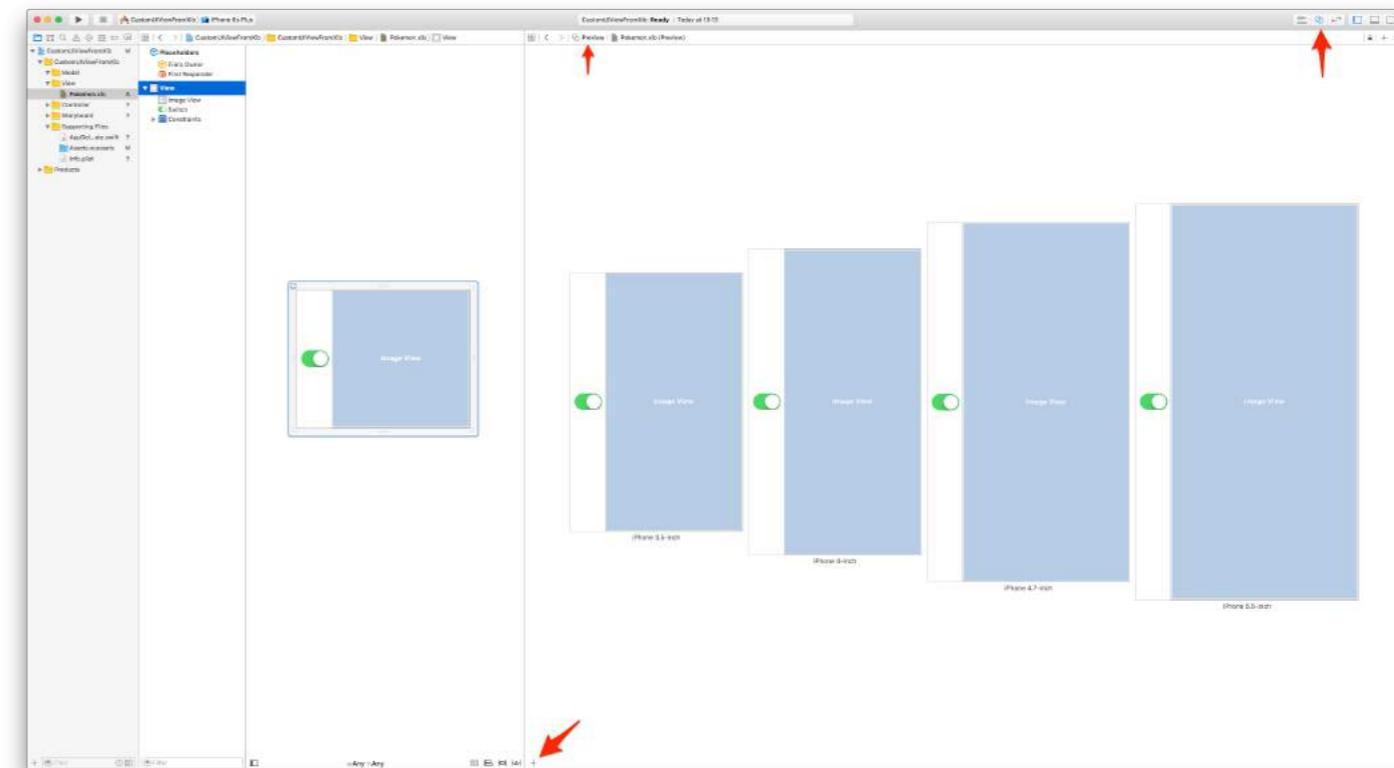


### 子类化 UIView

创建将管理 XIB 文件的类。

Xcode 菜单栏 > 文件 > 新建 > 文件。

选择 iOS / 源代码 / Cocoa Touch 类。点击“下一步”。



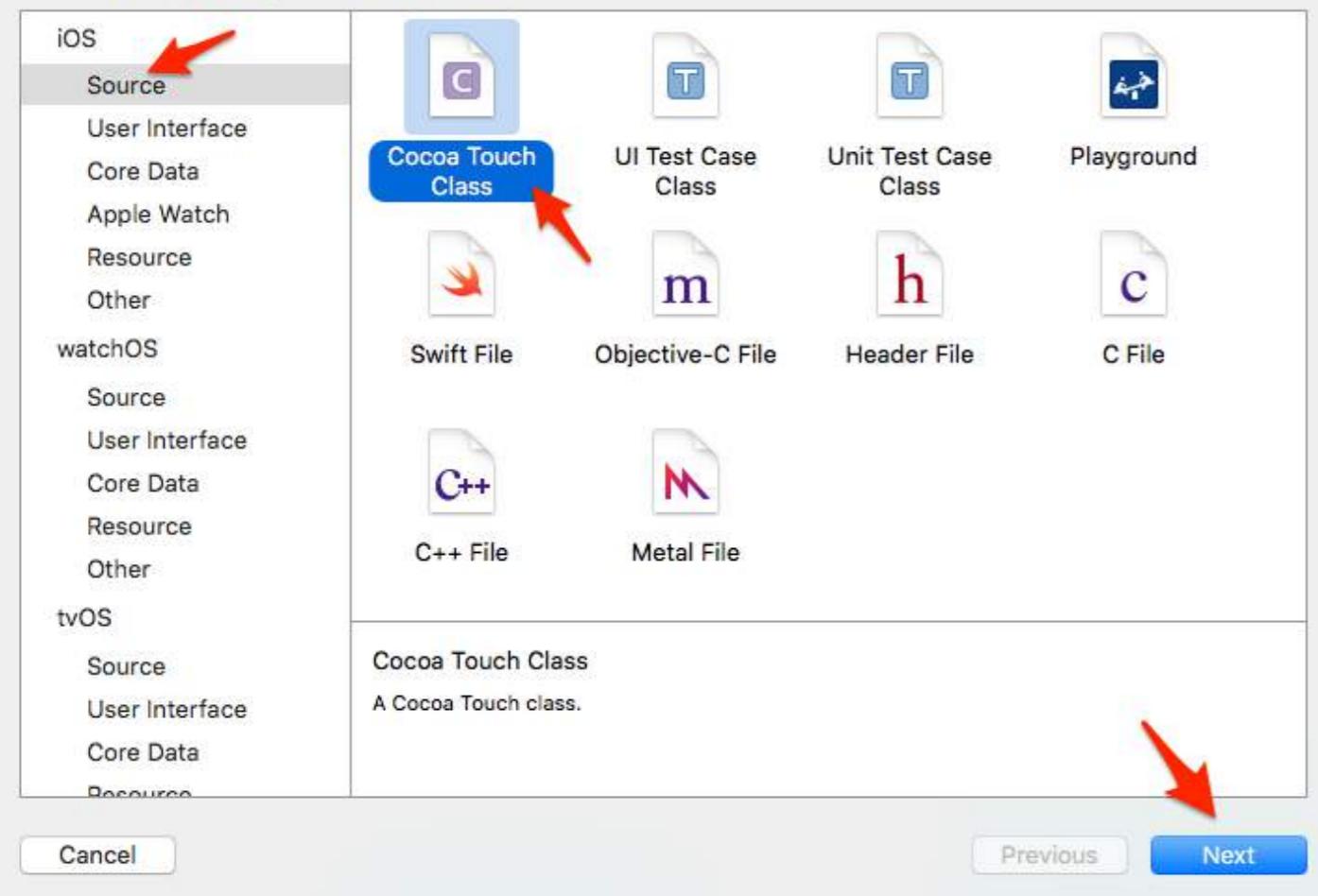
### Subclass UIView

Create the class that is going to manage the XIB file.

Xcode Menu Bar > File > New > File.

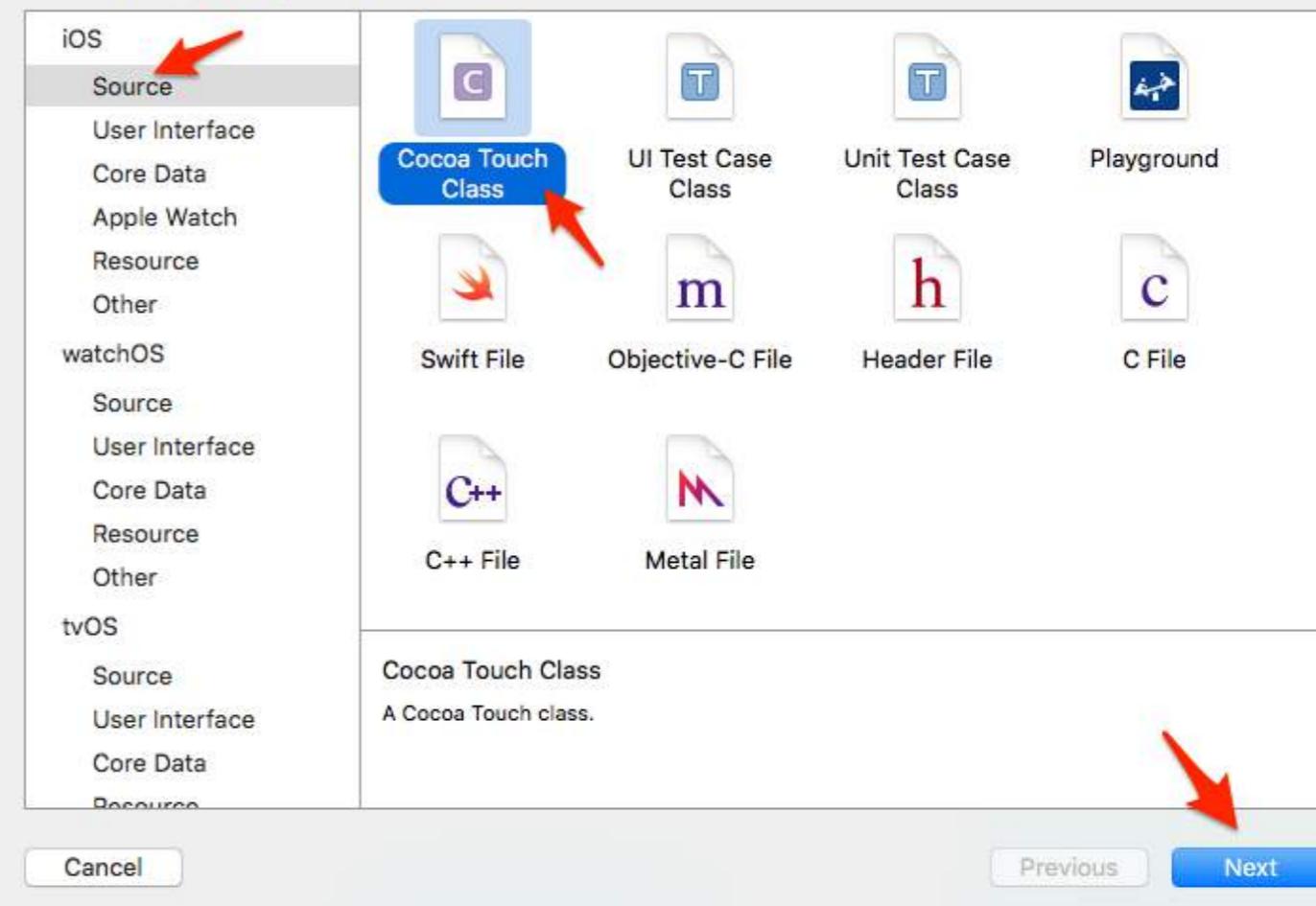
Select iOS / Source / Cocoa Touch Class. Hit "Next".

Choose a template for your new file:



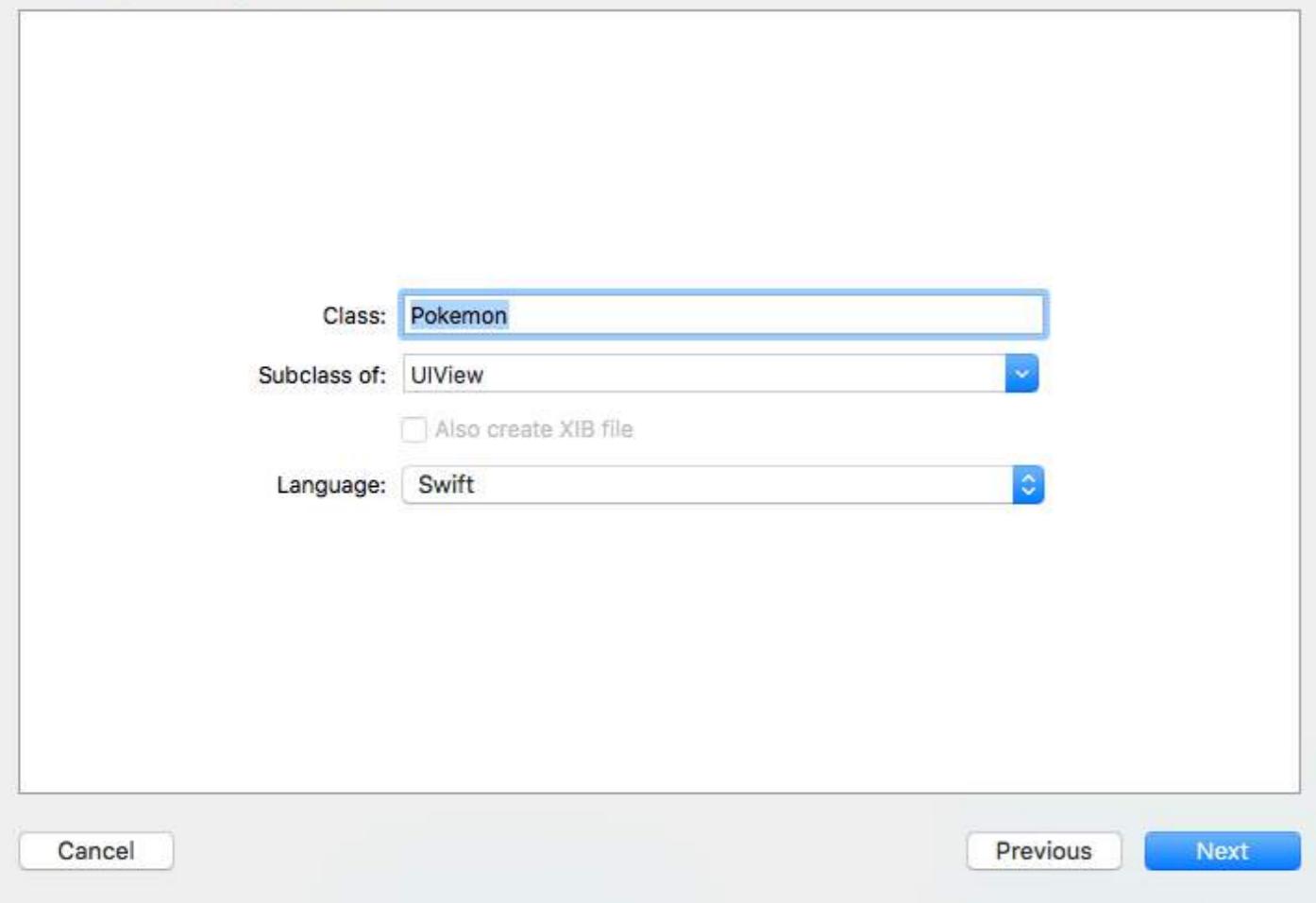
为类命名，名称必须与 XIB 文件 (Pokemon) 相同。  
选择 UIView 作为子类类型，然后点击“下一步”。

Choose a template for your new file:



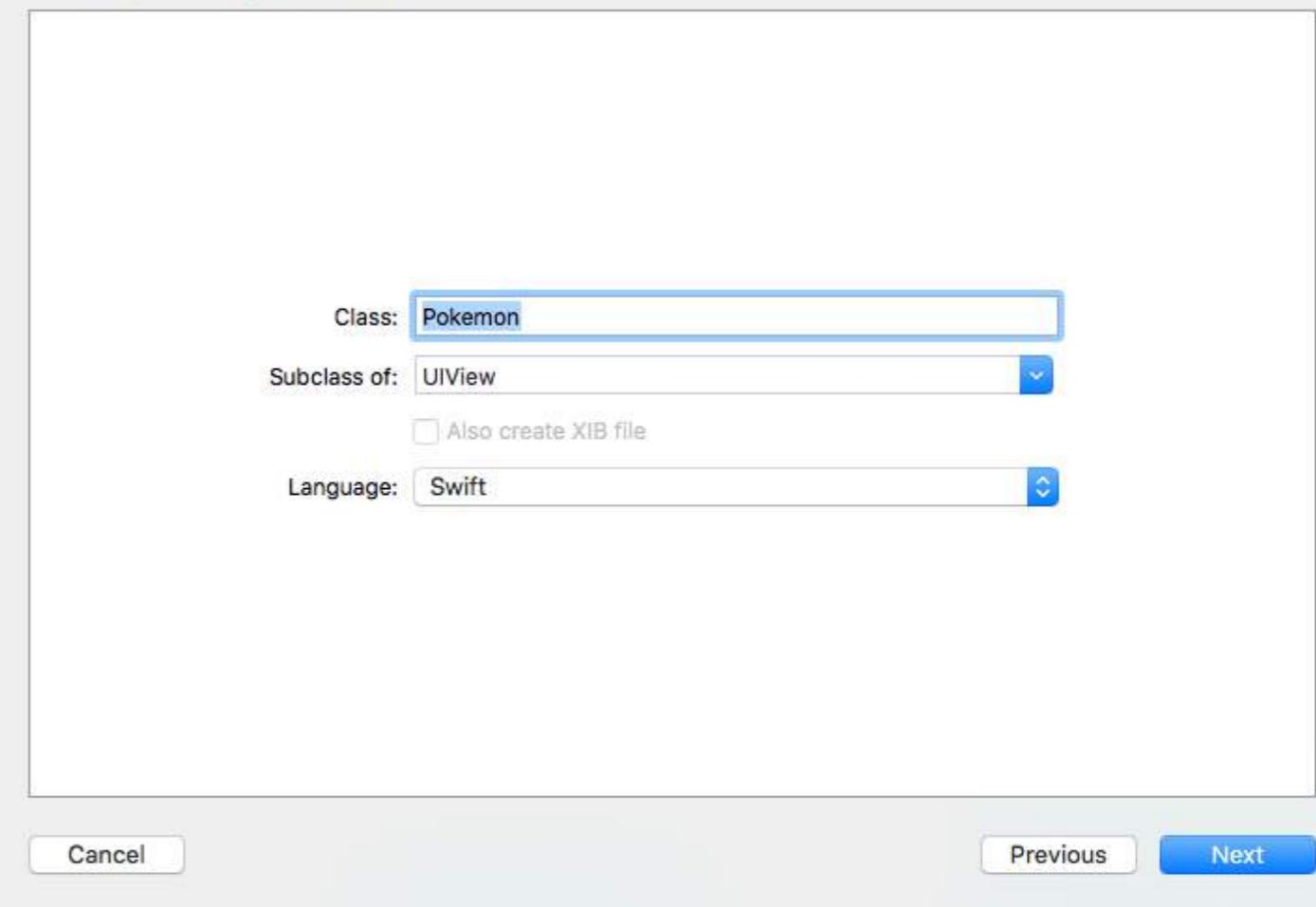
Give the class a name, which must be the same name as the XIB file (Pokemon).  
Select UIView as the subclass type, then hit "Next".

Choose options for your new file:

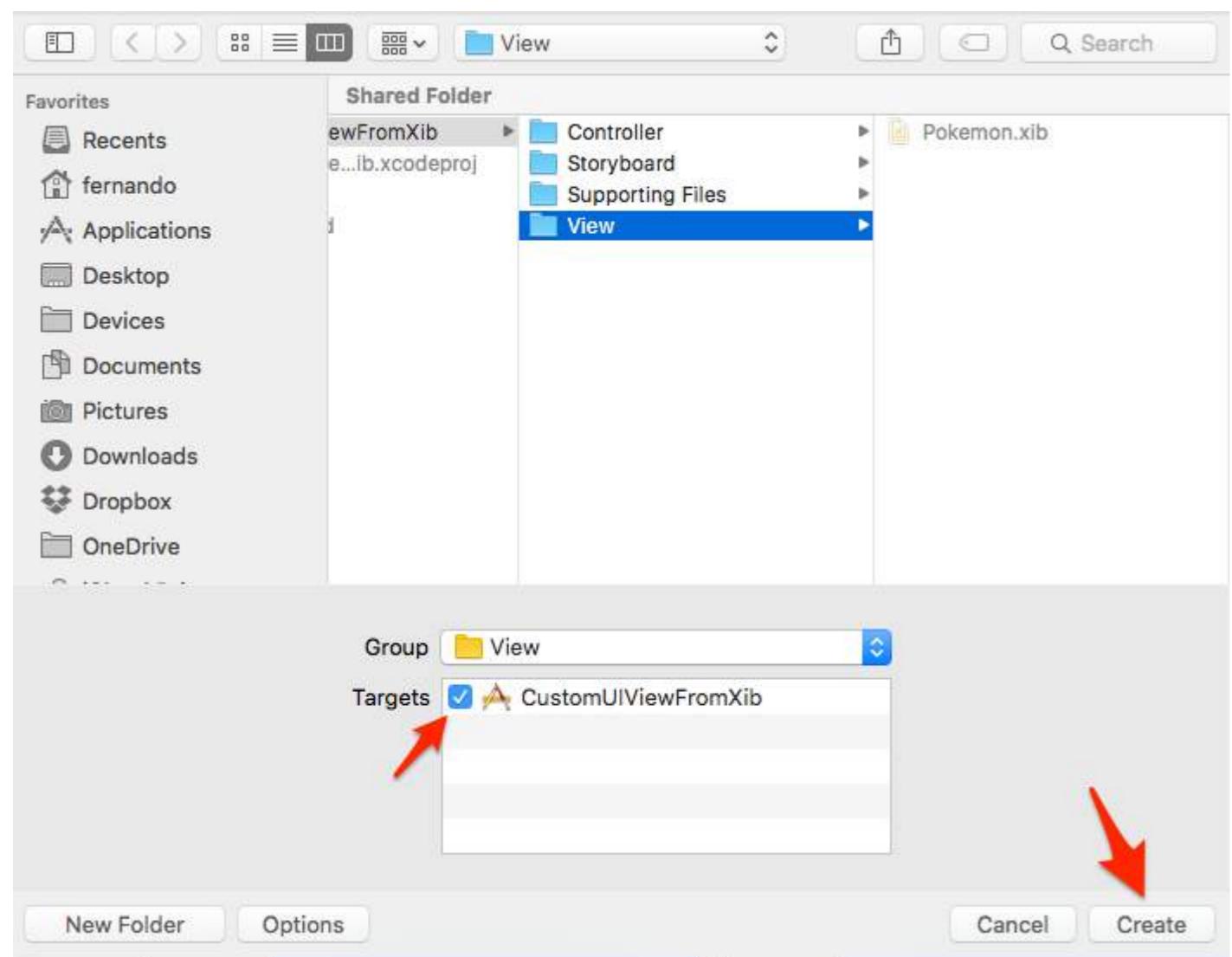


在下一个窗口中，选择你的目标并点击“创建”。

Choose options for your new file:



On the next window, select your target and hit "Create".

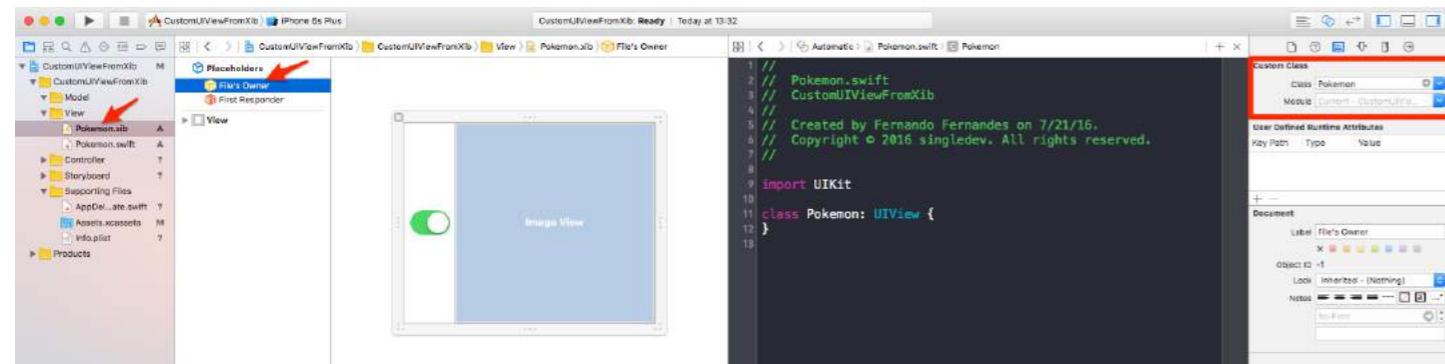


通过“文件’所有者”属性将 Pokemon.xib 连接到 Pokemon.swift

点击 Xcode 中的 Pokemon.xib 文件。

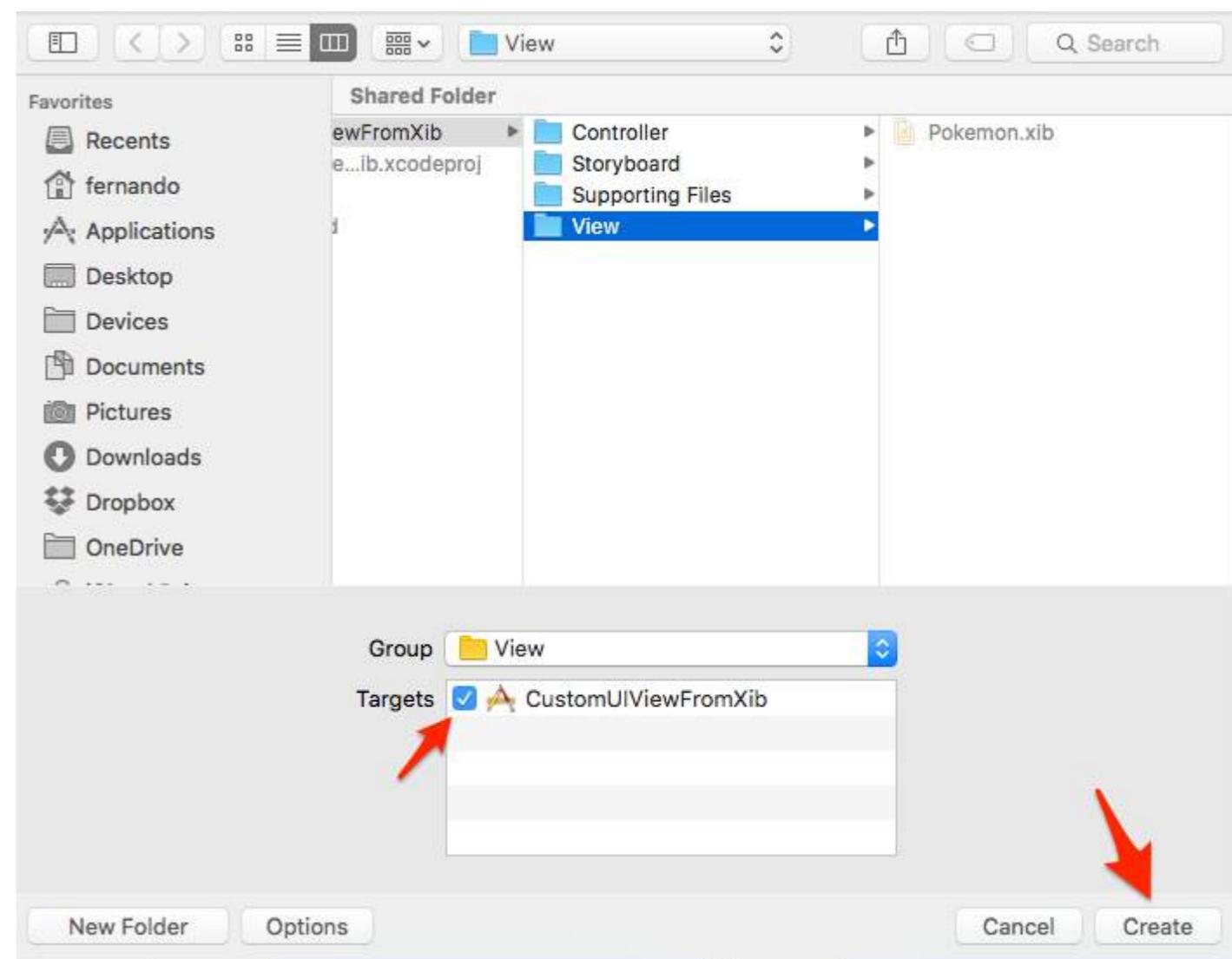
点击“文件所有者”出口。

在“身份检查器”（右上角），将类设置为我们刚创建的 Pokemon.swift 文件。



宝可梦！！！

是的！将一些宝可梦拖放到你的项目中，以完成我们的“基础设施”。  
这里我们添加了两个PGN文件，大小为256x256，透明背景。

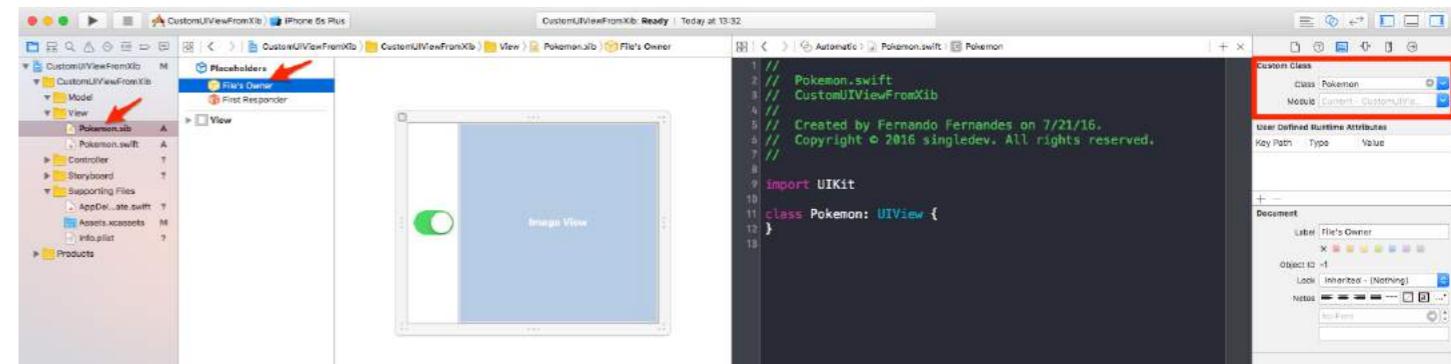


Connect Pokemon.xib to Pokemon.swift via "File's Owner" attribute

Click on the Pokemon.xib file in Xcode.

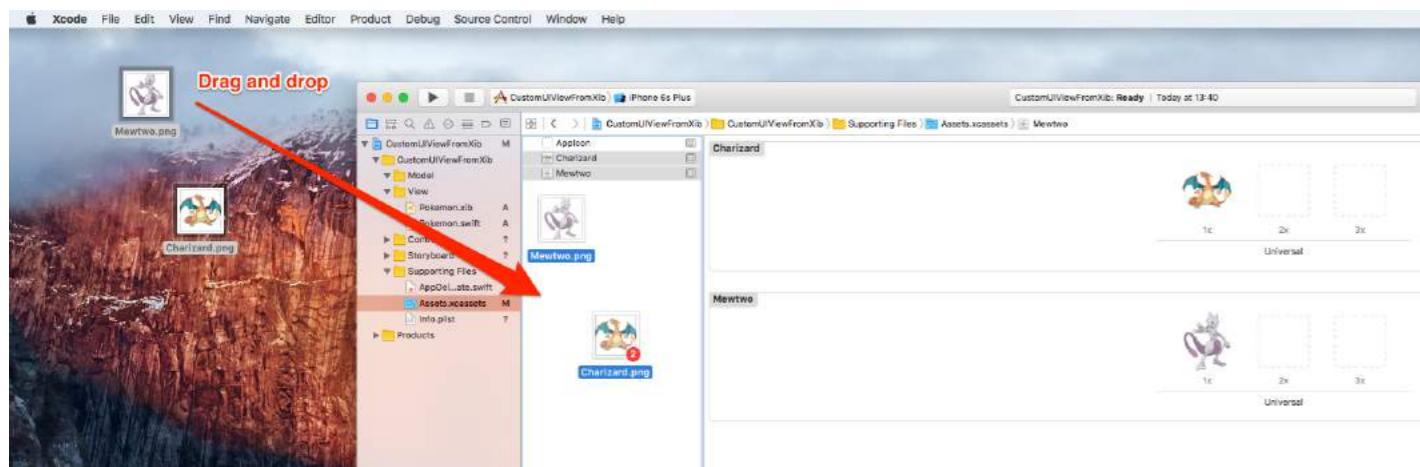
Click on the "File's Owner" outlet.

On the "Identity inspector" (top-right), set the Class to our recently created Pokemon.swift file.



POKEMONS!!!

Yes! Drag and drop some Pokemons into your project to finish up our "infrastructure".  
Here we are adding two PGN files, 256x256, transparent.



给我看代码吧。

好吧，好吧。

是时候给我们的 Pokemon.swift 类添加一些代码了。

其实很简单：

1. 实现必需的初始化方法
2. 加载 XIB 文件
3. 配置将显示 XIB 文件的视图
4. 显示上述视图

将以下代码添加到 Pokemon.swift 类中：

```
import UIKit

class Pokemon: UIView {

    // MARK: - 初始化方法

    override init(frame: CGRect) {
        super.init(frame: frame)
        setupView()
    }

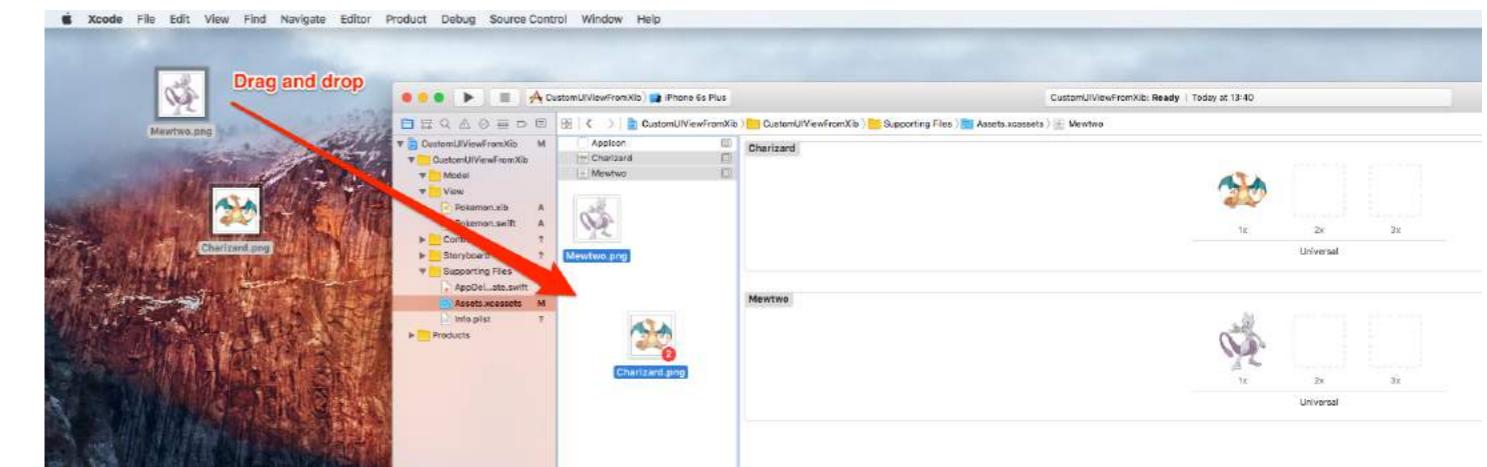
    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        setupView()
    }

    // MARK: - 私有辅助方法

    // 执行初始设置。
    private func setupView() {
        let view = viewFromNibForClass()
        view.frame = bounds

        // 自动布局相关内容。
        view.autoresizingMask = [
            UIViewAutoresizing.flexibleWidth,
            UIViewAutoresizing.flexibleHeight
        ]

        // 显示视图。
    }
}
```



Show me code already.

All right, all right.

Time to add some code to our Pokemon.swift class.

It's actually pretty simple:

1. Implement required initializers
2. Load the XIB file
3. Configure the view that will display the XIB file
4. Show the above view

Add the following code to the Pokemon.swift class:

```
import UIKit

class Pokemon: UIView {

    // MARK: - Initializers

    override init(frame: CGRect) {
        super.init(frame: frame)
        setupView()
    }

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        setupView()
    }

    // MARK: - Private Helper Methods

    // Performs the initial setup.
    private func setupView() {
        let view = viewFromNibForClass()
        view.frame = bounds

        // Auto-layout stuff.
        view.autoresizingMask = [
            UIViewAutoresizing.flexibleWidth,
            UIViewAutoresizing.flexibleHeight
        ]

        // Show the view.
    }
}
```

```

addSubview(view)
}

// 加载一个XIB文件到视图并返回该视图。
private func viewFromNibForClass() -> UIView {
    let bundle = Bundle(for: type(of: self))
    let nib = UINib(nibName: String(describing: type(of: self)), bundle: bundle)
    let view = nib.instantiate(withOwner: self, options: nil).first as! UIView

    /* 适用于 swift < 3.x 的用法
    let bundle = NSBundle(forClass: self.dynamicType)
    let nib = UINib(nibName: String(self.dynamicType), bundle: bundle)
    let view = nib.instantiateWithOwner(self, options: nil)[0] as! UIView
    */

    return view
}

```

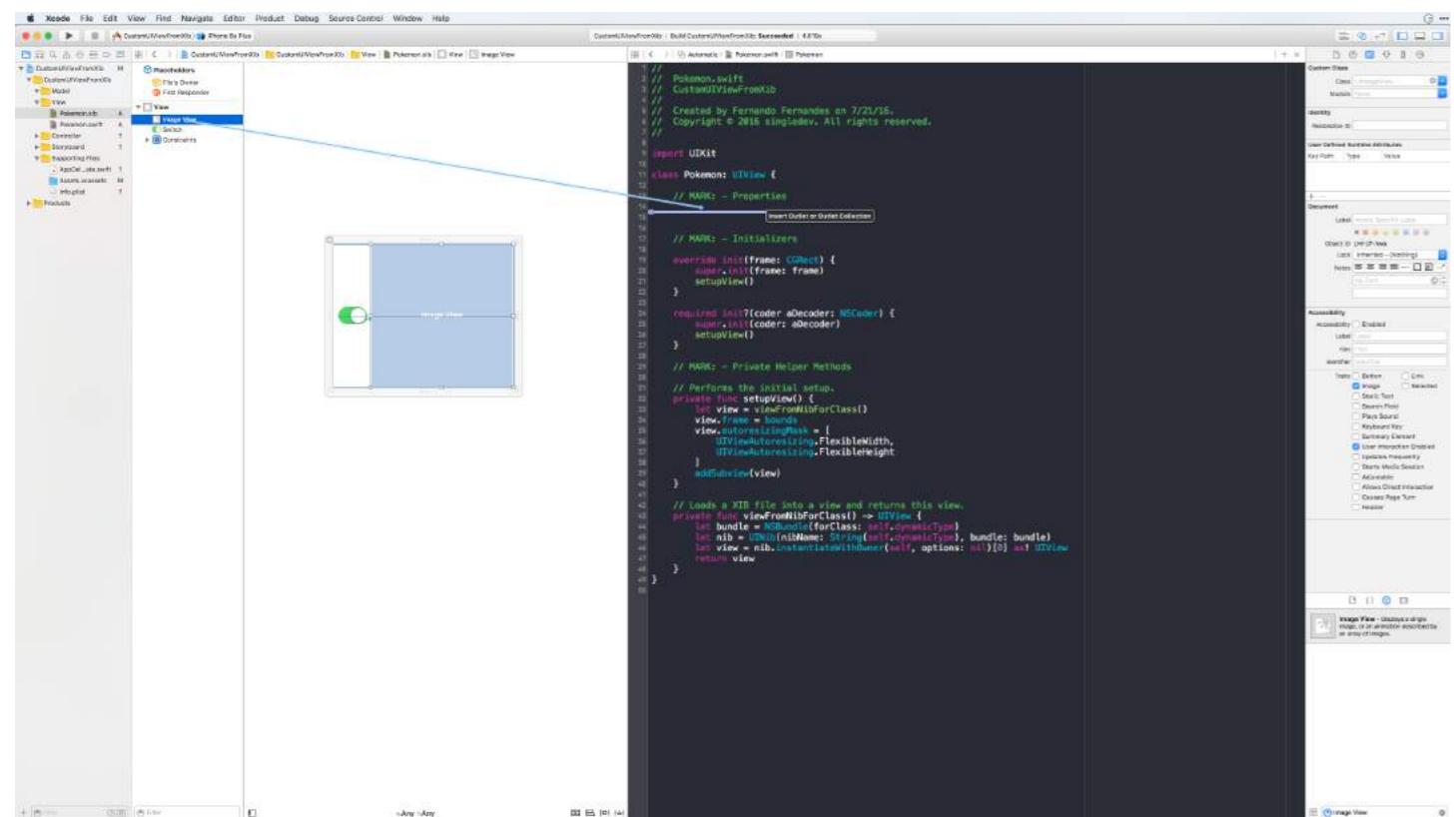
### @IBDesignable 和 @IBInspectable

通过在类上添加 `@IBDesignable`, 可以使其在界面构建器 (Interface Builder) 中实时渲染。

通过在类的属性上添加 `@IBInspectable`, 可以在界面构建器中实时看到自定义视图随着属性修改而变化。

让我们将自定义视图的 Image View 设为“Inspectable”。

首先, 将 `Pokemon.xib` 文件中的 Image View 连接到 `Pokemon.swift` 类。



将 `outlet` 命名为 `imageView`, 然后添加以下代码 (注意类名前的 `@IBDesignable`) :

```
@IBDesignable class Pokemon: UIView {
```

```

addSubview(view)
}

// Loads a XIB file into a view and returns this view.
private func viewFromNibForClass() -> UIView {
    let bundle = Bundle(for: type(of: self))
    let nib = UINib(nibName: String(describing: type(of: self)), bundle: bundle)
    let view = nib.instantiate(withOwner: self, options: nil).first as! UIView

    /* Usage for swift < 3.x
    let bundle = NSBundle(forClass: self.dynamicType)
    let nib = UINib(nibName: String(self.dynamicType), bundle: bundle)
    let view = nib.instantiateWithOwner(self, options: nil)[0] as! UIView
    */

    return view
}

```

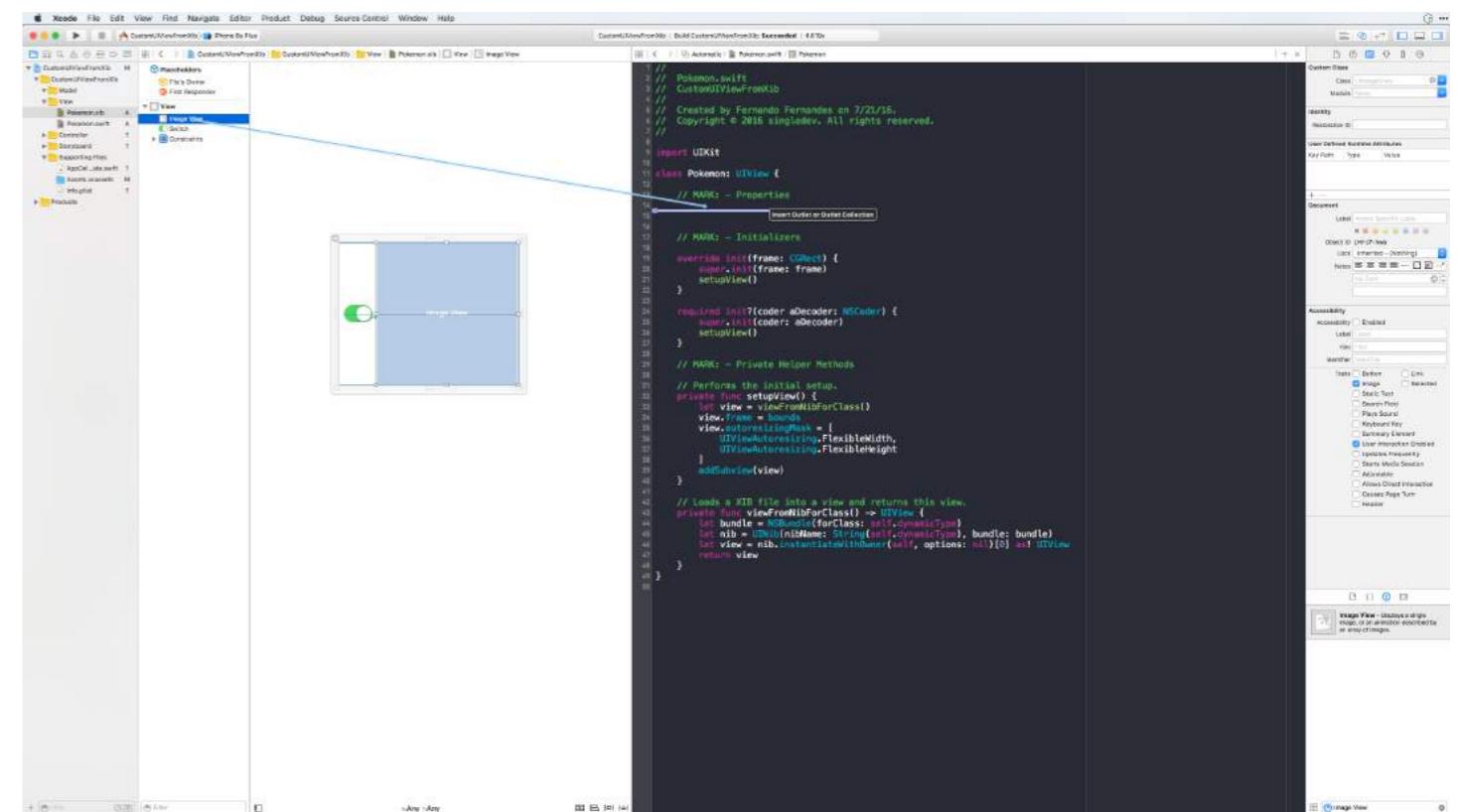
### @IBDesignable and @IBInspectable

By adding `@IBDesignable` to your class, you make possible for it to live-render in Interface Builder.

By adding `@IBInspectable` to the properties of your class, you can see your custom views changing in Interface Builder as soon as you modify those properties.

Let's make the Image View of our custom view "Inspectable".

First, hook up the Image View from the `Pokemon.xib` file to the `Pokemon.swift` class.



Call the outlet `imageView` and then add the following code (notice the `@IBDesignable` before the class name):

```
@IBDesignable class Pokemon: UIView {
```

```
// MARK: - 属性

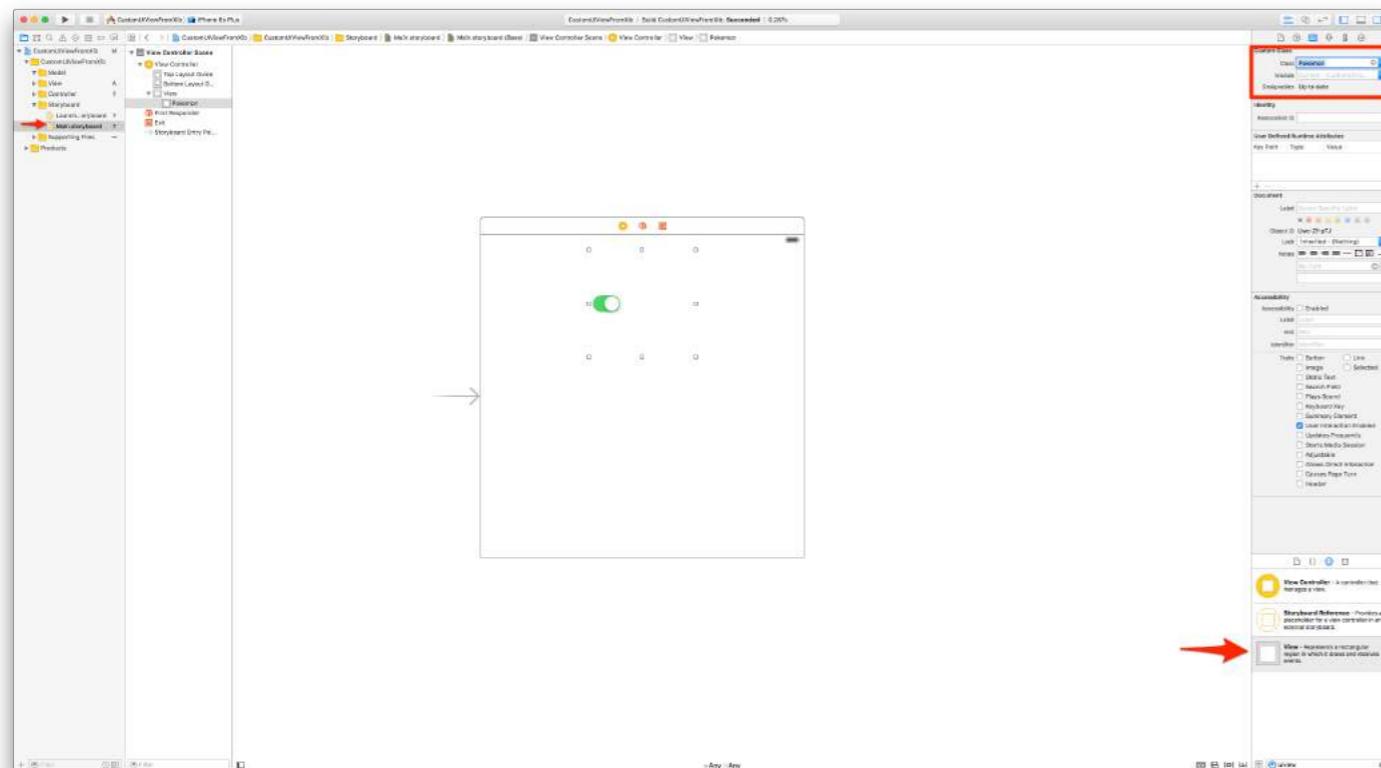
@IBOutlet weak var imageView: UIImageView!

@IBInspectable var image: UIImage? {
    get {
        return imageView.image
    }
    set(image) {
        imageView.image = image
    }
}

// MARK: - 初始化方法
...
```

### 使用您的自定义视图

打开您的主故事板文件，拖入一个UIView。  
将视图大小调整为例如200x200。居中。  
转到身份检查器（右上角），将类设置为Pokemon。



要选择一个宝可梦，请转到属性检查器（右上角），然后选择之前使用超棒的@IBInspectable图像属性添加的宝可梦图像之一。

```
// MARK: - Properties

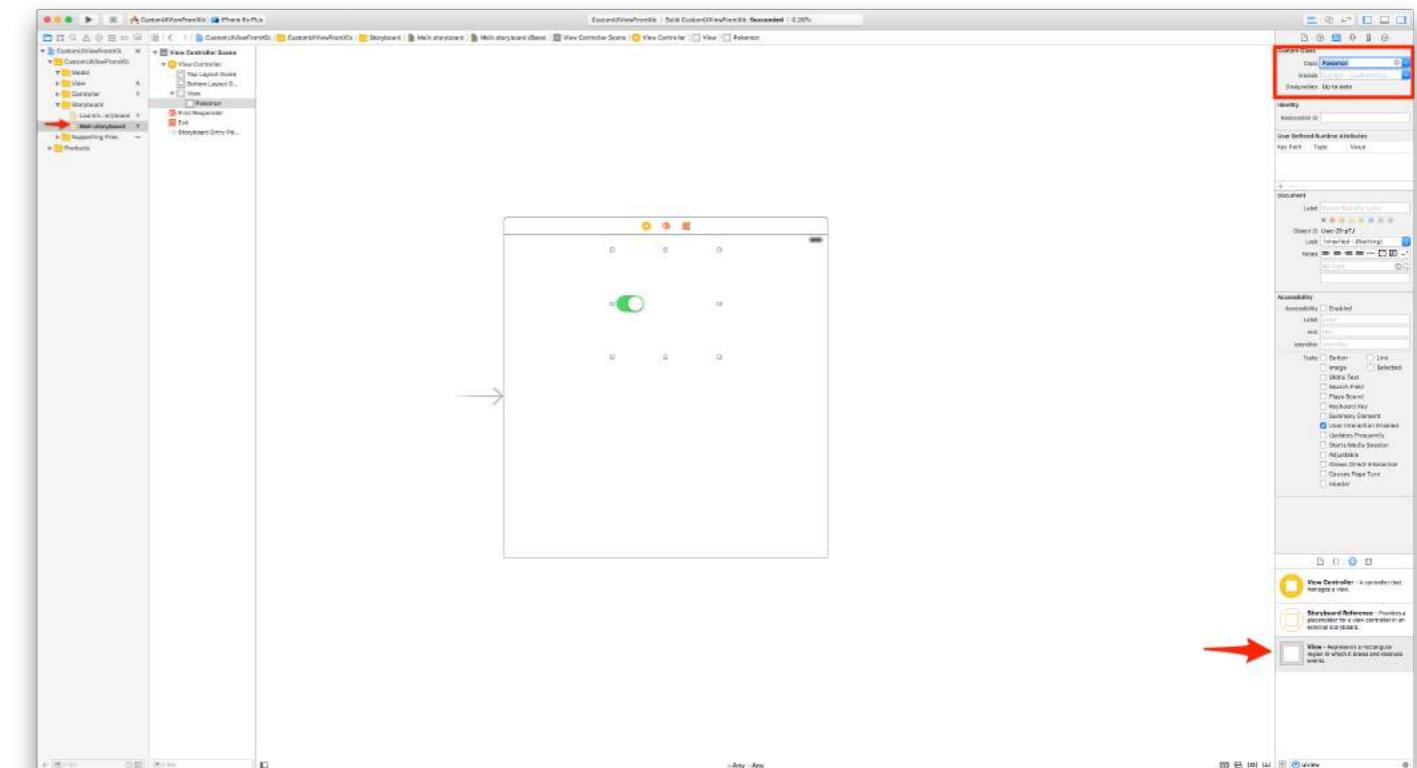
@IBOutlet weak var imageView: UIImageView!

@IBInspectable var image: UIImage? {
    get {
        return imageView.image
    }
    set(image) {
        imageView.image = image
    }
}

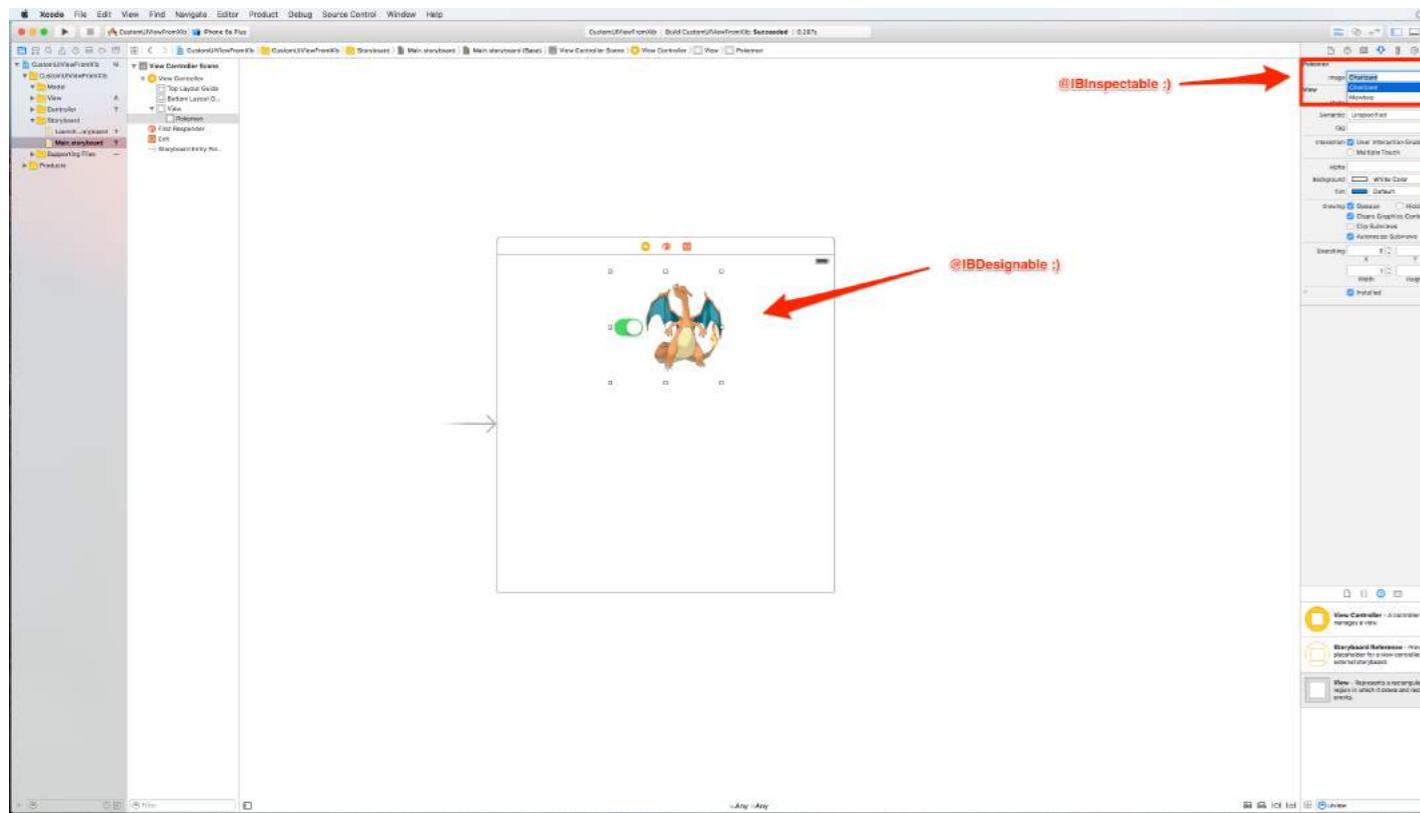
// MARK: - Initializers
...
```

### Using your Custom Views

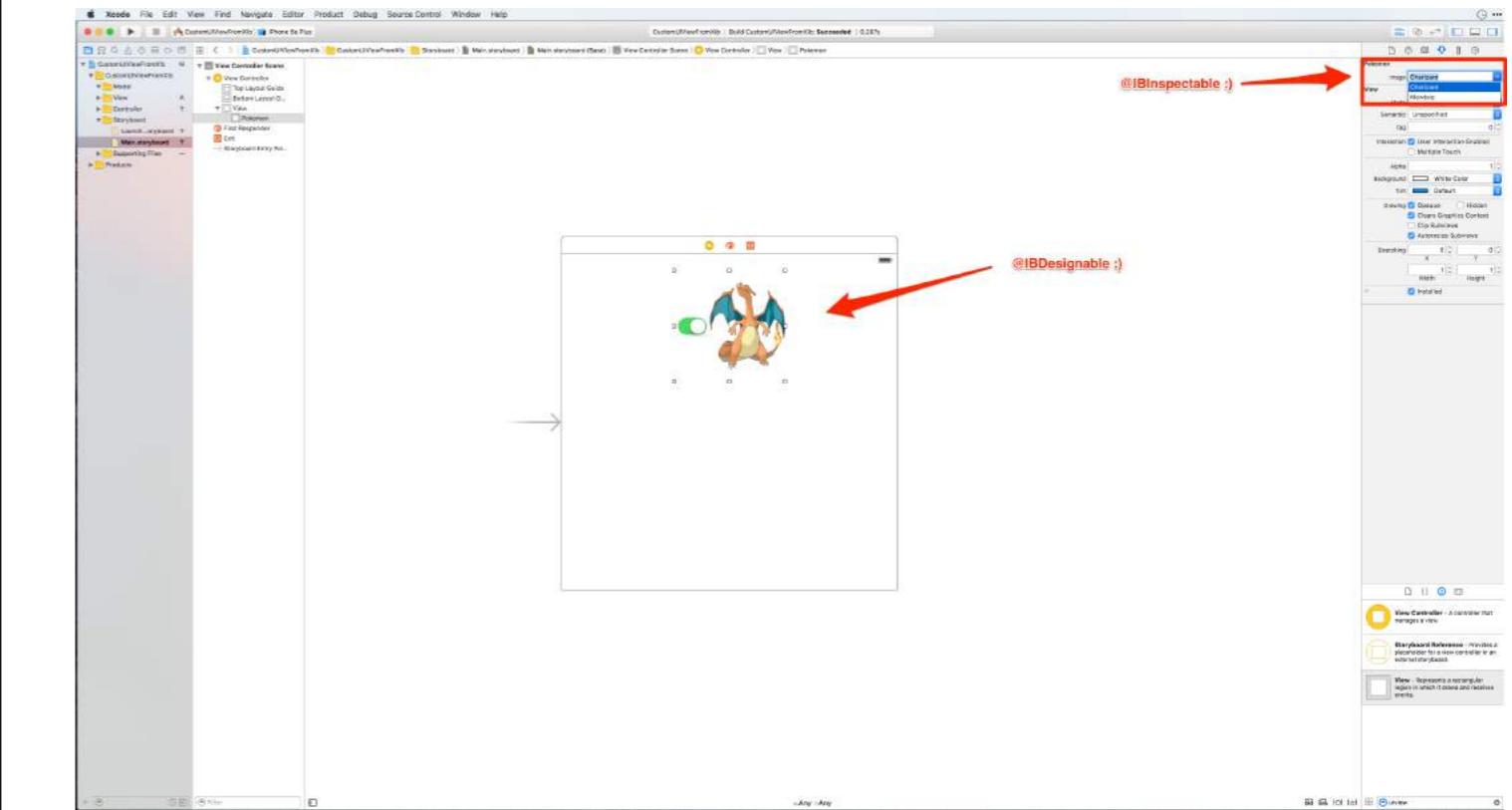
Got to your Main storyboard file, drag a UIView into it.  
Resize the view to, say 200x200. Centralize.  
Go to the Identity inspector (top-right) and set the Class to Pokemon.



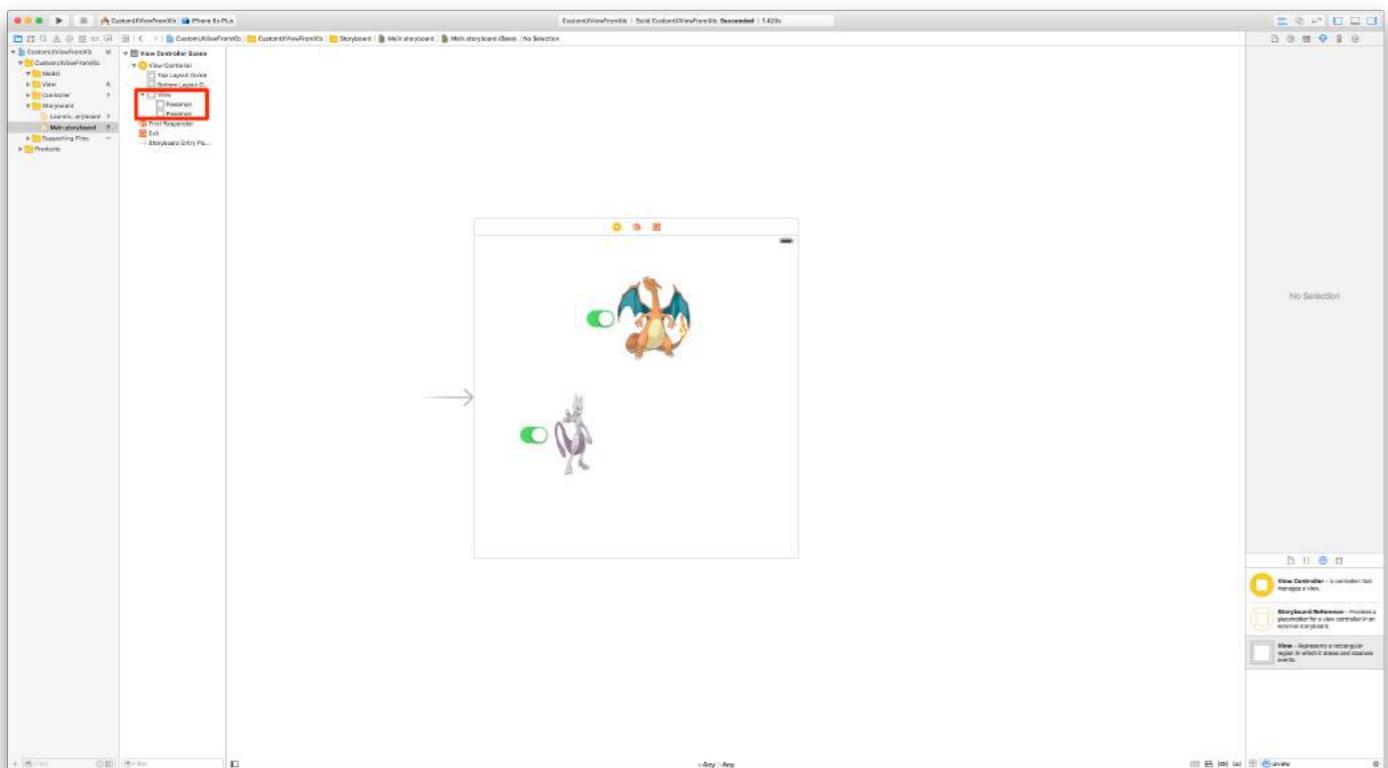
To select a Pokemon, go to the Attribute Inspector (top-right) and select one of the Pokemon images you previously added using the awesome @IBInspectable image property.



现在复制你的自定义宝可梦视图。  
给它一个不同的尺寸，比如150x150。  
选择另一个宝可梦图像，观察：

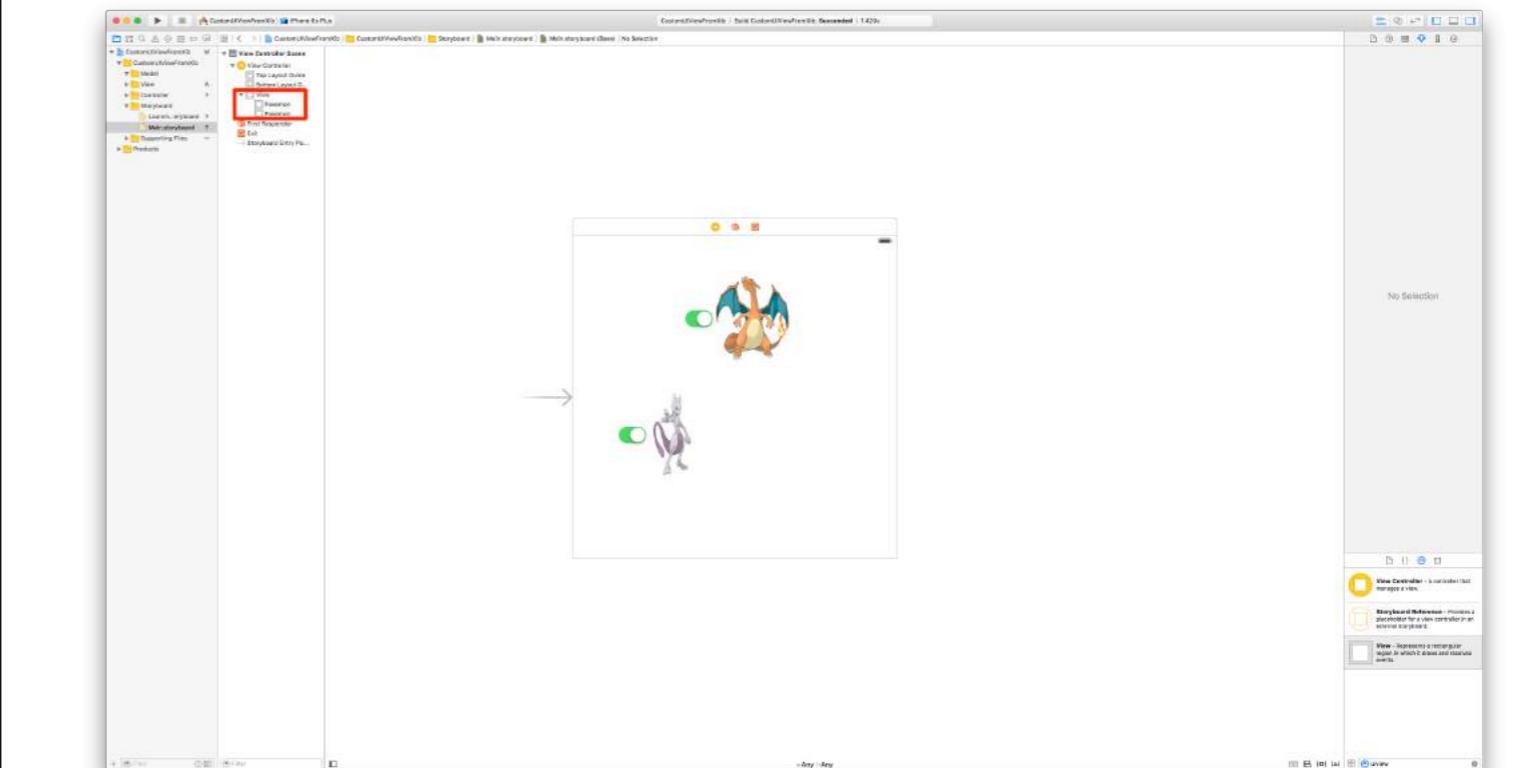


Now duplicate your custom Pokemon view.  
Give it a different size, say 150x150.  
Choose another Pokemon image, observe:



现在我们将为这个自包含的自定义UI元素添加更多逻辑。  
按钮将允许启用/禁用宝可梦。

从开关按钮创建一个IBAction到Pokemon.swift类。  
将该动作命名为类似switchTapped的名称。



Now we are going to add more logic to that self-containing custom UI element.  
The button will allow Pokemons to be enabled/disabled.

Create an IBAction from the Switch button to the Pokemon.swift class.  
Call the action something like switchTapped.

将以下代码添加到其中：

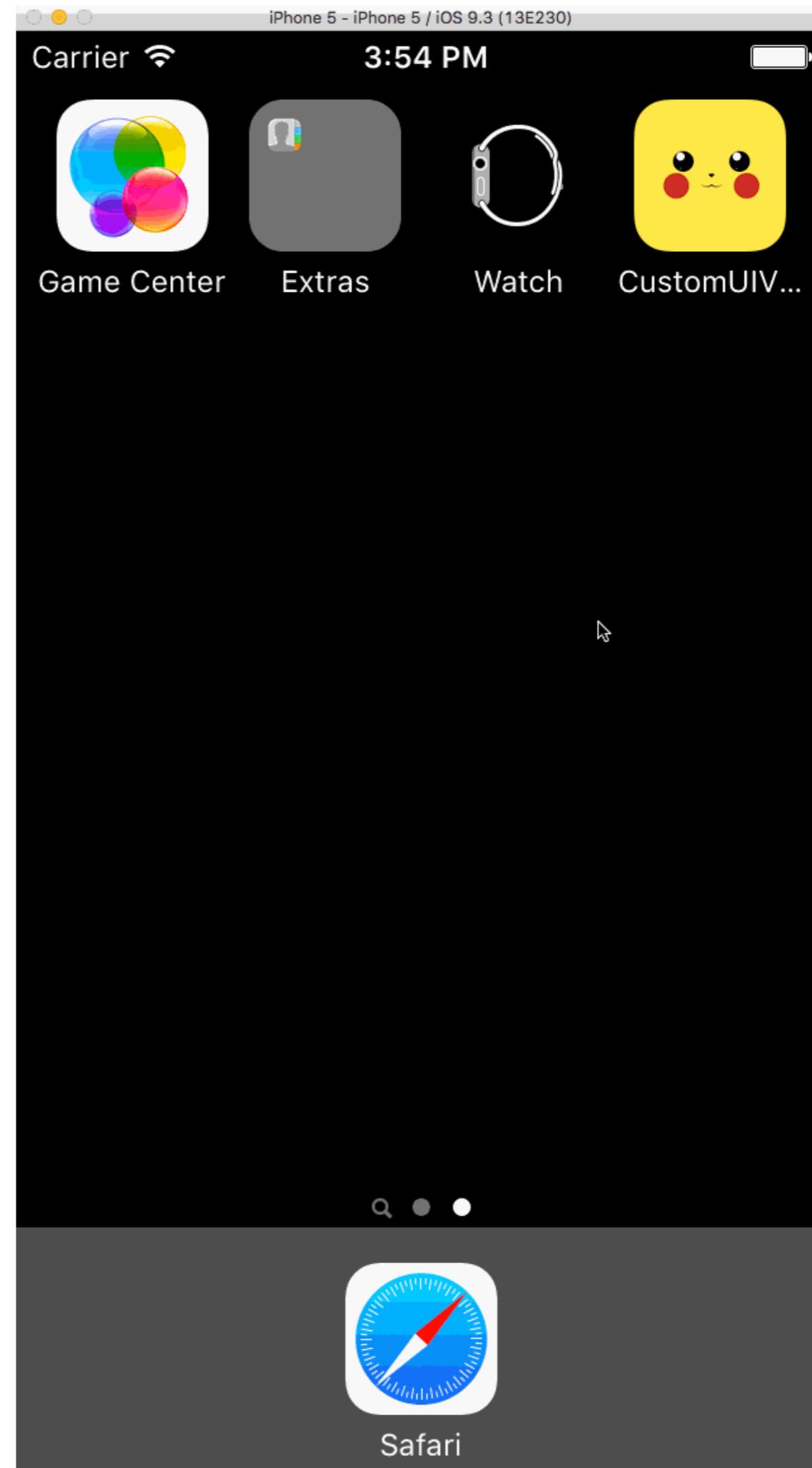
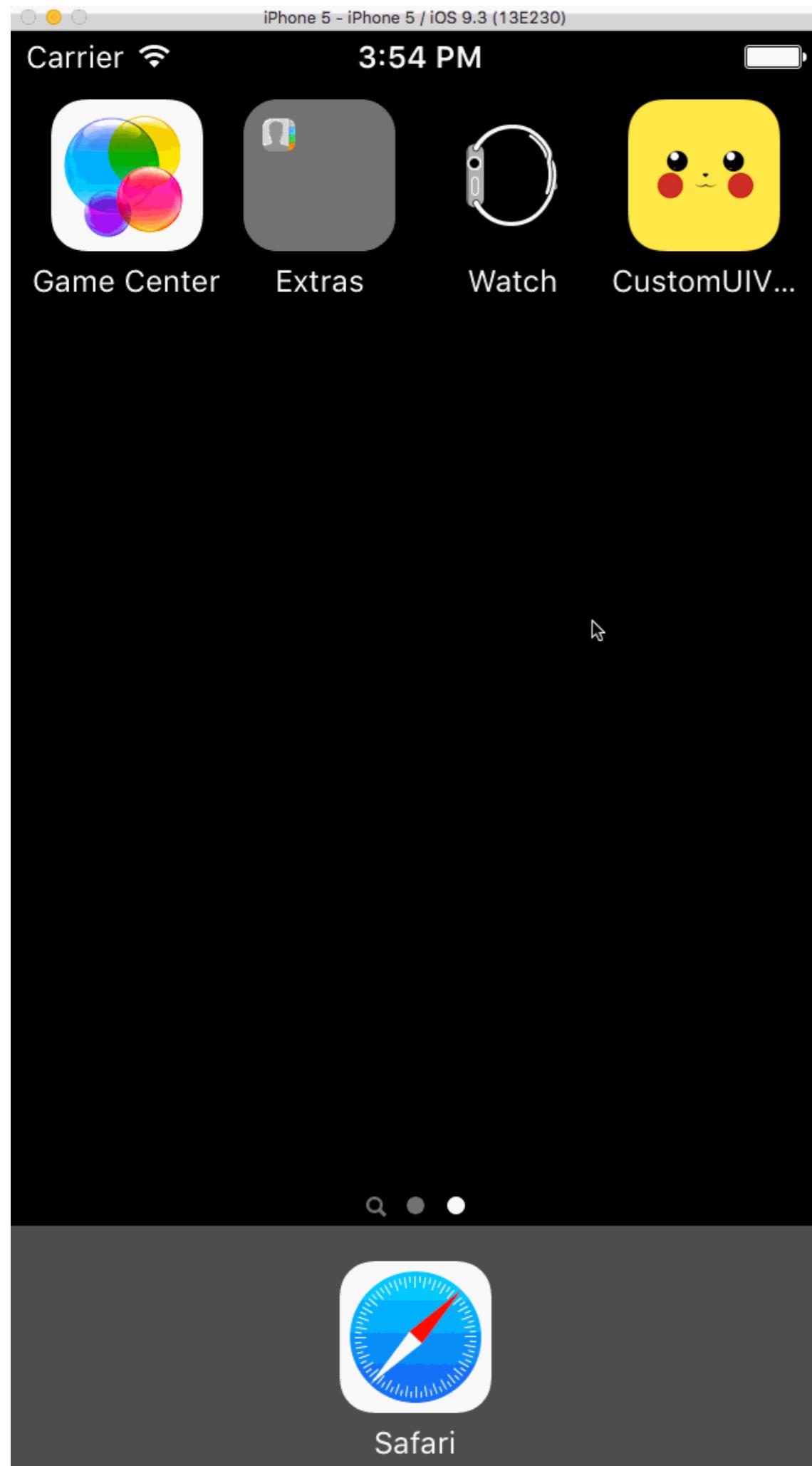
```
// MARK: - 操作  
  
@IBAction func switchTapped(sender: UISwitch) {  
    imageView.alpha = sender.on ? 1.0 : 0.2  
}  
  
// MARK: - 初始化方法  
...
```

最终结果：

Add the following code to it:

```
// MARK: - Actions  
  
@IBAction func switchTapped(sender: UISwitch) {  
    imageView.alpha = sender.on ? 1.0 : 0.2  
}  
  
// MARK: - Initializers  
...
```

Final result:



完成了！

现在你可以创建复杂的自定义视图，并在任何你想要的地方重用它们。

这将提高生产力，同时将代码隔离到自包含的UI元素中。

[最终项目可以在Github上克隆。](#)

(更新至Swift 3.1)

## 第52.2节：如何使用XIB制作自定义可重用UIView

以下示例展示了从XIB初始化视图所涉及的步骤。

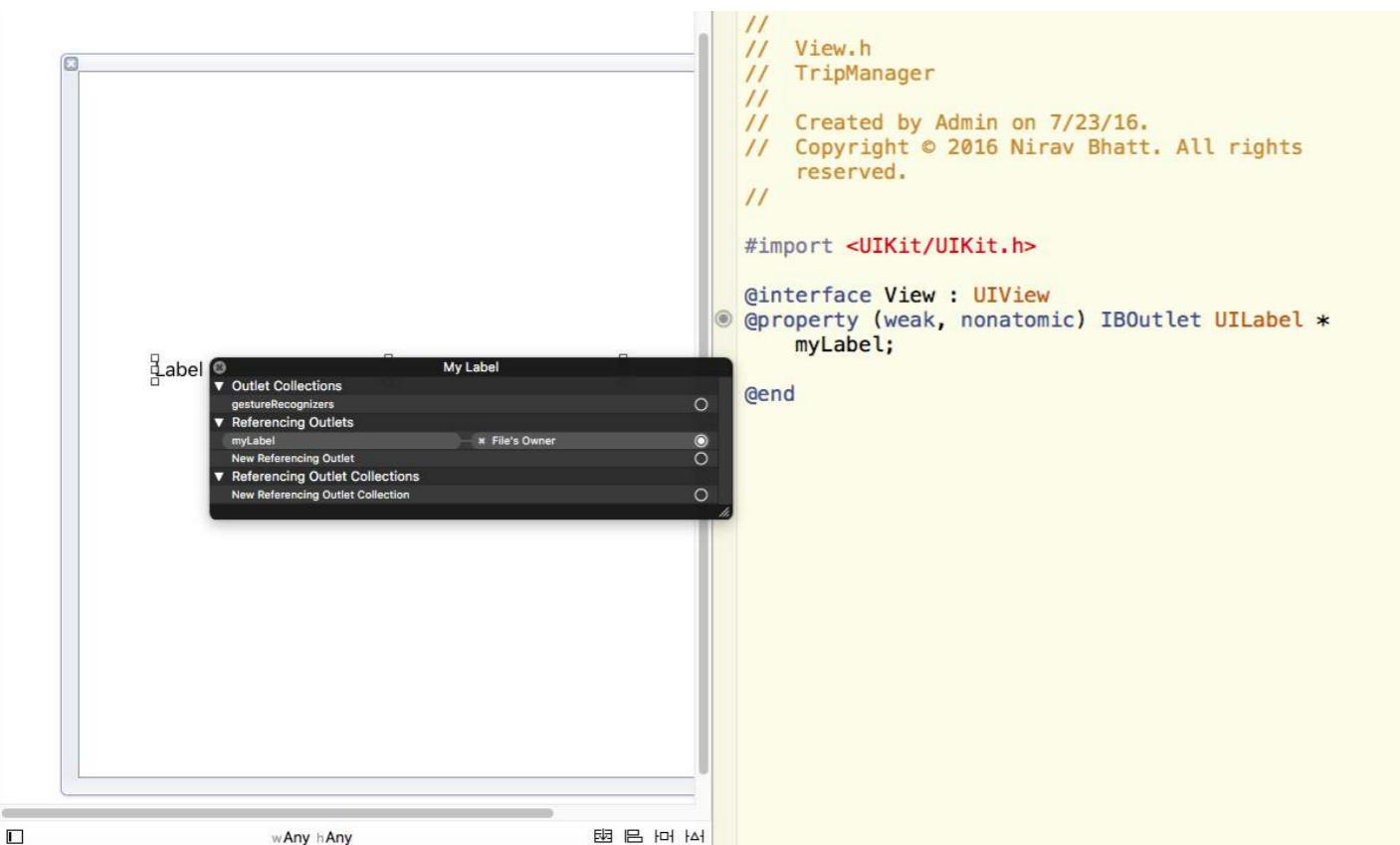
这不是一个复杂的操作，但需要按照准确的步骤进行，才能第一次正确完成，避免出现异常。

[loadNibName的工作原理](#)

主要步骤如下：

1. 创建XIB
2. 创建类的.h和.m文件
3. 在.h中定义outlets
4. 连接.h和XIB之间的outlets

请参见附带的截图：



5. 在.m文件的initWithCoder函数中调用loadNibName。这是为了确保你可以直接将UIView对象放入Storyboard/父UIView XIB文件中，并将其定义为自定义视图。无需其他初始化代码，一旦加载Storyboard/父XIB。你的自定义视图可以像XCode中提供的其他内置Objective C视图对象一样，添加到其他视图中。

You are done!

Now you can create complex custom views and reuse them anywhere you want.

This will increase productivity while isolating code into self-contained UI elements.

[The final project can be cloned in Github.](#)

(Updated to Swift 3.1)

## Section 52.2: How to make custom reusable UIView using XIB

Following example shows steps involved in initializing a view from XIB.

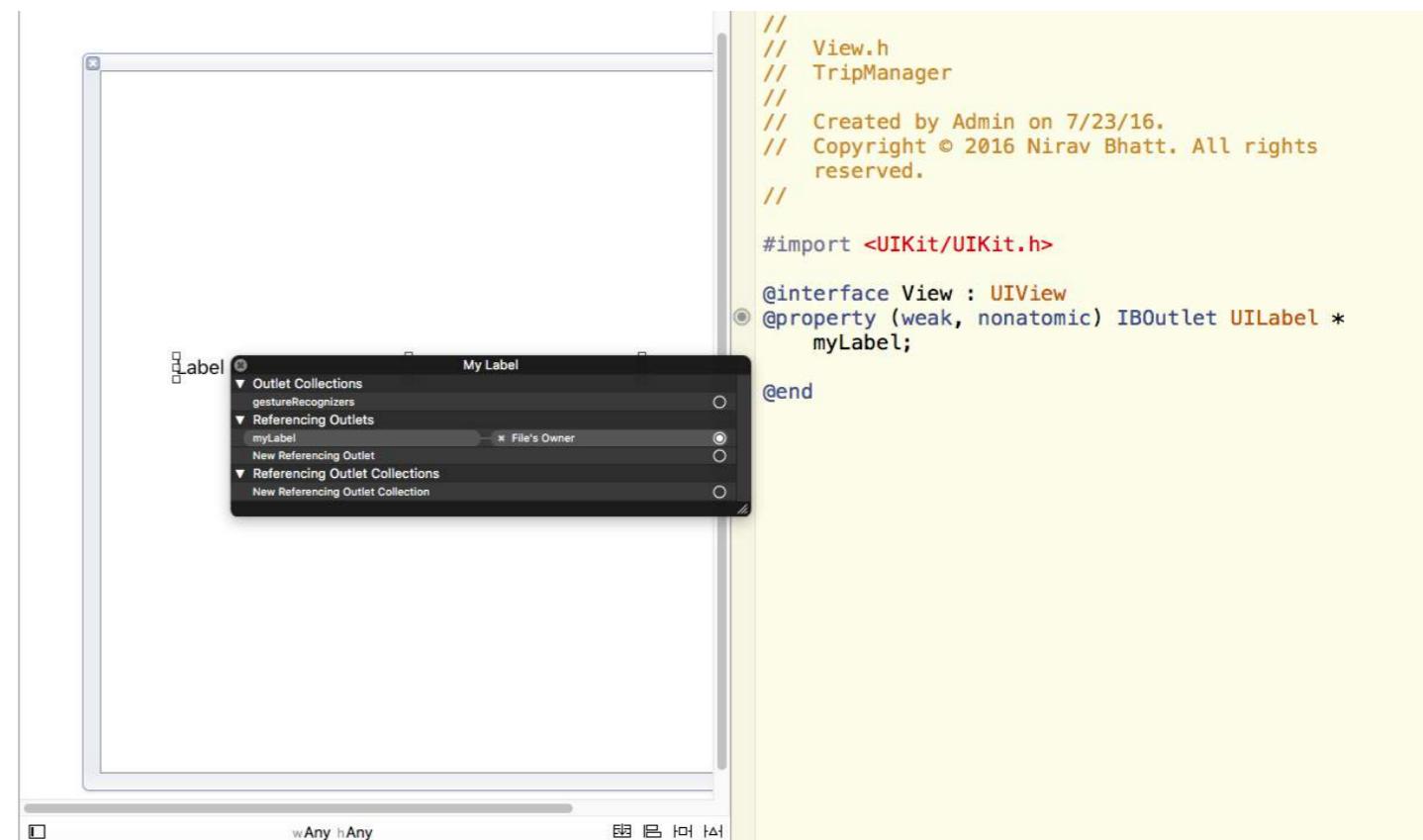
This is not a complex operation but exact steps need to be followed in order to do it right way first time, avoiding exceptions.

[How does loadNibName Works](#)

Main steps are:

1. Create XIB
2. Create class .h and .m
3. Define outlets in .h
4. Connect outlets between .h and XIB

See attached screenshot:



5. Invoke loadNibName inside initWithCoder function of .m file. This is needed to ensure you can directly place UIView object into storyboard / Parent UIView XIB file and define it as your custom view. No other initialization code is needed once you load the storyboard / parent XIB. Your custom view can be added to other views just like other built-in Objective C view objects given in XCode.

# 第53章：UIBezierPath

## 第53.1节：设计和绘制Bezier路径

本示例展示了从设计所需形状到在视图上绘制的过程。使用了一个特定的形状，但你学到的概念可以应用于任何形状。

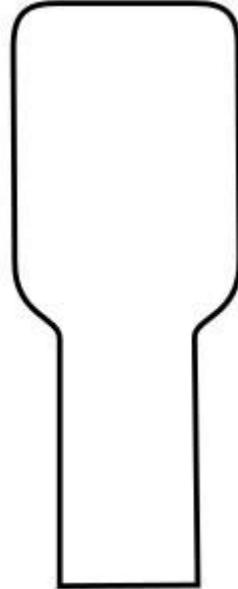
### 如何在自定义视图中绘制Bézier路径

主要步骤如下：

1. 设计你想要的形状轮廓。
2. 将轮廓路径划分为线段、弧线和曲线段。
3. 通过编程构建该路径。
4. 在 `drawRect` 中绘制路径，或使用 `CAShapeLayer`。

### 设计形状轮廓

你可以设计任何形状，但作为示例，我选择了下面的形状。它可以是键盘上的弹出键。



### 将路径划分为多个段

回顾你的形状设计，并将其分解为更简单的元素：直线（用于直线段）、弧线（用于圆形和圆角）以及曲线（用于其他任何形状）。

下面是我们的示例设计示意：

# Chapter 53: UIBezierPath

## Section 53.1: Designing and drawing a Bezier Path

This example shows the process from designing the shape you want to drawing it on a view. A specific shape is used but the concepts you learn can be applied to any shape.

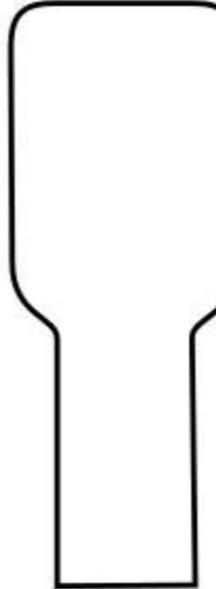
### How to draw a [Bézier path](#) in a custom view

These are the main steps:

1. Design the outline of the shape you want.
2. Divide the outline path into segments of lines, arcs, and curves.
3. Build that path programmatically.
4. Draw the path either in `drawRect` or using a `CAShapeLayer`.

### Design shape outline

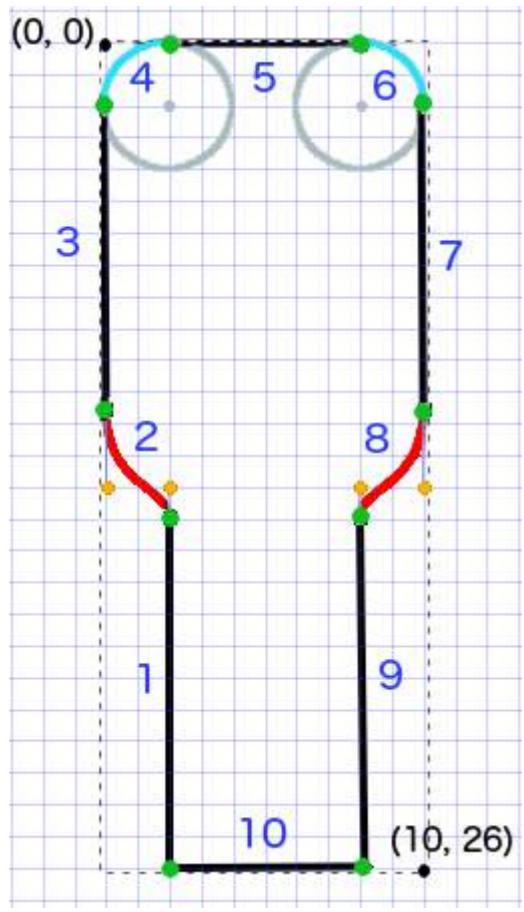
You could do anything, but as an example I have chosen the shape below. It could be a popup key on a keyboard.



### Divide the path into segments

Look back at your shape design and break it down into simpler elements of lines (for straight lines), arcs (for circles and round corners), and curves (for anything else).

Here is what our example design would look like:



- 黑色是线段
- 浅蓝色是弧线段
- 红色是曲线
- 橙色点是曲线的控制点
- 绿色点是路径段之间的点
- 虚线表示边界矩形
- 深蓝色数字表示按程序添加的路径段顺序

#### 通过程序构建路径

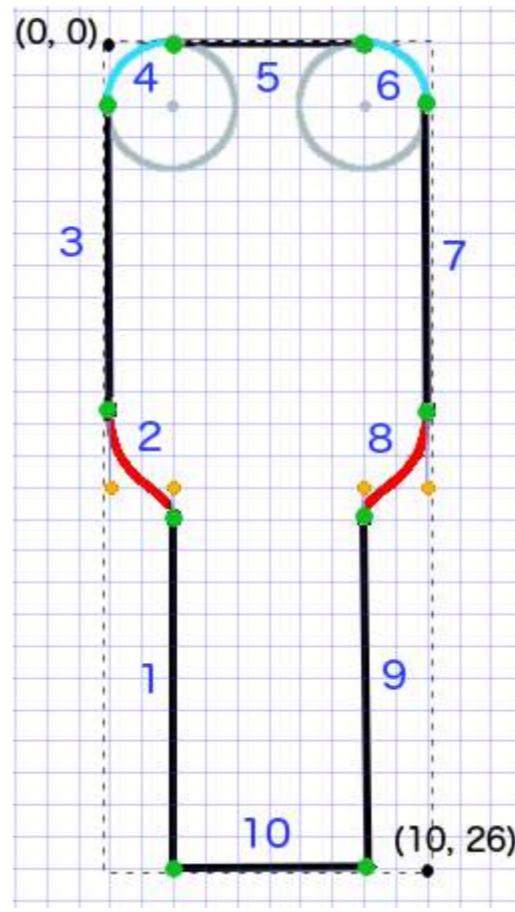
我们将任意从左下角开始，顺时针方向绘制。我会使用图中的网格来获取点的x和y坐标值。这里我会硬编码所有内容，但在实际项目中当然不会这样做。

基本流程是：

1. 创建一个新的UIBezierPath
2. 用moveToPoint选择路径的起点
3. 向路径添加段
  - 线段：addLineToPoint
  - 弧线：addArcWithCenter
  - 曲线：addCurveToPoint
4. 用closePath闭合路径

以下是制作上图路径的代码。

```
func createBezierPath() -> UIBezierPath {
    // 创建一个新路径
    let path = UIBezierPath()
```



- Black are line segments
- Light blue are arc segments
- Red are curves
- Orange dots are the control points for the curves
- Green dots are the points between path segments
- Dotted lines show the bounding rectangle
- Dark blue numbers are the segments in the order that they will be added programmatically

#### Build the path programmatically

We'll arbitrarily start in the bottom left corner and work clockwise. I'll use the grid in the image to get the x and y values for the points. I'll hardcode everything here, but of course you wouldn't do that in a real project.

The basic process is:

1. Create a new `UIBezierPath`
2. Choose a starting point on the path with `moveToPoint`
3. Add segments to the path
  - line: `addLineToPoint`
  - arc: `addArcWithCenter`
  - curve: `addCurveToPoint`
4. Close the path with `closePath`

Here is the code to make the path in the image above.

```
func createBezierPath() -> UIBezierPath {
    // create a new path
    let path = UIBezierPath()
```

```

// 路径起点 (左下角)
path.moveToPoint(CGPoint(x: 2, y: 26))

// *****
// ***** 左侧 *****
// *****

// 段1：线段
path.addLineToPoint(CGPoint(x: 2, y: 15))

// 段2：曲线
path.addCurveToPoint(CGPoint(x: 0, y: 12), // 终点
    controlPoint1: CGPoint(x: 2, y: 14),
controlPoint2: CGPoint(x: 0, y: 14))

// 段 3：线段
path.addLineToPoint(CGPoint(x: 0, y: 2))

// *****
// ***** 顶部边 *****
// *****

// 段 4：弧线
path.addArcWithCenter(CGPoint(x: 2, y: 2), // 圆心坐标
    radius: 2, // 使其与路径线相接
startAngle: CGFloat(M_PI), // π 弧度 = 180 度 = 正左
    endAngle: CGFloat(3*M_PI_2), // 3π/2 弧度 = 270 度 = 正上
    clockwise: true) // 从起始角到结束角顺时针方向

// 段 5：线段
path.addLineToPoint(CGPoint(x: 8, y: 0))

// 段 6：弧线
path.addArcWithCenter(CGPoint(x: 8, y: 2),
    radius: 2,
startAngle: CGFloat(3*M_PI_2), // 正上
    endAngle: CGFloat(0), // 0 弧度 = 正右
    clockwise: true)

// *****
// ***** 右侧 *****
// *****

// 段 7：线
path.addLineToPoint(CGPoint(x: 10, y: 12))

// 段 8：曲线
path.addCurveToPoint(CGPoint(x: 8, y: 15), // 终点
    controlPoint1: CGPoint(x: 10, y: 14),
controlPoint2: CGPoint(x: 8, y: 14))

// 段 9：线
path.addLineToPoint(CGPoint(x: 8, y: 26))

// *****
// *** 底部 ***
// *****

// 段 10：线
path.closePath() // 绘制闭合路径的最后一条线

```

```

// starting point for the path (bottom left)
path.moveToPoint(CGPoint(x: 2, y: 26))

// *****
// ***** Left side *****
// *****

// segment 1: line
path.addLineToPoint(CGPoint(x: 2, y: 15))

// segment 2: curve
path.addCurveToPoint(CGPoint(x: 0, y: 12), // ending point
    controlPoint1: CGPoint(x: 2, y: 14),
controlPoint2: CGPoint(x: 0, y: 14))

// segment 3: line
path.addLineToPoint(CGPoint(x: 0, y: 2))

// *****
// ***** Top side *****
// *****

// segment 4: arc
path.addArcWithCenter(CGPoint(x: 2, y: 2), // center point of circle
    radius: 2, // this will make it meet our path line
startAngle: CGFloat(M_PI), // π radians = 180 degrees = straight left
    endAngle: CGFloat(3*M_PI_2), // 3π/2 radians = 270 degrees = straight up
    clockwise: true) // startAngle to endAngle goes in a clockwise direction

// segment 5: line
path.addLineToPoint(CGPoint(x: 8, y: 0))

// segment 6: arc
path.addArcWithCenter(CGPoint(x: 8, y: 2),
    radius: 2,
startAngle: CGFloat(3*M_PI_2), // straight up
endAngle: CGFloat(0), // 0 radians = straight right
clockwise: true)

// *****
// ***** Right side *****
// *****

// segment 7: line
path.addLineToPoint(CGPoint(x: 10, y: 12))

// segment 8: curve
path.addCurveToPoint(CGPoint(x: 8, y: 15), // ending point
    controlPoint1: CGPoint(x: 10, y: 14),
controlPoint2: CGPoint(x: 8, y: 14))

// segment 9: line
path.addLineToPoint(CGPoint(x: 8, y: 26))

// *****
// *** Bottom side ***
// *****

// segment 10: line
path.closePath() // draws the final line to close the path

```

```
    return path  
}
```

注意：上述部分代码可以通过在单个命令中添加一条线和一条弧来简化（因为弧有一个隐含的起点）。详情请参见 [here](#)。

## 绘制路径

我们可以在图层中或在 `drawRect` 中绘制路径。

### 方法一：在图层中绘制路径

我们的自定义类如下。视图初始化时，我们将贝塞尔路径添加到一个新的 `CAShapeLayer` 中。

```
import UIKit  
class MyCustomView: UIView {  
  
    override init(frame: CGRect) {  
        super.init(frame: frame)  
        setup()  
    }  
  
    required init?(coder aDecoder: NSCoder) {  
        super.init(coder: aDecoder)  
        setup()  
    }  
  
    func setup() {  
  
        // 创建一个 CAShapeLayer  
        let shapeLayer = CAShapeLayer()  
  
        // 我们创建的贝塞尔路径需要转换为  
        // CGPath 才能用于图层。  
        shapeLayer.path = createBezierPath().CGPath  
  
        // 应用与路径相关的其他属性  
        shapeLayer.strokeColor = UIColor.blueColor().CGColor  
        shapeLayer.fillColor = UIColor.whiteColor().CGColor  
        shapeLayer.lineWidth = 1.0  
        shapeLayer.position = CGPointMake(x: 10, y: 10)  
  
        // 将新图层添加到我们的自定义视图中  
        self.layer.addSublayer(shapeLayer)  
    }  
  
    func createBezierPath() -> UIBezierPath {  
  
        // 参见之前创建贝塞尔路径的代码  
    }  
}
```

并且在视图控制器中这样创建我们的视图

```
override func viewDidLoad() {  
    super.viewDidLoad()  
  
    // 创建一个新的 UIView 并将其添加到视图控制器中  
    let myView = MyCustomView()  
    myView.frame = CGRectMake(x: 100, y: 100, width: 50, height: 50)
```

```
    return path  
}
```

Note: Some of the above code can be reduced by adding a line and an arc in a single command (since the arc has an implied starting point). See [here](#) for more details.

## Draw the path

We can draw the path either in a layer or in `drawRect`.

### Method 1: Draw path in a layer

Our custom class looks like this. We add our Bezier path to a new `CAShapeLayer` when the view is initialized.

```
import UIKit  
class MyCustomView: UIView {  
  
    override init(frame: CGRect) {  
        super.init(frame: frame)  
        setup()  
    }  
  
    required init?(coder aDecoder: NSCoder) {  
        super.init(coder: aDecoder)  
        setup()  
    }  
  
    func setup() {  
  
        // Create a CAShapeLayer  
        let shapeLayer = CAShapeLayer()  
  
        // The Bezier path that we made needs to be converted to  
        // a CGPath before it can be used on a layer.  
        shapeLayer.path = createBezierPath().CGPath  
  
        // apply other properties related to the path  
        shapeLayer.strokeColor = UIColor.blueColor().CGColor  
        shapeLayer.fillColor = UIColor.whiteColor().CGColor  
        shapeLayer.lineWidth = 1.0  
        shapeLayer.position = CGPointMake(x: 10, y: 10)  
  
        // add the new layer to our custom view  
        self.layer.addSublayer(shapeLayer)  
    }  
  
    func createBezierPath() -> UIBezierPath {  
  
        // see previous code for creating the Bezier path  
    }  
}
```

And creating our view in the View Controller like this

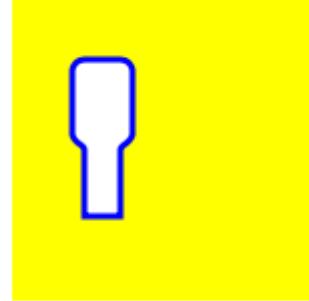
```
override func viewDidLoad() {  
    super.viewDidLoad()  
  
    // create a new UIView and add it to the view controller  
    let myView = MyCustomView()  
    myView.frame = CGRectMake(x: 100, y: 100, width: 50, height: 50)
```

```

myView.backgroundColor = UIColor.yellowColor()
view.addSubview(myView)
}

```

我们得到...

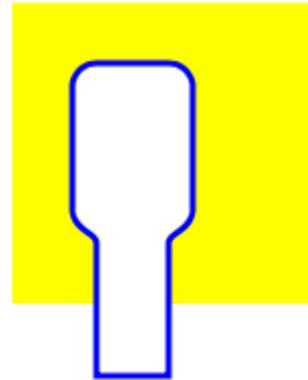


嗯，这有点小，因为我把所有数字都写死了。不过我可以这样放大路径的大小：

```

let path = createBezierPath()
let scale = CGAffineTransformMakeScale(2, 2)
path.applyTransform(scale)
shapeLayer.path = path.CGPath

```



## 方法二：在 drawRect 中绘制路径

使用 drawRect 绘制到图层要慢，所以如果不需要，建议不要用这种方法。

这是我们自定义视图的修改代码：

```

import UIKit
class MyCustomView: UIView {

    override func drawRect(rect: CGRect) {

        // 创建路径 (参见前面的代码)
        let path = createBezierPath()

        // 填充
        let fillColor = UIColor.whiteColor()
        fillColor.setFill()

        // 描边
        path.lineWidth = 1.0
        let strokeColor = UIColor.blueColor()

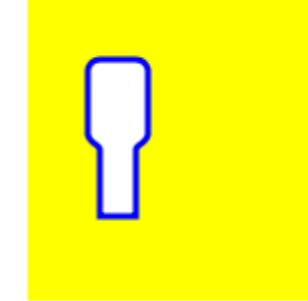
```

```

myView.backgroundColor = UIColor.yellowColor()
view.addSubview(myView)
}

```

We get...

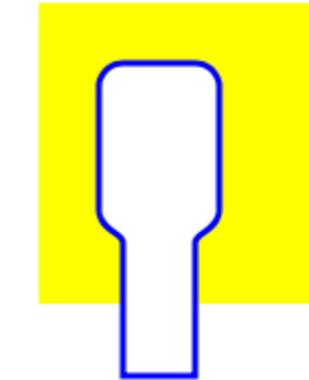


Hmm, that's a little small because I hardcoded all the numbers in. I can scale the path size up, though, like this:

```

let path = createBezierPath()
let scale = CGAffineTransformMakeScale(2, 2)
path.applyTransform(scale)
shapeLayer.path = path.CGPath

```



## Method 2: Draw path in drawRect

Using drawRect is slower than drawing to the layer, so this is not the recommended method if you don't need it.

Here is the revised code for our custom view:

```

import UIKit
class MyCustomView: UIView {

    override func drawRect(rect: CGRect) {

        // create path (see previous code)
        let path = createBezierPath()

        // fill
        let fillColor = UIColor.whiteColor()
        fillColor.setFill()

        // stroke
        path.lineWidth = 1.0
        let strokeColor = UIColor.blueColor()

```

```

strokeColor.setStroke()

    // 将路径移动到新位置
path.applyTransform(CGAffineTransformMakeTranslation(10, 10))

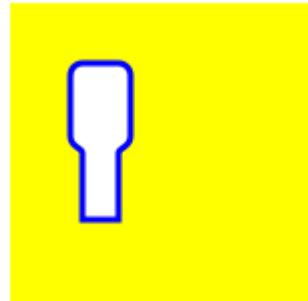
    // 填充并描边路径 (总是最后执行这些操作)
    path.fill()
path.stroke()

}

func createBezierPath() -> UIBezierPath {
    // 参见之前创建贝塞尔路径的代码
}
}

```

这给了我们相同的结果.....



## 进一步学习

理解贝塞尔路径的优秀文章。

- [像贝塞尔路径一样思考](#) (我读过的这位作者的所有内容都很棒，我上面示例的灵感也来自这里。)
- [编程数学：第19集 - 贝塞尔曲线](#) (有趣且配良好的视觉示例)
- [贝塞尔曲线](#) (它们在图形应用中的使用方式)
- [贝塞尔曲线](#) (关于数学公式推导的良好描述)

## 注意事项

- 此示例最初来自这个 [Stack Overflow 回答](#)。
- 在实际项目中，你可能不应该使用硬编码数字，而是应从视图的边界中获取尺寸。

## 第53.2节：如何将圆角半径应用于由UIBezierPath绘制的矩形

所有四个边的圆角半径：



```
strokeColor.setStroke()
```

```

// Move the path to a new location
path.applyTransform(CGAffineTransformMakeTranslation(10, 10))

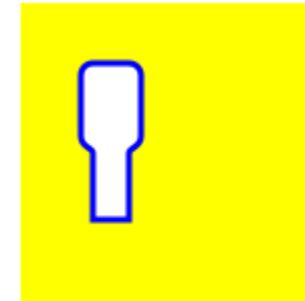
// fill and stroke the path (always do these last)
path.fill()
path.stroke()

}

func createBezierPath() -> UIBezierPath {
    // see previous code for creating the Bezier path
}
}

```

which gives us the same result...



## Further study

Excellent articles for understanding Bezier paths.

- [Thinking like a Bézier path](#) (Everything I've ever read from this author is good and the inspiration for my example above came from here.)
- [Coding Math: Episode 19 - Bezier Curves](#) (entertaining and good visual illustrations)
- [Bezier Curves](#) (how they are used in graphics applications)
- [Bezier Curves](#) (good description of how the mathematical formulas are derived)

## Notes

- This example originally comes from [this Stack Overflow answer](#).
- In your actual projects you probably shouldn't use hard coded numbers, but rather get the sizes from your view's bounds.

## Section 53.2: How to apply corner radius to rectangles drawn by UIBezierPath

Corner radius for all 4 edges:



```
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRoundedRect:  
CGRectMake(x,y,width,height) cornerRadius: 11];  
[UIColor.grayColor setFill];  
[rectanglePath fill];
```

左上角的圆角半径：



```
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRoundedRect:  
CGRectMake(x,y,width,height) byRoundingCorners: UIRectCornerTopLeft cornerRadii: CGSizeMake(11,  
11)];  
[rectanglePath closePath];  
[UIColor.grayColor setFill];  
[rectanglePath fill];
```

右上角的圆角半径：



```
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRoundedRect: CGRectMake(x,y,width,height)  
byRoundingCorners: UIRectCornerTopRight cornerRadii: CGSizeMake(11, 11)];  
[rectanglePath closePath];  
[UIColor.grayColor setFill];  
[rectanglePath fill];
```

左下角的圆角半径：



```
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRoundedRect: CGRectMake(x,y,width,height)  
byRoundingCorners: UIRectCornerBottomLeft cornerRadii: CGSizeMake(11, 11)];  
[rectanglePath closePath];  
[UIColor.grayColor setFill];  
[rectanglePath fill];
```

右下角的圆角半径：

```
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRoundedRect:  
CGRectMake(x,y,width,height) cornerRadius: 11];  
[UIColor.grayColor setFill];  
[rectanglePath fill];
```

Corner radius for top-left edge:



```
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRoundedRect:  
CGRectMake(x,y,width,height) byRoundingCorners: UIRectCornerTopLeft cornerRadii: CGSizeMake(11,  
11)];  
[rectanglePath closePath];  
[UIColor.grayColor setFill];  
[rectanglePath fill];
```

Corner radius for top-right edge:



```
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRoundedRect: CGRectMake(x,y,width,height)  
byRoundingCorners: UIRectCornerTopRight cornerRadii: CGSizeMake(11, 11)];  
[rectanglePath closePath];  
[UIColor.grayColor setFill];  
[rectanglePath fill];
```

corner radius for bottom-left edge:



```
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRoundedRect: CGRectMake(x,y,width,height)  
byRoundingCorners: UIRectCornerBottomLeft cornerRadii: CGSizeMake(11, 11)];  
[rectanglePath closePath];  
[UIColor.grayColor setFill];  
[rectanglePath fill];
```

corner radius for bottom-right edge:



```
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRoundedRect:  
    CGRectMake(x,y,width,height) byRoundingCorners: UIRectCornerBottomRight cornerRadii: CGSizeMake(11,  
    11)];  
[rectanglePath closePath];  
[UIColor.grayColor setFill];  
[rectanglePath fill];
```

底部边缘的圆角半径：



```
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRoundedRect: CGRectMake(x,y,width,height)  
byRoundingCorners: UIRectCornerBottomLeft | UIRectCornerBottomRight cornerRadii: CGSizeMake(11,  
11)];  
[rectanglePath closePath];  
[UIColor.grayColor setFill];  
[rectanglePath fill];
```

顶部边缘的圆角半径：



```
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRoundedRect: CGRectMake(x,y,width,height)  
byRoundingCorners: UIRectCornerTopLeft | UIRectCornerTopRight cornerRadii: CGSizeMake(11, 11)];  
[rectanglePath closePath];  
[UIColor.grayColor setFill];  
[rectanglePath fill];
```

### 第53.3节：如何为UIBezierPath应用阴影

考虑一个由贝塞尔路径绘制的简单矩形。



```
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRoundedRect:  
    CGRectMake(x,y,width,height) byRoundingCorners: UIRectCornerBottomRight cornerRadii: CGSizeMake(11,  
    11)];  
[rectanglePath closePath];  
[UIColor.grayColor setFill];  
[rectanglePath fill];
```

*corner radius for bottom edges:*



```
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRoundedRect: CGRectMake(x,y,width,height)  
byRoundingCorners: UIRectCornerBottomLeft | UIRectCornerBottomRight cornerRadii: CGSizeMake(11,  
11)];  
[rectanglePath closePath];  
[UIColor.grayColor setFill];  
[rectanglePath fill];
```

*corner radius for top edges:*



```
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRoundedRect: CGRectMake(x,y,width,height)  
byRoundingCorners: UIRectCornerTopLeft | UIRectCornerTopRight cornerRadii: CGSizeMake(11, 11)];  
[rectanglePath closePath];  
[UIColor.grayColor setFill];  
[rectanglePath fill];
```

### Section 53.3: How to apply shadows to UIBezierPath

Consider a simple rectangle that is drawn by the bezier path.



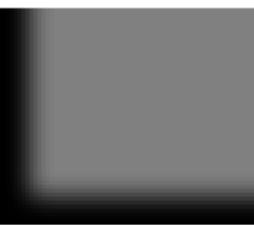
```
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRect: CGRectMake(x,y,width,height)];  
[UIColor.grayColor setFill];  
[rectanglePath fill];
```

基本外部填充阴影：



```
CGContextRef context = UIGraphicsGetCurrentContext();  
  
NSShadow* shadow = [[NSShadow alloc] init];  
[shadow setShadowColor: UIColor.blackColor];  
[shadow setShadowOffset: CGSizeMake(7.1, 5.1)];  
[shadow setShadowBlurRadius: 5];  
  
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRect: CGRectMake(x,y,width,height)];  
CGContextSaveGState(context);  
CGContextSetShadowWithColor(context, shadow.shadowOffset, shadow.shadowBlurRadius,  
[shadow.shadowColor CGColor]);  
[UIColor.grayColor setFill];  
[rectanglePath fill];  
CGContextRestoreGState(context);
```

基本内部填充阴影：



```
CGContextRef context = UIGraphicsGetCurrentContext();  
  
NSShadow* shadow = [[NSShadow alloc] init];  
[shadow setShadowColor: UIColor.blackColor];  
[shadow setShadowOffset: CGSizeMake(9.1, -7.1)];  
[shadow setShadowBlurRadius: 6];
```



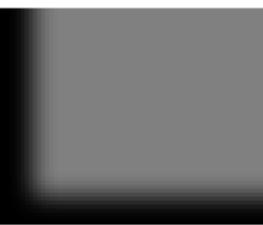
```
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRect: CGRectMake(x,y,width,height)];  
[UIColor.grayColor setFill];  
[rectanglePath fill];
```

Basic Outer-fill shadow:



```
CGContextRef context = UIGraphicsGetCurrentContext();  
  
NSShadow* shadow = [[NSShadow alloc] init];  
[shadow setShadowColor: UIColor.blackColor];  
[shadow setShadowOffset: CGSizeMake(7.1, 5.1)];  
[shadow setShadowBlurRadius: 5];  
  
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRect: CGRectMake(x,y,width,height)];  
CGContextSaveGState(context);  
CGContextSetShadowWithColor(context, shadow.shadowOffset, shadow.shadowBlurRadius,  
[shadow.shadowColor CGColor]);  
[UIColor.grayColor setFill];  
[rectanglePath fill];  
CGContextRestoreGState(context);
```

Basic Inner fill shadow:



```
CGContextRef context = UIGraphicsGetCurrentContext();  
  
NSShadow* shadow = [[NSShadow alloc] init];  
[shadow setShadowColor: UIColor.blackColor];  
[shadow setShadowOffset: CGSizeMake(9.1, -7.1)];  
[shadow setShadowBlurRadius: 6];
```

```

UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRect: CGRectMake(x,y,width,height)];
[UIColor.grayColor setFill];
[rectanglePath fill];

CGContextSaveGState(context);
UIRectClip(rectanglePath.bounds);
CGContextSetShadowWithColor(context, CGSizeZero, 0, NULL);

CGContextSetAlpha(context, CGColorGetAlpha([shadow.shadowColor CGColor]));
CGContextBeginTransparencyLayer(context, NULL);
{
    UIColor* opaqueShadow = [shadow.shadowColor colorWithAlphaComponent: 1];
    CGContextSetShadowWithColor(context, shadow.shadowOffset, shadow.shadowBlurRadius,
    [opaqueShadow CGColor]);
    CGContextSetBlendMode(context, kCGBlendModeSourceOut);
    CGContextBeginTransparencyLayer(context, NULL);

    [opaqueShadow setFill];
    [rectanglePath fill];

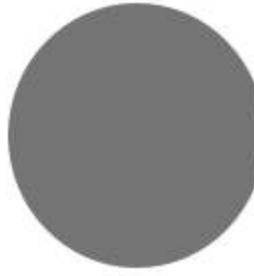
    CGContextEndTransparencyLayer(context);
}
CGContextEndTransparencyLayer(context);
CGContextRestoreGState(context);

```

## 第53.4节：如何使用

**UIBezierPath**创建简单形状

对于一个简单的圆形：



```

UIBezierPath* ovalPath = [UIBezierPath bezierPathWithOvalInRect: CGRectMake(0,0,50,50)];
[UIColor.grayColor setFill];
[ovalPath fill];

```

Swift :

```

let ovalPath = UIBezierPath(ovalInRect: CGRect(x: 0, y: 0, width: 50, height: 50))
UIColor.grayColor().setFill()
ovalPath.fill()

```

对于一个简单的矩形：

```

UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRect: CGRectMake(x,y,width,height)];
[UIColor.grayColor setFill];
[rectanglePath fill];

CGContextSaveGState(context);
UIRectClip(rectanglePath.bounds);
CGContextSetShadowWithColor(context, CGSizeZero, 0, NULL);

CGContextSetAlpha(context, CGColorGetAlpha([shadow.shadowColor CGColor]));
CGContextBeginTransparencyLayer(context, NULL);
{
    UIColor* opaqueShadow = [shadow.shadowColor colorWithAlphaComponent: 1];
    CGContextSetShadowWithColor(context, shadow.shadowOffset, shadow.shadowBlurRadius,
    [opaqueShadow CGColor]);
    CGContextSetBlendMode(context, kCGBlendModeSourceOut);
    CGContextBeginTransparencyLayer(context, NULL);

    [opaqueShadow setFill];
    [rectanglePath fill];

    CGContextEndTransparencyLayer(context);
}
CGContextEndTransparencyLayer(context);
CGContextRestoreGState(context);

```

## Section 53.4: How to create a simple shapes using UIBezierPath

*For a simple circle:*



```

UIBezierPath* ovalPath = [UIBezierPath bezierPathWithOvalInRect: CGRectMake(0,0,50,50)];
[UIColor.grayColor setFill];
[ovalPath fill];

```

Swift:

```

let ovalPath = UIBezierPath(ovalInRect: CGRect(x: 0, y: 0, width: 50, height: 50))
UIColor.grayColor().setFill()
ovalPath.fill()

```

*For a simple Rectangle:*



```
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRect: CGRectMake(0, 0, 50, 50)];
[UIColor.grayColor setFill];
[rectanglePath fill];
```

Swift:

```
let rectanglePath = UIBezierPath(rect: CGRect(x: 0, y: 0, width: 50, height: 50))
UIColor.grayColor().setFill()
rectanglePath.fill()
```

对于一条简单的线：

---

```
UIBezierPath* bezierPath = [UIBezierPath bezierPath];
[bezierPath moveToPoint: CGPointMake(x1,y1)];
[bezierPath addLineToPoint: CGPointMake(x2,y2)];
[UIColor.blackColor setStroke];
bezierPath.lineWidth = 1;
[bezierPath stroke];
```

Swift:

```
let bezierPath = UIBezierPath()
bezierPath.moveToPoint(CGPoint(x: x1, y: y1))
bezierPath.addLineToPoint(CGPoint(x: x2, y: y2))
UIColor.blackColor().setStroke()
bezierPath.lineWidth = 1
bezierPath.stroke()
```

对于半圆：



```
CGRect ovalRect = CGRectMake(x,y,width,height);
UIBezierPath* ovalPath = [UIBezierPath bezierPath];
[ovalPath addArcWithCenter: CGPointMake(0, 0) radius: CGRectGetWidth(ovalRect) / 2 startAngle: 180 * M_PI/180 endAngle: 0 * M_PI/180 clockwise: YES];
[ovalPath addLineToPoint: CGPointMake(0, 0)];
[ovalPath closePath];

CGAffineTransform ovalTransform = CGAffineTransformMakeTranslation(CGRectGetMidX(ovalRect),
CGRectGetMidY(ovalRect));
ovalTransform = CGAffineTransformScale(ovalTransform, 1, CGRectGetHeight(ovalRect) / CGRectGetWidth(ovalRect));
```



```
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRect: CGRectMake(0, 0, 50, 50)];
[UIColor.grayColor setFill];
[rectanglePath fill];
```

Swift:

```
let rectanglePath = UIBezierPath(rect: CGRect(x: 0, y: 0, width: 50, height: 50))
UIColor.grayColor().setFill()
rectanglePath.fill()
```

For a simple Line:

---

```
UIBezierPath* bezierPath = [UIBezierPath bezierPath];
[bezierPath moveToPoint: CGPointMake(x1,y1)];
[bezierPath addLineToPoint: CGPointMake(x2,y2)];
[UIColor.blackColor setStroke];
bezierPath.lineWidth = 1;
[bezierPath stroke];
```

Swift:

```
let bezierPath = UIBezierPath()
bezierPath.moveToPoint(CGPoint(x: x1, y: y1))
bezierPath.addLineToPoint(CGPoint(x: x2, y: y2))
UIColor.blackColor().setStroke()
bezierPath.lineWidth = 1
bezierPath.stroke()
```

For a half circle:



```
CGRect ovalRect = CGRectMake(x,y,width,height);
UIBezierPath* ovalPath = [UIBezierPath bezierPath];
[ovalPath addArcWithCenter: CGPointMake(0, 0) radius: CGRectGetWidth(ovalRect) / 2 startAngle: 180 * M_PI/180 endAngle: 0 * M_PI/180 clockwise: YES];
[ovalPath addLineToPoint: CGPointMake(0, 0)];
[ovalPath closePath];
```

```
CGAffineTransform ovalTransform = CGAffineTransformMakeTranslation(CGRectGetMidX(ovalRect),
CGRectGetMidY(ovalRect));
ovalTransform = CGAffineTransformScale(ovalTransform, 1, CGRectGetHeight(ovalRect) / CGRectGetWidth(ovalRect));
```

```
[ovalPath applyTransform: ovalTransform];
[UIColor.grayColor setFill];
[ovalPath fill];
```

Swift:

```
let ovalRect = CGRect(x: 0, y: 0, width: 50, height: 50)
let ovalPath = UIBezierPath()
ovalPath.addArcWithCenter(CGPoint.zero, radius: ovalRect.width / 2, startAngle: 180 * CGFloat(M_PI)/180, endAngle: 0 * CGFloat(M_PI)/180, clockwise: true)
ovalPath.addLineToPoint(CGPoint.zero)
ovalPath.closePath()

var ovalTransform = CGAffineTransformMakeTranslation(CGRectGetMidX(ovalRect),
CGRectGetMidY(ovalRect))
ovalTransform = CGAffineTransformScale(ovalTransform, 1, ovalRect.height / ovalRect.width)
ovalPath.applyTransform(ovalTransform)

UIColor.grayColor().setFill()
ovalPath.fill()
```

对于一个简单的三角形：



```
UIBezierPath* polygonPath = [UIBezierPath bezierPath];
[polygonPath moveToPoint: CGPointMake(x1, y1)];
[polygonPath addLineToPoint: CGPointMake(x2, y2)];
[polygonPath addLineToPoint: CGPointMake(x3, y2)];
[polygonPath closePath];
[UIColor.grayColor setFill];
[polygonPath fill];
```

Swift:

```
let polygonPath = UIBezierPath()
polygonPath.moveToPoint(CGPoint(x: x1, y: y1))
polygonPath.addLineToPoint(CGPoint(x: x2, y: y2))
polygonPath.addLineToPoint(CGPoint(x: x3, y: y3))
polygonPath.closePath()
UIColor.grayColor().setFill()
polygonPath.fill()
```

## 第53.5节：UIBezierPath + 自动布局

要根据视图框架调整贝塞尔路径的大小，请重写绘制贝塞尔路径的视图的 drawRect 方法：

```
- (void)drawRect:(CGRect)frame
{
    UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRect:
CGRectMake(CGRectGetMinX(frame), CGRectGetMinY(frame), CGRectGetWidth(frame),
CGRectGetHeight(frame))];
```

```
[ovalPath applyTransform: ovalTransform];
[UIColor.grayColor setFill];
[ovalPath fill];
```

Swift:

```
let ovalRect = CGRect(x: 0, y: 0, width: 50, height: 50)
let ovalPath = UIBezierPath()
ovalPath.addArcWithCenter(CGPoint.zero, radius: ovalRect.width / 2, startAngle: 180 * CGFloat(M_PI)/180, endAngle: 0 * CGFloat(M_PI)/180, clockwise: true)
ovalPath.addLineToPoint(CGPoint.zero)
ovalPath.closePath()

var ovalTransform = CGAffineTransformMakeTranslation(CGRectGetMidX(ovalRect),
CGRectGetMidY(ovalRect))
ovalTransform = CGAffineTransformScale(ovalTransform, 1, ovalRect.height / ovalRect.width)
ovalPath.applyTransform(ovalTransform)

UIColor.grayColor().setFill()
ovalPath.fill()
```

For a simple triangle:



```
UIBezierPath* polygonPath = [UIBezierPath bezierPath];
[polygonPath moveToPoint: CGPointMake(x1, y1)];
[polygonPath addLineToPoint: CGPointMake(x2, y2)];
[polygonPath addLineToPoint: CGPointMake(x3, y2)];
[polygonPath closePath];
[UIColor.grayColor setFill];
[polygonPath fill];
```

Swift:

```
let polygonPath = UIBezierPath()
polygonPath.moveToPoint(CGPoint(x: x1, y: y1))
polygonPath.addLineToPoint(CGPoint(x: x2, y: y2))
polygonPath.addLineToPoint(CGPoint(x: x3, y: y3))
polygonPath.closePath()
UIColor.grayColor().setFill()
polygonPath.fill()
```

## Section 53.5: UIBezierPath + AutoLayout

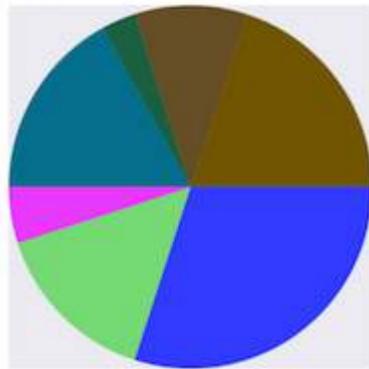
For bezier path to get resized based on the view frame, override the drawRect of view that you are drawing the bezier path :

```
- (void)drawRect:(CGRect)frame
{
    UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRect:
CGRectMake(CGRectGetMinX(frame), CGRectGetMinY(frame), CGRectGetWidth(frame),
CGRectGetHeight(frame))];
```

```
[UIColor.grayColor setFill];
[rectanglePath fill];
}
```

## 第53.6节：使用UIBezierPath的饼图视图和柱状图视图

- 饼图视图



```
- (void)drawRect:(CGRect)rect {
    NSArray *data = @[@30, @15, @5, @17, @3, @10, @20];

    // 1. 上下文
    CGContextRef ctxtRef = UIGraphicsGetCurrentContext();

    CGPoint center = CGPointMake(150, 150);
    CGFloat radius = 150;
    __block CGFloat startAngle = 0;
    [data enumerateObjectsUsingBlock:^(NSNumber * _Nonnull obj, NSUInteger idx, BOOL * _Nonnull stop) {

        // 2. 创建路径
        CGFloat endAngle = obj.floatValue / 100 * M_PI * 2 + startAngle;
        UIBezierPath *circlePath = [UIBezierPath bezierPathWithArcCenter:center radius:radius
startAngle:startAngle endAngle:endAngle clockwise:YES];
        [circlePath addLineToPoint:center];

        // 3. 添加路径
        CGContextAddPath(ctxtRef, circlePath.CGPath);

        // 设置颜色
        [[UIColor colorWithRed:@(float)arc4random_uniform(256) / 255.0
green:@(float)arc4random_uniform(256) / 255.0 blue:@(float)arc4random_uniform(256) / 255.0
alpha:1.0] setFill];

        // 4. 渲染
        CGContextDrawPath(ctxtRef, kCGPathFill);

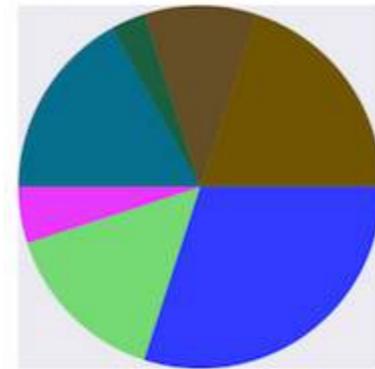
        // 重置角度
        startAngle = endAngle;
    }];

    override func draw(_ rect: CGRect) {
        // 定义用于创建饼图的数据
        let data: [Int] = [30, 15, 5, 17, 3, 10, 20]
        // 1. 找到绘制矩形的中心点
    }
}
```

```
[UIColor.grayColor setFill];
[rectanglePath fill];
}
```

## Section 53.6: pie view & column view with UIBezierPath

- pie view



```
- (void)drawRect:(CGRect)rect {
    NSArray *data = @[@30, @15, @5, @17, @3, @10, @20];

    // 1. context
    CGContextRef ctxtRef = UIGraphicsGetCurrentContext();

    CGPoint center = CGPointMake(150, 150);
    CGFloat radius = 150;
    __block CGFloat startAngle = 0;
    [data enumerateObjectsUsingBlock:^(NSNumber * _Nonnull obj, NSUInteger idx, BOOL * _Nonnull stop) {

        // 2. create path
        CGFloat endAngle = obj.floatValue / 100 * M_PI * 2 + startAngle;
        UIBezierPath *circlePath = [UIBezierPath bezierPathWithArcCenter:center radius:radius
startAngle:startAngle endAngle:endAngle clockwise:YES];
        [circlePath addLineToPoint:center];

        // 3. add path
        CGContextAddPath(ctxtRef, circlePath.CGPath);

        // set color
        [[UIColor colorWithRed:@(float)arc4random_uniform(256) / 255.0
green:@(float)arc4random_uniform(256) / 255.0 blue:@(float)arc4random_uniform(256) / 255.0
alpha:1.0] setFill];

        // 4. render
        CGContextDrawPath(ctxtRef, kCGPathFill);

        // reset angle
        startAngle = endAngle;
    }];

    override func draw(_ rect: CGRect) {
        // define data to create pie chart
        let data: [Int] = [30, 15, 5, 17, 3, 10, 20]
        // 1. find center of draw rect
    }
}
```

```

let center: CGPoint = CGPointMake(x: rect.midX, y: rect.midY)

// 2. 计算饼图的半径
let radius = min(rect.width, rect.height) / 2.0

var startAngle: CGFloat = 0.0
for value in data {

    // 3. 计算当前扇形的结束角度
    let endAngle = CGFloat(value) / 100.0 * CGFloat.pi * 2.0 + startAngle

    // 4. 创建当前扇形的 UIBezierPath
    let circlePath = UIBezierPath(arcCenter: center, radius: radius, startAngle: startAngle,
endAngle: endAngle, clockwise: true)

    // 5. 添加一条线到中心点以闭合路径
    circlePath.addLine(to: center)

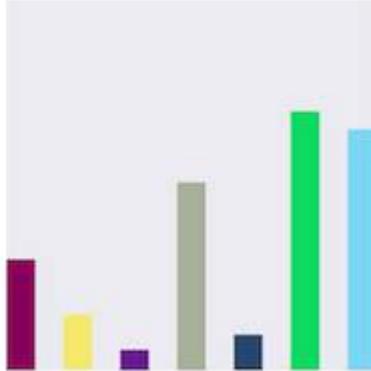
    // 6. 设置当前扇形的填充颜色
    UIColor(red: (CGFloat(arc4random_uniform(256)) / 255.0), green:
(CGFloat(arc4random_uniform(256)) / 255.0), blue: (CGFloat(arc4random_uniform(256)) / 255.0),
alpha: 1.0).setFill()

    // 7. 填充扇形路径
    circlePath.fill()

    // 8. 将结束角度设为下一扇形的起始角度
    startAngle = endAngle
}
}

```

• 柱状视图



```

- (void)drawRect:(CGRect)rect {

    NSArray *data = @[@300, @150.65, @55.3, @507.7, @95.8, @700, @650.65];

    // 1.
    CGContextRef ctxtRef = UIGraphicsGetCurrentContext();

    NSInteger columnCount = 7;
    CGFloat width = self.bounds.size.width / (columnCount + columnCount - 1);
    for (NSInteger i = 0; i < columnCount; i++) {

        // 2.
        CGFloat height = [data[i] floatValue] / 1000 * self.bounds.size.height; // floatValue
        CGFloat x = 0 + width * (2 * i);
        CGFloat y = self.bounds.size.height - height;
        UIBezierPath *rectPath = [UIBezierPath bezierPathWithRect:CGRectMake(x, y, width, height)];

```

```

let center: CGPoint = CGPointMake(x: rect.midX, y: rect.midY)

// 2. calculate radius of pie
let radius = min(rect.width, rect.height) / 2.0

var startAngle: CGFloat = 0.0
for value in data {

    // 3. calculate end angle for slice
    let endAngle = CGFloat(value) / 100.0 * CGFloat.pi * 2.0 + startAngle

    // 4. create UIBezierPath for slide
    let circlePath = UIBezierPath(arcCenter: center, radius: radius, startAngle: startAngle,
endAngle: endAngle, clockwise: true)

    // 5. add line to center to close path
    circlePath.addLine(to: center)

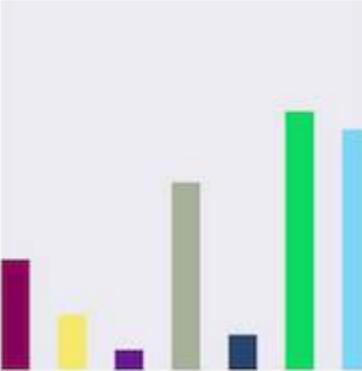
    // 6. set fill color for current slice
    UIColor(red: (CGFloat(arc4random_uniform(256)) / 255.0), green:
(CGFloat(arc4random_uniform(256)) / 255.0), blue: (CGFloat(arc4random_uniform(256)) / 255.0),
alpha: 1.0).setFill()

    // 7. fill slice path
    circlePath.fill()

    // 8. set end angle as start angle for next slice
    startAngle = endAngle
}
}

```

• column view



```

- (void)drawRect:(CGRect)rect {

    NSArray *data = @[@300, @150.65, @55.3, @507.7, @95.8, @700, @650.65];

    // 1.
    CGContextRef ctxtRef = UIGraphicsGetCurrentContext();

    NSInteger columnCount = 7;
    CGFloat width = self.bounds.size.width / (columnCount + columnCount - 1);
    for (NSInteger i = 0; i < columnCount; i++) {

        // 2.
        CGFloat height = [data[i] floatValue] / 1000 * self.bounds.size.height; // floatValue
        CGFloat x = 0 + width * (2 * i);
        CGFloat y = self.bounds.size.height - height;
        UIBezierPath *rectPath = [UIBezierPath bezierPathWithRect:CGRectMake(x, y, width, height)];

```

```

CGContextAddPath(ctxRef, rectPath.CGPath);

    // 3.
    [[UIColor colorWithRed:(arc4random_uniform(256) / 255.0)
green:(arc4random_uniform(256) / 255.0) blue:(arc4random_uniform(256) / 255.0)
alpha:1.0] setFill];
CGContextDrawPath(ctxRef, kCGPathFill);
}

override func draw(_ rect: CGRect) {
    // 定义图表数据
    let data: [CGFloat] = [300, 150.65, 55.3, 507.7, 95.8, 700, 650.65]

    // 1. 计算列数
    let columnCount = data.count

    // 2. 计算列宽
    let columnWidth = rect.width / CGFloat(columnCount + columnCount - 1)

    for (columnIndex, value) in data.enumerated() {
        // 3. 计算列高
        let columnHeight = value / 1000.0 * rect.height

        // 4. 计算列起点
        let columnOrigin = CGPoint(x: (columnWidth * 2.0 * CGFloat(columnIndex)), y: (rect.height -
columnHeight))

        // 5. 创建列路径
        let columnPath = UIBezierPath(rect: CGRect(origin: columnOrigin, size: CGSize(width:
columnWidth, height: columnHeight)))

        // 6. 设置当前列的填充颜色
        UIColor(red: (CGFloat(arc4random_uniform(256)) / 255.0), green:
(CGFloat(arc4random_uniform(256)) / 255.0), blue: (CGFloat(arc4random_uniform(256)) / 255.0),
alpha: 1.0).setFill()

        // 7. 填充列路径
        columnPath.fill()
    }
}

```

```

CGContextAddPath(ctxRef, rectPath.CGPath);

    // 3.
    [[UIColor colorWithRed:(arc4random_uniform(256) / 255.0)
green:(arc4random_uniform(256) / 255.0) blue:(arc4random_uniform(256) / 255.0)
alpha:1.0] setFill];
CGContextDrawPath(ctxRef, kCGPathFill);
}

override func draw(_ rect: CGRect) {
    // define data for chart
    let data: [CGFloat] = [300, 150.65, 55.3, 507.7, 95.8, 700, 650.65]

    // 1. calculate number of columns
    let columnCount = data.count

    // 2. calculate column width
    let columnWidth = rect.width / CGFloat(columnCount + columnCount - 1)

    for (columnIndex, value) in data.enumerated() {
        // 3. calculate column height
        let columnHeight = value / 1000.0 * rect.height

        // 4. calculate column origin
        let columnOrigin = CGPoint(x: (columnWidth * 2.0 * CGFloat(columnIndex)), y: (rect.height -
columnHeight))

        // 5. create path for column
        let columnPath = UIBezierPath(rect: CGRect(origin: columnOrigin, size: CGSize(width:
columnWidth, height: columnHeight)))

        // 6. set fill color for current column
        UIColor(red: (CGFloat(arc4random_uniform(256)) / 255.0), green:
(CGFloat(arc4random_uniform(256)) / 255.0), blue: (CGFloat(arc4random_uniform(256)) / 255.0),
alpha: 1.0).setFill()

        // 7. fill column path
        columnPath.fill()
    }
}

```

# 第54章：UIPickerView

## 第54.1节：基本示例

### Swift

```
class PickerViewExampleViewController : UIViewController, UIPickerViewDelegate, UIPickerViewDataSource {
    @IBOutlet weak var btnFolder: UIButton!
    let pickerView = UIPickerView()
    let pickerViewRows = ["第一行", "第二行", "第三行", "第四行"]

    override func viewDidLoad() {
        super.viewDidLoad()
        self.btnFolder.addTarget(self, action: #selector(CreateListVC.btnAddPress), forControlEvents: UIControlEvents.TouchUpInside)
    }

    @objc private func btnAddPress() {
        self.pickerView.delegate = self
        self.pickerView.dataSource = self
        self.view.addSubview(self.pickerView)
    }

    //MARK: UIPickerViewDelegate

    func pickerView(pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int) -> String? {
        return self.pickerViewRows[row]
    }

    //MARK: UIPickerViewDataSource

    func numberOfComponentsInPickerView(pickerView: UIPickerView) -> Int {
        return 1
    }

    func pickerView(pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {
        return self.pickerViewRows.count
    }
}
```

### Objective-C

```
@property (nonatomic, strong) UIPickerView *countryPicker;
@property (nonatomic, strong) NSArray *countryNames;

- (void)viewDidLoad {
    [super viewDidLoad];
    _countryNames = @[@"澳大利亚 (AUD)", @"中国 (CNY)",
                      @"法国 (EUR)", @"英国 (GBP)", @"日本 (JPY)", @"印度
                      (IN)", @"澳大利亚 (AUS)", @"纽约 (NW)"];
    [self pickCountry];
}

-(void)pickCountry {
    _countryPicker = [[UIPickerView alloc] init];
    _countryPicker.delegate = self;
    _countryPicker.dataSource = self;
```

# Chapter 54: UIPickerView

## Section 54.1: Basic example

### Swift

```
class PickerViewExampleViewController : UIViewController, UIPickerViewDelegate, UIPickerViewDataSource {
    @IBOutlet weak var btnFolder: UIButton!
    let pickerView = UIPickerView()
    let pickerViewRows = ["First row", "Second row", "Third row", "Fourth row"]

    override func viewDidLoad() {
        super.viewDidLoad()
        self.btnFolder.addTarget(self, action: #selector(CreateListVC.btnAddPress), forControlEvents: UIControlEvents.TouchUpInside)
    }

    @objc private func.btnAddPress() {
        self.pickerView.delegate = self
        self.pickerView.dataSource = self
        self.view.addSubview(self.pickerView)
    }

    //MARK: UIPickerViewDelegate

    func pickerView(pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int) -> String? {
        return self.pickerViewRows[row]
    }

    //MARK: UIPickerViewDataSource

    func numberOfComponentsInPickerView(pickerView: UIPickerView) -> Int {
        return 1
    }

    func pickerView(pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {
        return self.pickerViewRows.count
    }
}
```

### Objective-C

```
@property (nonatomic, strong) UIPickerView *countryPicker;
@property (nonatomic, strong) NSArray *countryNames;

- (void)viewDidLoad {
    [super viewDidLoad];
    _countryNames = @[@"Australia (AUD)", @"China (CNY)",
                      @"France (EUR)", @"Great Britain (GBP)", @"Japan (JPY)", @"INDIA
                      (IN)", @"AUSTRALIA (AUS)", @"NEW YORK (NW)"];
    [self pickCountry];
}

-(void)pickCountry {
    _countryPicker = [[UIPickerView alloc] init];
    _countryPicker.delegate = self;
    _countryPicker.dataSource = self;
```

```

[[UIPickerView appearance] setBackgroundColor:[UIColor colorWithRed:21/255.0 green:17/255.0
blue:50/255.0 alpha:1.0]];
}

#pragma mark- pickerView 代理和数据源

- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView {
    return 1;
}

- (NSInteger)pickerView:(UIPickerView *)pickerView numberOfRowsInComponent:(NSInteger)component {
    return _countryNames.count;
}

- (NSString *)pickerView:(UIPickerView *)pickerView
    titleForRow:(NSInteger)row
forComponent:(NSInteger)component {
    return _countryNames[row];
}

- (void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger)row
inComponent:(NSInteger)component {
    NSString *pickedCountryName = _countryNames[row];
}

```

## 第54.2节：更改pickerView背景颜色和文本颜色

### Objective-C

```

// 显示带有黑色背景和白色文本的国家选择器
[self. countryPicker setValue:[UIColor whiteColor] forKey:@"textColor"];
[self. countryPicker setValue:[UIColor blackColor] forKey:@"backgroundColor"];

```

### Swift

```

let color1 = UIColor(colorLiteralRed: 1, green: 1, blue: 1, alpha: 1)
let color2 = UIColor(colorLiteralRed: 0, green: 0, blue: 0, alpha: 1)
pickerView2.setValue(color1, forKey: "textColor")
pickerView2.setValue(color2, forKey: "backgroundColor")

```

```

[[UIPickerView appearance] setBackgroundColor:[UIColor colorWithRed:21/255.0 green:17/255.0
blue:50/255.0 alpha:1.0]];
}

#pragma mark- pickerView Delegates And datasource

- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView {
    return 1;
}

- (NSInteger)pickerView:(UIPickerView *)pickerView numberOfRowsInComponent:(NSInteger)component {
    return _countryNames.count;
}

- (NSString *)pickerView:(UIPickerView *)pickerView
    titleForRow:(NSInteger)row
forComponent:(NSInteger)component {
    return _countryNames[row];
}

- (void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger)row
inComponent:(NSInteger)component {
    NSString *pickedCountryName = _countryNames[row];
}

```

## Section 54.2: Changing pickerView Background Color and text color

### Objective-C

```

//Displays the country pickerView with black background and white text
[self. countryPicker setValue:[UIColor whiteColor] forKey:@"textColor"];
[self. countryPicker setValue:[UIColor blackColor] forKey:@"backgroundColor"];

```

### Swift

```

let color1 = UIColor(colorLiteralRed: 1, green: 1, blue: 1, alpha: 1)
let color2 = UIColor(colorLiteralRed: 0, green: 0, blue: 0, alpha: 1)
pickerView2.setValue(color1, forKey: "textColor")
pickerView2.setValue(color2, forKey: "backgroundColor")

```

# 第55章：UIFeedbackGenerator

UIFeedbackGenerator及其子类提供了对iOS设备上自iPhone 7起配备的Taptic Engine®的公共接口。触觉反馈，品牌名为Taptics，为屏幕上的事件提供触觉反馈。虽然许多系统控件开箱即用地提供触觉反馈，开发者也可以使用UIFeedbackGenerator的子类为自定义控件和其他事件添加触觉反馈。UIFeedbackGenerator是一个抽象类，不应直接使用，开发者应使用其子类之一。

## 第55.1节：触发冲击触觉

示例展示了如何在按钮按下后使用UIImpactFeedbackGenerator触发冲击触觉。

### Swift

```
class ViewController: UIViewController
{
    lazy var button: UIButton =
    {
        let button = UIButton()
        button.translatesAutoresizingMaskIntoConstraints = false
        self.view.addSubview(button)
        button.centerXAnchor.constraint(equalTo: self.view.centerXAnchor).isActive = true
        button.centerYAnchor.constraint(equalTo: self.view.centerYAnchor).isActive = true
        button.setTitle("Impact", for: .normal)
        button.setTitleColor(UIColor.gray, for: .normal)
        return button
    }()

    // 在 heavy、medium 和 light 之间选择样式
    let impactFeedbackGenerator = UIImpactFeedbackGenerator(style: .heavy)

    override func viewDidLoad()
    {
        super.viewDidLoad()
        button.addTarget(self, action: #selector(self.didPressButton(sender:)), for:
        .touchUpInside)

        // 为即将发生的事件准备反馈生成器并减少延迟
        impactFeedbackGenerator.prepare()
    }

    func didPressButton(sender: UIButton)
    {
        // 触发触觉反馈
        impactFeedbackGenerator.impactOccurred()
    }
}
```

### Objective-C

```
@interface ViewController ()
@property (nonatomic, strong) UIImpactFeedbackGenerator *impactFeedbackGenerator;
@property (nonatomic, strong) UIButton *button;
@end

@implementation ViewController

- (void)viewDidLoad
{
```

# Chapter 55: UIFeedbackGenerator

UIFeedbackGenerator and its subclasses offers a public interface to the Taptic Engine® found on iOS devices starting with iPhone 7. Haptics, branded Taptics, provide tactile feedback for on-screen events. While many system controls provide haptics out-of-the-box, developers can use UIImpactFeedbackGenerator subclasses to add haptics to custom controls and other events. UIFeedbackGenerator is an abstract class that should not be used directly, rather developers use one of its subclasses.

## Section 55.1: Trigger Impact Haptic

Example shows how to trigger an impact haptic using UIImpactFeedbackGenerator after a button press.

### Swift

```
class ViewController: UIViewController
{
    lazy var button: UIButton =
    {
        let button = UIButton()
        button.translatesAutoresizingMaskIntoConstraints = false
        self.view.addSubview(button)
        button.centerXAnchor.constraint(equalTo: self.view.centerXAnchor).isActive = true
        button.centerYAnchor.constraint(equalTo: self.view.centerYAnchor).isActive = true
        button.setTitle("Impact", for: .normal)
        button.setTitleColor(UIColor.gray, for: .normal)
        return button
    }()

    // Choose between heavy, medium, and light for style
    let impactFeedbackGenerator = UIImpactFeedbackGenerator(style: .heavy)

    override func viewDidLoad()
    {
        super.viewDidLoad()
        button.addTarget(self, action: #selector(self.didPressButton(sender:)), for:
        .touchUpInside)

        // Primes feedback generator for upcoming events and reduces latency
        impactFeedbackGenerator.prepare()
    }

    func didPressButton(sender: UIButton)
    {
        // Triggers haptic
        impactFeedbackGenerator.impactOccurred()
    }
}
```

### Objective-C

```
@interface ViewController ()
@property (nonatomic, strong) UIImpactFeedbackGenerator *impactFeedbackGenerator;
@property (nonatomic, strong) UIButton *button;
@end

@implementation ViewController

- (void)viewDidLoad
{
```

```

[super viewDidLoad];
[self.button addTarget:self action:@selector(didPressButton:)
forControlEvents:UIControlEventTouchUpInside];

// 在 heavy、medium 和 light 之间选择样式
self.impactFeedbackGenerator = [[UIImpactFeedbackGenerator alloc]
initWithStyle:UIImpactFeedbackStyleHeavy];

// 为即将发生的事件准备反馈生成器并减少延迟
[self.impactFeedbackGenerator prepare];
}

- (void)didPressButton:(UIButton *)sender
{
    // 触发触觉反馈
    [self.impactFeedbackGenerator impactOccurred];
}

#pragma mark - 延迟初始化
- (UIButton *)button
{
    if (!_button)
    {
        _button = [[UIButton alloc] init];
        _button.translatesAutoresizingMaskIntoConstraints = NO;
        [self.view addSubview:_button];
        [_button.centerXAnchor constraintEqualToAnchor:self.view.centerXAnchor].active = YES;
        [_button.centerYAnchor constraintEqualToAnchor:self.view.centerYAnchor].active = YES;
        [_button setTitle:@"Impact" forState:UIControlStateNormal];
        [_button setTitleColor:[UIColor grayColor] forState:UIControlStateNormal];
    }
    return _button;
}

@end

```

```

[super viewDidLoad];
[self.button addTarget:self action:@selector(didPressButton:)
forControlEvents:UIControlEventTouchUpInside];

// Choose between heavy, medium, and light for style
self.impactFeedbackGenerator = [[UIImpactFeedbackGenerator alloc]
initWithStyle:UIImpactFeedbackStyleHeavy];

// Primes feedback generator for upcoming events and reduces latency
[self.impactFeedbackGenerator prepare];
}

- (void)didPressButton:(UIButton *)sender
{
    // Triggers haptic
    [self.impactFeedbackGenerator impactOccurred];
}

#pragma mark - Lazy Init
- (UIButton *)button
{
    if (!_button)
    {
        _button = [[UIButton alloc] init];
        _button.translatesAutoresizingMaskIntoConstraints = NO;
        [self.view addSubview:_button];
        [_button.centerXAnchor constraintEqualToAnchor:self.view.centerXAnchor].active = YES;
        [_button.centerYAnchor constraintEqualToAnchor:self.view.centerYAnchor].active = YES;
        [_button setTitle:@"Impact" forState:UIControlStateNormal];
        [_button setTitleColor:[UIColor grayColor] forState:UIControlStateNormal];
    }
    return _button;
}

@end

```

# 第56章：UIAppearance

## 第56.1节：设置类的所有实例的外观

要自定义某个类的所有实例的外观，请访问该类的外观代理。例如：

### 设置UIButton的色调颜色

Swift :

```
UIButton.appearance().tintColor = UIColor.greenColor()
```

Objective-C :

```
[UIButton appearance].tintColor = [UIColor greenColor];
```

### 设置UIButton的背景颜色

Swift :

```
UIButton.appearance().backgroundColor = UIColor.blueColor()
```

Objective-C :

```
[UIButton appearance].backgroundColor = [UIColor blueColor];
```

### 设置UILabel的文本颜色

Swift :

```
UILabel.appearance().textColor = UIColor.redColor()
```

Objective-C :

```
[UILabel appearance].textColor = [UIColor redColor];
```

### 设置 UILabel 文字颜色

Swift :

```
UILabel.appearance().backgroundColor = UIColor.greenColor()
```

Objective-C :

```
[UILabel appearance].backgroundColor = [UIColor greenColor];
```

### 设置 UINavigationBar 颜色

Swift :

```
UINavigationBar.appearance().tintColor = UIColor.cyanColor()
```

Objective-C :

# Chapter 56: UIAppearance

## Section 56.1: Set appearance of all instances of the class

To customize appearance of all instances of a class, access appearance proxy of the desired class. For example:

### Set UIButton tint color

Swift:

```
UIButton.appearance().tintColor = UIColor.greenColor()
```

Objective-C:

```
[UIButton appearance].tintColor = [UIColor greenColor];
```

### Set UIButton background color

Swift:

```
UIButton.appearance().backgroundColor = UIColor.blueColor()
```

Objective-C:

```
[UIButton appearance].backgroundColor = [UIColor blueColor];
```

### Set UILabel text color

Swift:

```
UILabel.appearance().textColor = UIColor.redColor()
```

Objective-C:

```
[UILabel appearance].textColor = [UIColor redColor];
```

### Set UILabel background color

Swift:

```
UILabel.appearance().backgroundColor = UIColor.greenColor()
```

Objective-C:

```
[UILabel appearance].backgroundColor = [UIColor greenColor];
```

### Set UINavigationBar tint color

Swift:

```
UINavigationBar.appearance().tintColor = UIColor.cyanColor()
```

Objective-C:

```
[UINavigationBar appearance].tintColor = [UIColor cyanColor];
```

#### 设置 UINavigationBar 背景颜色

Swift :

```
UINavigationBar.appearance().backgroundColor = UIColor.redColor()
```

Objective-C :

```
[UINavigationBar appearance].backgroundColor = [UIColor redColor];
```

## 第56.2节：当包含在容器类中时的类外观

使用`appearanceWhenContainedInInstancesOfClasses`:来自定义当类的实例包含在容器类实例中时的外观。例如，自定义UILabel在ViewController类中的`textColor`和`backgroundColor`如下所示：

#### 设置UILabel的文本颜色

Swift :

```
UILabel.appearanceWhenContainedInInstancesOfClasses([ViewController.self]).textColor =  
UIColor.whiteColor()
```

Objective-C :

```
[UILabel appearanceWhenContainedInInstancesOfClasses:@[[ViewController class]]].textColor =  
[UIColor whiteColor];
```

#### 设置 UILabel 文字颜色

Swift :

```
UILabel.appearanceWhenContainedInInstancesOfClasses([ViewController.self]).backgroundColor =  
UIColor.blueColor()
```

Objective-C :

```
[UILabel appearanceWhenContainedInInstancesOfClasses:@[[ViewController class]]].backgroundColor =  
[UIColor blueColor];
```

```
[UINavigationBar appearance].tintColor = [UIColor cyanColor];
```

#### Set UINavigationBar background color

Swift:

```
UINavigationBar.appearance().backgroundColor = UIColor.redColor()
```

Objective-C:

```
[UINavigationBar appearance].backgroundColor = [UIColor redColor];
```

## Section 56.2: Appearance for class when contained in container class

Use `appearanceWhenContainedInInstancesOfClasses`: to customize the appearance for instance of a class when contained within an instance of container class. For example customization of `UILabel`'s `textColor` and `backgroundColor` within `ViewController` class will look like this:

#### Set UILabel text color

Swift:

```
UILabel.appearanceWhenContainedInInstancesOfClasses([ViewController.self]).textColor =  
UIColor.whiteColor()
```

Objective-C:

```
[UILabel appearanceWhenContainedInInstancesOfClasses:@[[ViewController class]]].textColor =  
[UIColor whiteColor];
```

#### Set UILabel background color

Swift:

```
UILabel.appearanceWhenContainedInInstancesOfClasses([ViewController.self]).backgroundColor =  
UIColor.blueColor()
```

Objective-C:

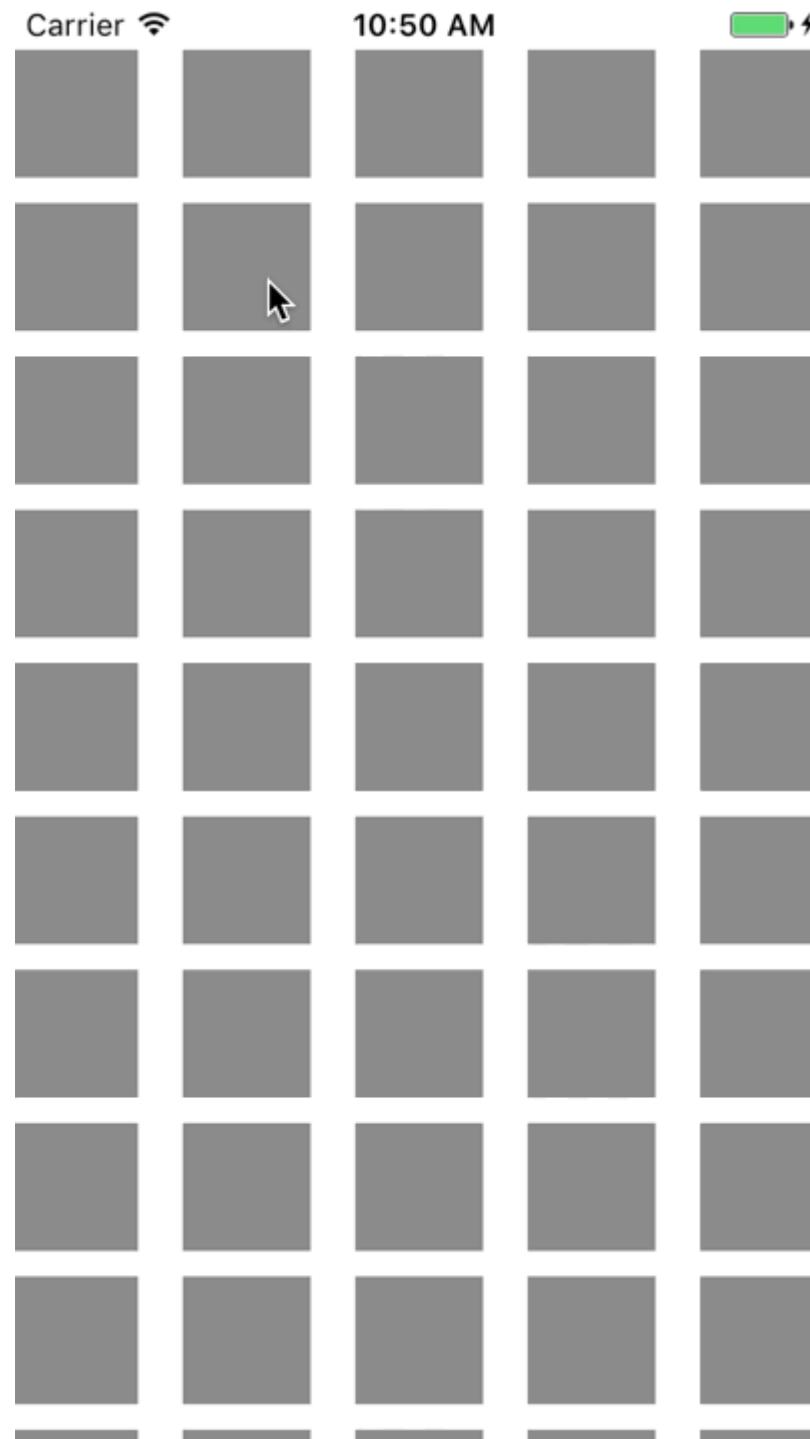
```
[UILabel appearanceWhenContainedInInstancesOfClasses:@[[ViewController class]]].backgroundColor =  
[UIColor blueColor];
```

# 第57章：带UICollectionView的UIKit动力学

UIKit动力学是集成在UIKit中的物理引擎。UIKit动力学提供了一组API，支持与UICollectionView和UICollectionViewLayout的互操作性

## 第57.1节：使用UIDynamicAnimator创建自定义拖拽行为

本示例展示了如何通过子类化UIDynamicBehavior和子类化UICollectionViewFlowLayout来创建自定义拖拽行为。示例中，我们有一个允许多项选择的UICollectionView。然后通过长按手势，这些选中的项可以被拖动，表现为由UIDynamicAnimator驱动的弹性“弹簧”动画。

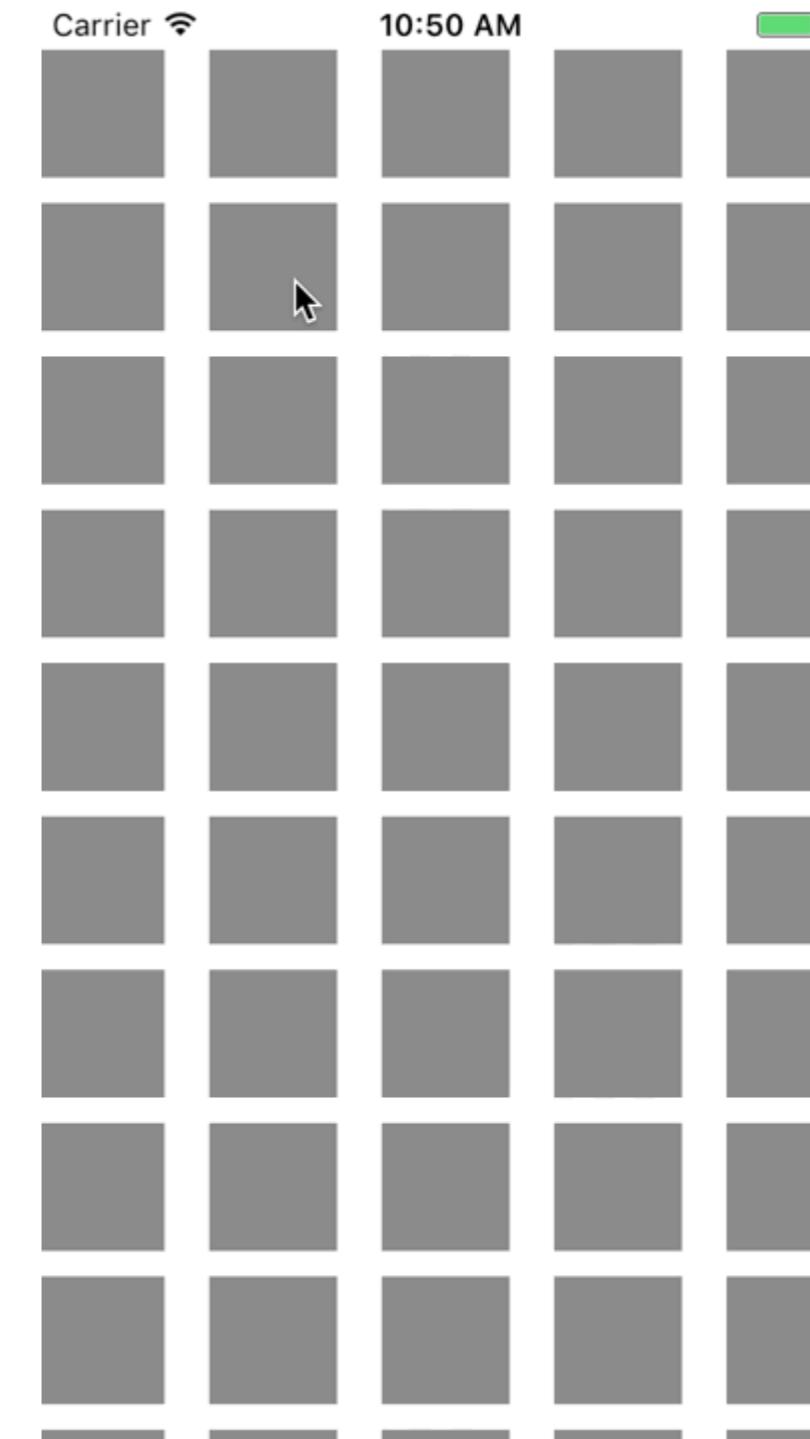


# Chapter 57: UIKit Dynamics with UICollectionView

UIKit Dynamics is a physics engine integrated into UIKit. UIKit Dynamics offers a set of API that offers interoperability with a [UICollectionView](#) and [UICollectionViewLayout](#)

## Section 57.1: Creating a Custom Dragging Behavior with UIDynamicAnimator

This example shows how to create a custom dragging behavior by Subclassing [UIDynamicBehavior](#) and subclassing [UICollectionViewFlowLayout](#). In the example, we have [UICollectionView](#) that allows for the selection of multiple items. Then with a long press gesture those items can be dragged in an elastic, "springy" animation driven by a [UIDynamicAnimator](#).



拖拽行为是通过结合一个低级行为和一个高级行为来实现的。低级行为是在UIDynamicItem的四个角添加UIAttachmentBehavior，高级行为则管理多个低级行为。

UIDynamicItems。

我们可以先创建这个低级行为，称之为RectangleAttachmentBehavior

### Swift

```
final class RectangleAttachmentBehavior: UIDynamicBehavior
{
    init(item: UIDynamicItem, point: CGPoint)
    {
        // 高频率，形成更“刚性”的效果
        let frequency: CGFloat = 8.0

        // 低阻尼，动画耗时更长，最终停下
        let damping: CGFloat = 0.6

        super.init()

        // 附着点是物体的四个角
        let points = self.attachmentPoints(for: point)

        let attachmentBehaviors: [UIAttachmentBehavior] = points.map
        {
            let attachmentBehavior = UIAttachmentBehavior(item: item, attachedToAnchor: $0)
            attachmentBehavior.frequency = frequency
            attachmentBehavior.damping = damping
            return attachmentBehavior
        }

        attachmentBehaviors.forEach
        {
            addChildBehavior($0)
        }
    }

    func updateAttachmentLocation(with point: CGPoint)
    {
        // 更新锚点到新的附着点
        let points = self.attachmentPoints(for: point)
        let attachments = self.childBehaviors.flatMap { $0 as? UIAttachmentBehavior }
        let pairs = zip(points, attachments)
        pairs.forEach { $0.1.anchorPoint = $0.0 }
    }

    func attachmentPoints(for point: CGPoint) -> [CGPoint]
    {
        // 宽度和高度应接近项目的宽度和高度
        let width: CGFloat = 40.0
        let height: CGFloat = 40.0

        let topLeft = CGPoint(x: point.x - width * 0.5, y: point.y - height * 0.5)
        let topRight = CGPoint(x: point.x + width * 0.5, y: point.y - height * 0.5)
        let bottomLeft = CGPoint(x: point.x - width * 0.5, y: point.y + height * 0.5)
        let bottomRight = CGPoint(x: point.x + width * 0.5, y: point.y + height * 0.5)
        let points = [topLeft, topRight, bottomLeft, bottomRight]
        return points
    }
}
```

The dragging behavior is produced by combining a low-level behavior that adds a [UIAttachmentBehavior](#) to the four corners of a [UIDynamicItem](#) and a high-level behavior that manages the low-level behavior for a number of [UIDynamicItems](#).

We can begin by creating this low-level behavior, we'll call RectangleAttachmentBehavior

### Swift

```
final class RectangleAttachmentBehavior: UIDynamicBehavior
{
    init(item: UIDynamicItem, point: CGPoint)
    {
        // Higher frequency more "ridged" formation
        let frequency: CGFloat = 8.0

        // Lower damping longer animation takes to come to rest
        let damping: CGFloat = 0.6

        super.init()

        // Attachment points are four corners of item
        let points = self.attachmentPoints(for: point)

        let attachmentBehaviors: [UIAttachmentBehavior] = points.map
        {
            let attachmentBehavior = UIAttachmentBehavior(item: item, attachedToAnchor: $0)
            attachmentBehavior.frequency = frequency
            attachmentBehavior.damping = damping
            return attachmentBehavior
        }

        attachmentBehaviors.forEach
        {
            addChildBehavior($0)
        }
    }

    func updateAttachmentLocation(with point: CGPoint)
    {
        // Update anchor points to new attachment points
        let points = self.attachmentPoints(for: point)
        let attachments = self.childBehaviors.flatMap { $0 as? UIAttachmentBehavior }
        let pairs = zip(points, attachments)
        pairs.forEach { $0.1.anchorPoint = $0.0 }
    }

    func attachmentPoints(for point: CGPoint) -> [CGPoint]
    {
        // Width and height should be close to the width and height of the item
        let width: CGFloat = 40.0
        let height: CGFloat = 40.0

        let topLeft = CGPoint(x: point.x - width * 0.5, y: point.y - height * 0.5)
        let topRight = CGPoint(x: point.x + width * 0.5, y: point.y - height * 0.5)
        let bottomLeft = CGPoint(x: point.x - width * 0.5, y: point.y + height * 0.5)
        let bottomRight = CGPoint(x: point.x + width * 0.5, y: point.y + height * 0.5)
        let points = [topLeft, topRight, bottomLeft, bottomRight]
        return points
    }
}
```

## Objective-C

```
@implementation RectangleAttachmentBehavior

- (instancetype)initWithItem:(id<UIDynamicItem>)item point:(CGPoint)point
{
    CGFloat frequency = 8.0f;
    CGFloat damping = 0.6f;
    self = [super init];
    if (self)
    {
        NSArray <NSValue *> *pointValues = [self attachmentPointValuesForPoint:point];
        for (NSValue *value in pointValues)
        {
            UIAttachmentBehavior *attachment = [[UIAttachmentBehavior alloc] initWithItem:item
attachedToAnchor:[value CGPointValue]];
            attachment.frequency = frequency;
            attachment.damping = damping;
            [self addChildBehavior:attachment];
        }
    }
    return self;
}

- (void)updateAttachmentLocationWithPoint:(CGPoint)point
{
    NSArray <NSValue *> *pointValues = [self attachmentPointValuesForPoint:point];
    for (NSInteger i = 0; i < pointValues.count; i++)
    {
        NSValue *pointValue = pointValues[i];
        UIAttachmentBehavior *attachment = self.childBehaviors[i];
        attachment.anchorPoint = [pointValue CGPointValue];
    }
}

- (NSArray <NSValue *> *)attachmentPointValuesForPoint:(CGPoint)point
{
    CGFloat width = 40.0f;
    CGFloat height = 40.0f;

    CGPoint topLeft = CGPointMake(point.x - width * 0.5, point.y - height * 0.5);
    CGPoint topRight = CGPointMake(point.x + width * 0.5, point.y - height * 0.5);
    CGPoint bottomLeft = CGPointMake(point.x - width * 0.5, point.y + height * 0.5);
    CGPoint bottomRight = CGPointMake(point.x + width * 0.5, point.y + height * 0.5);

    NSArray <NSValue *> *pointValues = @[[NSValue valueWithCGPoint:topLeft], [NSValue
valueWithCGPoint:topRight], [NSValue valueWithCGPoint:bottomLeft], [NSValue
valueWithCGPoint:bottomRight]];
    return pointValues;
}

@end
```

接下来我们可以创建一个高级行为，它将组合多个RectangleAttachmentBehavior。

## Swift

```
final class DragBehavior: UIDynamicBehavior
{
    init(items: [UIDynamicItem], point: CGPoint)
    {
        super.init()
        items.forEach
```

## Objective-C

```
@implementation RectangleAttachmentBehavior

- (instancetype)initWithItem:(id<UIDynamicItem>)item point:(CGPoint)point
{
    CGFloat frequency = 8.0f;
    CGFloat damping = 0.6f;
    self = [super init];
    if (self)
    {
        NSArray <NSValue *> *pointValues = [self attachmentPointValuesForPoint:point];
        for (NSValue *value in pointValues)
        {
            UIAttachmentBehavior *attachment = [[UIAttachmentBehavior alloc] initWithItem:item
attachedToAnchor:[value CGPointValue]];
            attachment.frequency = frequency;
            attachment.damping = damping;
            [self addChildBehavior:attachment];
        }
    }
    return self;
}

- (void)updateAttachmentLocationWithPoint:(CGPoint)point
{
    NSArray <NSValue *> *pointValues = [self attachmentPointValuesForPoint:point];
    for (NSInteger i = 0; i < pointValues.count; i++)
    {
        NSValue *pointValue = pointValues[i];
        UIAttachmentBehavior *attachment = self.childBehaviors[i];
        attachment.anchorPoint = [pointValue CGPointValue];
    }
}

- (NSArray <NSValue *> *)attachmentPointValuesForPoint:(CGPoint)point
{
    CGFloat width = 40.0f;
    CGFloat height = 40.0f;

    CGPoint topLeft = CGPointMake(point.x - width * 0.5, point.y - height * 0.5);
    CGPoint topRight = CGPointMake(point.x + width * 0.5, point.y - height * 0.5);
    CGPoint bottomLeft = CGPointMake(point.x - width * 0.5, point.y + height * 0.5);
    CGPoint bottomRight = CGPointMake(point.x + width * 0.5, point.y + height * 0.5);

    NSArray <NSValue *> *pointValues = @[[NSValue valueWithCGPoint:topLeft], [NSValue
valueWithCGPoint:topRight], [NSValue valueWithCGPoint:bottomLeft], [NSValue
valueWithCGPoint:bottomRight]];
    return pointValues;
}

@end
```

Next we can create the high-level behavior that will combine a number of RectangleAttachmentBehavior.

## Swift

```
final class DragBehavior: UIDynamicBehavior
{
    init(items: [UIDynamicItem], point: CGPoint)
    {
        super.init()
        items.forEach
```

```

    {
        let rectAttachment = RectangleAttachmentBehavior(item: $0, point: point)
        self.addChildBehavior(rectAttachment)
    }

    func updateDragLocation(with point: CGPoint)
    {
        // 告诉底层行为位置已更改
        self.childBehaviors.flatMap { $0 as? RectangleAttachmentBehavior }.forEach {
            $0.updateAttachmentLocation(with: point)
        }
    }
}

```

## Objective-C

```

@implementation DragBehavior

- (instancetype)initWithItems:(NSArray <id<UIDynamicItem>> *)items point: (CGPoint)point
{
    self = [super init];
    if (self)
    {
        for (id<UIDynamicItem> item in items)
        {
            RectangleAttachmentBehavior *rectAttachment = [[RectangleAttachmentBehavior
alloc]initWithItem:item point:point];
            [self addChildBehavior:rectAttachment];
        }
    }
    return self;
}

- (void)updateDragLocationWithPoint:(CGPoint)point
{
    for (RectangleAttachmentBehavior *rectAttachment in self.childBehaviors)
    {
        [rectAttachment updateAttachmentLocationWithPoint:point];
    }
}

@end

```

现在我们的行为已经就绪，下一步是在合适的时候将它们添加到我们的集合视图中。因为通常我们想要一个标准的网格布局，所以可以子类化UICollectionViewFlowLayout，并且只在拖动时更改属性。我们主要通过重写layoutAttributesForElementsInRect并使用UIDynamicAnimator的便利方法itemsInRect来实现。

## Swift

```

final class DraggableLayout: UICollectionViewFlowLayout
{
    // 保存被拖动的索引路径数组
    var indexPathsForDraggingElements: [IndexPath]?

    // 用于动画拖动行为的动态动画器
    var animator: UIDynamicAnimator?

    // 指定拖动动画的自定义高级行为
    var dragBehavior: DragBehavior?
}

```

```

    {
        let rectAttachment = RectangleAttachmentBehavior(item: $0, point: point)
        self.addChildBehavior(rectAttachment)
    }

    func updateDragLocation(with point: CGPoint)
    {
        // Tell low-level behaviors location has changed
        self.childBehaviors.flatMap { $0 as? RectangleAttachmentBehavior }.forEach {
            $0.updateAttachmentLocation(with: point)
        }
    }
}

```

## Objective-C

```

@implementation DragBehavior

- (instancetype)initWithItems:(NSArray <id<UIDynamicItem>> *)items point: (CGPoint)point
{
    self = [super init];
    if (self)
    {
        for (id<UIDynamicItem> item in items)
        {
            RectangleAttachmentBehavior *rectAttachment = [[RectangleAttachmentBehavior
alloc]initWithItem:item point:point];
            [self addChildBehavior:rectAttachment];
        }
    }
    return self;
}

- (void)updateDragLocationWithPoint:(CGPoint)point
{
    for (RectangleAttachmentBehavior *rectAttachment in self.childBehaviors)
    {
        [rectAttachment updateAttachmentLocationWithPoint:point];
    }
}

@end

```

Now with our behaviors in place, the next step is to add them to our collection view when. Because normally we want a standard grid layout we can subclass `UICollectionViewFlowLayout` and only change attributes when dragging. We do this mainly through overriding `layoutAttributesForElementsInRect` and using the `UIDynamicAnimator`'s convenience method `itemsInRect`.

## Swift

```

final class DraggableLayout: UICollectionViewFlowLayout
{
    // Array that holds dragged index paths
    var indexPathsForDraggingElements: [IndexPath]?

    // The dynamic animator that will animate drag behavior
    var animator: UIDynamicAnimator?

    // Custom high-level behavior that dictates drag animation
    var dragBehavior: DragBehavior?
}

```

```

// 拖动开始的位置，以便拖动结束后可以返回
var startDragPoint = CGPoint.zero

// 用于跟踪拖动是否结束的布尔值
var isFinishedDragging = false

// 通知布局拖动已开始的方法
func startDragging(indexPaths: [IndexPath], from point: CGPoint)
{
    indexPathsForDraggingElements = indexPaths
    animator = UIDynamicAnimator(collectionViewLayout: self)
    animator?.delegate = self

    // 获取所有可拖动的属性，但更改 zIndex 以置于其他单元格之上
    let draggableAttributes: [UICollectionViewLayoutAttributes] = indexPaths.flatMap {
        let attribute = super.layoutAttributesForItem(at: $0)
        attribute?.zIndex = 1
        return attribute
    }

    startDragPoint = point

    // 将它们添加到高级行为中
    dragBehavior = DragBehavior(items: draggableAttributes, point: point)

    // 将高级行为添加到动画器中
    animator?.addBehavior(dragBehavior!)

}

func updateDragLocation(_ point: CGPoint)
{
    // 告诉高级行为该点已更新
    dragBehavior?.updateDragLocation(with: point)
}

func endDragging()
{
    isFinishedDragging = true

    // 将高级行为返回到起始点
    dragBehavior?.updateDragLocation(with: startDragPoint)
}

func clearDraggedIndexPaths()
{
    // 重置状态以准备下一次拖动事件
    animator = nil
    indexPathsForDraggingElements = nil
    isFinishedDragging = false
}

override func layoutAttributesForElements(in rect: CGRect) -> [UICollectionViewLayoutAttributes]?
{
    let existingAttributes: [UICollectionViewLayoutAttributes] =
super.layoutAttributesForElements(in: rect) ?? []
    var allAttributes = [UICollectionViewLayoutAttributes]()

    // 获取非拖动项目的普通流布局属性
    for attributes in existingAttributes
    {

```

```

// Where dragging starts so can return there once dragging ends
var startDragPoint = CGPoint.zero

// Bool to keep track if dragging has ended
var isFinishedDragging = false

// Method to inform layout that dragging has started
func startDragging(indexPaths: [IndexPath], from point: CGPoint)
{
    indexPathsForDraggingElements = indexPaths
    animator = UIDynamicAnimator(collectionViewLayout: self)
    animator?.delegate = self

    // Get all of the draggable attributes but change zIndex so above other cells
    let draggableAttributes: [UICollectionViewLayoutAttributes] = indexPaths.flatMap {
        let attribute = super.layoutAttributesForItem(at: $0)
        attribute?.zIndex = 1
        return attribute
    }

    startDragPoint = point

    // Add them to high-level behavior
    dragBehavior = DragBehavior(items: draggableAttributes, point: point)

    // Add high-level behavior to animator
    animator?.addBehavior(dragBehavior!)
}

func updateDragLocation(_ point: CGPoint)
{
    // Tell high-level behavior that point has updated
    dragBehavior?.updateDragLocation(with: point)
}

func endDragging()
{
    isFinishedDragging = true

    // Return high-level behavior to starting point
    dragBehavior?.updateDragLocation(with: startDragPoint)
}

func clearDraggedIndexPaths()
{
    // Reset state for next drag event
    animator = nil
    indexPathsForDraggingElements = nil
    isFinishedDragging = false
}

override func layoutAttributesForElements(in rect: CGRect) -> [UICollectionViewLayoutAttributes]?
{
    let existingAttributes: [UICollectionViewLayoutAttributes] =
super.layoutAttributesForElements(in: rect) ?? []
    var allAttributes = [UICollectionViewLayoutAttributes]()

    // Get normal flow layout attributes for non-drag items
    for attributes in existingAttributes
    {

```

```

        if (indexPathsForDraggingElements?.contains(attributesIndexPath) ?? false) == false
    {
        allAttributes.append(attributes)
    }
}

// 通过询问动画器添加被拖动项的属性
if let animator = self.animator
{
    let animatorAttributes: [UICollectionViewLayoutAttributes] = animator.items(in:
rect).flatMap { $0 as? UICollectionViewLayoutAttributes }
    allAttributes.append(contentsOf: animatorAttributes)
}
return allAttributes
}

extension DraggableLayout: UIDynamicAnimatorDelegate
{
    func dynamicAnimatorDidPause(_ animator: UIDynamicAnimator)
    {
        // 动画器已暂停且拖动完成；重置状态
        guard isFinishedDragging else { return }
        clearDraggedIndexPaths()
    }
}

```

## Objective-C

```

@interface DraggableLayout () <UIDynamicAnimatorDelegate>
@property (nonatomic, strong) NSArray <NSIndexPath *> *indexPathsForDraggingElements;
@property (nonatomic, strong) UIDynamicAnimator *animator;
@property (nonatomic, assign) CGPoint startDragPoint;
@property (nonatomic, assign) BOOL finishedDragging;
@property (nonatomic, strong) DragBehavior *dragBehavior;
@end

@implementation DraggableLayout

- (void)startDraggingWithIndexPaths:(NSArray <NSIndexPath *> *)selectedIndexPaths
fromPoint:(CGPoint)point
{
    self.indexPathsForDraggingElements = selectedIndexPaths;
    self.animator = [[UIDynamicAnimator alloc] initWithCollectionViewLayout:self];
    self.animator.delegate = self;
    NSMutableArray *draggableAttributes = [[NSMutableArray
alloc]initWithCapacity:selectedIndexPaths.count];
    for (NSIndexPath *indexPath in selectedIndexPaths)
    {
        UICollectionViewLayoutAttributes *attributes = [super
layoutAttributesForItemAtIndexPath:indexPath];
        attributes.zIndex = 1;
        [draggableAttributes addObject:attributes];
    }
    self.startDragPoint = point;
    self.dragBehavior = [[DragBehavior alloc] initWithItems:draggableAttributes point:point];
    [self.animator addBehavior:self.dragBehavior];
}

- (void)updateDragLoactionWithPoint:(CGPoint)point
{
    [self.dragBehavior updateDragLocationWithPoint:point];
}

```

```

        if (indexPathsForDraggingElements?.contains(attributesIndexPath) ?? false) == false
    {
        allAttributes.append(attributes)
    }
}

// Add dragged item attributes by asking animator for them
if let animator = self.animator
{
    let animatorAttributes: [UICollectionViewLayoutAttributes] = animator.items(in:
rect).flatMap { $0 as? UICollectionViewLayoutAttributes }
    allAttributes.append(contentsOf: animatorAttributes)
}
return allAttributes
}

extension DraggableLayout: UIDynamicAnimatorDelegate
{
    func dynamicAnimatorDidPause(_ animator: UIDynamicAnimator)
    {
        // Animator has paused and done dragging; reset state
        guard isFinishedDragging else { return }
        clearDraggedIndexPaths()
    }
}

```

## Objective-C

```

@interface DraggableLayout () <UIDynamicAnimatorDelegate>
@property (nonatomic, strong) NSArray <NSIndexPath *> *indexPathsForDraggingElements;
@property (nonatomic, strong) UIDynamicAnimator *animator;
@property (nonatomic, assign) CGPoint startDragPoint;
@property (nonatomic, assign) BOOL finishedDragging;
@property (nonatomic, strong) DragBehavior *dragBehavior;
@end

@implementation DraggableLayout

- (void)startDraggingWithIndexPaths:(NSArray <NSIndexPath *> *)selectedIndexPaths
fromPoint:(CGPoint)point
{
    self.indexPathsForDraggingElements = selectedIndexPaths;
    self.animator = [[UIDynamicAnimator alloc] initWithCollectionViewLayout:self];
    self.animator.delegate = self;
    NSMutableArray *draggableAttributes = [[NSMutableArray
alloc]initWithCapacity:selectedIndexPaths.count];
    for (NSIndexPath *indexPath in selectedIndexPaths)
    {
        UICollectionViewLayoutAttributes *attributes = [super
layoutAttributesForItemAtIndexPath:indexPath];
        attributes.zIndex = 1;
        [draggableAttributes addObject:attributes];
    }
    self.startDragPoint = point;
    self.dragBehavior = [[DragBehavior alloc] initWithItems:draggableAttributes point:point];
    [self.animator addBehavior:self.dragBehavior];
}

- (void)updateDragLoactionWithPoint:(CGPoint)point
{
    [self.dragBehavior updateDragLocationWithPoint:point];
}

```

```

- (void)endDragging
{
    self.finishedDragging = YES;
    [self.dragBehavior updateDragLocationWithPoint:self.startDragPoint];
}

- (void)clearDraggedIndexPath
{
    self.animator = nil;
    self.indexPathsForDraggingElements = nil;
    self.finishedDragging = NO;
}

- (void)dynamicAnimatorDidPause:(UIDynamicAnimator *)animator
{
    if (self.finishedDragging)
    {
        [self clearDraggedIndexPath];
    }
}

- (NSArray<UICollectionViewLayoutAttributes *> *)layoutAttributesForElementsInRect:(CGRect)rect
{
    NSArray *existingAttributes = [super layoutAttributesForElementsInRect:rect];
    NSMutableArray *allAttributes = [[NSMutableArray alloc] initWithCapacity:existingAttributes.count];
    for (UICollectionViewLayoutAttributes *attributes in existingAttributes)
    {
        if (![self.indexPathsForDraggingElements containsObject:attributesIndexPath])
        {
            [allAttributes addObject:attributes];
        }
    }
    [allAttributes addObjectsFromArray:[self.animator itemsInRect:rect]];
    return allAttributes;
}

@end

```

最后，我们将创建一个视图控制器，用于创建我们的UICollectionView并处理长按手势。

## Swift

```

final class ViewController: UIViewController
{
    // 显示单元格的集合视图
    lazy var collectionView: UICollectionView =
    {
        let collectionView = UICollectionView(frame: .zero, collectionViewLayout:
        DraggableLayout())
        collectionView.backgroundColor = .white
        collectionView.translatesAutoresizingMaskIntoConstraints = false
        self.view.addSubview(collectionView)
        collectionView.topAnchor.constraint(equalTo: self.topLayoutGuide.bottomAnchor).isActive =
        true
        collectionView.leadingAnchor.constraint(equalTo: self.view.leadingAnchor).isActive = true
        collectionView.trailingAnchor.constraint(equalTo: self.view.trailingAnchor).isActive = true
        collectionView.bottomAnchor.constraint(equalTo: self.bottomLayoutGuide.topAnchor).isActive =
        true

        return collectionView
    }()
}

```

```

- (void)endDragging
{
    self.finishedDragging = YES;
    [self.dragBehavior updateDragLocationWithPoint:self.startDragPoint];
}

- (void)clearDraggedIndexPath
{
    self.animator = nil;
    self.indexPathsForDraggingElements = nil;
    self.finishedDragging = NO;
}

- (void)dynamicAnimatorDidPause:(UIDynamicAnimator *)animator
{
    if (self.finishedDragging)
    {
        [self clearDraggedIndexPath];
    }
}

- (NSArray<UICollectionViewLayoutAttributes *> *)layoutAttributesForElementsInRect:(CGRect)rect
{
    NSArray *existingAttributes = [super layoutAttributesForElementsInRect:rect];
    NSMutableArray *allAttributes = [[NSMutableArray alloc] initWithCapacity:existingAttributes.count];
    for (UICollectionViewLayoutAttributes *attributes in existingAttributes)
    {
        if (![self.indexPathsForDraggingElements containsObject:attributesIndexPath])
        {
            [allAttributes addObject:attributes];
        }
    }
    [allAttributes addObjectsFromArray:[self.animator itemsInRect:rect]];
    return allAttributes;
}

@end

```

Finally, we'll create a view controller that will create our `UICollectionView` and handle our long press gesture.

## Swift

```

final class ViewController: UIViewController
{
    // Collection view that displays cells
    lazy var collectionView: UICollectionView =
    {
        let collectionView = UICollectionView(frame: .zero, collectionViewLayout:
        DraggableLayout())
        collectionView.backgroundColor = .white
        collectionView.translatesAutoresizingMaskIntoConstraints = false
        self.view.addSubview(collectionView)
        collectionView.topAnchor.constraint(equalTo: self.topLayoutGuide.bottomAnchor).isActive =
        true
        collectionView.leadingAnchor.constraint(equalTo: self.view.leadingAnchor).isActive = true
        collectionView.trailingAnchor.constraint(equalTo: self.view.trailingAnchor).isActive = true
        collectionView.bottomAnchor.constraint(equalTo: self.bottomLayoutGuide.topAnchor).isActive =
        true

        return collectionView
    }()
}

```

```

// 驱动拖拽的手势
lazy var longPress: UILongPressGestureRecognizer =
{
    let longPress = UILongPressGestureRecognizer(target: self, action:
#selector(self.handleLongPress(sender:)))
    return longPress
}()

// 保存选中索引路径的数组
var selectedIndexPaths = [IndexPath]()

override func viewDidLoad()
{
    super.viewDidLoad()
collectionView.delegate = self
collectionView.dataSource = self
collectionView.register(UICollectionViewCell.self, forCellWithReuseIdentifier: "Cell")
    collectionView.addGestureRecognizer(longPress)
}

func handleLongPress(sender: UILongPressGestureRecognizer)
{
guard let draggableLayout = collectionView.collectionViewLayout as? DraggableLayout else {
return }
    let location = sender.location(in: collectionView)
    switch sender.state
    {
        case .began:
draggableLayout.startDragging(indexPaths: selectedIndexPaths, from: location)
            case .changed:
draggableLayout.updateDragLocation(location)
            case .ended, .failed, .cancelled:
draggableLayout.endDragging()
            case .possible:
                break
    }
}
extension ViewController: UICollectionViewDelegate, UICollectionViewDataSource
{
    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int
    {
        返回 1000
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell
    {
        let cell = collectionView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath)
cell.backgroundColor = .gray
        如果 selectedIndexPaths.contains(indexPath) == true
        {
            cell.backgroundColor = .red
        }
        return cell
    }

    func collectionView(_ collectionView: UICollectionView, didSelectItemAt indexPath: IndexPath)
    {
        // 布尔值，判断单元格是被选中还是取消选中
    }
}

```

```

// Gesture that drives dragging
lazy var longPress: UILongPressGestureRecognizer =
{
    let longPress = UILongPressGestureRecognizer(target: self, action:
#selector(self.handleLongPress(sender:)))
    return longPress
}()

// Array that holds selected index paths
var selectedIndexPaths = [IndexPath]()

override func viewDidLoad()
{
    super.viewDidLoad()
collectionView.delegate = self
collectionView.dataSource = self
collectionView.register(UICollectionViewCell.self, forCellWithReuseIdentifier: "Cell")
    collectionView.addGestureRecognizer(longPress)
}

func handleLongPress(sender: UILongPressGestureRecognizer)
{
guard let draggableLayout = collectionView.collectionViewLayout as? DraggableLayout else {
return }
    let location = sender.location(in: collectionView)
    switch sender.state
    {
        case .began:
draggableLayout.startDragging(indexPaths: selectedIndexPaths, from: location)
            case .changed:
draggableLayout.updateDragLocation(location)
            case .ended, .failed, .cancelled:
draggableLayout.endDragging()
            case .possible:
                break
    }
}
extension ViewController: UICollectionViewDelegate, UICollectionViewDataSource
{
    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int
    {
        返回 1000
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell
    {
        let cell = collectionView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath)
cell.backgroundColor = .gray
        if selectedIndexPaths.contains(indexPath) == true
        {
            cell.backgroundColor = .red
        }
        return cell
    }

    func collectionView(_ collectionView: UICollectionView, didSelectItemAt indexPath: IndexPath)
    {
        // Bool that determines if cell is being selected or unselected
    }
}

```

```

let isSelected = !selectedIndexPaths.contains(indexPath)
let cell = collectionView.cellForItem(at: indexPath)
cell?.backgroundColor = isSelected ? .red : .gray
    如果 isSelected
    {
selectedIndexPaths.append(indexPath)
}
else
{
selectedIndexPaths.remove(at: selectedIndexPaths.index(of: indexPath)!)
}
}
}

```

### Objective-C

```

@interface ViewController () <UICollectionViewDelegate, UICollectionViewDataSource>
@property (nonatomic, strong) UICollectionView *collectionView;
@property (nonatomic, strong) UILongPressGestureRecognizer *longPress;
@property (nonatomic, strong) NSMutableArray <NSIndexPath *> *selectedIndexPaths;
@end

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    self.collectionView.delegate = self;
    self.collectionView.dataSource = self;
    [self.collectionView registerClass:[UICollectionViewCell class]
forCellWithReuseIdentifier:@"Cell"];
    [self.collectionView addGestureRecognizer:self.longPress];
    self.selectedIndexPaths = [[NSMutableArray alloc] init];
}

- (UICollectionView *)collectionView
{
    if (!_collectionView)
    {
_collectionView = [[UICollectionView alloc] initWithFrame:CGRectZero
collectionViewLayout:[[DraggableLayout alloc]init]];
_collectionView.backgroundColor = [UIColor whiteColor];
    _collectionView.translatesAutoresizingMaskIntoConstraintsIntoConstraints = NO;
    [self.view addSubview:_collectionView];
    [_collectionView.topAnchor constraintEqualToAnchor:self.topLayoutGuide.bottomAnchor].active
= YES;
    [_collectionView.leadingAnchor constraintEqualToAnchor:self.view.leadingAnchor].active =
YES;
    [_collectionView.trailingAnchor constraintEqualToAnchor:self.view.trailingAnchor].active =
YES;
    [_collectionView.bottomAnchor
constraintEqualToAnchor:self.bottomLayoutGuide.topAnchor].active = YES;
    }
    return _collectionView;
}

- (UILongPressGestureRecognizer *)longPress
{
    if (!_longPress)
    {
_longPress = [[UILongPressGestureRecognizer alloc] initWithTarget:self
action:@selector(handleLongPress:)];
}

```

```

let isSelected = !selectedIndexPaths.contains(indexPath)
let cell = collectionView.cellForItem(at: indexPath)
cell?.backgroundColor = isSelected ? .red : .gray
if isSelected
{
    selectedIndexPaths.append(indexPath)
}
else
{
    selectedIndexPaths.remove(at: selectedIndexPaths.index(of: indexPath)!)
}
}
}

```

### Objective-C

```

@interface ViewController () <UICollectionViewDelegate, UICollectionViewDataSource>
@property (nonatomic, strong) UICollectionView *collectionView;
@property (nonatomic, strong) UILongPressGestureRecognizer *longPress;
@property (nonatomic, strong) NSMutableArray <NSIndexPath *> *selectedIndexPaths;
@end

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    self.collectionView.delegate = self;
    self.collectionView.dataSource = self;
    [self.collectionView registerClass:[UICollectionViewCell class]
forCellWithReuseIdentifier:@"Cell"];
    [self.collectionView addGestureRecognizer:self.longPress];
    self.selectedIndexPaths = [[NSMutableArray alloc] init];
}

- (UICollectionView *)collectionView
{
    if (!_collectionView)
    {
_collectionView = [[UICollectionView alloc] initWithFrame:CGRectZero
collectionViewLayout:[[DraggableLayout alloc]init]];
_collectionView.backgroundColor = [UIColor whiteColor];
    _collectionView.translatesAutoresizingMaskIntoConstraintsIntoConstraints = NO;
    [self.view addSubview:_collectionView];
    [_collectionView.topAnchor constraintEqualToAnchor:self.topLayoutGuide.bottomAnchor].active
= YES;
    [_collectionView.leadingAnchor constraintEqualToAnchor:self.view.leadingAnchor].active =
YES;
    [_collectionView.trailingAnchor constraintEqualToAnchor:self.view.trailingAnchor].active =
YES;
    [_collectionView.bottomAnchor
constraintEqualToAnchor:self.bottomLayoutGuide.topAnchor].active = YES;
    }
    return _collectionView;
}

- (UILongPressGestureRecognizer *)longPress
{
    if (!_longPress)
    {
        _longPress = [[UILongPressGestureRecognizer alloc] initWithTarget:self
action:@selector(handleLongPress:)];
}

```

```

    }
    return _longPress;
}

- (void)handleLongPress:(UILongPressGestureRecognizer *)sender
{
    DraggableLayout *draggableLayout = (DraggableLayout *)self.collectionView.collectionViewLayout;
    CGPoint location = [sender locationInView:self.collectionView];
    if (sender.state == UIGestureRecognizerStateBegan)
    {
        [draggableLayout startDraggingWithIndexPaths:self.selectedIndexPaths fromPoint:location];
    }
    else if (sender.state == UIGestureRecognizerStateChanged)
    {
        [draggableLayout updateDragLoactionWithPoint:location];
    }
    else if (sender.state == UIGestureRecognizerStateChanged || sender.state ==
    UIGestureRecognizerStateCancelled || sender.state == UIGestureRecognizerStateFailed)
    {
        [draggableLayout endDragging];
    }
}

- (NSInteger)collectionView:(UICollectionView *)collectionView
numberOfItemsInSection:(NSInteger)section
{
    return 1000;
}

- (UICollectionViewCell *)collectionView:(UICollectionView *)collectionView
cellForItemAtIndexPath:(NSIndexPath *)indexPath
{
    UICollectionViewCell *cell = [collectionView dequeueReusableCellWithReuseIdentifier:@"Cell"
forIndexPath:indexPath];
    cell.backgroundColor = [UIColor grayColor];
    if ([self.selectedIndexPaths containsObject:indexPath])
    {
        cell.backgroundColor = [UIColor redColor];
    }
    return cell;
}

- (void)collectionView:(UICollectionView *)collectionView didSelectItemAtIndexPath:(NSIndexPath *)
indexPath
{
    BOOL isSelected = !*[self.selectedIndexPaths containsObject:indexPath];
    UICollectionViewCell *cell = [collectionView cellForItemAtIndexPath:indexPath];
    if (isSelected)
    {
        cell.backgroundColor = [UIColor redColor];
        [self.selectedIndexPaths addObject:indexPath];
    }
    else
    {
        cell.backgroundColor = [UIColor grayColor];
        [self.selectedIndexPaths removeObject:indexPath];
    }
}
@end

```

更多信息请参见 2013 WWDC 会议“UIKit Dynamics 高级技巧”

```

    }
    return _longPress;
}

- (void)handleLongPress:(UILongPressGestureRecognizer *)sender
{
    DraggableLayout *draggableLayout = (DraggableLayout *)self.collectionView.collectionViewLayout;
    CGPoint location = [sender locationInView:self.collectionView];
    if (sender.state == UIGestureRecognizerStateBegan)
    {
        [draggableLayout startDraggingWithIndexPaths:self.selectedIndexPaths fromPoint:location];
    }
    else if (sender.state == UIGestureRecognizerStateChanged)
    {
        [draggableLayout updateDragLoactionWithPoint:location];
    }
    else if (sender.state == UIGestureRecognizerStateChanged || sender.state ==
    UIGestureRecognizerStateCancelled || sender.state == UIGestureRecognizerStateFailed)
    {
        [draggableLayout endDragging];
    }
}

- (NSInteger)collectionView:(UICollectionView *)collectionView
numberOfItemsInSection:(NSInteger)section
{
    return 1000;
}

- (UICollectionViewCell *)collectionView:(UICollectionView *)collectionView
cellForItemAtIndexPath:(NSIndexPath *)indexPath
{
    UICollectionViewCell *cell = [collectionView dequeueReusableCellWithReuseIdentifier:@"Cell"
forIndexPath:indexPath];
    cell.backgroundColor = [UIColor grayColor];
    if ([self.selectedIndexPaths containsObject:indexPath])
    {
        cell.backgroundColor = [UIColor redColor];
    }
    return cell;
}

- (void)collectionView:(UICollectionView *)collectionView didSelectItemAtIndexPath:(NSIndexPath *)
indexPath
{
    BOOL isSelected = !*[self.selectedIndexPaths containsObject:indexPath];
    UICollectionViewCell *cell = [collectionView cellForItemAtIndexPath:indexPath];
    if (isSelected)
    {
        cell.backgroundColor = [UIColor redColor];
        [self.selectedIndexPaths addObject:indexPath];
    }
    else
    {
        cell.backgroundColor = [UIColor grayColor];
        [self.selectedIndexPaths removeObject:indexPath];
    }
}
@end

```

For more information [2013 WWDC Session "Advanced Techniques with UIKit Dynamics"](#)

# 第58章：UIPheonix - 简单、灵活、动态且高度可扩展的UI框架

受游戏开发启发，UIPheonix 是一个超级简单、灵活、动态且高度可扩展的UI框架+概念，用于构建面向macOS、iOS和tvOS的可重用组件/控件驱动的应用程序。相同的API适用于跨平台开发！可以把它想象成使用乐高积木，你可以使用类似的积木并轻松地移动它们。

<https://github.com/MKGitHub/UIPheonix>

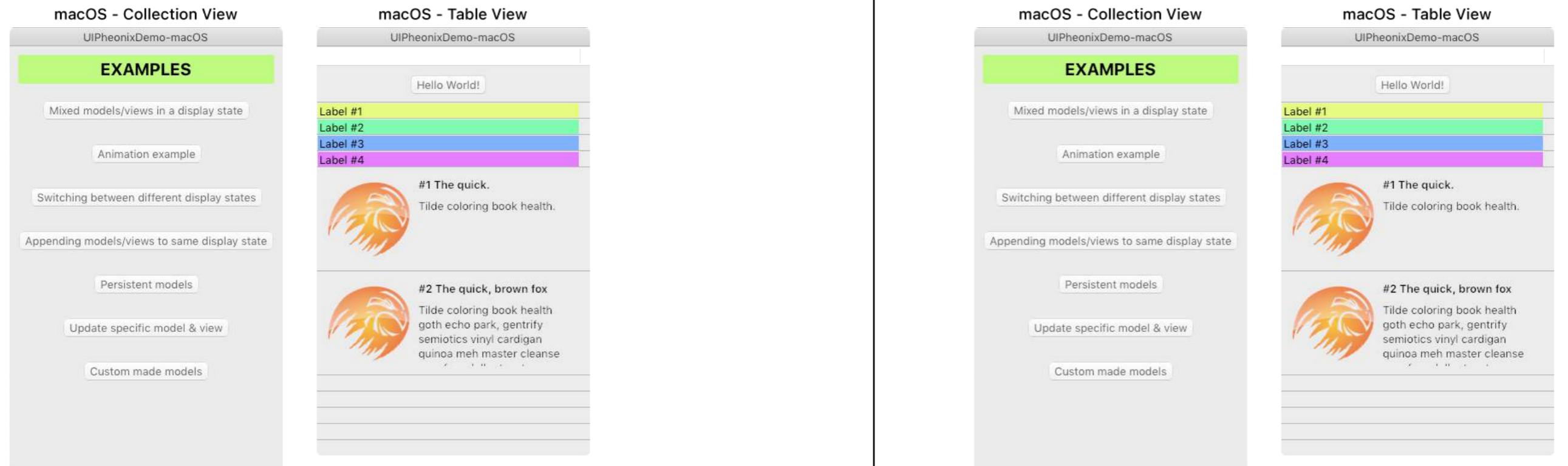
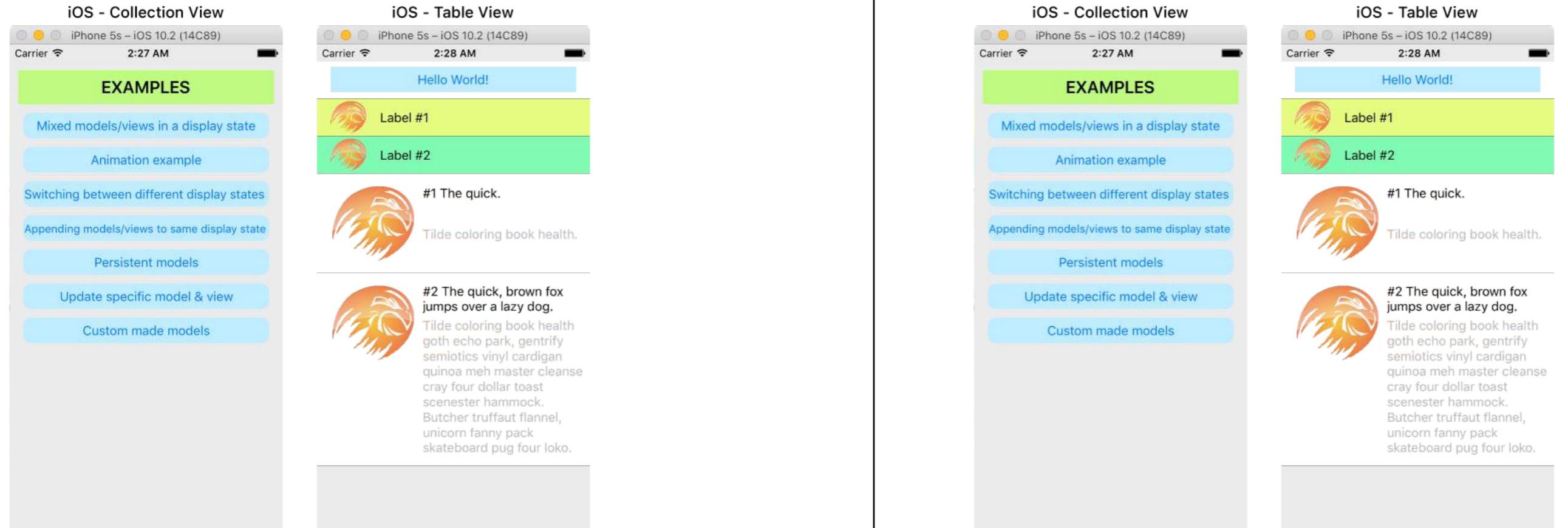
## 第58.1节：示例用户界面组件

# Chapter 58: UIPheonix - easy, flexible, dynamic & highly scalable UI framework

Inspired by game development UIPheonix is a super easy, flexible, dynamic and highly scalable UI framework + concept for building reusable component/control-driven apps for macOS, iOS and tvOS. The same API apply for cross platform development! Think of it as using Lego blocks, you can use similar ones and move them around easy as pie.

<https://github.com/MKGitHub/UIPheonix>

## Section 58.1: Example UI Components



## 第58.2节：示例用法

```
// 初始化  
mUIPheonix = UIPheonix(with:myCollectionView)  
mUIPheonix = UIPheonix(with:myTableView)  
  
// 连接模型-视图  
mUIPheonix.setModelViewRelationships([MyModel.nameOfClass:MyView.nameOfClass])  
  
// 为UI添加模型  
models.append(SimpleButtonModel(id:1, title:"Hello World!"))  
  
// 渲染，更新UI  
mUIPheonix.setDisplayModels(models)
```

## Section 58.2: Example Usage

```
// init  
mUIPheonix = UIPheonix(with:myCollectionView)  
mUIPheonix = UIPheonix(with:myTableView)  
  
// connect model-view  
mUIPheonix.setModelViewRelationships([MyModel.nameOfClass:MyView.nameOfClass])  
  
// add models for the UI  
models.append(SimpleButtonModel(id:1, title:"Hello World!"))  
  
// render, update UI  
mUIPheonix.setDisplayModels(models)
```

# 第59章：UIKit 动力学

UIKit 动力学是一个集成在 UIKit 中的完整真实物理引擎。它允许你通过添加重力、附着、碰撞和力等行为，创建感觉真实的界面。你定义希望界面元素采用的物理特性，动力学引擎负责其余部分。

## 第59.1节：基于手势速度的快速滑动视图

此示例展示了如何让视图跟踪平移手势并以基于物理的方式离开。



### Swift

```
class ViewController: UIViewController
{
    // 调整以改变视图的滑动速度
    let magnitudeMultiplier: CGFloat = 0.0008

    lazy var dynamicAnimator: UIDynamicAnimator =
    {
        let dynamicAnimator = UIDynamicAnimator(referenceView: self.view)
        return dynamicAnimator
    }()

    lazy var gravity: UIGravityBehavior =
    {
        let gravity = UIGravityBehavior(items: [self.orangeView])
        return gravity
    }()

    lazy var collision: UICollisionBehavior =
    {
        let collision = UICollisionBehavior(items: [self.orangeView])
        collision.translatesReferenceBoundsIntoBoundary = true
        return collision
    }()

    lazy var orangeView: UIView =
    {
        let widthHeight: CGFloat = 40.0
        let orangeView = UIView(frame: CGRect(x: 0.0, y: 0.0, width: widthHeight, height: widthHeight))
        orangeView.backgroundColor = UIColor.orange
        return orangeView
    }()
}
```

# Chapter 59: UIKit Dynamics

UIKit Dynamics is a full real-world physics engine integrated into UIKit. It allows you to create interfaces that feel real by adding behaviors such as gravity, attachments, collision and forces. You define the physical traits that you would like your interface elements to adopt, and the dynamics engine takes care of the rest.

## Section 59.1: Flick View Based on Gesture Velocity

This example shows how to have a view track a pan gesture and depart in a physics-based manner.



### Swift

```
class ViewController: UIViewController
{
    // Adjust to change speed of view from flick
    let magnitudeMultiplier: CGFloat = 0.0008

    lazy var dynamicAnimator: UIDynamicAnimator =
    {
        let dynamicAnimator = UIDynamicAnimator(referenceView: self.view)
        return dynamicAnimator
    }()

    lazy var gravity: UIGravityBehavior =
    {
        let gravity = UIGravityBehavior(items: [self.orangeView])
        return gravity
    }()

    lazy var collision: UICollisionBehavior =
    {
        let collision = UICollisionBehavior(items: [self.orangeView])
        collision.translatesReferenceBoundsIntoBoundary = true
        return collision
    }()

    lazy var orangeView: UIView =
    {
        let widthHeight: CGFloat = 40.0
        let orangeView = UIView(frame: CGRect(x: 0.0, y: 0.0, width: widthHeight, height: widthHeight))
        orangeView.backgroundColor = UIColor.orange
        return orangeView
    }()
}
```

```

        self.view.addSubview(orangeView)
        return orangeView
   }()

lazy var panGesture: UIPanGestureRecognizer =
{
    let panGesture = UIPanGestureRecognizer(target: self, action:
#selector(self.handlePan(sender:)))
    return panGesture
}()

lazy var attachment: UIAttachmentBehavior =
{
    let attachment = UIAttachmentBehavior(item: self.orangeView, attachedToAnchor: .zero)
    return attachment
}()

override func viewDidLoad()
{
    super.viewDidLoad()
dynamicAnimator.addBehavior(gravity)
    dynamicAnimator.addBehavior(collision)
    orangeView.addGestureRecognizer(panGesture)
}

override func viewDidLayoutSubviews()
{
    super.viewDidLayoutSubviews()
orangeView.center = view.center
    dynamicAnimator.updateItem(usingCurrentState: orangeView)
}

func handlePan(sender: UIPanGestureRecognizer)
{
    let location = sender.location(in: view)
    let velocity = sender.velocity(in: view)
    let magnitude = sqrt((velocity.x * velocity.x) + (velocity.y * velocity.y))
    switch sender.state
    {
        case .began:
attachment.anchorPoint = location
            dynamicAnimator.addBehavior(attachment)
        case .changed:
attachment.anchorPoint = location
            case .cancelled, .ended, .failed, .possible:
                let push = UIPushBehavior(items: [self.orangeView], mode: .instantaneous)
                push.pushDirection = CGVector(dx: velocity.x, dy: velocity.y)
                push.magnitude = magnitude * magnitudeMultiplier
dynamicAnimator.removeBehavior(attachment)
            dynamicAnimator.addBehavior(push)
    }
}
}

```

## Objective-C

```

@interface ViewController ()

@property (nonatomic, assign) CGFloat magnitudeMultiplier;
@property (nonatomic, strong) UIDynamicAnimator *dynamicAnimator;
@property (nonatomic, strong) UIGravityBehavior *gravity;
@property (nonatomic, strong) UICollisionBehavior *collision;

```

```

        self.view.addSubview(orangeView)
        return orangeView
   }()

lazy var panGesture: UIPanGestureRecognizer =
{
    let panGesture = UIPanGestureRecognizer(target: self, action:
#selector(self.handlePan(sender:)))
    return panGesture
}()

lazy var attachment: UIAttachmentBehavior =
{
    let attachment = UIAttachmentBehavior(item: self.orangeView, attachedToAnchor: .zero)
    return attachment
}()

override func viewDidLoad()
{
    super.viewDidLoad()
dynamicAnimator.addBehavior(gravity)
    dynamicAnimator.addBehavior(collision)
    orangeView.addGestureRecognizer(panGesture)
}

override func viewDidLayoutSubviews()
{
    super.viewDidLayoutSubviews()
orangeView.center = view.center
    dynamicAnimator.updateItem(usingCurrentState: orangeView)
}

func handlePan(sender: UIPanGestureRecognizer)
{
    let location = sender.location(in: view)
    let velocity = sender.velocity(in: view)
    let magnitude = sqrt((velocity.x * velocity.x) + (velocity.y * velocity.y))
    switch sender.state
    {
        case .began:
attachment.anchorPoint = location
            dynamicAnimator.addBehavior(attachment)
        case .changed:
attachment.anchorPoint = location
            case .cancelled, .ended, .failed, .possible:
                let push = UIPushBehavior(items: [self.orangeView], mode: .instantaneous)
                push.pushDirection = CGVector(dx: velocity.x, dy: velocity.y)
                push.magnitude = magnitude * magnitudeMultiplier
dynamicAnimator.removeBehavior(attachment)
            dynamicAnimator.addBehavior(push)
    }
}
}

```

## Objective-C

```

@interface ViewController ()

@property (nonatomic, assign) CGFloat magnitudeMultiplier;
@property (nonatomic, strong) UIDynamicAnimator *dynamicAnimator;
@property (nonatomic, strong) UIGravityBehavior *gravity;
@property (nonatomic, strong) UICollisionBehavior *collision;

```

```

@property (nonatomic, strong) UIView *orangeView;
@property (nonatomic, strong) UIPanGestureRecognizer *panGesture;
@property (nonatomic, strong) UIAttachmentBehavior *attachment;

@end

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    [self.dynamicAnimator addBehavior:self.gravity];
    [self.dynamicAnimator addBehavior:self.collision];
    [self.orangeView addGestureRecognizer:self.panGesture];
    // 调整以改变视图的滑动速度
    self.magnitudeMultiplier = 0.0008f;
}

- (void)viewDidLayoutSubviews
{
    [super viewDidLayoutSubviews];
    self.orangeView.center = self.view.center;
    [self.dynamicAnimator updateItemUsingCurrentState:self.orangeView];
}

- (void)handlePan:(UIPanGestureRecognizer *)sender
{
    CGPoint location = [sender locationInView:self.view];
    CGPoint velocity = [sender velocityInView:self.view];
    CGFloat magnitude = sqrt((velocity.x * velocity.x) + (velocity.y * velocity.y));
    if (sender.state == UIGestureRecognizerStateBegan)
    {
        self.attachment.anchorPoint = location;
        [self.dynamicAnimator addBehavior:self.attachment];
    }
    else if (sender.state == UIGestureRecognizerStateChanged)
    {
        self.attachment.anchorPoint = location;
    }
    else if (sender.state == UIGestureRecognizerStateCancelled ||
              sender.state == UIGestureRecognizerStateEnded ||
              sender.state == UIGestureRecognizerStateFailed ||
              sender.state == UIGestureRecognizerStatePossible)
    {
        UIPushBehavior *push = [[UIPushBehavior alloc] initWithItems:@[self.orangeView]
mode:UIPushBehaviorModeInstantaneous];
        push.pushDirection = CGVectorMake(velocity.x, velocity.y);
        push.magnitude = magnitude * self.magnitudeMultiplier;
        [self.dynamicAnimator removeBehavior:self.attachment];
        [self.dynamicAnimator addBehavior:push];
    }
}

#pragma mark - 延迟初始化
- (UIDynamicAnimator *)dynamicAnimator
{
    if (!_dynamicAnimator)
    {
        _dynamicAnimator = [[UIDynamicAnimator alloc] initWithReferenceView:self.view];
    }
    return _dynamicAnimator;
}

```

```

@property (nonatomic, strong) UIView *orangeView;
@property (nonatomic, strong) UIPanGestureRecognizer *panGesture;
@property (nonatomic, strong) UIAttachmentBehavior *attachment;

@end

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    [self.dynamicAnimator addBehavior:self.gravity];
    [self.dynamicAnimator addBehavior:self.collision];
    [self.orangeView addGestureRecognizer:self.panGesture];
    // Adjust to change speed of view from flick
    self.magnitudeMultiplier = 0.0008f;
}

- (void)viewDidLayoutSubviews
{
    [super viewDidLayoutSubviews];
    self.orangeView.center = self.view.center;
    [self.dynamicAnimator updateItemUsingCurrentState:self.orangeView];
}

- (void)handlePan:(UIPanGestureRecognizer *)sender
{
    CGPoint location = [sender locationInView:self.view];
    CGPoint velocity = [sender velocityInView:self.view];
    CGFloat magnitude = sqrt((velocity.x * velocity.x) + (velocity.y * velocity.y));
    if (sender.state == UIGestureRecognizerStateBegan)
    {
        self.attachment.anchorPoint = location;
        [self.dynamicAnimator addBehavior:self.attachment];
    }
    else if (sender.state == UIGestureRecognizerStateChanged)
    {
        self.attachment.anchorPoint = location;
    }
    else if (sender.state == UIGestureRecognizerStateCancelled ||
              sender.state == UIGestureRecognizerStateEnded ||
              sender.state == UIGestureRecognizerStateFailed ||
              sender.state == UIGestureRecognizerStatePossible)
    {
        UIPushBehavior *push = [[UIPushBehavior alloc] initWithItems:@[self.orangeView]
mode:UIPushBehaviorModeInstantaneous];
        push.pushDirection = CGVectorMake(velocity.x, velocity.y);
        push.magnitude = magnitude * self.magnitudeMultiplier;
        [self.dynamicAnimator removeBehavior:self.attachment];
        [self.dynamicAnimator addBehavior:push];
    }
}

#pragma mark - Lazy Init
- (UIDynamicAnimator *)dynamicAnimator
{
    if (!_dynamicAnimator)
    {
        _dynamicAnimator = [[UIDynamicAnimator alloc] initWithReferenceView:self.view];
    }
    return _dynamicAnimator;
}

```

```

- (UIGravityBehavior *)gravity
{
    if (!_gravity)
    {
        _gravity = [[UIGravityBehavior alloc] initWithItems:@[self.orangeView]];
    }
    return _gravity;
}

- (UICollisionBehavior *)collision
{
    if (!_collision)
    {
        _collision = [[UICollisionBehavior alloc] initWithItems:@[self.orangeView]];
        _collision.translatesReferenceBoundsIntoBoundary = YES;
    }
    return _collision;
}

- (UIView *)orangeView
{
    if (!_orangeView)
    {
        CGFloat widthHeight = 40.0f;
        _orangeView = [[UIView alloc]initWithFrame:CGRectMake(0.0, 0.0, widthHeight, widthHeight)];
        _orangeView.backgroundColor = [UIColor orangeColor];
        [self.view addSubview:_orangeView];
    }
    return _orangeView;
}

- (UIPanGestureRecognizer *)panGesture
{
    如果(!_panGesture)
    {
        _panGesture = [[UIPanGestureRecognizer 分配]initWithTarget:self
action:@selector(handlePan:)];
    }
    return _panGesture;
}

- (UIAttachmentBehavior *)attachment
{
    如果(!_attachment)
    {
        _attachment = [[UIAttachmentBehavior 分配]initWithItem:self.orangeView
attachedToAnchor:CGPointZero];
    }
    return _attachment;
}

@end

```

## 第59.2节：“粘性角落”效果使用UIFieldBehaviors

此示例展示了如何实现类似FaceTime的效果，当视图进入特定区域时会被吸引到该点，在此例中是上下两个区域。

```

- (UIGravityBehavior *)gravity
{
    if (!_gravity)
    {
        _gravity = [[UIGravityBehavior alloc] initWithItems:@[self.orangeView]];
    }
    return _gravity;
}

- (UICollisionBehavior *)collision
{
    if (!_collision)
    {
        _collision = [[UICollisionBehavior alloc] initWithItems:@[self.orangeView]];
        _collision.translatesReferenceBoundsIntoBoundary = YES;
    }
    return _collision;
}

- (UIView *)orangeView
{
    if (!_orangeView)
    {
        CGFloat widthHeight = 40.0f;
        _orangeView = [[UIView alloc]initWithFrame:CGRectMake(0.0, 0.0, widthHeight, widthHeight)];
        _orangeView.backgroundColor = [UIColor orangeColor];
        [self.view addSubview:_orangeView];
    }
    return _orangeView;
}

- (UIPanGestureRecognizer *)panGesture
{
    if (!_panGesture)
    {
        _panGesture = [[UIPanGestureRecognizer alloc] initWithTarget:self
action:@selector(handlePan:)];
    }
    return _panGesture;
}

- (UIAttachmentBehavior *)attachment
{
    if (!_attachment)
    {
        _attachment = [[UIAttachmentBehavior alloc] initWithItem:self.orangeView
attachedToAnchor:CGPointZero];
    }
    return _attachment;
}

@end

```

## Section 59.2: "Sticky Corners" Effect Using UIFieldBehaviors

This example shows how to achieve an effect similar to FaceTime where a view is attracted to point once it enters a particular region, in this case two regions a top and bottom.

**Swift**

```

class ViewController: UIViewController
{
    lazy var dynamicAnimator: UIDynamicAnimator =
    {
        let dynamicAnimator = UIDynamicAnimator(referenceView: self.view)
        return dynamicAnimator
    }()

    lazy var collision: UICollisionBehavior =
    {
        let collision = UICollisionBehavior(items: [self.orangeView])
        collision.translatesReferenceBoundsIntoBoundary = true
        return collision
    }()

    lazy var fieldBehaviors: [UIFieldBehavior] =
    {
        var fieldBehaviors = [UIFieldBehavior]()
        for _ in 0 ..< 2
        {
            let field = UIFieldBehavior.springField()
            field.addItem(self.orangeView)
            fieldBehaviors.append(field)
        }
        return fieldBehaviors
    }()

    lazy var itemBehavior: UIDynamicItemBehavior =
    {
        let itemBehavior = UIDynamicItemBehavior(items: [self.orangeView])
        // 调整这些值以改变视图的“粘性”
        itemBehavior.density = 0.01
        itemBehavior.resistance = 10
        itemBehavior.friction = 0.0
        itemBehavior.allowsRotation = false
        return itemBehavior
    }()

    lazy var orangeView: UIView =
    {
        let widthHeight: CGFloat = 40.0
        let orangeView = UIView(frame: CGRect(x: 0.0, y: 0.0, width: widthHeight, height:
    
```

**Swift**

```

class ViewController: UIViewController
{
    lazy var dynamicAnimator: UIDynamicAnimator =
    {
        let dynamicAnimator = UIDynamicAnimator(referenceView: self.view)
        return dynamicAnimator
    }()

    lazy var collision: UICollisionBehavior =
    {
        let collision = UICollisionBehavior(items: [self.orangeView])
        collision.translatesReferenceBoundsIntoBoundary = true
        return collision
    }()

    lazy var fieldBehaviors: [UIFieldBehavior] =
    {
        var fieldBehaviors = [UIFieldBehavior]()
        for _ in 0 ..< 2
        {
            let field = UIFieldBehavior.springField()
            field.addItem(self.orangeView)
            fieldBehaviors.append(field)
        }
        return fieldBehaviors
    }()

    lazy var itemBehavior: UIDynamicItemBehavior =
    {
        let itemBehavior = UIDynamicItemBehavior(items: [self.orangeView])
        // Adjust these values to change the "stickiness" of the view
        itemBehavior.density = 0.01
        itemBehavior.resistance = 10
        itemBehavior.friction = 0.0
        itemBehavior.allowsRotation = false
        return itemBehavior
    }()

    lazy var orangeView: UIView =
    {
        let widthHeight: CGFloat = 40.0
        let orangeView = UIView(frame: CGRect(x: 0.0, y: 0.0, width: widthHeight, height:
    
```

```

widthHeight))
orangeView.backgroundColor = UIColor.orange
    self.view.addSubview(orangeView)
    return orangeView
}()

lazy var panGesture: UIPanGestureRecognizer =
{
    let panGesture = UIPanGestureRecognizer(target: self, action:
#selector(self.handlePan(sender:)))
    return panGesture
}()

lazy var attachment: UIAttachmentBehavior =
{
    let attachment = UIAttachmentBehavior(item: self.orangeView, attachedToAnchor: .zero)
    return attachment
}()

override func viewDidLoad()
{
    super.viewDidLoad()
dynamicAnimator.addBehavior(collision)
    dynamicAnimator.addBehavior(itemBehavior)
    for field in fieldBehaviors
    {
dynamicAnimator.addBehavior(field)
    }

orangeView.addGestureRecognizer(panGesture)
}

override func viewDidLayoutSubviews()
{
    super.viewDidLayoutSubviews()

orangeView.center = view.center
    dynamicAnimator.updateItem(usingCurrentState: orangeView)

    for (index, field) in fieldBehaviors.enumerated()
    {
field.position = CGPoint(x: view.bounds
        .midX, y: view.bounds.height * (0.25 + 0.5 * CGFloat(index)))
        field.region = UIRegion(size: CGSize(width: view.bounds.width, height:
view.bounds.height * 0.5))
    }
}

func handlePan(sender: UIPanGestureRecognizer)
{
    let location = sender.location(in: view)
    let velocity = sender.velocity(in: view)
    switch sender.state
    {
        case .began:
attachment.anchorPoint = location
            dynamicAnimator.addBehavior(attachment)
        case .changed:
attachment.anchorPoint = location
            case .cancelled, .ended, .failed, .possible:
itemBehavior.addLinearVelocity(velocity, for: self.orangeView)
            dynamicAnimator.removeBehavior(attachment)
    }
}

```

```

widthHeight))
orangeView.backgroundColor = UIColor.orange
    self.view.addSubview(orangeView)
    return orangeView
}()

lazy var panGesture: UIPanGestureRecognizer =
{
    let panGesture = UIPanGestureRecognizer(target: self, action:
#selector(self.handlePan(sender:)))
    return panGesture
}()

lazy var attachment: UIAttachmentBehavior =
{
    let attachment = UIAttachmentBehavior(item: self.orangeView, attachedToAnchor: .zero)
    return attachment
}()

override func viewDidLoad()
{
    super.viewDidLoad()
dynamicAnimator.addBehavior(collision)
    dynamicAnimator.addBehavior(itemBehavior)
    for field in fieldBehaviors
    {
        dynamicAnimator.addBehavior(field)
    }

    orangeView.addGestureRecognizer(panGesture)
}

override func viewDidLayoutSubviews()
{
    super.viewDidLayoutSubviews()

orangeView.center = view.center
    dynamicAnimator.updateItem(usingCurrentState: orangeView)

    for (index, field) in fieldBehaviors.enumerated()
    {
        field.position = CGPoint(x: view.bounds
            .midX, y: view.bounds.height * (0.25 + 0.5 * CGFloat(index)))
        field.region = UIRegion(size: CGSize(width: view.bounds.width, height:
view.bounds.height * 0.5))
    }
}

func handlePan(sender: UIPanGestureRecognizer)
{
    let location = sender.location(in: view)
    let velocity = sender.velocity(in: view)
    switch sender.state
    {
        case .began:
            attachment.anchorPoint = location
            dynamicAnimator.addBehavior(attachment)
        case .changed:
            attachment.anchorPoint = location
        case .cancelled, .ended, .failed, .possible:
            itemBehavior.addLinearVelocity(velocity, for: self.orangeView)
            dynamicAnimator.removeBehavior(attachment)
    }
}

```

```
    }
}
```

## Objective-C

```
@interface ViewController ()  
  
@property (nonatomic, strong) UIDynamicAnimator *dynamicAnimator;  
@property (nonatomic, strong) UICollisionBehavior *collision;  
@property (nonatomic, strong) UIAttachmentBehavior *attachment;  
@property (nonatomic, strong) UIDynamicItemBehavior *itemBehavior;  
@property (nonatomic, strong) NSArray <UIFieldBehavior *> *fieldBehaviors;  
@property (nonatomic, strong) UIView *orangeView;  
@property (nonatomic, strong) UIPanGestureRecognizer *panGesture;  
  
@end  
  
@implementation ViewController  
  
- (void)viewDidLoad  
{  
    [super viewDidLoad];  
    [self.dynamicAnimator addBehavior:self.collision];  
    [self.dynamicAnimator addBehavior:self.itemBehavior];  
    for (UIFieldBehavior *field in self.fieldBehaviors)  
    {  
        [self.dynamicAnimator addBehavior:field];  
    }  
  
    [self.orangeView addGestureRecognizer:self.panGesture];  
}  
  
- (void)viewDidLayoutSubviews  
{  
    [super viewDidLayoutSubviews];  
    self.orangeView.center = self.view.center;  
    [self.dynamicAnimator updateItemUsingCurrentState:self.orangeView];  
  
    for (NSInteger i = 0; i < self.fieldBehaviors.count; i++)  
    {  
        UIFieldBehavior *field = self.fieldBehaviors[i];  
        field.position = CGPointMake(CGRectGetMidX(self.view.bounds),  
                                     CGRectGetHeight(self.view.bounds) * (0.25f + 0.5f * i));  
        field.region = [[UIRegion alloc] initWithSize:CGSizeMake(CGRectGetWidth(self.view.bounds),  
                                                               CGRectGetHeight(self.view.bounds) * 0.5)];  
    }  
}  
  
- (void)handlePan:(UIPanGestureRecognizer *)sender  
{  
    CGPoint location = [sender locationInView:self.view];  
    CGPoint velocity = [sender velocityInView:self.view];  
    if (sender.state == UIGestureRecognizerStateBegan)  
    {  
        self.attachment.anchorPoint = location;  
        [self.dynamicAnimator addBehavior:self.attachment];  
    }  
    else if (sender.state == UIGestureRecognizerStateChanged)  
    {  
        self.attachment.anchorPoint = location;  
    }  
}
```

```
    }
}
```

## Objective-C

```
@interface ViewController ()  
  
@property (nonatomic, strong) UIDynamicAnimator *dynamicAnimator;  
@property (nonatomic, strong) UICollisionBehavior *collision;  
@property (nonatomic, strong) UIAttachmentBehavior *attachment;  
@property (nonatomic, strong) UIDynamicItemBehavior *itemBehavior;  
@property (nonatomic, strong) NSArray <UIFieldBehavior *> *fieldBehaviors;  
@property (nonatomic, strong) UIView *orangeView;  
@property (nonatomic, strong) UIPanGestureRecognizer *panGesture;  
  
@end  
  
@implementation ViewController  
  
- (void)viewDidLoad  
{  
    [super viewDidLoad];  
    [self.dynamicAnimator addBehavior:self.collision];  
    [self.dynamicAnimator addBehavior:self.itemBehavior];  
    for (UIFieldBehavior *field in self.fieldBehaviors)  
    {  
        [self.dynamicAnimator addBehavior:field];  
    }  
  
    [self.orangeView addGestureRecognizer:self.panGesture];  
}  
  
- (void)viewDidLayoutSubviews  
{  
    [super viewDidLayoutSubviews];  
    self.orangeView.center = self.view.center;  
    [self.dynamicAnimator updateItemUsingCurrentState:self.orangeView];  
  
    for (NSInteger i = 0; i < self.fieldBehaviors.count; i++)  
    {  
        UIFieldBehavior *field = self.fieldBehaviors[i];  
        field.position = CGPointMake(CGRectGetMidX(self.view.bounds),  
                                     CGRectGetHeight(self.view.bounds) * (0.25f + 0.5f * i));  
        field.region = [[UIRegion alloc] initWithSize:CGSizeMake(CGRectGetWidth(self.view.bounds),  
                                                               CGRectGetHeight(self.view.bounds) * 0.5)];  
    }  
}  
  
- (void)handlePan:(UIPanGestureRecognizer *)sender  
{  
    CGPoint location = [sender locationInView:self.view];  
    CGPoint velocity = [sender velocityInView:self.view];  
    if (sender.state == UIGestureRecognizerStateBegan)  
    {  
        self.attachment.anchorPoint = location;  
        [self.dynamicAnimator addBehavior:self.attachment];  
    }  
    else if (sender.state == UIGestureRecognizerStateChanged)  
    {  
        self.attachment.anchorPoint = location;  
    }  
}
```

```

else if (sender.state == UIGestureRecognizerStateCancelled ||
    sender.state == UIGestureRecognizerStateEnded ||
    sender.state == UIGestureRecognizerStateFailed ||
    sender.state == UIGestureRecognizerStatePossible)
{
    [self.itemBehavior addLinearVelocity:velocity forItem:self.orangeView];
    [self.dynamicAnimator removeBehavior:self.attachment];
}
}

#pragma mark - 延迟初始化
- (UIDynamicAnimator *)dynamicAnimator
{
    if (!_dynamicAnimator)
    {
        _dynamicAnimator = [[UIDynamicAnimator alloc] initWithReferenceView:self.view];
    }
    return _dynamicAnimator;
}

- (UICollisionBehavior *)collision
{
    if (!_collision)
    {
        _collision = [[UICollisionBehavior alloc] initWithItems:@[self.orangeView]];
        _collision.translatesReferenceBoundsIntoBoundary = YES;
    }
    return _collision;
}

- (NSArray <UIFieldBehavior *> *)fieldBehaviors
{
    if (!_fieldBehaviors)
    {
        NSMutableArray *fields = [[NSMutableArray alloc] init];
        for (NSInteger i = 0; i < 2; i++)
        {
            UIFieldBehavior *field = [UIFieldBehavior springField];
            [field addItem:self.orangeView];
            [fields addObject:field];
        }
        _fieldBehaviors = fields;
    }
    return _fieldBehaviors;
}

- (UIDynamicItemBehavior *)itemBehavior
{
    if (!_itemBehavior)
    {
        _itemBehavior = [[UIDynamicItemBehavior alloc] initWithItems:@[self.orangeView]];
        // 调整这些值以改变视图的“粘性”
        _itemBehavior.density = 0.01;
        _itemBehavior.resistance = 10;
        _itemBehavior.friction = 0.0;
        _itemBehavior.allowsRotation = NO;
    }
    return _itemBehavior;
}

- (UIView *)orangeView
{

```

```

else if (sender.state == UIGestureRecognizerStateCancelled ||
    sender.state == UIGestureRecognizerStateEnded ||
    sender.state == UIGestureRecognizerStateFailed ||
    sender.state == UIGestureRecognizerStatePossible)
{
    [self.itemBehavior addLinearVelocity:velocity forItem:self.orangeView];
    [self.dynamicAnimator removeBehavior:self.attachment];
}
}

#pragma mark - Lazy Init
- (UIDynamicAnimator *)dynamicAnimator
{
    if (!_dynamicAnimator)
    {
        _dynamicAnimator = [[UIDynamicAnimator alloc] initWithReferenceView:self.view];
    }
    return _dynamicAnimator;
}

- (UICollisionBehavior *)collision
{
    if (!_collision)
    {
        _collision = [[UICollisionBehavior alloc] initWithItems:@[self.orangeView]];
        _collision.translatesReferenceBoundsIntoBoundary = YES;
    }
    return _collision;
}

- (NSArray <UIFieldBehavior *> *)fieldBehaviors
{
    if (!_fieldBehaviors)
    {
        NSMutableArray *fields = [[NSMutableArray alloc] init];
        for (NSInteger i = 0; i < 2; i++)
        {
            UIFieldBehavior *field = [UIFieldBehavior springField];
            [field addItem:self.orangeView];
            [fields addObject:field];
        }
        _fieldBehaviors = fields;
    }
    return _fieldBehaviors;
}

- (UIDynamicItemBehavior *)itemBehavior
{
    if (!_itemBehavior)
    {
        _itemBehavior = [[UIDynamicItemBehavior alloc] initWithItems:@[self.orangeView]];
        // Adjust these values to change the "stickiness" of the view
        _itemBehavior.density = 0.01;
        _itemBehavior.resistance = 10;
        _itemBehavior.friction = 0.0;
        _itemBehavior.allowsRotation = NO;
    }
    return _itemBehavior;
}

- (UIView *)orangeView
{

```

```

if (!_orangeView)
{
    CGFloat widthHeight = 40.0f;
    _orangeView = [[UIView alloc] initWithFrame:CGRectMake(0.0, 0.0, widthHeight, widthHeight)];
    _orangeView.backgroundColor = [UIColor orangeColor];
    [self.view addSubview:_orangeView];
}
返回 _orangeView;
}

- (UIPanGestureRecognizer *)panGesture
{
    如果 (!_panGesture)
    {
        _panGesture = [[UIPanGestureRecognizer 分配]initWithTarget:self
action:@selector(handlePan:)];
    }
    返回 _panGesture;
}

- (UIAttachmentBehavior *)attachment
{
    如果 (!_attachment)
    {
        _attachment = [[UIAttachmentBehavior 分配]initWithItem:self.orangeView
attachedToAnchor:CGPointZero];
    }
    返回 _attachment;
}

@end

```

有关UIFieldBehaviors的更多信息，您可以查看2015 WWDC 会议“[UIKit Dynamics 和视觉效果的新特性](#)”及其附带的示例代码。

## 第59.3节：UIDynamicBehavior驱动的自定义转场



本示例展示了如何创建由复合  
UIDynamicBehavior驱动的自定义展示转场。我们可以从创建一个将呈现模态视图的展示视图控制器开始。

### Swift

```
class PresentingViewController: UIViewController
```

```

if (!_orangeView)
{
    CGFloat widthHeight = 40.0f;
    _orangeView = [[UIView alloc] initWithFrame:CGRectMake(0.0, 0.0, widthHeight, widthHeight)];
    _orangeView.backgroundColor = [UIColor orangeColor];
    [self.view addSubview:_orangeView];
}
return _orangeView;
}

- (UIPanGestureRecognizer *)panGesture
{
    if (!_panGesture)
    {
        _panGesture = [[UIPanGestureRecognizer alloc] initWithTarget:self
action:@selector(handlePan:)];
    }
    return _panGesture;
}

- (UIAttachmentBehavior *)attachment
{
    if (!_attachment)
    {
        _attachment = [[UIAttachmentBehavior alloc] initWithItem:self.orangeView
attachedToAnchor:CGPointZero];
    }
    return _attachment;
}

@end

```

For more information about UIFieldBehaviors you can see the [2015 WWDC Session "What's New in UIKit Dynamics and Visual Effects"](#) and accompanying [sample code](#).

## Section 59.3: UIDynamicBehavior Driven Custom Transition



This example shows how to create a custom presentation transition that is driven by a composite  
UIDynamicBehavior. We can start by creating a presenting view controller that will present a modal.

### Swift

```
class PresentingViewController: UIViewController
```

```

{
    lazy var button: UIButton =
    {
        let button = UIButton()
        button.translatesAutoresizingMaskIntoConstraints = false
        self.view.addSubview(button)
        button.centerXAnchor.constraint(equalTo: self.view.centerXAnchor).isActive
            = true
        button.centerYAnchor.constraint(equalTo: self.view.centerYAnchor).isActive = true
            button.setTitle("Present", for: .normal)
        button.setTextColor(UIColor.blue, for: .normal)

        return button
    }()

    override func viewDidLoad()
    {
        super.viewDidLoad()
        button.addTarget(self, action: #selector(self.didPressPresent), for: .touchUpInside)
    }

    func didPressPresent()
    {
        let modal = ModalViewController()
        modal.view.frame = CGRect(x: 0.0, y: 0.0, width: 200.0, height: 200.0)
        modal.modalPresentationStyle = .custom
        modal.transitioningDelegate = modal
        self.present(modal, animated: true)
    }
}

```

## Objective-C

```

@interface PresentingViewController ()
@property (nonatomic, strong) UIButton *button;
@end

@implementation PresentingViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    [self.button 添加目标:self 动作:@selector(didPressPresent)
针对控制事件: UIControlEventTouchUpInside];
}

- (void)didPressPresent
{
    ModalViewController *modal = [[ModalViewController alloc] 初始化];
    modal.view.frame = CGRectMake(0.0, 0.0, 200.0, 200.0);
    modal.modalPresentationStyle = UIModalPresentationCustom;
    modal.transitioningDelegate = modal;
    [self presentViewController:modal 动画:YES 完成:nil];
}

- (UIButton *)button
{
    if (!_button)
    {
        _button = [[UIButton 分配] 初始化];
        _button.translatesAutoresizingMaskIntoConstraints = NO;
        [self.view 添加子视图:_button];
    }
}

```

```

{
    lazy var button: UIButton =
    {
        let button = UIButton()
        button.translatesAutoresizingMaskIntoConstraints = false
        self.view.addSubview(button)
        button.centerXAnchor.constraint(equalTo: self.view.centerXAnchor).isActive
            = true
        button.centerYAnchor.constraint(equalTo: self.view.centerYAnchor).isActive = true
            button.setTitle("Present", for: .normal)
        button.setTextColor(UIColor.blue, for: .normal)

        return button
    }()

    override func viewDidLoad()
    {
        super.viewDidLoad()
        button.addTarget(self, action: #selector(self.didPressPresent), for: .touchUpInside)
    }

    func didPressPresent()
    {
        let modal = ModalViewController()
        modal.view.frame = CGRect(x: 0.0, y: 0.0, width: 200.0, height: 200.0)
        modal.modalPresentationStyle = .custom
        modal.transitioningDelegate = modal
        self.present(modal, animated: true)
    }
}

```

## Objective-C

```

@interface PresentingViewController ()
@property (nonatomic, strong) UIButton *button;
@end

@implementation PresentingViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    [self.button addTarget:self action:@selector(didPressPresent)
forControlEvents:UIControlEventTouchUpInside];
}

- (void)didPressPresent
{
    ModalViewController *modal = [[ModalViewController alloc] init];
    modal.view.frame = CGRectMake(0.0, 0.0, 200.0, 200.0);
    modal.modalPresentationStyle = UIModalPresentationCustom;
    modal.transitioningDelegate = modal;
    [self presentViewController:modal animated:YES completion:nil];
}

- (UIButton *)button
{
    if (!_button)
    {
        _button = [[UIButton alloc] init];
        _button.translatesAutoresizingMaskIntoConstraints = NO;
        [self.view addSubview:_button];
    }
}

```

```

[_button.centerXAnchor 约束等于锚点:self.view.centerXAnchor].激活 = YES;
[_button.centerYAnchor 约束等于锚点:self.view.centerYAnchor].激活 = YES;
[_button 设置标题:@"Present" 状态:UIControlStateNormal];
[_button 设置标题颜色:[UIColor 蓝色] 状态:UIControlStateNormal];
}
return _button;
}

@end

```

当点击 present 按钮时，我们创建一个 ModalViewController 并将其展示样式设置为 .custom，并将其 transitionDelegate 设置为自身。这将允许我们提供一个动画控制器来驱动其模态转换。我们还设置了 modal 的视图的框架，使其比全屏小。

现在让我们来看一下ModalViewController：

### Swift

```

class ModalViewController: UIViewController
{
    lazy var button: UIButton =
    {
        let button = UIButton()
        button.translatesAutoresizingMaskIntoConstraints = false
        self.view.addSubview(button)
        button.centerXAnchor.constraint(equalTo: self.view.centerXAnchor).isActive
            = true
        button.centerYAnchor.constraint(equalTo: self.view.centerYAnchor).isActive = true
        button.setTitle("Dismiss", for: .normal)
        button.setTitleColor(.white, for: .normal)

        return button
    }()

    override func viewDidLoad()
    {
        super.viewDidLoad()
        button.addTarget(self, action: #selector(self.didPressDismiss), for: .touchUpInside)
        view.backgroundColor = .red
        view.layer.cornerRadius = 15.0
    }

    func didPressDismiss()
    {
        dismiss(animated: true)
    }
}

extension ModalViewController: UIViewControllerTransitioningDelegate
{
    func animationController(forPresented presented: UIViewController, presenting: UIViewController, 来源: UIViewController) -> UIViewControllerAnimatedTransitioning?
    {
        return DropOutAnimator(duration: 1.5, isAppearing: true)
    }

    func animationController(forDismissed dismissed: UIViewController) -> UIViewControllerAnimatedTransitioning?
    {
        return DropOutAnimator(duration: 4.0, isAppearing: false)
    }
}

```

```

[_button.centerXAnchor constraintEqualToAnchor:self.view.centerXAnchor].active = YES;
[_button.centerYAnchor constraintEqualToAnchor:self.view.centerYAnchor].active = YES;
[_button setTitle:@"Present" forState:UIControlStateNormal];
[_button setTitleColor:[UIColor blueColor] forState:UIControlStateNormal];
}
return _button;
}

@end

```

When the present button is tapped, we create a ModalViewController and set its presentation style to .custom and set its transitionDelegate to itself. This will allow us to vend an animator that will drive its modal transition. We also set modal's view's frame so it will be smaller than the full screen.

Let's now look at ModalViewController:

### Swift

```

class ModalViewController: UIViewController
{
    lazy var button: UIButton =
    {
        let button = UIButton()
        button.translatesAutoresizingMaskIntoConstraints = false
        self.view.addSubview(button)
        button.centerXAnchor.constraint(equalTo: self.view.centerXAnchor).isActive
            = true
        button.centerYAnchor.constraint(equalTo: self.view.centerYAnchor).isActive = true
        button.setTitle("Dismiss", for: .normal)
        button.setTitleColor(.white, for: .normal)

        return button
    }()

    override func viewDidLoad()
    {
        super.viewDidLoad()
        button.addTarget(self, action: #selector(self.didPressDismiss), for: .touchUpInside)
        view.backgroundColor = .red
        view.layer.cornerRadius = 15.0
    }

    func didPressDismiss()
    {
        dismiss(animated: true)
    }
}

extension ModalViewController: UIViewControllerTransitioningDelegate
{
    func animationController(forPresented presented: UIViewController, presenting: UIViewController, source: UIViewController) -> UIViewControllerAnimatedTransitioning?
    {
        return DropOutAnimator(duration: 1.5, isAppearing: true)
    }

    func animationController(forDismissed dismissed: UIViewController) -> UIViewControllerAnimatedTransitioning?
    {
        return DropOutAnimator(duration: 4.0, isAppearing: false)
    }
}

```

}

## Objective-C

```

@interface ModalViewController () <UIViewControllerAnimatedTransitioningDelegate>
@property (nonatomic, strong) UIButton *button;
@end

@implementation ModalViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    [self.button addTarget:self action:@selector(didPressPresent)
forControlEvents:UIControlEventTouchUpInside];
    self.view.backgroundColor = [UIColor redColor];
    self.view.layer.cornerRadius = 15.0f;
}

- (void)didPressPresent
{
    [self dismissViewControllerAnimated:YES completion:nil];
}

- (UIButton *)button
{
    if (!_button)
    {
        _button = [[UIButton 分配] 初始化];
        _button.translatesAutoresizingMaskIntoConstraints = NO;
        [self.view 添加子视图:_button];
        [_button.centerXAnchor constraintEqualToAnchor:self.view.centerXAnchor].active = YES;
        [_button.centerYAnchor constraintEqualToAnchor:self.view.centerYAnchor].active = YES;
        [_button setTitle:@"Dismiss" forState:UIControlStateNormal];
        [_button 设置标题颜色:[UIColor 蓝色] 状态:UIControlStateNormal];
    }
    return _button;
}

- (id<UIViewControllerAnimatedTransitioning>)animationControllerForPresentedController:(UIViewController *)presented presentingController:(UIViewController *)presenting
sourceController:(UIViewController *)source
{
    return [[DropOutAnimator alloc] initWithDuration: 1.5 appearing:YES];
}

- (id<UIViewControllerAnimatedTransitioning>)animationControllerForDismissedController:(UIViewController *)dismissed
{
    return [[DropOutAnimator alloc] initWithDuration:4.0 appearing:NO];
}

```

这里我们创建了被呈现的视图控制器。由于ModalViewController是它自己的transitioningDelegate，因此它也负责提供一个管理其过渡动画的对象。对我们来说，这意味着传递一个我们复合UIDynamicBehavior子类的实例。

我们的动画器将有两种不同的过渡：一种用于呈现，一种用于消失。对于呈现，

}

## Objective-C

```

@interface ModalViewController () <UIViewControllerAnimatedTransitioningDelegate>
@property (nonatomic, strong) UIButton *button;
@end

@implementation ModalViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    [self.button addTarget:self action:@selector(didPressPresent)
forControlEvents:UIControlEventTouchUpInside];
    self.view.backgroundColor = [UIColor redColor];
    self.view.layer.cornerRadius = 15.0f;
}

- (void)didPressPresent
{
    [self dismissViewControllerAnimated:YES completion:nil];
}

- (UIButton *)button
{
    if (!_button)
    {
        _button = [[UIButton alloc] init];
        _button.translatesAutoresizingMaskIntoConstraints = NO;
        [self.view addSubview:_button];
        [_button.centerXAnchor constraintEqualToAnchor:self.view.centerXAnchor].active = YES;
        [_button.centerYAnchor constraintEqualToAnchor:self.view.centerYAnchor].active = YES;
        [_button setTitle:@"Dismiss" forState:UIControlStateNormal];
        [_button setTitleColor:[UIColor blueColor] forState:UIControlStateNormal];
    }
    return _button;
}

- (id<UIViewControllerAnimatedTransitioning>)animationControllerForPresentedController:(UIViewController *)presented presentingController:(UIViewController *)presenting
sourceController:(UIViewController *)source
{
    return [[DropOutAnimator alloc] initWithDuration: 1.5 appearing:YES];
}

- (id<UIViewControllerAnimatedTransitioning>)animationControllerForDismissedController:(UIViewController *)dismissed
{
    return [[DropOutAnimator alloc] initWithDuration:4.0 appearing:NO];
}

```

Here we create the view controller that is presented. Also because ModalViewController is its own transitioningDelegate it is also responsible for vending an object that will manage its transition animation. For us that means passing on an instance of our composite `UIDynamicBehavior` subclass.

Our animator will have two different transitions: one for presenting and one for dismissing. For presenting, the

呈现视图控制器的视图将从上方掉落。对于消失，视图看起来像是从一根绳子上摆动然后掉落。因为DropOutAnimator遵循UIViewControllerAnimatedTransitioning协议，大部分工作将在其func animateTransition(using transitionContext:)的实现中完成。

UIViewControllerContextTransitioning).

## Swift

```
class DropOutAnimator: UIDynamicBehavior
{
    let duration: TimeInterval
    let isAppearing: Bool

    var transitionContext: UIViewControllerContextTransitioning?
    var hasElapsedDurationExceeded = false
    var finishTime: TimeInterval = 0.0
    var collisionBehavior: UICollisionBehavior?
    var attachmentBehavior: UIAttachmentBehavior?
    var animator: UIDynamicAnimator?

    init(duration: TimeInterval = 1.0, isAppearing: Bool)
    {
        self.duration = duration
        self.isAppearing = isAppearing
        super.init()
    }

extension DropOutAnimator: UIViewControllerAnimatedTransitioning
{
    func animateTransition(using transitionContext: UIViewControllerContextTransitioning)
    {
        // 从 transitionContext 获取相关视图和视图控制器
        guard let fromVC = transitionContext.viewController(forKey: .from),
              let toVC = transitionContext.viewController(forKey: .to),
              let fromView = fromVC.view,
              let toView = toVC.view else { return }

        let containerView = transitionContext.containerView
        let duration = self.transitionDuration(using: transitionContext)

        // 持有 transitionContext 的引用以通知其完成
        self.transitionContext = transitionContext

        // 创建动态动画器
        let animator = UIDynamicAnimator(referenceView: containerView)
        animator.delegate = self
        self.animator = animator

        // 展示动画
        if self.isAppearing
        {
            fromView.isUserInteractionEnabled = false

            // 视图 toView 位置刚好在屏幕外
            let fromViewInitialFrame = transitionContext.initialFrame(for: fromVC)
            var toViewInitialFrame = toView.frame
            toViewInitialFrame.origin.y -= toViewInitialFrame.height
            toViewInitialFrame.origin.x = fromViewInitialFrame.width * 0.5 -
            toViewInitialFrame.width * 0.5
            toView.frame = toViewInitialFrame

            containerView.addSubview(toView)
        }
    }
}
```

presenting view controller's view will drop in from above. And for dismissing, the view will seem to swing from a rope and then drop out. Because DropOutAnimator conforms to `UIViewControllerAnimatedTransitioning` most of this work will be done in its implementation of `func animateTransition(using transitionContext: UIViewControllerContextTransitioning)`.

## Swift

```
class DropOutAnimator: UIDynamicBehavior
{
    let duration: TimeInterval
    let isAppearing: Bool

    var transitionContext: UIViewControllerContextTransitioning?
    var hasElapsedDurationExceeded = false
    var finishTime: TimeInterval = 0.0
    var collisionBehavior: UICollisionBehavior?
    var attachmentBehavior: UIAttachmentBehavior?
    var animator: UIDynamicAnimator?

    init(duration: TimeInterval = 1.0, isAppearing: Bool)
    {
        self.duration = duration
        self.isAppearing = isAppearing
        super.init()
    }

extension DropOutAnimator: UIViewControllerAnimatedTransitioning
{
    func animateTransition(using transitionContext: UIViewControllerContextTransitioning)
    {
        // Get relevant views and view controllers from transitionContext
        guard let fromVC = transitionContext.viewController(forKey: .from),
              let toVC = transitionContext.viewController(forKey: .to),
              let fromView = fromVC.view,
              let toView = toVC.view else { return }

        let containerView = transitionContext.containerView
        let duration = self.transitionDuration(using: transitionContext)

        // Hold reference to transitionContext to notify it of completion
        self.transitionContext = transitionContext

        // Create dynamic animator
        let animator = UIDynamicAnimator(referenceView: containerView)
        animator.delegate = self
        self.animator = animator

        // Presenting Animation
        if self.isAppearing
        {
            fromView.isUserInteractionEnabled = false

            // Position toView just off-screen
            let fromViewInitialFrame = transitionContext.initialFrame(for: fromVC)
            var toViewInitialFrame = toView.frame
            toViewInitialFrame.origin.y -= toViewInitialFrame.height
            toViewInitialFrame.origin.x = fromViewInitialFrame.width * 0.5 -
            toViewInitialFrame.width * 0.5
            toView.frame = toViewInitialFrame

            containerView.addSubview(toView)
        }
    }
}
```

```

// 防止旋转并调整弹跳
let bodyBehavior = UIDynamicItemBehavior(items: [toView])
bodyBehavior.elasticity = 0.7
bodyBehavior.allowsRotation = false

// 以夸张的力度添加重力，使动画看起来不慢
let gravityBehavior = UIGravityBehavior(items: [toView])
gravityBehavior.magnitude = 10.0

// 设置碰撞边界以包含屏幕外视图，并在中心发生碰撞
// 这是我们最终视图应停留的位置
let collisionBehavior = UICollisionBehavior(items: [toView])
let insets = UIEdgeInsets(top: toViewInitialFrame.minY, left: 0.0, bottom:
fromViewInitialFrame.height * 0.5 - toViewInitialFrame.height * 0.5, right: 0.0)
collisionBehavior.setTranslatesReferenceBoundsIntoBoundary(with: insets)
self.collisionBehavior = collisionBehavior

// 跟踪结束时间，以防需要在动画暂停前结束动画器
self.finishTime = duration + (self.animator?.elapsedTime ?? 0.0)

// 动画器每次“滴答”后调用的闭包
// 检查是否超过持续时间
self.action =
{ [weak self] in
guard let strongSelf = self,
      (strongSelf.animator?.elapsedTime ?? 0.0) >= strongSelf.finishTime else { return
}
strongSelf.hasElapsedDurationExceeded = true
strongSelf.animator?.removeBehavior(strongSelf)
}

// `DropOutAnimator` 是一个组合行为，因此将子行为添加到自身
self.addChildBehavior(collisionBehavior)
self.addChildBehavior(bodyBehavior)
self.addChildBehavior(gravityBehavior)

// 将自身添加到动态动画器
self.animator?.addBehavior(self)
}

// 取消动画
else
{
    // 创建允许旋转并具有弹性的项目
    let bodyBehavior = UIDynamicItemBehavior(items: [fromView])
    bodyBehavior.elasticity = 0.8
    bodyBehavior.angularResistance = 5.0
    bodyBehavior.allowsRotation = true

    // 创建具有夸大幅度的重力
    let gravityBehavior = UIGravityBehavior(items: [fromView])
    gravityBehavior.magnitude = 10.0

    // 碰撞边界设置为屏幕底部正下方有一个地板
    let collisionBehavior = UICollisionBehavior(items: [fromView])
    let insets = UIEdgeInsets(top: 0.0, left: -1000, bottom: -225, right: -1000)
    collisionBehavior.setTranslatesReferenceBoundsIntoBoundary(with: insets)
    self.collisionBehavior = collisionBehavior

    // 附着行为，使视图具有悬挂在绳索上的效果
    let offset = UIOffset(horizontal: 70.0, vertical: fromView.bounds.height * 0.5)

```

```

// Prevent rotation and adjust bounce
let bodyBehavior = UIDynamicItemBehavior(items: [toView])
bodyBehavior.elasticity = 0.7
bodyBehavior.allowsRotation = false

// Add gravity at exaggerated magnitude so animation doesn't seem slow
let gravityBehavior = UIGravityBehavior(items: [toView])
gravityBehavior.magnitude = 10.0

// Set collision bounds to include off-screen view and have collision in center
// where our final view should come to rest
let collisionBehavior = UICollisionBehavior(items: [toView])
let insets = UIEdgeInsets(top: toViewInitialFrame.minY, left: 0.0, bottom:
fromViewInitialFrame.height * 0.5 - toViewInitialFrame.height * 0.5, right: 0.0)
collisionBehavior.setTranslatesReferenceBoundsIntoBoundary(with: insets)
self.collisionBehavior = collisionBehavior

// Keep track of finish time in case we need to end the animator before the animator
pauses
self.finishTime = duration + (self.animator?.elapsedTime ?? 0.0)

// Closure that is called after every "tick" of the animator
// Check if we exceed duration
self.action =
{ [weak self] in
guard let strongSelf = self,
      (strongSelf.animator?.elapsedTime ?? 0.0) >= strongSelf.finishTime else { return
}
strongSelf.hasElapsedDurationExceeded = true
strongSelf.animator?.removeBehavior(strongSelf)
}

// `DropOutAnimator` is a composit behavior, so add child behaviors to self
self.addChildBehavior(collisionBehavior)
self.addChildBehavior(bodyBehavior)
self.addChildBehavior(gravityBehavior)

// Add self to dynamic animator
self.animator?.addBehavior(self)
}

// Dismissing Animation
else
{
    // Create allow rotation and have a elastic item
let bodyBehavior = UIDynamicItemBehavior(items: [fromView])
bodyBehavior.elasticity = 0.8
bodyBehavior.angularResistance = 5.0
bodyBehavior.allowsRotation = true

// Create gravity with exaggerated magnitude
let gravityBehavior = UIGravityBehavior(items: [fromView])
gravityBehavior.magnitude = 10.0

// Collision boundary is set to have a floor just below the bottom of the screen
let collisionBehavior = UICollisionBehavior(items: [fromView])
let insets = UIEdgeInsets(top: 0.0, left: -1000, bottom: -225, right: -1000)
collisionBehavior.setTranslatesReferenceBoundsIntoBoundary(with: insets)
self.collisionBehavior = collisionBehavior

// Attachment behavior so view will have effect of hanging from a rope
let offset = UIOffset(horizontal: 70.0, vertical: fromView.bounds.height * 0.5)

```

```

var anchorPoint = CGPoint(x: fromView.bounds.maxX - 40.0, y: fromView.bounds.minY)
anchorPoint = containerView.convert(anchorPoint, from: fromView)
let attachmentBehavior = UIAttachmentBehavior(item: fromView, offsetFromCenter: offset,
attachedToAnchor: anchorPoint)
attachmentBehavior.frequency = 3.0
attachmentBehavior.damping = 3.0
self.attachmentBehavior = attachmentBehavior

// `DropOutAnimator` 是一个组合行为，因此将子行为添加到自身
self.addChildBehavior(collisionBehavior)
self.addChildBehavior(bodyBehavior)
self.addChildBehavior(gravityBehavior)
self.addChildBehavior(attachmentBehavior)

// 将自身添加到动态动画器中
self.animator?.addBehavior(self)

// 动画分为两部分，第一部分是悬挂在绳子上。
// 第二部分是在屏幕外弹跳
// 将持续时间分成两部分
self.finishTime = (2.0 / 3.0) * duration + (self.animator?.elapsedTime ?? 0.0)

// 每次动画器“滴答”后检查是否超过时间限制
self.action =
{ [weak self] in
guard let strongSelf = self,
      (strongSelf.animator?.elapsedTime ?? 0.0) >= strongSelf.finishTime else { return
}
strongSelf.hasElapsedTimeExceededDuration = true
strongSelf.animator?.removeBehavior(strongSelf)
}

func transitionDuration(using transitionContext: UIViewControllerContextTransitioning?) ->
TimeInterval
{
    // 返回动画的持续时间
    return self.duration
}

extension DropOutAnimator: UIDynamicAnimatorDelegate
{
    func dynamicAnimatorDidPause(_ animator: UIDynamicAnimator)
    {
        // 动画器已达到静止状态
        if self.isAppearing
        {
            // 检查是否超出时间
            if self.hasElapsedTimeExceededDuration
            {
                // 移动到最终位置
                let toView = self.transitionContext?.viewController(forKey: .to)?.view
                let containerView = self.transitionContext?.containerView
                toView?.center = containerView?.center ?? .zero
                self.hasElapsedTimeExceededDuration = false
            }
            // 清理并调用完成回调
        }
    }
}

```

```

var anchorPoint = CGPoint(x: fromView.bounds.maxX - 40.0, y: fromView.bounds.minY)
anchorPoint = containerView.convert(anchorPoint, from: fromView)
let attachmentBehavior = UIAttachmentBehavior(item: fromView, offsetFromCenter: offset,
attachedToAnchor: anchorPoint)
attachmentBehavior.frequency = 3.0
attachmentBehavior.damping = 3.0
self.attachmentBehavior = attachmentBehavior

// `DropOutAnimator` is a composite behavior, so add child behaviors to self
self.addChildBehavior(collisionBehavior)
self.addChildBehavior(bodyBehavior)
self.addChildBehavior(gravityBehavior)
self.addChildBehavior(attachmentBehavior)

// Add self to dynamic animator
self.animator?.addBehavior(self)

// Animation has two parts part one is hanging from rope.
// Part two is bouncy off-screen
// Divide duration in two
self.finishTime = (2.0 / 3.0) * duration + (self.animator?.elapsedTime ?? 0.0)

// After every "tick" of animator check if past time limit
self.action =
{ [weak self] in
guard let strongSelf = self,
      (strongSelf.animator?.elapsedTime ?? 0.0) >= strongSelf.finishTime else { return
}
strongSelf.hasElapsedTimeExceededDuration = true
strongSelf.animator?.removeBehavior(strongSelf)
}

func transitionDuration(using transitionContext: UIViewControllerContextTransitioning?) ->
TimeInterval
{
    // Return the duration of the animation
    return self.duration
}

extension DropOutAnimator: UIDynamicAnimatorDelegate
{
    func dynamicAnimatorDidPause(_ animator: UIDynamicAnimator)
    {
        // Animator has reached stasis
        if self.isAppearing
        {
            // Check if we are out of time
            if self.hasElapsedTimeExceededDuration
            {
                // Move to final positions
                let toView = self.transitionContext?.viewController(forKey: .to)?.view
                let containerView = self.transitionContext?.containerView
                toView?.center = containerView?.center ?? .zero
                self.hasElapsedTimeExceededDuration = false
            }
            // Clean up and call completion
        }
    }
}

```

```

self.transitionContext?.completeTransition(!(self.transitionContext?.transitionWasCancelled ??
false))
    self.childBehaviors.forEach { self.removeChildBehavior($0) }
    animator.removeAllBehaviors()
    self.transitionContext = nil
}
else
{
    if let attachmentBehavior = self.attachmentBehavior
    {
        // 如果存在附着行为，表示第一部分结束，第二部分开始。
        self.removeChildBehavior(attachmentBehavior)
        self.attachmentBehavior = nil
    }
    animator.addBehavior(self)
    let duration = self.transitionDuration(using: self.transitionContext)
    self.finishTime = 1.0 / 3.0 * duration + animator.elapsedTime
}
else
{
    // 清理并调用完成回调
    let fromView = self.transitionContext?.viewController(forKey: .from)?.view
    let toView = self.transitionContext?.viewController(forKey: .to)?.view
    fromView?.removeFromSuperview()
    toView?.isUserInteractionEnabled = true

    self.transitionContext?.completeTransition(!(self.transitionContext?.transitionWasCancelled ??
false))
        self.childBehaviors.forEach { self.removeChildBehavior($0) }
        animator.removeAllBehaviors()
        self.transitionContext = nil
    }
}
}

```

## Objective-C

```

@interface ObjcDropOutAnimator() <UIDynamicAnimatorDelegate, UIViewControllerAnimatedTransitioning>
@property (nonatomic, strong) id<UIViewControllerContextTransitioning> transitionContext;
@property (nonatomic, strong) UIDynamicAnimator *animator;
@property (nonatomic, assign) NSTimeInterval finishTime;
@property (nonatomic, assign) BOOL elapsedTimeExceededDuration;
@property (nonatomic, assign, getter=isAppearing) BOOL appearing;
@property (nonatomic, assign) NSTimeInterval duration;
@property (nonatomic, strong) UIAttachmentBehavior *attachBehavior;
@property (nonatomic, strong) UICollisionBehavior * collisionBehavior;

@end

@implementation ObjcDropOutAnimator

- (instancetype)initWithDuration:(NSTimeInterval)duration appearing:(BOOL)appearing
{
    self = [super init];
    if (self)
    {
        _duration = duration;
        _appearing = appearing;
    }
    return self;
}

```

```

self.transitionContext?.completeTransition(!(self.transitionContext?.transitionWasCancelled ??
false))
    self.childBehaviors.forEach { self.removeChildBehavior($0) }
    animator.removeAllBehaviors()
    self.transitionContext = nil
}
else
{
    if let attachmentBehavior = self.attachmentBehavior
    {
        // If we have an attachment, we are at the end of part one and start part two.
        self.removeChildBehavior(attachmentBehavior)
        self.attachmentBehavior = nil
        animator.addBehavior(self)
        let duration = self.transitionDuration(using: self.transitionContext)
        self.finishTime = 1.0 / 3.0 * duration + animator.elapsedTime
    }
    else
    {
        // Clean up and call completion
        let fromView = self.transitionContext?.viewController(forKey: .from)?.view
        let toView = self.transitionContext?.viewController(forKey: .to)?.view
        fromView?.removeFromSuperview()
        toView?.isUserInteractionEnabled = true

        self.transitionContext?.completeTransition(!(self.transitionContext?.transitionWasCancelled ??
false))
            self.childBehaviors.forEach { self.removeChildBehavior($0) }
            animator.removeAllBehaviors()
            self.transitionContext = nil
        }
    }
}

```

## Objective-C

```

@interface ObjcDropOutAnimator() <UIDynamicAnimatorDelegate, UIViewControllerAnimatedTransitioning>
@property (nonatomic, strong) id<UIViewControllerContextTransitioning> transitionContext;
@property (nonatomic, strong) UIDynamicAnimator *animator;
@property (nonatomic, assign) NSTimeInterval finishTime;
@property (nonatomic, assign) BOOL elapsedTimeExceededDuration;
@property (nonatomic, assign, getter=isAppearing) BOOL appearing;
@property (nonatomic, assign) NSTimeInterval duration;
@property (nonatomic, strong) UIAttachmentBehavior *attachBehavior;
@property (nonatomic, strong) UICollisionBehavior * collisionBehavior;

@end

@implementation ObjcDropOutAnimator

- (instancetype)initWithDuration:(NSTimeInterval)duration appearing:(BOOL)appearing
{
    self = [super init];
    if (self)
    {
        _duration = duration;
        _appearing = appearing;
    }
    return self;
}

```

```

- (void) animateTransition:(id<UIViewControllerContextTransitioning>)transitionContext
{
    // 从transitionContext获取相关视图和视图控制器
    UIViewController *fromVC = [transitionContext
viewControllerForKey:UITransitionContextFromViewControllerKey];
    UIViewController *toVC = [transitionContext
viewControllerForKey:UITransitionContextToViewControllerKey];
    UIView *fromView = fromVC.view;
    UIView *toView = toVC.view;

    UIView *containerView = transitionContext.containerView;
    NSTimeInterval duration = [self transitionDuration:transitionContext];

    // 持有transitionContext的引用以通知其完成
    self.transitionContext = transitionContext;

    // 创建动态动画器
    UIDynamicAnimator *animator = [[UIDynamicAnimator alloc] initWithReferenceView:containerView];
    animator.delegate = self;
    self.animator = animator;

    // 展示动画
    if (self.isAppearing)
    {
        fromView.userInteractionEnabled = NO;

        // 将 toView 位置设置在屏幕正上方
        CGRect fromViewInitialFrame = [transitionContext initialFrameForViewController:fromVC];
        CGRect toViewInitialFrame = toView.frame;
        toViewInitialFrame.origin.y -= CGRectGetHeight(toViewInitialFrame);
        toViewInitialFrame.origin.x = CGRectGetWidth(fromViewInitialFrame) * 0.5 -
        CGRectGetWidth(toViewInitialFrame) * 0.5;
        toView.frame = toViewInitialFrame;

        [containerView addSubview:toView];

        // 防止旋转并调整弹跳效果
        UIDynamicItemBehavior *bodyBehavior = [[UIDynamicItemBehavior
alloc] initWithItems:@[toView]];
        bodyBehavior.elasticity = 0.7;
        bodyBehavior.allowsRotation = NO;

        // 以夸张的重力加速度添加重力，使动画看起来不慢
        UIGravityBehavior *gravityBehavior = [[UIGravityBehavior alloc] initWithItems:@[toView]];
        gravityBehavior.magnitude = 10.0f;

        // 设置碰撞边界，包含屏幕外视图，并在中心设置碰撞地面
        // 这是最终视图应停留的位置
        UICollisionBehavior *collisionBehavior = [[UICollisionBehavior
alloc] initWithItems:@[toView]];
        UIEdgeInsets insets = UIEdgeInsetsMake(CGRectGetMinY(toViewInitialFrame), 0.0,
        CGRectGetHeight(fromViewInitialFrame) * 0.5 - CGRectGetHeight(toViewInitialFrame) * 0.5, 0.0);
        [collisionBehavior setTranslatesReferenceBoundsIntoBoundaryWithInsets:insets];
        self.collisionBehavior = collisionBehavior;

        // 记录结束时间，以防我们需要在动画暂停前结束动画器
        self.finishTime = duration + self.animator.elapsedTime;

        // 动画器每次“滴答”后调用的闭包
        // 检查是否超过持续时间
        __weak ObjcDropOutAnimator *weakSelf = self;
        self.action = ^{

```

```

- (void) animateTransition:(id<UIViewControllerContextTransitioning>)transitionContext
{
    // Get relevant views and view controllers from transitionContext
    UIViewController *fromVC = [transitionContext
viewControllerForKey:UITransitionContextFromViewControllerKey];
    UIViewController *toVC = [transitionContext
viewControllerForKey:UITransitionContextToViewControllerKey];
    UIView *fromView = fromVC.view;
    UIView *toView = toVC.view;

    UIView *containerView = transitionContext.containerView;
    NSTimeInterval duration = [self transitionDuration:transitionContext];

    // Hold reference to transitionContext to notify it of completion
    self.transitionContext = transitionContext;

    // Create dynamic animator
    UIDynamicAnimator *animator = [[UIDynamicAnimator alloc] initWithReferenceView:containerView];
    animator.delegate = self;
    self.animator = animator;

    // Presenting Animation
    if (self.isAppearing)
    {
        fromView.userInteractionEnabled = NO;

        // Position toView just above screen
        CGRect fromViewInitialFrame = [transitionContext initialFrameForViewController:fromVC];
        CGRect toViewInitialFrame = toView.frame;
        toViewInitialFrame.origin.y -= CGRectGetHeight(toViewInitialFrame);
        toViewInitialFrame.origin.x = CGRectGetWidth(fromViewInitialFrame) * 0.5 -
        CGRectGetWidth(toViewInitialFrame) * 0.5;
        toView.frame = toViewInitialFrame;

        [containerView addSubview:toView];

        // Prevent rotation and adjust bounce
        UIDynamicItemBehavior *bodyBehavior = [[UIDynamicItemBehavior
alloc] initWithItems:@[toView]];
        bodyBehavior.elasticity = 0.7;
        bodyBehavior.allowsRotation = NO;

        // Add gravity at exaggerated magnitude so animation doesn't seem slow
        UIGravityBehavior *gravityBehavior = [[UIGravityBehavior alloc] initWithItems:@[toView]];
        gravityBehavior.magnitude = 10.0f;

        // Set collision bounds to include off-screen view and have collision floor in center
        // where our final view should come to rest
        UICollisionBehavior *collisionBehavior = [[UICollisionBehavior
alloc] initWithItems:@[toView]];
        UIEdgeInsets insets = UIEdgeInsetsMake(CGRectGetMinY(toViewInitialFrame), 0.0,
        CGRectGetHeight(fromViewInitialFrame) * 0.5 - CGRectGetHeight(toViewInitialFrame) * 0.5, 0.0);
        [collisionBehavior setTranslatesReferenceBoundsIntoBoundaryWithInsets:insets];
        self.collisionBehavior = collisionBehavior;

        // Keep track of finish time in case we need to end the animator before the animator pauses
        self.finishTime = duration + self.animator.elapsedTime;

        // Closure that is called after every "tick" of the animator
        // Check if we exceed duration
        __weak ObjcDropOutAnimator *weakSelf = self;
        self.action = ^{

```

```

__strong ObjcDropOutAnimator *strongSelf = weakSelf;
    if (strongSelf)
    {
        if (strongSelf.animator.elapsedTime >= strongSelf.finishTime)
        {
            strongSelf.elapsedTimeExceededDuration = YES;
            [strongSelf.animator removeBehavior:strongSelf];
        }
    }
};

// `DropOutAnimator` 是一个组合行为，因此将子行为添加到自身
[self addChildBehavior:collisionBehavior];
[self addChildBehavior:bodyBehavior];
[self addChildBehavior:gravityBehavior];

// 将自身添加到动态动画器中
[self.animator 添加行为:self];
}

// 取消动画
else
{
    // 允许旋转并具有弹性效果的项目
    UIDynamicItemBehavior *bodyBehavior = [[UIDynamicItemBehavior 分配]
用项目初始化:@[fromView]];
bodyBehavior.弹性 = 0.8;
bodyBehavior.角阻力 = 5.0;
bodyBehavior.允许旋转 = 是;

// 创建具有夸大幅度的重力
UIGravityBehavior *gravityBehavior = [[UIGravityBehavior 分配] 用项目初始化:@[fromView]];
gravityBehavior.幅度 = 10.0f;

// 碰撞边界设置为屏幕底部正下方的地板
UICollisionBehavior *collisionBehavior = [[UICollisionBehavior 分配]
用项目初始化:@[fromView]];
UIEdgeInsets 内边距 = UIEdgeInsetsMake(0, -1000, -225, -1000);
[collisionBehavior 设置参考边界转换为带内边距的边界:内边距];
self.collisionBehavior = collisionBehavior;

// 附着行为，使视图具有悬挂绳索的效果
UIOffset 偏移 = UIOffsetMake(70, -(CGRectGetHeight(fromView.边界) / 2.0));

CGPoint 锚点 = CGPointMake(CGRectGetMaxX(fromView.边界) - 40,
                           CGRectGetMinY(fromView.边界));
锚点 = [containerView 从fromView转换点:锚点];
UIAttachmentBehavior *attachBehavior = [[UIAttachmentBehavior 分配] 用项目初始化:fromView
中心偏移:偏移 附着到锚点:锚点];
attachBehavior.频率 = 3.0;
attachBehavior.阻尼 = 0.3;
attachBehavior.长度 = 40;
self.attachBehavior = attachBehavior;

// `DropOutAnimator` 是一个组合行为，因此将子行为添加到自身
[self addChildBehavior:collisionBehavior];
[self 添加子行为:bodyBehavior];
[self 添加子行为:gravityBehavior];
[self 添加子行为:attachBehavior];

// 将自身添加到动态动画器中
[self.animator 添加行为:self];
}

```

```

__strong ObjcDropOutAnimator *strongSelf = weakSelf;
if (strongSelf)
{
    if (strongSelf.animator.elapsedTime >= strongSelf.finishTime)
    {
        strongSelf.elapsedTimeExceededDuration = YES;
        [strongSelf.animator removeBehavior:strongSelf];
    }
}

// `DropOutAnimator` is a composit behavior, so add child behaviors to self
[self addChildBehavior:collisionBehavior];
[self addChildBehavior:bodyBehavior];
[self addChildBehavior:gravityBehavior];

// Add self to dynamic animator
[self.animator addBehavior:self];
}

// Dismissing Animation
else
{
    // Allow rotation and have a elastic item
    UIDynamicItemBehavior *bodyBehavior = [[UIDynamicItemBehavior alloc]
initWithItems:@[fromView]];
bodyBehavior.elasticity = 0.8;
bodyBehavior.angularResistance = 5.0;
bodyBehavior.allowsRotation = YES;

// Create gravity with exaggerated magnitude
UIGravityBehavior *gravityBehavior = [[UIGravityBehavior alloc] initWithItems:@[fromView]];
gravityBehavior.magnitude = 10.0f;

// Collision boundary is set to have a floor just below the bottom of the screen
UICollisionBehavior *collisionBehavior = [[UICollisionBehavior alloc]
initWithItems:@[fromView]];
UIEdgeInsets insets = UIEdgeInsetsMake(0, -1000, -225, -1000);
[collisionBehavior setTranslatesReferenceBoundsIntoBoundaryWithInsets:insets];
self.collisionBehavior = collisionBehavior;

// Attachment behavior so view will have effect of hanging from a rope
UIOffset offset = UIOffsetMake(70, -(CGRectGetHeight(fromView.bounds) / 2.0));

CGPoint anchorPoint = CGPointMake(CGRectGetMaxX(fromView.bounds) - 40,
                                   CGRectGetMinY(fromView.bounds));
anchorPoint = [containerView convertPoint:anchorPoint fromView:fromView];
UIAttachmentBehavior *attachBehavior = [[UIAttachmentBehavior alloc] initWithItem:fromView
offsetFromCenter:offset attachedToAnchor:anchorPoint];
attachBehavior.frequency = 3.0;
attachBehavior.damping = 0.3;
attachBehavior.length = 40;
self.attachBehavior = attachBehavior;

// `DropOutAnimator` is a composit behavior, so add child behaviors to self
[self addChildBehavior:collisionBehavior];
[self addChildBehavior:bodyBehavior];
[self addChildBehavior:gravityBehavior];
[self addChildBehavior:attachBehavior];

// Add self to dynamic animator
[self.animator addBehavior:self];
}

```

```

// 动画分为两部分，第一部分是悬挂在绳子上。
// 第二部分是在屏幕外弹跳
// 将持续时间分成两部分
self.finishTime = (2./3.) * duration + [self.animator 已用时间];

// 每次动画器“滴答”后检查是否超过时间限制
__weak ObjcDropOutAnimator *weakSelf = self;
self.action = ^{
    __strong ObjcDropOutAnimator *strongSelf = weakSelf;
    if (strongSelf)
    {
        if ([strongSelf.animator 已用时间] >= strongSelf.finishTime)
        {
            strongSelf.elapsedTimeExceededDuration = YES;
            [strongSelf.animator removeBehavior:strongSelf];
        }
    }
};

- (NSTimeInterval)transitionDuration:(id<UIViewControllerContextTransitioning>)transitionContext
{
    return self.duration;
}

- (void)dynamicAnimatorDidPause:(UIDynamicAnimator *)animator
{
    // 动画器已达到静止状态
    if (self.isAppearing)
    {
        // 检查是否超时
        if (self.elapsedTimeExceededDuration)
        {
            // 移动到最终位置
            UIView *toView = [self.transitionContext
viewControllerForKey:UITransitionContextToViewControllerKey].view;
            UIView *containerView = [self.transitionContext containerView];
            toView.center = containerView.center;
            self.elapsedTimeExceededDuration = NO;
        }

        // 清理并调用完成回调
        [self.transitionContext completeTransition:![[self.transitionContext
transitionWasCancelled]];
        for (UIDynamicBehavior *behavior in self.childBehaviors)
        {
            [self removeChildBehavior:behavior];
        }
        [animator removeAllBehaviors];
        self.transitionContext = nil;
    }
    // Dismissing
    else
    {
        if (self.attachBehavior)
        {
            // 如果存在附着行为，表示第一部分结束，第二部分开始。
            [self removeChildBehavior:self.attachBehavior];
            self.attachBehavior = nil;
            [animator addBehavior:self];
        }
    }
    NSTimeInterval duration = [self transitionDuration:self.transitionContext];
}

```

```

// Animation has two parts part one is hanging from rope.
// Part two is bouncing off-screen
// Divide duration in two
self.finishTime = (2./3.) * duration + [self.animator elapsedTime];

// After every "tick" of animator check if past time limit
__weak ObjcDropOutAnimator *weakSelf = self;
self.action = ^{
    __strong ObjcDropOutAnimator *strongSelf = weakSelf;
    if (strongSelf)
    {
        if ([strongSelf.animator elapsedTime] >= strongSelf.finishTime)
        {
            strongSelf.elapsedTimeExceededDuration = YES;
            [strongSelf.animator removeBehavior:strongSelf];
        }
    }
};

- (NSTimeInterval)transitionDuration:(id<UIViewControllerContextTransitioning>)transitionContext
{
    return self.duration;
}

- (void)dynamicAnimatorDidPause:(UIDynamicAnimator *)animator
{
    // Animator has reached stasis
    if (self.isAppearing)
    {
        // Check if we are out of time
        if (self.elapsedTimeExceededDuration)
        {
            // Move to final positions
            UIView *toView = [self.transitionContext
viewControllerForKey:UITransitionContextToViewControllerKey].view;
            UIView *containerView = [self.transitionContext containerView];
            toView.center = containerView.center;
            self.elapsedTimeExceededDuration = NO;
        }

        // Clean up and call completion
        [self.transitionContext completeTransition:![[self.transitionContext
transitionWasCancelled]];
        for (UIDynamicBehavior *behavior in self.childBehaviors)
        {
            [self removeChildBehavior:behavior];
        }
        [animator removeAllBehaviors];
        self.transitionContext = nil;
    }
    // Dismissing
    else
    {
        if (self.attachBehavior)
        {
            // If we have an attachment, we are at the end of part one and start part two.
            [self removeChildBehavior:self.attachBehavior];
            self.attachBehavior = nil;
            [animator addBehavior:self];
        }
    }
    NSTimeInterval duration = [self transitionDuration:self.transitionContext];
}

```

```

        self.finishTime = 1./3. * duration + [animator elapsedTime];
    }
    else
    {
        // 清理并调用完成回调
        UIView *fromView = [self.transitionContext
viewControllerForKey:UITransitionContextFromViewControllerKey].view;
        UIView *toView = [self.transitionContext
viewControllerForKey:UITransitionContextToViewControllerKey].view;
        [fromView removeFromSuperview];
        toView.userInteractionEnabled = YES;

        [self.transitionContext completeTransition:![[self.transitionContext
transitionWasCancelled]];
        for (UIDynamicBehavior *behavior in self.childBehaviors)
        {
            [self removeChildBehavior:behavior];
        }
        [animator removeAllBehaviors];
        self.transitionContext = nil;
    }
}

```

作为组合行为，DropOutAnimator可以组合多种不同的行为来执行其呈现和消失动画。DropOutAnimator还演示了如何使用行为的action块来检查其项目的位置以及经过的时间，这是一种可以用来移除移出屏幕的视图或截断尚未达到静止状态的动画的技术。

更多信息请参见2013 WWDC 会议“UIKit 动力学的高级技术”以及  
[SOLPresentingFun](#)

## 第59.4节：使用UIDynamicBehaviors实现具有真实物理效果的遮罩过渡

此示例展示了如何制作一个具有“真实世界”物理效果的交互式呈现过渡，类似于iOS的通知屏幕。



Swipe Down From Top

```

        self.finishTime = 1./3. * duration + [animator elapsedTime];
    }
    else
    {
        // Clean up and call completion
        UIView *fromView = [self.transitionContext
viewControllerForKey:UITransitionContextFromViewControllerKey].view;
        UIView *toView = [self.transitionContext
viewControllerForKey:UITransitionContextToViewControllerKey].view;
        [fromView removeFromSuperview];
        toView.userInteractionEnabled = YES;

        [self.transitionContext completeTransition:![[self.transitionContext
transitionWasCancelled]];
        for (UIDynamicBehavior *behavior in self.childBehaviors)
        {
            [self removeChildBehavior:behavior];
        }
        [animator removeAllBehaviors];
        self.transitionContext = nil;
    }
}

```

As composite behavior, DropOutAnimator, can combine a number of different behaviors to perform its presenting and dismissing animations. DropOutAnimator also demonstrates how to use the action block of a behavior to inspect the locations of its items as well as the time elapsed a technique that can be used to remove views that move offscreen or truncate animations that have yet to reach stasis.

For more information [2013 WWDC Session "Advanced Techniques with UIKit Dynamics"](#) as well as  
[SOLPresentingFun](#)

## Section 59.4: Shade Transition with Real-World Physics Using UIDynamicBehaviors

This example shows how to make an interactive presentation transition with "real-world" physics similar to iOS's notifications screen.



Swipe Down From Top

首先，我们需要一个呈现视图控制器，遮罩将显示在其上。该视图控制器还将作为我们呈现视图控制器的UIViewControllerTransitioningDelegate，并为我们的过渡提供动画器。因此，我们将创建交互式动画器的实例（一个用于呈现，一个用于消失）。我们还将创建遮罩视图控制器的实例，在本例中，它只是一个带有标签的视图控制器。因为我们希望同一个平移手势驱动整个交互，所以我们将呈现视图控制器和遮罩的引用传递给我们的交互式动画器。

## Swift

```
class ViewController: UIViewController
{
    var presentingAnimator: ShadeAnimator!
    var dismissingAnimator: ShadeAnimator!
    let shadeVC = ShadeViewController()

    lazy var label: UILabel =
    {
        let label = UILabel()
        label.textColor = .blue
        label.translatesAutoresizingMaskIntoConstraints = false
        self.view.addSubview(label)
        label.centerXAnchor.constraint(equalTo: self.view.centerXAnchor).isActive = true
        label.centerYAnchor.constraint(equalTo: self.view.centerYAnchor).isActive = true
        return label
    }()

    override func viewDidLoad()
    {
        super.viewDidLoad()
        label.text = "从顶部向下滑动"
        presentingAnimator = ShadeAnimator(isAppearing: true, presentingVC: self, presentedVC: shadeVC, transitionDelegate: self)
        dismissingAnimator = ShadeAnimator(isAppearing: false, presentingVC: self, presentedVC: shadeVC, transitionDelegate: self)
    }

    extension ViewController: UIViewControllerTransitioningDelegate
    {
        func animationController(forPresented presented: UIViewController, presenting: UIViewController, source: UIViewController) -> UIViewControllerAnimatedTransitioning?
        {
            return EmptyAnimator()
        }

        func animationController(forDismissed dismissed: UIViewController) -> UIViewControllerAnimatedTransitioning?
        {
            return EmptyAnimator()
        }

        func interactionControllerForPresentation(using animator: UIViewControllerAnimatedTransitioning) -> UIViewControllerInteractiveTransitioning?
        {
            return presentingAnimator
        }

        func interactionControllerForDismissal(using animator: UIViewControllerAnimatedTransitioning) -> UIViewControllerInteractiveTransitioning?
        {
            return dismissingAnimator
        }
    }
}
```

To start with, we need a presenting view controller that the shade will appear over. This view controller will also act as our `UIViewControllerTransitioningDelegate` for our presented view controller and will vend animators for our transition. So we'll create instances of our interactive animators (one for presenting, one for dismissing). We'll also create an instance of the shade view controller, which, in this example, is just a view controller with a label. Because we want the same pan gesture to drive the entire interaction we pass references to the presenting view controller and the shade into our interactive animators.

## Swift

```
class ViewController: UIViewController
{
    var presentingAnimator: ShadeAnimator!
    var dismissingAnimator: ShadeAnimator!
    let shadeVC = ShadeViewController()

    lazy var label: UILabel =
    {
        let label = UILabel()
        label.textColor = .blue
        label.translatesAutoresizingMaskIntoConstraints = false
        self.view.addSubview(label)
        label.centerXAnchor.constraint(equalTo: self.view.centerXAnchor).isActive = true
        label.centerYAnchor.constraint(equalTo: self.view.centerYAnchor).isActive = true
        return label
    }()

    override func viewDidLoad()
    {
        super.viewDidLoad()
        label.text = "Swipe Down From Top"
        presentingAnimator = ShadeAnimator(isAppearing: true, presentingVC: self, presentedVC: shadeVC, transitionDelegate: self)
        dismissingAnimator = ShadeAnimator(isAppearing: false, presentingVC: self, presentedVC: shadeVC, transitionDelegate: self)
    }

    extension ViewController: UIViewControllerTransitioningDelegate
    {
        func animationController(forPresented presented: UIViewController, presenting: UIViewController, source: UIViewController) -> UIViewControllerAnimatedTransitioning?
        {
            return EmptyAnimator()
        }

        func animationController(forDismissed dismissed: UIViewController) -> UIViewControllerAnimatedTransitioning?
        {
            return EmptyAnimator()
        }

        func interactionControllerForPresentation(using animator: UIViewControllerAnimatedTransitioning) -> UIViewControllerInteractiveTransitioning?
        {
            return presentingAnimator
        }

        func interactionControllerForDismissal(using animator: UIViewControllerAnimatedTransitioning) -> UIViewControllerInteractiveTransitioning?
        {
            return dismissingAnimator
        }
    }
}
```

}

## Objective-C

```

@interface ObjCViewController () <UIViewControllerTransitioningDelegate>
@property (nonatomic, strong) ShadeAnimator *presentingAnimator;
@property (nonatomic, strong) ShadeAnimator *dismissingAnimator;
@property (nonatomic, strong) UILabel *label;
@property (nonatomic, strong) ShadeViewController *shadeVC;
@end

@implementation ObjCViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    self.label.text = @"从顶部向下滑动";
    self.shadeVC = [[ShadeViewController alloc] init];
    self.presentingAnimator = [[ShadeAnimator alloc] initWithIsAppearing:YES presentingVC:self
presentedVC:self.shadeVC transitionDelegate:self];
    self.dismissingAnimator = [[ShadeAnimator alloc] initWithIsAppearing:NO presentingVC:self
presentedVC:self.shadeVC transitionDelegate:self];
}

- (UILabel *)label
{
    if (!_label)
    {
        _label = [[UILabel alloc] init];
        _label.textColor = [UIColor blueColor];
        _label.translatesAutoresizingMaskIntoConstraints = NO;
        [self.view addSubview:_label];
        [_label.centerXAnchor constraintEqualToAnchor:self.view.centerXAnchor].active = YES;
        [_label.centerYAnchor constraintEqualToAnchor:self.view.centerYAnchor].active = YES;
    }
    return _label;
}

#pragma mark - UIViewControllerTransitioningDelegate

- (id<UIViewControllerAnimatedTransitioning>)animationControllerForPresentedController:(UIViewController *)presented presentingController:(UIViewController *)presenting
sourceController:(UIViewController *)source
{
    return [[EmptyAnimator alloc] init];
}

- (id<UIViewControllerAnimatedTransitioning>)animationControllerForDismissedController:(UIViewController *)dismissed
{
    return [[EmptyAnimator alloc] init];
}
{
    id<UIViewControllerAnimatedTransitioning>动画器
{
    return self.presentingAnimator;
}
}

```

}

## Objective-C

```

@interface ObjCViewController () <UIViewControllerTransitioningDelegate>
@property (nonatomic, strong) ShadeAnimator *presentingAnimator;
@property (nonatomic, strong) ShadeAnimator *dismissingAnimator;
@property (nonatomic, strong) UILabel *label;
@property (nonatomic, strong) ShadeViewController *shadeVC;
@end

@implementation ObjCViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    self.label.text = @"Swipe Down From Top";
    self.shadeVC = [[ShadeViewController alloc] init];
    self.presentingAnimator = [[ShadeAnimator alloc] initWithIsAppearing:YES presentingVC:self
presentedVC:self.shadeVC transitionDelegate:self];
    self.dismissingAnimator = [[ShadeAnimator alloc] initWithIsAppearing:NO presentingVC:self
presentedVC:self.shadeVC transitionDelegate:self];
}

- (UILabel *)label
{
    if (!_label)
    {
        _label = [[UILabel alloc] init];
        _label.textColor = [UIColor blueColor];
        _label.translatesAutoresizingMaskIntoConstraints = NO;
        [self.view addSubview:_label];
        [_label.centerXAnchor constraintEqualToAnchor:self.view.centerXAnchor].active = YES;
        [_label.centerYAnchor constraintEqualToAnchor:self.view.centerYAnchor].active = YES;
    }
    return _label;
}

#pragma mark - UIViewControllerTransitioningDelegate

- (id<UIViewControllerAnimatedTransitioning>)animationControllerForPresentedController:(UIViewController *)presented presentingController:(UIViewController *)presenting
sourceController:(UIViewController *)source
{
    return [[EmptyAnimator alloc] init];
}

- (id<UIViewControllerAnimatedTransitioning>)animationControllerForDismissedController:(UIViewController *)dismissed
{
    return [[EmptyAnimator alloc] init];
}

- (id<UIViewControllerInteractiveTransitioning>)interactionControllerForPresentation:(id<UIViewControllerAnimatedTransitioning>)animator
{
    return self.presentingAnimator;
}

```

```

        id <
        >_ : (id<UIViewControllerAnimatedTransitioning>)animatedTransitioning)动画器
{
    return self.dismissingAnimator;
}

@end

```

我们实际上只想通过交互式过渡来呈现我们的阴影，但由于其方式  
`UIViewControllerTransitioningDelegate` 的工作原理是，如果我们不返回一个常规的动画控制器，我们的交互式控制器将永远不会被使用。正因为如此，我们创建了一个符合要求的`EmptyAnimator`类。  
`UIViewControllerAnimatedTransitioning`.

### Swift

```

class EmptyAnimator: NSObject
{

extension EmptyAnimator: UIViewControllerAnimatedTransitioning
{
    func animateTransition(using transitionContext: UIViewControllerContextTransitioning)
    {

        func transitionDuration(using transitionContext: UIViewControllerContextTransitioning?) ->
TimeInterval
        {
            return 0.0
        }
}

```

### Objective-C

```

@implementation EmptyAnimator

- (void)animateTransition:(id<UIViewControllerContextTransitioning>)transitionContext
{
}

- (NSTimeInterval)transitionDuration:(id<UIViewControllerContextTransitioning>)transitionContext
{
    return 0.0;
}

@end

```

最后我们需要实际创建`ShadeAnimator`，它是`UIDynamicBehavior`的子类，并且遵循  
`UIViewControllerInteractiveTransitioning`协议。

### Swift

```

class ShadeAnimator: UIDynamicBehavior
{
    // 我们是正在呈现还是正在消失
    let isAppearing: Bool

```

```

- (id<UIViewControllerInteractiveTransitioning>)interactionControllerForDismissal:(id<UIViewControllerAnimatedTransitioning>)animatedTransitioning) animator
{
    return self.dismissingAnimator;
}

@end

```

We want really only ever want to present our shade through an interactive transition but because of how  
`UIViewControllerTransitioningDelegate` works if we don't return a regular animation controller our interactive  
controller will never be used. Because of that we create an `EmptyAnimator` class that conforms to  
`UIViewControllerAnimatedTransitioning`.

### Swift

```

class EmptyAnimator: NSObject
{

extension EmptyAnimator: UIViewControllerAnimatedTransitioning
{
    func animateTransition(using transitionContext: UIViewControllerContextTransitioning)
    {

        func transitionDuration(using transitionContext: UIViewControllerContextTransitioning?) ->
TimeInterval
        {
            return 0.0
        }
}

```

### Objective-C

```

@implementation EmptyAnimator

- (void)animateTransition:(id<UIViewControllerContextTransitioning>)transitionContext
{
}

- (NSTimeInterval)transitionDuration:(id<UIViewControllerContextTransitioning>)transitionContext
{
    return 0.0;
}

@end

```

Finally we need to actually create the `ShadeAnimator` which is a subclass of `UIDynamicBehavior` which conforms to  
`UIViewControllerInteractiveTransitioning`.

### Swift

```

class ShadeAnimator: UIDynamicBehavior
{
    // Whether we are presenting or dismissing
    let isAppearing: Bool

```

```

// 不是遮罩的视图控制器
weak var presentingVC: UIViewController?

// 作为遮罩的视图控制器
weak var presentedVC: UIViewController?

// 委托将提供动画器
weak var transitionDelegate: UIViewControllerTransitioningDelegate?

// 碰撞时的触觉反馈生成器
let impactFeedbackGenerator = UIImpactFeedbackGenerator(style: .light)

// 动画开始时传递给动画器的上下文
var transitionContext: UIViewControllerContextTransitioning?

// 动画动态部分的时间限制
var finishTime: TimeInterval = 4.0

// 驱动过渡的平移手势。未使用边缘平移手势，因为会触发通知
screen
lazy var pan: UIPanGestureRecognizer =
{
    let pan = UIPanGestureRecognizer(target: self, action: #selector(self.handlePan(sender:)))
    return pan
}()

// 我们添加 `ShadeAnimator` 的动态动画器
lazy var animator: UIDynamicAnimator! =
{
    let animator = UIDynamicAnimator(referenceView: self.transitionContext!.containerView)
    return animator
}()

// 使用我们所有依赖项进行初始化
init(isAppearing: Bool, presentingVC: UIViewController, presentedVC: UIViewController,
transitionDelegate: UIViewControllerTransitioningDelegate)
{
    self.isAppearing = isAppearing
    self.presentingVC = presentingVC
    self.presentedVC = presentedVC
    self.transitionDelegate = transitionDelegate
    super.init()
    self.impactFeedbackGenerator.prepare()

    if isAppearing
    {
        self.presentingVC?.view.addGestureRecognizer(pan)
    }
    else
    {
        self.presentedVC?.view.addGestureRecognizer(pan)
    }
}

// 设置并将遮罩视图控制器移动到屏幕上方（如果正在出现）
func setupViewsForTransition(with transitionContext: UIViewControllerContextTransitioning)
{
    // 从 transitionContext 获取相关视图和视图控制器
    guard let fromVC = transitionContext.viewController(forKey: .from),
          let toVC = transitionContext.viewController(forKey: .to),
          let toView = toVC.view else { return }
}

```

```

// The view controller that is not the shade
weak var presentingVC: UIViewController?

// The view controller that is the shade
weak var presentedVC: UIViewController?

// The delegate will vend the animator
weak var transitionDelegate: UIViewControllerTransitioningDelegate?

// Feedback generator for haptics on collisions
let impactFeedbackGenerator = UIImpactFeedbackGenerator(style: .light)

// The context given to the animator at the start of the transition
var transitionContext: UIViewControllerContextTransitioning?

// Time limit of the dynamic part of the animation
var finishTime: TimeInterval = 4.0

// The Pan Gesture that drives the transition. Not using EdgePan because triggers Notifications
screen
lazy var pan: UIPanGestureRecognizer =
{
    let pan = UIPanGestureRecognizer(target: self, action: #selector(self.handlePan(sender:)))
    return pan
}()

// The dynamic animator that we add `ShadeAnimator` to
lazy var animator: UIDynamicAnimator! =
{
    let animator = UIDynamicAnimator(referenceView: self.transitionContext!.containerView)
    return animator
}()

// init with all of our dependencies
init(isAppearing: Bool, presentingVC: UIViewController, presentedVC: UIViewController,
transitionDelegate: UIViewControllerTransitioningDelegate)
{
    self.isAppearing = isAppearing
    self.presentingVC = presentingVC
    self.presentedVC = presentedVC
    self.transitionDelegate = transitionDelegate
    super.init()
    self.impactFeedbackGenerator.prepare()

    if isAppearing
    {
        self.presentingVC?.view.addGestureRecognizer(pan)
    }
    else
    {
        self.presentedVC?.view.addGestureRecognizer(pan)
    }
}

// Setup and moves shade view controller to just above screen if appearing
func setupViewsForTransition(with transitionContext: UIViewControllerContextTransitioning)
{
    // Get relevant views and view controllers from transitionContext
    guard let fromVC = transitionContext.viewController(forKey: .from),
          let toVC = transitionContext.viewController(forKey: .to),
          let toView = toVC.view else { return }
}

```

```

let containerView = transitionContext.containerView

// 持有对 transitionContext 的引用以通知其完成
self.transitionContext = transitionContext
if isAppearing
{
    // 视图 toView 位置刚好在屏幕外
    let fromViewInitialFrame = transitionContext.initialFrame(for: fromVC)
    var toViewInitialFrame = toView.frame
    toViewInitialFrame.origin.y -= toViewInitialFrame.height
    toViewInitialFrame.origin.x = fromViewInitialFrame.width * 0.5 -
    toViewInitialFrame.width * 0.5
    toView.frame = toViewInitialFrame

    containerView.addSubview(toView)
}
else
{
fromVC.view.addGestureRecognizer(pan)
}

// 处理从展示/消失到完成的整个交互过程
func handlePan(sender: UIPanGestureRecognizer)
{
    let location = sender.location(in: transitionContext?.containerView)
    let velocity = sender.velocity(in: transitionContext?.containerView)
    let fromVC = transitionContext?.viewController(forKey: .from)
    let toVC = transitionContext?.viewController(forKey: .to)

    let touchStartHeight: CGFloat = 90.0
    let touchLocationFromBottom: CGFloat = 20.0

    switch sender.state
    {
        case .began:
            let beginLocation = sender.location(in: sender.view)
            if isAppearing
            {
guard beginLocation.y <= touchStartHeight,
                let presentedVC = self.presentedVC else { break }
                presentedVC.modalPresentationStyle = .custom
                presentedVC.transitioningDelegate = transitionDelegate
                presentingVC?.present(presentedVC, animated: true)
            }
            else
            {
guard beginLocation.y >= (sender.view?.frame.height ?? 0.0) - touchStartHeight else
{ break }
                presentedVC?.dismiss(animated: true)
            }
            case .changed:
                guard let view = isAppearing ? toVC?.view : fromVC?.view else { return }
                UIView.animate(withDuration: 0.2)
                {
                    view.frame.origin.y = location.y - view.bounds.height + touchLocationFromBottom
                }

transitionContext?.updateInteractiveTransition(view.frame.maxY / view.frame.height
                )
            case .ended, .cancelled:

```

```

let containerView = transitionContext.containerView

// Hold reference to transitionContext to notify it of completion
self.transitionContext = transitionContext
if isAppearing
{
    // Position toView just off-screen
    let fromViewInitialFrame = transitionContext.initialFrame(for: fromVC)
    var toViewInitialFrame = toView.frame
    toViewInitialFrame.origin.y -= toViewInitialFrame.height
    toViewInitialFrame.origin.x = fromViewInitialFrame.width * 0.5 -
    toViewInitialFrame.width * 0.5
    toView.frame = toViewInitialFrame

    containerView.addSubview(toView)
}
else
{
    fromVC.view.addGestureRecognizer(pan)
}

// Handles the entire interaction from presenting/dismissing to completion
func handlePan(sender: UIPanGestureRecognizer)
{
    let location = sender.location(in: transitionContext?.containerView)
    let velocity = sender.velocity(in: transitionContext?.containerView)
    let fromVC = transitionContext?.viewController(forKey: .from)
    let toVC = transitionContext?.viewController(forKey: .to)

    let touchStartHeight: CGFloat = 90.0
    let touchLocationFromBottom: CGFloat = 20.0

    switch sender.state
    {
        case .began:
            let beginLocation = sender.location(in: sender.view)
            if isAppearing
            {
                guard beginLocation.y <= touchStartHeight,
                    let presentedVC = self.presentedVC else { break }
                presentedVC.modalPresentationStyle = .custom
                presentedVC.transitioningDelegate = transitionDelegate
                presentingVC?.present(presentedVC, animated: true)
            }
            else
            {
                guard beginLocation.y >= (sender.view?.frame.height ?? 0.0) - touchStartHeight else
{ break }
                    presentedVC?.dismiss(animated: true)
            }
            case .changed:
                guard let view = isAppearing ? toVC?.view : fromVC?.view else { return }
                UIView.animate(withDuration: 0.2)
                {
                    view.frame.origin.y = location.y - view.bounds.height + touchLocationFromBottom
                }

transitionContext?.updateInteractiveTransition(view.frame.maxY / view.frame.height
                )
            case .ended, .cancelled:

```

```

guard let view = isAppearing ? toVC?.view : fromVC?.view else { return }
    let isCancelled = isAppearing ? (velocity.y < 0.5 || view.center.y < 0.0) : (velocity.y
> 0.5 || view.center.y > 0.0)
addAttachmentBehavior(with: view, isCancelled: isCancelled)
    addCollisionBehavior(with: view)
addItemBehavior(with: view)

    animator.addBehavior(self)
    animator.delegate = self

    self.action =
    { [weak self] in
guard let strongSelf = self else { return }
    if strongSelf.animator.elapsedTime > strongSelf.finishTime
    {
strongSelf.animator.removeAllBehaviors()
}
else
{
strongSelf.transitionContext?.updateInteractiveTransition(view.frame.maxY /
view.frame.height
)
}
默认:
break
}

// 添加碰撞行为，完成时产生弹跳效果
func addCollisionBehavior(with view: UIView)
{
    let collisionBehavior = UICollisionBehavior(items: [view])
    let insets = UIEdgeInsets(top: -view.bounds.height, left: 0.0, bottom: 0.0, right: 0.0)
    collisionBehavior.setTranslatesReferenceBoundsIntoBoundary(with: insets)
    collisionBehavior.collisionDelegate = self
    self.addChildBehavior(collisionBehavior)
}

// 添加附着行为，将遮罩拉向顶部或底部
func addAttachmentBehavior(with view: UIView, isCancelled: Bool)
{
    let anchor: CGPoint
    switch (isAppearing, isCancelled)
    {
        case (true, true), (false, false):
anchor = CGPoint(x: view.center.x, y: -view.frame.height)
        case (true, false), (false, true):
anchor = CGPoint(x: view.center.x, y: view.frame.height)
    }
    let attachmentBehavior = UIAttachmentBehavior(item: view, attachedToAnchor: anchor)
    attachmentBehavior.damping = 0.1
attachmentBehavior.frequency = 3.0
attachmentBehavior.length = 0.5 * view.frame.height
    self.addChildBehavior(attachmentBehavior)
}

// 使视图更有弹性
func addItemBehavior(with view: UIView)
{
    let itemBehavior = UIDynamicItemBehavior(items: [view])
    itemBehavior.allowsRotation = false
}

```

```

guard let view = isAppearing ? toVC?.view : fromVC?.view else { return }
    let isCancelled = isAppearing ? (velocity.y < 0.5 || view.center.y < 0.0) : (velocity.y
> 0.5 || view.center.y > 0.0)
addAttachmentBehavior(with: view, isCancelled: isCancelled)
    addCollisionBehavior(with: view)
addItemBehavior(with: view)

    animator.addBehavior(self)
    animator.delegate = self

    self.action =
    { [weak self] in
guard let strongSelf = self else { return }
    if strongSelf.animator.elapsedTime > strongSelf.finishTime
    {
        strongSelf.animator.removeAllBehaviors()
}
else
{
        strongSelf.transitionContext?.updateInteractiveTransition(view.frame.maxY /
view.frame.height
)
}
default:
break
}

// Add collision behavior that causes bounce when finished
func addCollisionBehavior(with view: UIView)
{
    let collisionBehavior = UICollisionBehavior(items: [view])
    let insets = UIEdgeInsets(top: -view.bounds.height, left: 0.0, bottom: 0.0, right: 0.0)
    collisionBehavior.setTranslatesReferenceBoundsIntoBoundary(with: insets)
    collisionBehavior.collisionDelegate = self
    self.addChildBehavior(collisionBehavior)
}

// Add attachment behavior that pulls shade either to top or bottom
func addAttachmentBehavior(with view: UIView, isCancelled: Bool)
{
    let anchor: CGPoint
    switch (isAppearing, isCancelled)
    {
        case (true, true), (false, false):
            anchor = CGPoint(x: view.center.x, y: -view.frame.height)
        case (true, false), (false, true):
            anchor = CGPoint(x: view.center.x, y: view.frame.height)
    }
    let attachmentBehavior = UIAttachmentBehavior(item: view, attachedToAnchor: anchor)
    attachmentBehavior.damping = 0.1
attachmentBehavior.frequency = 3.0
attachmentBehavior.length = 0.5 * view.frame.height
    self.addChildBehavior(attachmentBehavior)
}

// Makes view more bouncy
func addItemBehavior(with view: UIView)
{
    let itemBehavior = UIDynamicItemBehavior(items: [view])
    itemBehavior.allowsRotation = false
}

```

```

itemBehavior.elasticity = 0.6
    self.addChildBehavior(itemBehavior)
}

}

extension ShadeAnimator: UIDynamicAnimatorDelegate
{
    // 判断过渡是否结束
    func dynamicAnimatorDidPause(_ animator: UIDynamicAnimator)
    {
        guard let transitionContext = self.transitionContext else { return }
        let fromVC = transitionContext.viewController(forKey: .from)
        let toVC = transitionContext.viewController(forKey: .to)
        guard let view = isAppearing ? toVC?.view : fromVC?.view else { return }
        switch (view.center.y < 0.0, isAppearing)
        {
            case (true, true), (true, false):
                view.removeFromSuperview()
                transitionContext.finishInteractiveTransition()
                transitionContext.completeTransition(!isAppearing)
            case (false, true):
                toVC?.view.frame = transitionContext.finalFrame(for: toVC!)
                transitionContext.finishInteractiveTransition()
        }
        transitionContext.completeTransition(true)
        case (false, false):
            fromVC?.view.frame = transitionContext.initialFrame(for: fromVC!)
            transitionContext.cancelInteractiveTransition()
        transitionContext.completeTransition(false)
    }
    childBehaviors.forEach { removeChildBehavior($0) }
    animator.removeAllBehaviors()
    self.animator = nil
    self.transitionContext = nil
}
}

extension ShadeAnimator: UICollisionBehaviorDelegate
{
    // 触发触觉反馈
    func collisionBehavior(_ behavior: UICollisionBehavior, beganContactFor item: UIDynamicItem,
    withBoundaryIdentifier identifier: NSCopying?, at p: CGPoint)
    {
        guard p.y > 0.0 else { return }
        impactFeedbackGenerator.impactOccurred()
    }
}

extension ShadeAnimator: UIViewControllerInteractiveTransitioning
{
    // 开始过渡
    func startInteractiveTransition(_ transitionContext: UIViewControllerContextTransitioning)
    {
        setupViewsForTransition(with: transitionContext)
    }
}

```

## Objective-C

```

@interface ShadeAnimator() <UIDynamicAnimatorDelegate, UICollisionBehaviorDelegate>
@property (nonatomic, assign) BOOL isAppearing;
@property (nonatomic, weak) UIViewController *presentingVC;
@property (nonatomic, weak) UIViewController *presentedVC;
@property (nonatomic, weak) NSObject<UIViewControllerTransitioningDelegate> *transitionDelegate;
@property (nonatomic, strong) UIImpactFeedbackGenerator *impactFeedbackGenerator;

```

```

itemBehavior.elasticity = 0.6
    self.addChildBehavior(itemBehavior)
}

}

extension ShadeAnimator: UIDynamicAnimatorDelegate
{
    // Determines transition has ended
    func dynamicAnimatorDidPause(_ animator: UIDynamicAnimator)
    {
        guard let transitionContext = self.transitionContext else { return }
        let fromVC = transitionContext.viewController(forKey: .from)
        let toVC = transitionContext.viewController(forKey: .to)
        guard let view = isAppearing ? toVC?.view : fromVC?.view else { return }
        switch (view.center.y < 0.0, isAppearing)
        {
            case (true, true), (true, false):
                view.removeFromSuperview()
                transitionContext.finishInteractiveTransition()
                transitionContext.completeTransition(!isAppearing)
            case (false, true):
                toVC?.view.frame = transitionContext.finalFrame(for: toVC!)
                transitionContext.finishInteractiveTransition()
                transitionContext.completeTransition(true)
            case (false, false):
                fromVC?.view.frame = transitionContext.initialFrame(for: fromVC!)
                transitionContext.cancelInteractiveTransition()
                transitionContext.completeTransition(false)
        }
        childBehaviors.forEach { removeChildBehavior($0) }
        animator.removeAllBehaviors()
        self.animator = nil
        self.transitionContext = nil
    }
}

extension ShadeAnimator: UICollisionBehaviorDelegate
{
    // Triggers haptics
    func collisionBehavior(_ behavior: UICollisionBehavior, beganContactFor item: UIDynamicItem,
    withBoundaryIdentifier identifier: NSCopying?, at p: CGPoint)
    {
        guard p.y > 0.0 else { return }
        impactFeedbackGenerator.impactOccurred()
    }
}

extension ShadeAnimator: UIViewControllerInteractiveTransitioning
{
    // Starts transition
    func startInteractiveTransition(_ transitionContext: UIViewControllerContextTransitioning)
    {
        setupViewsForTransition(with: transitionContext)
    }
}

```

## Objective-C

```

@interface ShadeAnimator() <UIDynamicAnimatorDelegate, UICollisionBehaviorDelegate>
@property (nonatomic, assign) BOOL isAppearing;
@property (nonatomic, weak) UIViewController *presentingVC;
@property (nonatomic, weak) UIViewController *presentedVC;
@property (nonatomic, weak) NSObject<UIViewControllerTransitioningDelegate> *transitionDelegate;
@property (nonatomic, strong) UIImpactFeedbackGenerator *impactFeedbackGenerator;

```

```

@property (nonatomic, strong) id<UIViewControllerContextTransitioning> transitionContext;
@property (nonatomic, assign) NSTimeInterval finishTime;
@property (nonatomic, strong) UIPanGestureRecognizer *pan;
@property (nonatomic, strong) UIDynamicAnimator *animator;
@end

@implementation ShadeAnimator

- (instancetype)initWithIsAppearing:(BOOL)isAppearing presentingVC:(UIViewController *)presentingVC
presentedVC:(UIViewController *)presentedVC
transitionDelegate:(id<UIViewControllerTransitioningDelegate>)transitionDelegate
{
    self = [super init];
    if (self)
    {
        _isAppearing = isAppearing;
        _presentingVC = presentingVC;
        _presentedVC = presentedVC;
        _transitionDelegate = transitionDelegate;
        _impactFeedbackGenerator = [[UIImpactFeedbackGenerator
alloc]initWithStyle:UIImpactFeedbackStyleLight];
        [_impactFeedbackGenerator prepare];
        if (_isAppearing)
        {
            [_presentingVC.view addGestureRecognizer:self.pan];
        }
        else
        {
            [_presentedVC.view addGestureRecognizer:self.pan];
        }
    }
    return self;
}

#pragma mark - 延迟初始化
- (UIPanGestureRecognizer *)pan
{
    if (!_pan)
    {
        _pan = [[UIPanGestureRecognizer alloc] initWithTarget:self action:@selector(handlePan:)];
    }
    return _pan;
}

- (UIDynamicAnimator *)animator
{
    if (!_animator)
    {
        _animator = [[UIDynamicAnimator
alloc] initWithReferenceView:self.transitionContext.containerView];
    }
    return _animator;
}

#pragma mark - 设置
...
... (id<UIViewControllerContextTransitioning>)transitionContext
{
    UIViewController *fromVC = [transitionContext
viewControllerForKey:UITransitionContextFromViewControllerKey];
    UIViewController *toVC = [transitionContext
viewControllerForKey:UITransitionContextToViewControllerKey];
}

```

```

@property (nonatomic, strong) id<UIViewControllerContextTransitioning> transitionContext;
@property (nonatomic, assign) NSTimeInterval finishTime;
@property (nonatomic, strong) UIPanGestureRecognizer *pan;
@property (nonatomic, strong) UIDynamicAnimator *animator;
@end

@implementation ShadeAnimator

- (instancetype)initWithIsAppearing:(BOOL)isAppearing presentingVC:(UIViewController *)presentingVC
presentedVC:(UIViewController *)presentedVC
transitionDelegate:(id<UIViewControllerTransitioningDelegate>)transitionDelegate
{
    self = [super init];
    if (self)
    {
        _isAppearing = isAppearing;
        _presentingVC = presentingVC;
        _presentedVC = presentedVC;
        _transitionDelegate = transitionDelegate;
        _impactFeedbackGenerator = [[UIImpactFeedbackGenerator
alloc] initWithStyle:UIImpactFeedbackStyleLight];
        [_impactFeedbackGenerator prepare];
        if (_isAppearing)
        {
            [_presentingVC.view addGestureRecognizer:self.pan];
        }
        else
        {
            [_presentedVC.view addGestureRecognizer:self.pan];
        }
    }
    return self;
}

#pragma mark - Lazy Init
- (UIPanGestureRecognizer *)pan
{
    if (!_pan)
    {
        _pan = [[UIPanGestureRecognizer alloc] initWithTarget:self action:@selector(handlePan:)];
    }
    return _pan;
}

- (UIDynamicAnimator *)animator
{
    if (!_animator)
    {
        _animator = [[UIDynamicAnimator
alloc] initWithReferenceView:self.transitionContext.containerView];
    }
    return _animator;
}

#pragma mark - Setup
-
(void)setupViewForTransitionWithContext:(id<UIViewControllerContextTransitioning>)transitionContext
{
    UIViewController *fromVC = [transitionContext
viewControllerForKey:UITransitionContextFromViewControllerKey];
    UIViewController *toVC = [transitionContext
viewControllerForKey:UITransitionContextToViewControllerKey];
}

```

```

UIView *toView = toVC.view;
UIView *containerView = transitionContext.containerView;
self.transitionContext = transitionContext;
if (self.isAppearing)
{
    CGRect fromViewInitialFrame = [transitionContext initialFrameForViewController:fromVC];
    CGRect toViewInitialFrame = toView.frame;
    toViewInitialFrame.origin.y -= CGRectGetHeight(toViewInitialFrame);
    toViewInitialFrame.origin.x = CGRectGetWidth(fromViewInitialFrame) * 0.5 -
    CGRectGetWidth(toViewInitialFrame) * 0.5;

    [containerView addSubview:toView];
}
else
{
    [fromVC.view addGestureRecognizer:self.pan];
}
}

#pragma mark - 手势
- (void)handlePan:(UIPanGestureRecognizer *)sender
{
    CGPoint location = [sender locationInView:self.transitionContext.containerView];
    CGPoint velocity = [sender velocityInView:self.transitionContext.containerView];
    UIViewController *fromVC = [self.transitionContext
viewControllerForKey:UITransitionContextFromViewControllerKey];
    UIViewController *toVC = [self.transitionContext
viewControllerForKey:UITransitionContextToViewControllerKey];

    CGFloat touchStartHeight = 90.0;
    CGFloat touchLocationFromBottom = 20.0;

    if (sender.state == UIGestureRecognizerStateBegan)
    {
        CGPoint beginLocation = [sender locationInView:sender.view];
        if (self.isAppearing)
        {
            if (beginLocation.y <= touchStartHeight)
            {
                self.presentedVC.modalPresentationStyle = UIModalPresentationCustom;
                self.presentedVC.transitioningDelegate = self.transitionDelegate;
                [self.presentingVC presentViewController:self.presentedVC animated:YES
completion:nil];
            }
        }
        else
        {
            if (beginLocation.y >= [sender locationInView:sender.view].y - touchStartHeight)
            {
                [self.presentedVC dismissViewControllerAnimated:true completion:nil];
            }
        }
    }
    else if (sender.state == UIGestureRecognizerStateChanged)
    {
        UIView *view = self.isAppearing ? toVC.view : fromVC.view;
        [UIView animateWithDuration:0.2 animations:^{
            CGRect frame = view.frame;
            frame.origin.y = location.y - CGRectGetHeight(view.bounds) + touchLocationFromBottom;
            view.frame = frame;
        }];
        [self.transitionContext updateInteractiveTransition:CGRectGetMaxY(view.frame) /
    }
}

```

```

UIView *toView = toVC.view;
UIView *containerView = transitionContext.containerView;
self.transitionContext = transitionContext;
if (self.isAppearing)
{
    CGRect fromViewInitialFrame = [transitionContext initialFrameForViewController:fromVC];
    CGRect toViewInitialFrame = toView.frame;
    toViewInitialFrame.origin.y -= CGRectGetHeight(toViewInitialFrame);
    toViewInitialFrame.origin.x = CGRectGetWidth(fromViewInitialFrame) * 0.5 -
    CGRectGetWidth(toViewInitialFrame) * 0.5;

    [containerView addSubview:toView];
}
else
{
    [fromVC.view addGestureRecognizer:self.pan];
}

#pragma mark - Gesture
- (void)handlePan:(UIPanGestureRecognizer *)sender
{
    CGPoint location = [sender locationInView:self.transitionContext.containerView];
    CGPoint velocity = [sender velocityInView:self.transitionContext.containerView];
    UIViewController *fromVC = [self.transitionContext
viewControllerForKey:UITransitionContextFromViewControllerKey];
    UIViewController *toVC = [self.transitionContext
viewControllerForKey:UITransitionContextToViewControllerKey];

    CGFloat touchStartHeight = 90.0;
    CGFloat touchLocationFromBottom = 20.0;

    if (sender.state == UIGestureRecognizerStateBegan)
    {
        CGPoint beginLocation = [sender locationInView:sender.view];
        if (self.isAppearing)
        {
            if (beginLocation.y <= touchStartHeight)
            {
                self.presentedVC.modalPresentationStyle = UIModalPresentationCustom;
                self.presentedVC.transitioningDelegate = self.transitionDelegate;
                [self.presentingVC presentViewController:self.presentedVC animated:YES
completion:nil];
            }
        }
        else
        {
            if (beginLocation.y >= [sender locationInView:sender.view].y - touchStartHeight)
            {
                [self.presentedVC dismissViewControllerAnimated:true completion:nil];
            }
        }
    }
    else if (sender.state == UIGestureRecognizerStateChanged)
    {
        UIView *view = self.isAppearing ? toVC.view : fromVC.view;
        [UIView animateWithDuration:0.2 animations:^{
            CGRect frame = view.frame;
            frame.origin.y = location.y - CGRectGetHeight(view.bounds) + touchLocationFromBottom;
            view.frame = frame;
        }];
        [self.transitionContext updateInteractiveTransition:CGRectGetMaxY(view.frame) /
    }
}

```

```

CGRectGetHeight(view.frame)];
}
else if (sender.state == UIGestureRecognizerStateChanged || sender.state ==
UIGestureRecognizerStateCancelled)
{
    UIView *view = self.isAppearing ? toVC.view : fromVC.view;
    BOOL isCancelled = self.isAppearing ? (velocity.y < 0.5 || view.center.y < 0.0) :
(velocity.y > 0.5 || view.center.y > 0.0);
    [self addAttachmentBehaviorWithView:view isCancelled:isCancelled];
    [self addCollisionBehaviorWithView:view];
    [self addItemBehaviorWithView:view];

    [self.animator addBehavior:self];
    self.animator.delegate = self;

__weak ShadeAnimator *weakSelf = self;
self.action =
 ^{
    if (weakSelf.animator.elapsedTime > weakSelf.finishTime)
    {
        [weakSelf.animator removeAllBehaviors];
    }
    else
    {
        [weakSelf.transitionContext updateInteractiveTransition:CGRectGetMaxY(view.frame) /
CGRectGetHeight(view.frame)];
    }
};

#pragma mark - UIViewControllerInteractiveTransitioning
- (void)startInteractiveTransition:(id<UIViewControllerContextTransitioning>)transitionContext
{
    [self setupViewForTransitionWithContext:transitionContext];
}

#pragma mark - 行为
- (void)addCollisionBehaviorWithView:(UIView *)view
{
    UICollisionBehavior *collisionBehavior = [[UICollisionBehavior alloc] initWithItems:@[view]];
    UIEdgeInsets insets = UIEdgeInsetsMake(-CGRectGetHeight(view.bounds), 0.0, 0.0, 0.0);
    [collisionBehavior setTranslatesReferenceBoundsIntoBoundaryWithInsets:insets];
    collisionBehavior.collisionDelegate = self;
    [self addChildBehavior:collisionBehavior];
}

- (void)addItemBehaviorWithView:(UIView *)view
{
    UIDynamicItemBehavior *itemBehavior = [[UIDynamicItemBehavior alloc] initWithItems:@[view]];
    itemBehavior.allowsRotation = NO;
    itemBehavior.elasticity = 0.6;
    [self addChildBehavior:itemBehavior];
}

- (void)addAttachmentBehaviorWithView:(UIView *)view isCancelled:(BOOL)isCancelled
{
    CGPoint锚点;
    if ((self.isAppearing && isCancelled) || (!self.isAppearing && isCancelled))
    {
        anchor = CGPointMake(view.center.x, -CGRectGetHeight(view.frame));
    }
}

```

```

CGRectGetHeight(view.frame)];
}
else if (sender.state == UIGestureRecognizerStateChanged || sender.state ==
UIGestureRecognizerStateCancelled)
{
    UIView *view = self.isAppearing ? toVC.view : fromVC.view;
    BOOL isCancelled = self.isAppearing ? (velocity.y < 0.5 || view.center.y < 0.0) :
(velocity.y > 0.5 || view.center.y > 0.0);
    [self addAttachmentBehaviorWithView:view isCancelled:isCancelled];
    [self addCollisionBehaviorWithView:view];
    [self addItemBehaviorWithView:view];

    [self.animator addBehavior:self];
    self.animator.delegate = self;

__weak ShadeAnimator *weakSelf = self;
self.action =
 ^{
    if (weakSelf.animator.elapsedTime > weakSelf.finishTime)
    {
        [weakSelf.animator removeAllBehaviors];
    }
    else
    {
        [weakSelf.transitionContext updateInteractiveTransition:CGRectGetMaxY(view.frame) /
CGRectGetHeight(view.frame)];
    }
};

#pragma mark - UIViewControllerInteractiveTransitioning
- (void)startInteractiveTransition:(id<UIViewControllerContextTransitioning>)transitionContext
{
    [self setupViewForTransitionWithContext:transitionContext];
}

#pragma mark - Behaviors
- (void)addCollisionBehaviorWithView:(UIView *)view
{
    UICollisionBehavior *collisionBehavior = [[UICollisionBehavior alloc] initWithItems:@[view]];
    UIEdgeInsets insets = UIEdgeInsetsMake(-CGRectGetHeight(view.bounds), 0.0, 0.0, 0.0);
    [collisionBehavior setTranslatesReferenceBoundsIntoBoundaryWithInsets:insets];
    collisionBehavior.collisionDelegate = self;
    [self addChildBehavior:collisionBehavior];
}

- (void)addItemBehaviorWithView:(UIView *)view
{
    UIDynamicItemBehavior *itemBehavior = [[UIDynamicItemBehavior alloc] initWithItems:@[view]];
    itemBehavior.allowsRotation = NO;
    itemBehavior.elasticity = 0.6;
    [self addChildBehavior:itemBehavior];
}

- (void)addAttachmentBehaviorWithView:(UIView *)view isCancelled:(BOOL)isCancelled
{
    CGPoint anchor;
    if ((self.isAppearing && isCancelled) || (!self.isAppearing && isCancelled))
    {
        anchor = CGPointMake(view.center.x, -CGRectGetHeight(view.frame));
    }
}

```

```

else
{
    anchor = CGPointMake(view.center.x, -CGRectGetHeight(view.frame));
}
UIAttachmentBehavior *attachmentBehavior = [[UIAttachmentBehavior alloc] initWithItem:view
attachedToAnchor:anchor];
attachmentBehavior.damping = 0.1;
attachmentBehavior.frequency = 3.0;
attachmentBehavior.length = 0.5 * CGRectGetHeight(view.frame);
    [self addChildBehavior:attachmentBehavior];
}

#pragma mark - UICollisionBehaviorDelegate
- (void)collisionBehavior:(UICollisionBehavior *)behavior
beganContactForItem:(id<UIDynamicItem>)item withBoundaryIdentifier:(id<NSCopying>)identifier
atPoint:(CGPoint)p
{
    if (p.y > 0.0)
    {
        [self.impactFeedbackGenerator impactOccurred];
    }
}

#pragma mark - UIDynamicAnimatorDelegate
- (void)dynamicAnimatorDidPause:(UIDynamicAnimator *)animator
{
    UIViewController *fromVC = [self.transitionContext
viewControllerForKey:UITransitionContextFromViewControllerKey];
    UIViewController *toVC = [self.transitionContext
viewControllerForKey:UITransitionContextToViewControllerKey];
    UIView *view = self.isAppearing ? toVC.view : fromVC.view;
    if (view.center.y < 0.0 && (self.isAppearing || !self.isAppearing))
    {
        [view removeFromSuperview];
        [self.transitionContext finishInteractiveTransition];
        [self.transitionContext completeTransition:!self.isAppearing];
    }
    else if (view.center.y >= 0.0 && self.isAppearing)
    {
        toVC.view.frame = [self.transitionContext finalFrameForViewController:toVC];
        [self.transitionContext finishInteractiveTransition];
        [self.transitionContext completeTransition:YES];
    }
    else
    {
        fromVC.view.frame = [self.transitionContext initialFrameForViewController:fromVC];
        [self.transitionContext cancelInteractiveTransition];
        [self.transitionContext completeTransition:NO];
    }
    for (UIDynamicBehavior *behavior in self.childBehaviors)
    {
        [self removeChildBehavior:behavior];
    }
    [animator removeAllBehaviors];
    self.animator = nil;
    self.transitionContext = nil;
}
}

@end

```

动画器在平移手势开始时触发过渡的开始。并且仅在手势变化时移动视图

```

else
{
    anchor = CGPointMake(view.center.x, -CGRectGetHeight(view.frame));
}
UIAttachmentBehavior *attachmentBehavior = [[UIAttachmentBehavior alloc] initWithItem:view
attachedToAnchor:anchor];
attachmentBehavior.damping = 0.1;
attachmentBehavior.frequency = 3.0;
attachmentBehavior.length = 0.5 * CGRectGetHeight(view.frame);
    [self addChildBehavior:attachmentBehavior];
}

#pragma mark - UICollisionBehaviorDelegate
- (void)collisionBehavior:(UICollisionBehavior *)behavior
beganContactForItem:(id<UIDynamicItem>)item withBoundaryIdentifier:(id<NSCopying>)identifier
atPoint:(CGPoint)p
{
    if (p.y > 0.0)
    {
        [self.impactFeedbackGenerator impactOccurred];
    }
}

#pragma mark - UIDynamicAnimatorDelegate
- (void)dynamicAnimatorDidPause:(UIDynamicAnimator *)animator
{
    UIViewController *fromVC = [self.transitionContext
viewControllerForKey:UITransitionContextFromViewControllerKey];
    UIViewController * toVC = [self.transitionContext
viewControllerForKey:UITransitionContextToViewControllerKey];
    UIView *view = self.isAppearing ? toVC.view : fromVC.view;
    if (view.center.y < 0.0 && (self.isAppearing || !self.isAppearing))
    {
        [view removeFromSuperview];
        [self.transitionContext finishInteractiveTransition];
        [self.transitionContext completeTransition:!self.isAppearing];
    }
    else if (view.center.y >= 0.0 && self.isAppearing)
    {
        toVC.view.frame = [self.transitionContext finalFrameForViewController:toVC];
        [self.transitionContext finishInteractiveTransition];
        [self.transitionContext completeTransition:YES];
    }
    else
    {
        fromVC.view.frame = [self.transitionContext initialFrameForViewController:fromVC];
        [self.transitionContext cancelInteractiveTransition];
        [self.transitionContext completeTransition:NO];
    }
    for (UIDynamicBehavior *behavior in self.childBehaviors)
    {
        [self removeChildBehavior:behavior];
    }
    [animator removeAllBehaviors];
    self.animator = nil;
    self.transitionContext = nil;
}
}

@end

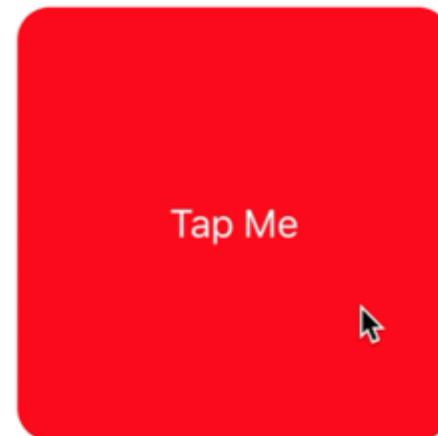
```

The animator triggers the start of the transition when the pan gesture begins. And simply moves the view as the

手势结束时，`UIDynamicBehaviors` 决定过渡是完成还是取消。为此，它使用了附着和碰撞行为。更多信息请参见2013年WWDC会议“UIKit Dynamics高级技巧”。

## 第59.5节：将动态动画位置变化映射到边界

本示例展示了如何自定义`UIDynamicItem`协议，将动态动画视图的位置变化映射到边界变化，从而创建一个以弹性方式扩展和收缩的UIButton。



首先，我们需要创建一个新的协议，该协议实现`UIDynamicItem`，同时具有可设置和可获取的`bounds`属性。

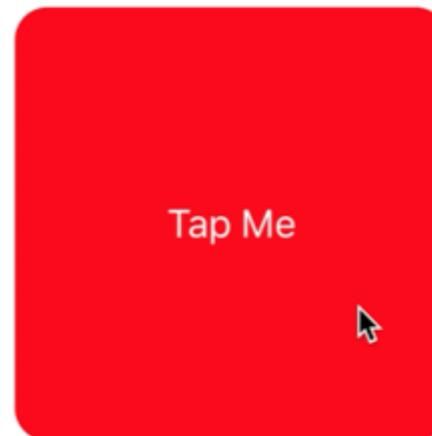
### Swift

```
protocol ResizableDynamicItem: UIDynamicItem
```

gesture changes. But when the gesture ends that is when `UIDynamicBehaviors` determines if the transition should be completed or cancelled. To do so it uses an attachment and collision behavior. For more information see the [2013 WWDC Session "Advanced Techniques with UIKit Dynamics"](#).

## Section 59.5: Map Dynamic Animation Position Changes to Bounds

This example shows how to customize the `UIDynamicItem` protocol to map position changes of a view being dynamically animated to bounds changes to create a `UIButton` that expands and contracts in a elastic fashion.



To start we need to create a new protocol that implements `UIDynamicItem` but that also has a settable and gettable `bounds` property.

### Swift

```
protocol ResizableDynamicItem: UIDynamicItem
```

```

{
    var bounds: CGRect { set get }
}
extension UIView: ResizableDynamicItem {}

```

### Objective-C

```

@protocol ResizableDynamicItem <UIDynamicItem>
@property (nonatomic, readwrite) CGRect bounds;
@end

```

然后我们将创建一个包装对象，该对象将包装一个UIDynamicItem，但会将中心点的变化映射到该项的宽度和高度。我们还将为底层项的bounds和transform提供透传。这将导致动态动画器对底层项的中心x和y值所做的任何更改都应用到该项的宽度和高度上。

### Swift

```

final class PositionToBoundsMapping: NSObject, UIDynamicItem
{
    var target: ResizableDynamicItem

    init(target: ResizableDynamicItem)
    {
        self.target = target
        super.init()
    }

    var bounds: CGRect
    {
        get
        {
            return self.target.bounds
        }
    }

    var center: CGPoint
    {
        get
        {
            return CGPoint(x: self.target.bounds.width, y: self.target.bounds.height)
        }

        set
        {
            self.target.bounds = CGRect(x: 0.0, y: 0.0, 宽度: newValue.x, 高度: newValue.y)
        }
    }

    var transform: CGAffineTransform
    {
        get
        {
            return self.target.transform
        }

        set
        {
            self.target.transform = newValue
        }
    }
}

```

```

{
    var bounds: CGRect { set get }
}
extension UIView: ResizableDynamicItem {}

```

### Objective-C

```

@protocol ResizableDynamicItem <UIDynamicItem>
@property (nonatomic, readwrite) CGRect bounds;
@end

```

We'll then create a wrapper object that will wrap a `UIDynamicItem` but will map center changes to the item's width and height. We will also provide passthroughs for `bounds` and `transform` of the underlying item. This will cause any changes the dynamic animator makes to the center x and y values of the underlying item will be applied to the items width and height.

### Swift

```

final class PositionToBoundsMapping: NSObject, UIDynamicItem
{
    var target: ResizableDynamicItem

    init(target: ResizableDynamicItem)
    {
        self.target = target
        super.init()
    }

    var bounds: CGRect
    {
        get
        {
            return self.target.bounds
        }
    }

    var center: CGPoint
    {
        get
        {
            return CGPoint(x: self.target.bounds.width, y: self.target.bounds.height)
        }

        set
        {
            self.target.bounds = CGRect(x: 0.0, y: 0.0, width: newValue.x, height: newValue.y)
        }
    }

    var transform: CGAffineTransform
    {
        get
        {
            return self.target.transform
        }

        set
        {
            self.target.transform = newValue
        }
    }
}

```

```
}
```

## Objective-C

```
@interface PositionToBoundsMapping ()  
@property (nonatomic, strong) id<ResizableDynamicItem> target;  
@end  
  
@implementation PositionToBoundsMapping  
  
- (instancetype)initWithTarget:(id<ResizableDynamicItem>)target  
{  
    self = [super init];  
    if (self)  
    {  
        _target = target;  
    }  
    return self;  
}  
  
- (CGRect)bounds  
{  
    return self.target.bounds;  
}  
  
- (CGPoint)center  
{  
    return CGPointMake(self.target.bounds.size.width, self.target.bounds.size.height);  
}  
  
- (void)setCenter:(CGPoint)center  
{  
    self.target.bounds = CGRectMake(0, 0, center.x, center.y);  
}  
  
- (CGAffineTransform)transform  
{  
    return self.target.transform;  
}  
  
- (void)setTransform:(CGAffineTransform)transform  
{  
    self.target.transform = transform;  
}  
  
@end
```

最后，我们将创建一个 UIViewController，其中包含一个按钮。当按钮被按下时，我们将创建一个 PositionToBoundsMapping，使用该按钮作为包装的动态项。我们创建一个 UIAttachmentBehavior 使其保持当前位置，然后添加一个瞬时的 UIPushBehavior。然而，由于我们映射了其 bounds 的变化，按钮不会移动，而是会放大和缩小。

## Swift

```
final class ViewController: UIViewController  
{  
    lazy var button: UIButton =  
    {  
        let button = UIButton(frame: CGRect(x: 0.0, y: 0.0, width: 300.0, height: 200.0))  
        button.backgroundColor = .red  
        button.layer.cornerRadius = 15.0  
    }
```

```
}
```

## Objective-C

```
@interface PositionToBoundsMapping ()  
@property (nonatomic, strong) id<ResizableDynamicItem> target;  
@end  
  
@implementation PositionToBoundsMapping  
  
- (instancetype)initWithTarget:(id<ResizableDynamicItem>)target  
{  
    self = [super init];  
    if (self)  
    {  
        _target = target;  
    }  
    return self;  
}  
  
- (CGRect)bounds  
{  
    return self.target.bounds;  
}  
  
- (CGPoint)center  
{  
    return CGPointMake(self.target.bounds.size.width, self.target.bounds.size.height);  
}  
  
- (void)setCenter:(CGPoint)center  
{  
    self.target.bounds = CGRectMake(0, 0, center.x, center.y);  
}  
  
- (CGAffineTransform)transform  
{  
    return self.target.transform;  
}  
  
- (void)setTransform:(CGAffineTransform)transform  
{  
    self.target.transform = transform;  
}  
  
@end
```

Finally, we'll create a UIViewController that will have a button. When the button is pressed we will create PositionToBoundsMapping with the button as the wrapped dynamic item. We create a UIAttachmentBehavior to its current position then add an instantaneous UIPushBehavior to it. However because we have mapped changes its bounds, the button does not move but rather grows and shrinks.

## Swift

```
final class ViewController: UIViewController  
{  
    lazy var button: UIButton =  
    {  
        let button = UIButton(frame: CGRect(x: 0.0, y: 0.0, width: 300.0, height: 200.0))  
        button.backgroundColor = .red  
        button.layer.cornerRadius = 15.0  
    }
```

```

button.setTitle("点击我", for: .normal)
self.view.addSubview(button)
return button
}()

var buttonBounds = CGRect.zero
var animator: UIDynamicAnimator?

override func viewDidLoad()
{
    super.viewDidLoad()
view.backgroundColor = .white
button.addTarget(self, action: #selector(self.didPressButton(sender:)), for:
.touchesInside)
buttonBounds = button.bounds
}

override func viewDidLayoutSubviews()
{
    super.viewDidLayoutSubviews()
button.center = view.center
}

func didPressButton(sender: UIButton)
{
    // 重置 bounds, 以防按钮连续按两次时, 之前的更改不会传播
button.bounds = buttonBounds
let animator = UIDynamicAnimator(referenceView: view)

// 创建映射
let buttonBoundsDynamicItem = PositionToBoundsMapping(target: button)

// 添加附着行为
let attachmentBehavior = UIAttachmentBehavior(item: buttonBoundsDynamicItem,
attachedToAnchor: buttonBoundsDynamicItem.center)

// 频率越高, 振荡越快
attachmentBehavior.frequency = 2.0

// 阻尼越低, 振荡持续时间越长
attachmentBehavior.damping = 0.1
animator.addBehavior(attachmentBehavior)

let pushBehavior = UIPushBehavior(items: [buttonBoundsDynamicItem], mode: .instantaneous)

// 改变角度以确定高度/宽度的变化, 45°表示高:宽为
1:1
pushBehavior.angle = .pi / 4.0

// 幅度越大, 变化越大
pushBehavior.magnitude = 30.0
animator.addBehavior(pushBehavior)
pushBehavior.active = true

// 保持引用, 防止animator被释放
self.animator = animator
}
}

```

## Objective-C

```
@interface ViewController ()
```

```

button.setTitle("Tap Me", for: .normal)
self.view.addSubview(button)
return button
}()

var buttonBounds = CGRect.zero
var animator: UIDynamicAnimator?

override func viewDidLoad()
{
    super.viewDidLoad()
view.backgroundColor = .white
button.addTarget(self, action: #selector(self.didPressButton(sender:)), for:
.touchesInside)
buttonBounds = button.bounds
}

override func viewDidLayoutSubviews()
{
    super.viewDidLayoutSubviews()
button.center = view.center
}

func didPressButton(sender: UIButton)
{
    // Reset bounds so if button is press twice in a row, previous changes don't propagate
button.bounds = buttonBounds
let animator = UIDynamicAnimator(referenceView: view)

// Create mapping
let buttonBoundsDynamicItem = PositionToBoundsMapping(target: button)

// Add Attachment behavior
let attachmentBehavior = UIAttachmentBehavior(item: buttonBoundsDynamicItem,
attachedToAnchor: buttonBoundsDynamicItem.center)

// Higher frequency faster oscillation
attachmentBehavior.frequency = 2.0

// Lower damping longer oscillation lasts
attachmentBehavior.damping = 0.1
animator.addBehavior(attachmentBehavior)

let pushBehavior = UIPushBehavior(items: [buttonBoundsDynamicItem], mode: .instantaneous)

// Change angle to determine how much height/ width should change 45° means height:width is
1:1
pushBehavior.angle = .pi / 4.0

// Larger magnitude means bigger change
pushBehavior.magnitude = 30.0
animator.addBehavior(pushBehavior)
pushBehavior.active = true

// Hold reference so animator is not released
self.animator = animator
}
}

```

## Objective-C

```
@interface ViewController ()
```

```

@property (nonatomic, strong) UIButton *button;
@property (nonatomic, assign) CGRect buttonBounds;
@property (nonatomic, strong) UIDynamicAnimator *animator;
@end

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    self.view.backgroundColor = [UIColor whiteColor];
    [self.button addTarget:self action:@selector(didTapButton:)
forControlEvents:UIControlEventTouchUpInside];
    self.buttonBounds = self.button.bounds;
}

- (void)viewDidLayoutSubviews
{
    [super viewDidLayoutSubviews];
    self.button.center = self.view.center;
}

- (UIButton *)button
{
    if (!_button)
    {
        _button = [[UIButton alloc]initWithFrame:CGRectMake(0.0, 0.0, 200.0, 200.0)];
        _button.backgroundColor = [UIColor redColor];
        _button.layer.cornerRadius = 15.0;
        [_button setTitle:@"Tap Me" forState:UIControlStateNormal];
        [self.view addSubview:_button];
    }
    return _button;
}

- (void)didTapButton:(id)sender
{
    self.button.bounds = self.buttonBounds;
    UIDynamicAnimator *animator = [[UIDynamicAnimator alloc] initWithReferenceView:self.view];
    PositionToBoundsMapping *buttonBoundsDynamicItem = [[PositionToBoundsMapping
alloc]initWithTarget:sender];
    UIAttachmentBehavior *attachmentBehavior = [[UIAttachmentBehavior
alloc]initWithItem:buttonBoundsDynamicItem attachedToAnchor:buttonBoundsDynamicItem.center];
    [attachmentBehavior setFrequency:2.0];
    [attachmentBehavior setDamping:0.3];
    [animator addBehavior:attachmentBehavior];

    UIPushBehavior *pushBehavior = [[UIPushBehavior alloc] initWithItems:@[buttonBoundsDynamicItem]
mode:UIPushBehaviorModeInstantaneous];
    pushBehavior.angle = M_PI_4;
    pushBehavior.magnitude = 2.0;
    [animator addBehavior:pushBehavior];

    [pushBehavior setActive:TRUE];

    self.animator = animator;
}
@end

```

更多信息请参见[UIKit Dynamics Catalog](#)

```

@property (nonatomic, strong) UIButton *button;
@property (nonatomic, assign) CGRect buttonBounds;
@property (nonatomic, strong) UIDynamicAnimator *animator;
@end

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    self.view.backgroundColor = [UIColor whiteColor];
    [self.button addTarget:self action:@selector(didTapButton:)
forControlEvents:UIControlEventTouchUpInside];
    self.buttonBounds = self.button.bounds;
}

- (void)viewDidLayoutSubviews
{
    [super viewDidLayoutSubviews];
    self.button.center = self.view.center;
}

- (UIButton *)button
{
    if (!_button)
    {
        _button = [[UIButton alloc]initWithFrame:CGRectMake(0.0, 0.0, 200.0, 200.0)];
        _button.backgroundColor = [UIColor redColor];
        _button.layer.cornerRadius = 15.0;
        [_button setTitle:@"Tap Me" forState:UIControlStateNormal];
        [self.view addSubview:_button];
    }
    return _button;
}

- (void)didTapButton:(id)sender
{
    self.button.bounds = self.buttonBounds;
    UIDynamicAnimator *animator = [[UIDynamicAnimator alloc] initWithReferenceView:self.view];
    PositionToBoundsMapping *buttonBoundsDynamicItem = [[PositionToBoundsMapping
alloc]initWithTarget:sender];
    UIAttachmentBehavior *attachmentBehavior = [[UIAttachmentBehavior
alloc]initWithItem:buttonBoundsDynamicItem attachedToAnchor:buttonBoundsDynamicItem.center];
    [attachmentBehavior setFrequency:2.0];
    [attachmentBehavior setDamping:0.3];
    [animator addBehavior:attachmentBehavior];

    UIPushBehavior *pushBehavior = [[UIPushBehavior alloc] initWithItems:@[buttonBoundsDynamicItem]
mode:UIPushBehaviorModeInstantaneous];
    pushBehavior.angle = M_PI_4;
    pushBehavior.magnitude = 2.0;
    [animator addBehavior:pushBehavior];

    [pushBehavior setActive:TRUE];

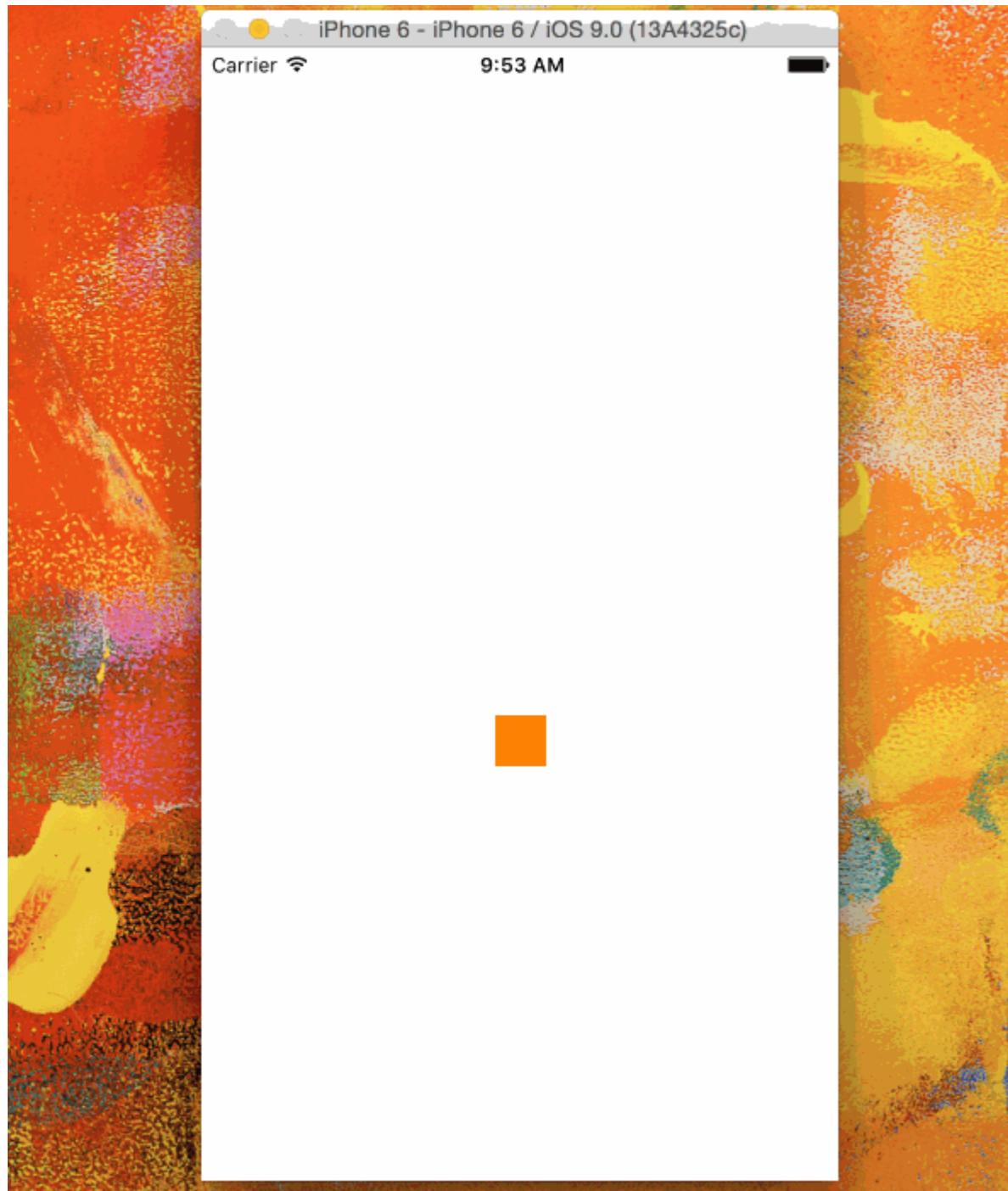
    self.animator = animator;
}
@end

```

For more information see [UIKit Dynamics Catalog](#)

## 第59.6节：下落的正方形

让我们在视图中央绘制一个正方形，使其下落到底部并在底部边缘与屏幕底部边界碰撞后停止。

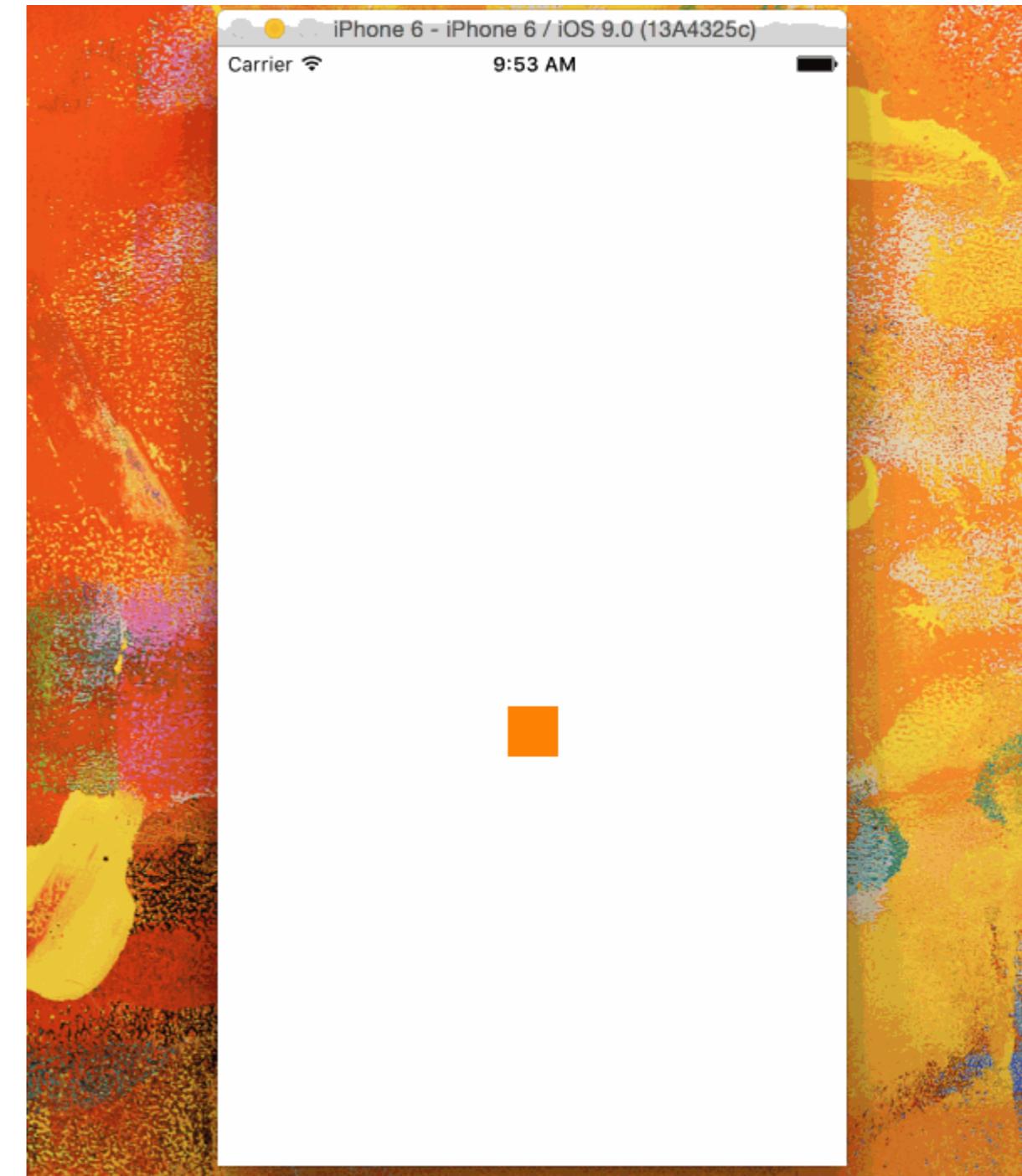


```
@IBOutlet var animationView: UIView!
var squareView: UIView!
var collision: UICollisionBehavior!
var animator: UIDynamicAnimator!
var gravity: UIGravityBehavior!

override func viewDidLoad() {
    super.viewDidLoad()
    let squareSize = CGSize(width: 30.0, height: 30.0)
    let centerPoint = CGPoint(x: self.animationView.bounds.midX - (squareSize.width/2), y:
self.animationView.bounds.midY - (squareSize.height/2))
    let frame = CGRect(origin: centerPoint, size: squareSize)
    squareView = UIView(frame: frame)
```

## Section 59.6: The Falling Square

Lets draw a square in the middle of our view and make it fall to the bottom and stop at the bottom edge collising with the screen bottom boundary.



```
@IBOutlet var animationView: UIView!
var squareView: UIView!
var collision: UICollisionBehavior!
var animator: UIDynamicAnimator!
var gravity: UIGravityBehavior!

override func viewDidLoad() {
    super.viewDidLoad()
    let squareSize = CGSize(width: 30.0, height: 30.0)
    let centerPoint = CGPoint(x: self.animationView.bounds.midX - (squareSize.width/2), y:
self.animationView.bounds.midY - (squareSize.height/2))
    let frame = CGRect(origin: centerPoint, size: squareSize)
    squareView = UIView(frame: frame)
```

```
squareView.backgroundColor = UIColor.orangeColor()
    animationView.addSubview(squareView)
animator = UIDynamicAnimator(参考视图: view)
    gravity = UIGravityBehavior(项目: [squareView])
    animator.addBehavior(gravity)
collision = UICollisionBehavior(项目: [square])
    collision.translatesReferenceBoundsIntoBoundary = true
    animator.addBehavior(collision)
}
```

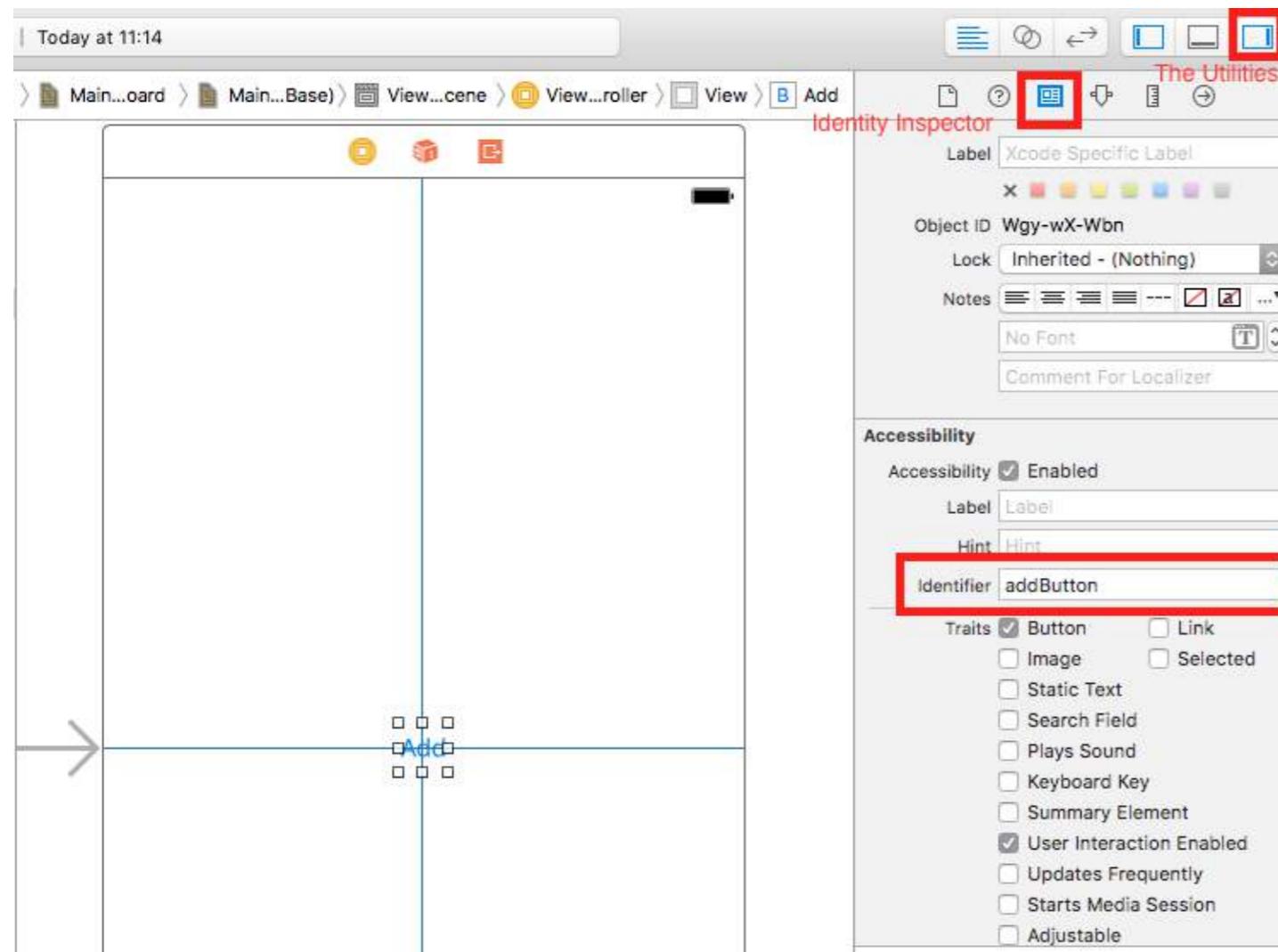
```
squareView.backgroundColor = UIColor.orangeColor()
    animationView.addSubview(squareView)
animator = UIDynamicAnimator(referenceView: view)
    gravity = UIGravityBehavior(items: [squareView])
    animator.addBehavior(gravity)
collision = UICollisionBehavior(items: [square])
    collision.translatesReferenceBoundsIntoBoundary = true
    animator.addBehavior(collision)
}
```

# 第60章：UI测试

## 第60.1节：辅助功能标识符

当在实用工具中启用辅助功能时

- 选择 storyboard。
- 展开实用工具
- 选择身份检查器
- 在故事板上选择你的元素
- 添加新的辅助功能标识符（例如addButton）



当在实用工具中禁用辅助功能时

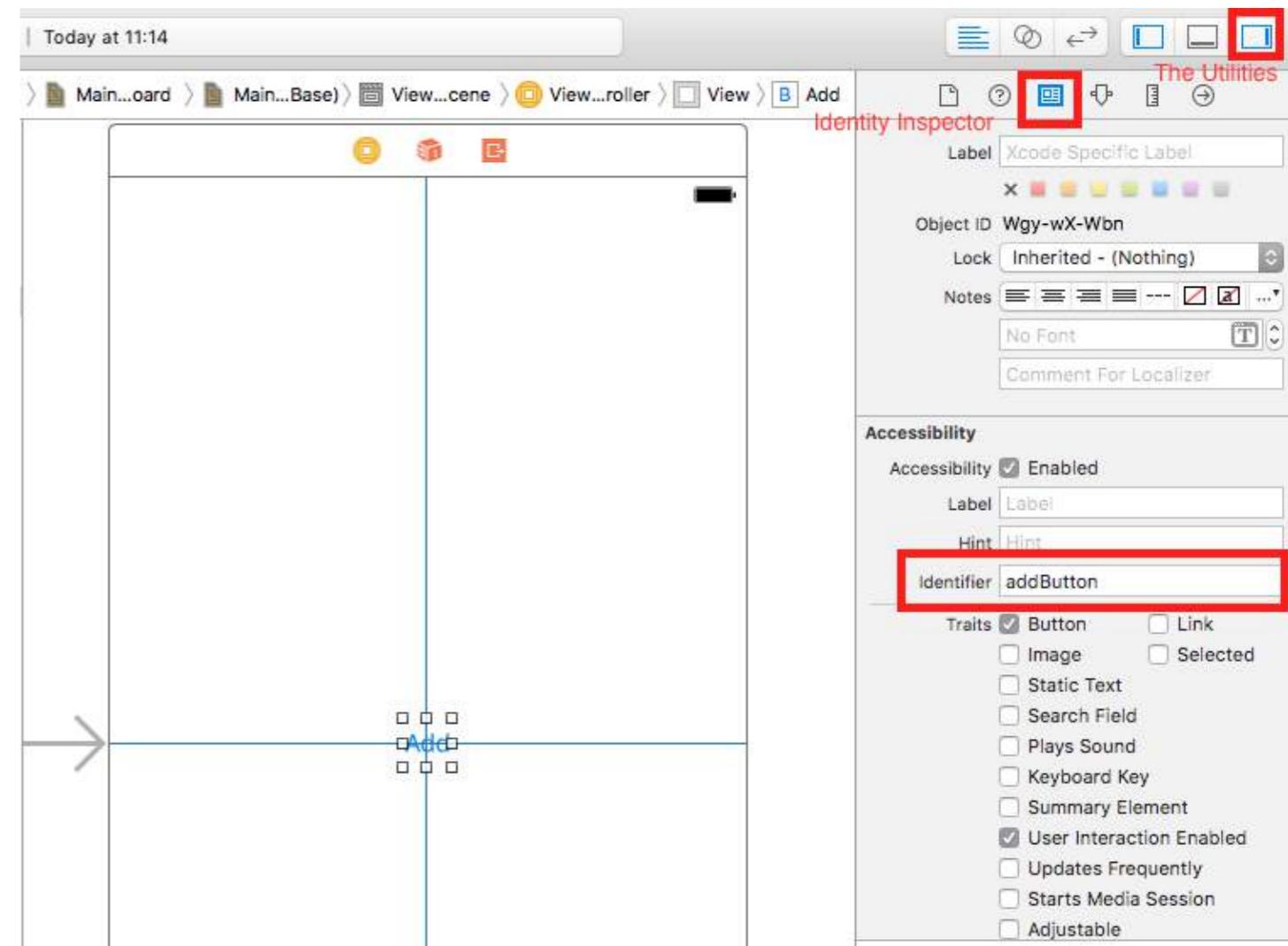
- 选择 storyboard。
- 展开实用工具
- 选择身份检查器
- 在故事板上选择你的元素
- 在用户定义的运行时属性中添加属性
- 对于键路径输入 - accessibilityIdentifier
- 对于类型 - `String`  
• 对于值 - 你的元素的新辅助功能标识符（例如view）

# Chapter 60: UI Testing

## Section 60.1: Accessibility Identifier

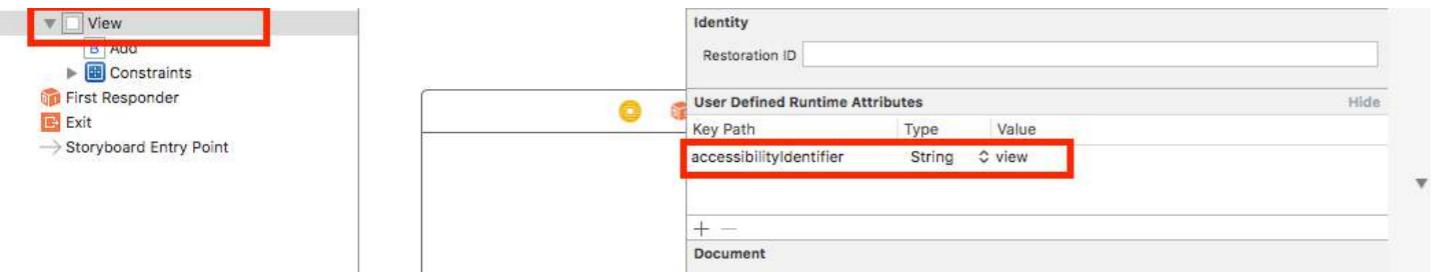
When Accessibility enabled in Utilities

- Select storyboard.
- Expand the Utilities
- Select Identity Inspector
- Select your element on storyboard
- Add new Accessibility Identifier (in example addButton)



When Accessibility disabled in Utilities

- Select storyboard.
- Expand the Utilities
- Select Identity Inspector
- Select your element on storyboard
- Add attribute in User Defined Runtime Attributes
  - For Key Path type - accessibilityIdentifier
  - For Type - `String`  
• For Value - new accessibility identifier for your element (in example view)



## 在UITest文件中设置

导入 XCTest

```
类 StackOverFlowUITests: XCTestCase {
```

```
    private let app = XCUIApplication()
```

//视图

```
private var view: XCUIElement!
```

//按钮

```
private var addButton: XCUIElement!
```

```
override func setUp() {
    super.setUp()
```

```
app.launch()
```

//视图

```
view = app.otherElements["view"]
```

//按钮

```
addButton = app.buttons["addButton"]
}
```

```
func testMyApp() {
```

```
    addButton.tap()
    view.tap()
}
```

在[ ]中为元素添加辅助功能标识符。

### UIView, UIImageView, UIScrollView

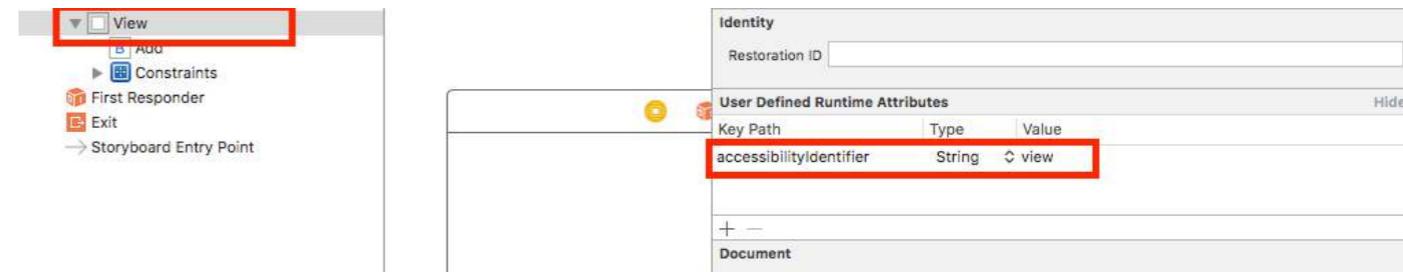
```
let imageView = app.images["imageView"]
let scrollView = app.scrollViews["scrollView"]
let view = app.otherElements["view"]
```

### UILabel

```
let label = app.staticTexts["label"]
```

### UIStackView

```
let stackView = app.otherElements["stackView"]
```



## Setting up in UITest file

```
import XCTest
```

```
class StackOverFlowUITests: XCTestCase {
```

```
    private let app = XCUIApplication()
```

//Views

```
private var view: XCUIElement!
```

//Buttons

```
private var addButton: XCUIElement!
```

```
override func setUp() {
    super.setUp()
```

```
    app.launch()
```

//Views

```
view = app.otherElements["view"]
```

//Buttons

```
addButton = app.buttons["addButton"]
}
```

```
func testMyApp() {
```

```
    addButton.tap()
    view.tap()
}
}
```

In [ ] add Accessibility Identifier for element.

### UIView, UIImageView, UIScrollView

```
let imageView = app.images["imageView"]
let scrollView = app.scrollViews["scrollView"]
let view = app.otherElements["view"]
```

### UILabel

```
let label = app.staticTexts["label"]
```

### UIStackView

```
let stackView = app.otherElements["stackView"]
```

## UITableView

```
let tableView = app.tables["tableView"]
```

## UITableViewCell

```
let tableViewCell = tableView.cells["tableViewCell"]
```

## UITableViewCell 元素

```
let tableViewCellButton = tableView.cells.element(boundBy: 0).buttons["button"]
```

## UICollectionView

```
let collectionView = app.collectionViews["collectionView"]
```

## UIButton, UIBarButtonItem

```
let button = app.buttons["button"]
let barButtonItem = app.buttons["barButtonItem"]
```

## UITextField

- 普通 UITextField

```
let textField = app.textFields["textField"]
```

- 密码 UITextField

```
let passwordTextField = app.secureTextFields["passwordTextField"]
```

## UITextView

```
let textView = app.textViews["textView"]
```

## UISwitch

```
let switch = app.switches["switch"]
```

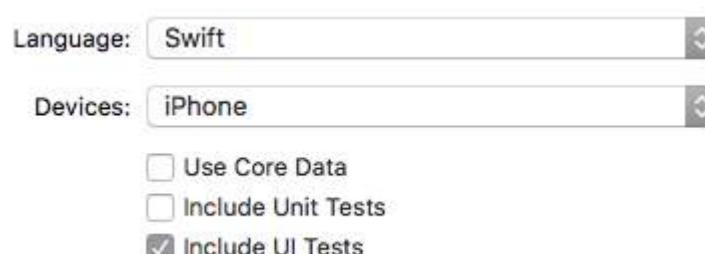
## Alerts

```
let alert = app.alerts["About yourself"] // 弹出警告的标题
```

## 第60.2节：向Xcode项目添加测试文件

### 创建项目时

您应在项目创建对话框中勾选“包含UI测试”。



### 创建项目后

## UITableView

```
let tableView = app.tables["tableView"]
```

## UITableViewCell

```
let tableViewCell = tableView.cells["tableViewCell"]
```

## UITableViewCell elements

```
let tableViewCellButton = tableView.cells.element(boundBy: 0).buttons["button"]
```

## UICollectionView

```
let collectionView = app.collectionViews["collectionView"]
```

## UIButton, UIBarButtonItem

```
let button = app.buttons["button"]
let barButtonItem = app.buttons["barButtonItem"]
```

## UITextField

- normal UITextField

```
let textField = app.textFields["textField"]
```

- password UITextField

```
let passwordTextField = app.secureTextFields["passwordTextField"]
```

## UITextView

```
let textView = app.textViews["textView"]
```

## UISwitch

```
let switch = app.switches["switch"]
```

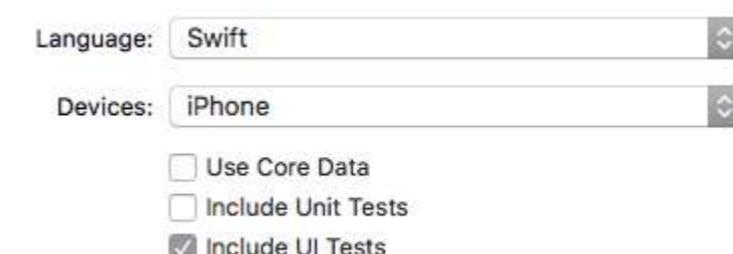
## Alerts

```
let alert = app.alerts["About yourself"] // Title of presented alert
```

## Section 60.2: Adding Test Files to Xcode Project

### When creating the project

You should check "Include UI Tests" in the project creation dialog.



### After creating the project

如果在创建项目时错过了勾选UI目标，也可以随时后续添加测试目标。

步骤：

- 打开项目后，进入文件 -> 新建 -> 目标
- 找到iOS UI测试包



## 第60.3节：在UI测试期间禁用动画

在测试中，可以通过在setUp中添加以下代码来禁用动画：

```
app.launchEnvironment = ["animations": "0"]
```

其中app是XCUIApplication的实例。

## 第60.4节：执行期间启动和终止应用程序

### 启动测试应用程序

```
override func setUp() {
    super.setUp()

    let app = XCUIApplication()
    app.launch()
}
```

### 终止应用程序

```
func testStacOverFlowApp() {
    app.terminate()
}
```

## 第60.5节：旋转设备

设备可以通过更改XCUIDevice.shared().orientation中的方向来旋转：

```
XCUIDevice.shared().orientation = .landscapeLeft
XCUIDevice.shared().orientation = .portrait
```

If you missed checking UI target while creating project, you could always add test target later.

Steps:

- While project open go to File -> New -> Target
- Find iOS UI Testing Bundle



## Section 60.3: Disable animations during UI Testing

In a test you can disable animations by adding in setUp:

```
app.launchEnvironment = ["animations": "0"]
```

Where app is instance of XCUIApplication.

## Section 60.4: Lunch and Terminate application while executing

### Lunch application for testing

```
override func setUp() {
    super.setUp()

    let app = XCUIApplication()
    app.launch()
}
```

### Terminating application

```
func testStacOverFlowApp() {
    app.terminate()
}
```

## Section 60.5: Rotate devices

Device can be rotate by changing orientation in XCUIDevice.shared().orientation:

```
XCUIDevice.shared().orientation = .landscapeLeft
XCUIDevice.shared().orientation = .portrait
```

# 第61章：更改状态栏颜色

## 第61.1节：针对非UINavigationBar状态栏

- 在 info.plist 中将 View controller-based status bar appearance 设置为 YES
- 在不包含于 UINavigationController 的视图控制器中实现此方法。

在 Objective-C 中：

```
- (UIStatusBarStyle)preferredStatusBarStyle
{
    return UIStatusBarStyleLightContent;
}
```

在 Swift 中：

```
override func preferredStatusBarStyle() -> UIStatusBarStyle {
    return UIStatusBarStyle.LightContent
}
```

## 第61.2节：针对 UINavigationBar 状态栏

子类化 UINavigationController，然后重写以下方法：

在 Objective-C 中：

```
- (UIStatusBarStyle)preferredStatusBarStyle
{
    return UIStatusBarStyleLightContent;
}
```

在 Swift 中：

```
override func preferredStatusBarStyle() -> UIStatusBarStyle {
    return .lightContent
}
```

或者，你可以在 UINavigationBar 实例上设置 barStyle：

Objective C:

```
// 例如，在你的视图控制器的 viewDidLoad 方法中：
self.navigationController.navigationBar.barStyle = UIBarStyleBlack; // 这将使状态栏变为白色
```

Swift

```
// 例如，在你的视图控制器的 viewDidLoad 方法中：
navigationController?.navigationBar.barStyle = .black // 这将使状态栏变为白色
```

UIBarStyle 选项有 default、black、blackOpaque、blackTranslucent。后面三个选项都会让状态栏文字变为白色，只是最后两个指定了导航栏的不透明度。

注意：你仍然可以根据需要更改导航栏的外观。

# Chapter 61: Change Status Bar Color

## Section 61.1: For non-UINavigationBar status bars

- In info.plist set View controller-based status bar appearance to YES
- In view controllers not contained by `UINavigationController` implement this method.

In Objective-C:

```
- (UIStatusBarStyle)preferredStatusBarStyle
{
    return UIStatusBarStyleLightContent;
}
```

In Swift:

```
override func preferredStatusBarStyle() -> UIStatusBarStyle {
    return UIStatusBarStyle.LightContent
}
```

## Section 61.2: For UINavigationBar status bars

Subclass UINavigationController and then override these methods:

In Objective-C:

```
- (UIStatusBarStyle)preferredStatusBarStyle
{
    return UIStatusBarStyleLightContent;
}
```

In Swift:

```
override func preferredStatusBarStyle() -> UIStatusBarStyle {
    return .lightContent
}
```

Alternatively, you can set barStyle on the `UINavigationBar` instance:

Objective C:

```
// e.g. in your view controller's viewDidLoad method:
self.navigationController.navigationBar.barStyle = UIBarStyleBlack; // this will give you a white
status bar
```

Swift

```
// e.g. in your view controller's viewDidLoad method:
navigationController?.navigationBar.barStyle = .black // this will give you a white status bar
```

UIBarStyle options are `default`, `black`, `blackOpaque`, `blackTranslucent`. The latter 3 should all give you a status bar with white text, just the last two specify the opacity of the bar.

Note: you can still change the appearance of your navigation bar as you like.

## 第61.3节：关于ViewController的包含

如果你使用UIViewControllerContainment，有一些其他方法值得关注。

当你希望子视图控制器控制状态栏的显示（例如子视图控制器位于屏幕顶部时）

在Swift中

```
class RootViewController: UIViewController {  
  
    private let messageBarViewController = MessageBarViewController()  
  
    override func childViewControllerForStatusBarStyle() -> UIViewController? {  
        return messageBarViewController  
    }  
  
    override func viewDidLoad() {  
        super.viewDidLoad()  
  
        // 在这里添加子视图控制器代码...  
  
        setNeedsStatusBarAppearanceUpdate()  
    }  
  
    class MessageBarViewController: UIViewController {  
  
        override func preferredStatusBarStyle() -> UIStatusBarStyle {  
            return .Default  
        }  
    }  
}
```

## 第61.4节：如果你无法更改ViewController的代码

如果你使用的库中包含（例如）状态栏颜色错误的AwesomeViewController，你可以

尝试如下操作：

```
let awesomeViewController = AwesomeViewController()  
awesomeViewController.navigationBar.barStyle = .blackTranslucent // 或其他样式
```

## 第61.5节：更改整个 应用程序的状态栏样式

SWIFT:

步骤1：

在你的Info.plist中添加以下属性：

基于视图控制器的状态栏外观

并将其值设置为

否

## Section 61.3: For ViewController containment

If you are using UIViewControllerContainment there are a few other methods that are worth looking at.

When you want a child viewController to control the presentation of the status bar (i.e. if the child is positioned at the top of the screen

in Swift

```
class RootViewController: UIViewController {  
  
    private let messageBarViewController = MessageBarViewController()  
  
    override func childViewControllerForStatusBarStyle() -> UIViewController? {  
        return messageBarViewController  
    }  
  
    override func viewDidLoad() {  
        super.viewDidLoad()  
  
        // add child vc code here...  
  
        setNeedsStatusBarAppearanceUpdate()  
    }  
  
    class MessageBarViewController: UIViewController {  
  
        override func preferredStatusBarStyle() -> UIStatusBarStyle {  
            return .Default  
        }  
    }  
}
```

## Section 61.4: If you cannot change ViewController's code

If you are using library that contains (for example) AwesomeViewController with a wrong status bar color you can try this:

```
let awesomeViewController = AwesomeViewController()  
awesomeViewController.navigationBar.barStyle = .blackTranslucent // or other style
```

## Section 61.5: Changing the status bar style for the entire application

SWIFT:

Step 1:

In your **Info.plist** add the following attribute:

View controller-based status bar appearance

and set its value to

NO

如下面图片所示：

Key	Type	Value
▼ Information Property List	Dictionary	(15 items)
View controller-based status...	Boolean	NO

步骤 2：

在你的AppDelegate.swift文件中， didFinishLaunchingWithOptions方法里，添加以下代码：

```
UIApplication.shared.statusBarStyle = .lightContent
```

或者

```
UIApplication.shared.statusBarStyle = .default
```

- .lightContent选项会将整个应用的statusBar颜色设置为白色。
- .default选项会将整个应用的statusBar颜色设置为原始的黑色。

OBJECTIVE-C：

按照SWIFT部分的第一步操作。然后将此代码添加到AppDelegate.m文件中：

```
[[UIApplication sharedApplication] setStatusBarStyle:UIStatusBarStyleLightContent];
```

或者

```
[[UIApplication sharedApplication] setStatusBarStyle:UIStatusBarStyleDefault];
```

as described in the image below:

Key	Type	Value
▼ Information Property List	Dictionary	(15 items)
View controller-based status...	Boolean	NO

Step 2:

In your **AppDelegate.swift** file, in `didFinishLaunchingWithOptions` method, add this code:

```
UIApplication.shared.statusBarStyle = .lightContent
```

or

```
UIApplication.shared.statusBarStyle = .default
```

- The `.lightContent` option will set the colour of the `statusBar` to white, for the entire app.
- The `.default` option will set the colour of the `statusBar` to the original black colour, for the entire app.

OBJECTIVE-C:

Follow the first step from the **SWIFT** Section. Then add this code to the **AppDelegate.m** file:

```
[[UIApplication sharedApplication] setStatusBarStyle:UIStatusBarStyleLightContent];
```

or

```
[[UIApplication sharedApplication] setStatusBarStyle:UIStatusBarStyleDefault];
```

# 第62章：UISegmentedControl

UISegmentedControl对象是由多个分段组成的水平控件，每个分段作为一个独立的按钮。分段控件提供了一种紧凑的方式将多个控件组合在一起。

## 第62.1节：通过代码创建UISegmentedControl

1. 创建一个包含3个项目（分段）的UISegmentedControl新实例：

```
let mySegmentedControl = UISegmentedControl(items: ["One", "Two", "Three"])
```

2. 设置框架；

```
mySegmentedControl.frame = CGRect(x: 0.0, y: 0.0, width: 300, height: 50)
```

3. 设置默认选中（注意分段的索引从0开始）：

```
mySegmentedControl.selectedSegmentIndex = 0
```

4. 配置目标：

```
mySegmentedControl.addTarget(self, action: #selector(segmentedValueChanged(_:)), for:  
.valueChanged)
```

5. 处理值变化：

```
func segmentedValueChanged(_ sender:UISegmentedControl!) {  
    print("Selected Segment Index is : \(sender.selectedSegmentIndex)")  
}
```

6. 将 UISegmentedControl 添加到视图层级

```
yourView.addSubview(mySegmentedControl)
```

# Chapter 62: UISegmentedControl

A UISegmentedControl object is a horizontal control made of multiple segments, each segment functioning as a discrete button. A segmented control affords a compact means to group together a number of controls.

## Section 62.1: Creating UISegmentedControl via code

1. Create new instance of UISegmentedControl filled with 3 items (segments):

```
let mySegmentedControl = UISegmentedControl(items: ["One", "Two", "Three"])
```

2. Setup frame;

```
mySegmentedControl.frame = CGRect(x: 0.0, y: 0.0, width: 300, height: 50)
```

3. Make default selection (note that segments are indexed by 0):

```
mySegmentedControl.selectedSegmentIndex = 0
```

4. Configure target:

```
mySegmentedControl.addTarget(self, action: #selector(segmentedValueChanged(_:)), for:  
.valueChanged)
```

- 5 Handle value changed:

```
func segmentedValueChanged(_ sender:UISegmentedControl!) {  
    print("Selected Segment Index is : \(sender.selectedSegmentIndex)")  
}
```

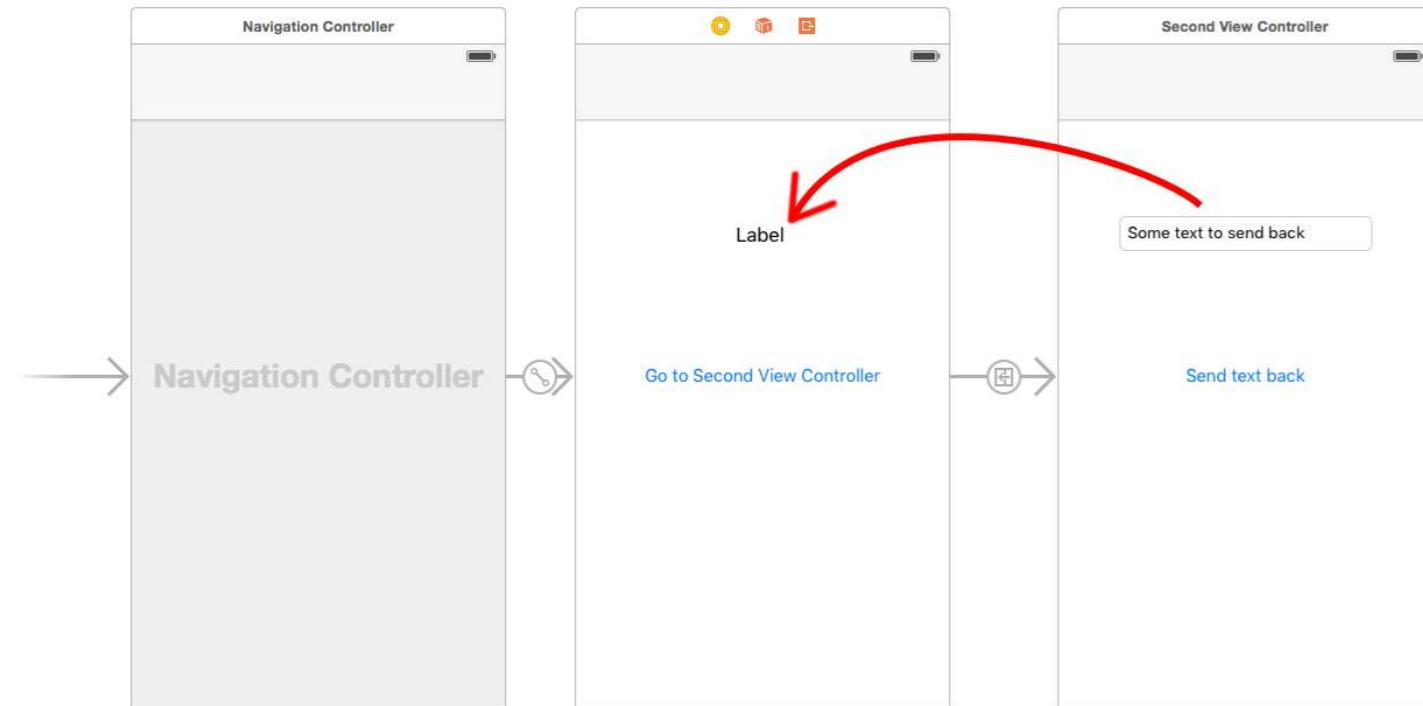
6. Add UISegmentedControl to views hierarchy

```
yourView.addSubview(mySegmentedControl)
```

# 第63章：视图控制器之间传递数据

## 第63.1节：使用代理模式（传递数据回调）

要将数据从当前视图控制器传回到上一个视图控制器，可以使用代理模式。



此示例假设您已在界面构建器中创建了一个segue，并且将segue标识符设置为showSecondViewController。还必须将outlets和actions连接到以下代码中的名称。

### 第一个视图控制器

第一个视图控制器的代码是

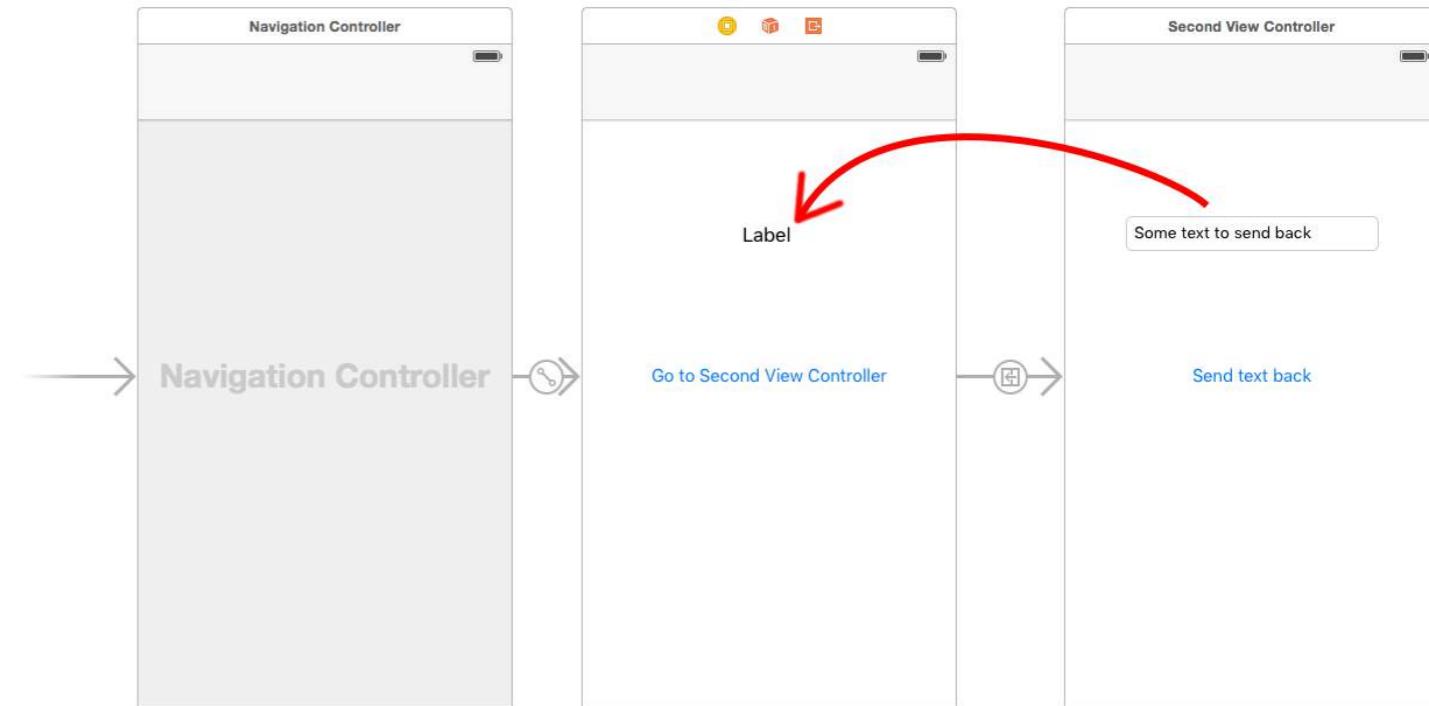
#### Swift

```
class FirstViewController: UIViewController, DataEnteredDelegate {  
  
    @IBOutlet weak var label: UILabel!  
  
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
        if segue.identifier == "showSecondViewController", let secondViewController = segue.destinationViewController as? SecondViewController {  
            secondViewController.delegate = self  
        }  
    }  
  
    // 我们自定义的DataEnteredDelegate协议的必需方法  
    func userDidEnterInformation(info: String) {  
        label.text = info  
        navigationController?.popViewControllerAnimated(true)  
    }  
}
```

# Chapter 63: Passing Data between View Controllers

## Section 63.1: Using the Delegate Pattern (passing data back)

To pass data from the current view controller back to the previous view controller, you can use the delegate pattern.



This example assumes that you have made a segue in the Interface Builder and that you set the segue identifier to showSecondViewController. The outlets and actions must also be hooked up to the names in the following code.

### First View Controller

The code for the First View Controller is

#### Swift

```
class FirstViewController: UIViewController, DataEnteredDelegate {  
  
    @IBOutlet weak var label: UILabel!  
  
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
        if segue.identifier == "showSecondViewController", let secondViewController = segue.destinationViewController as? SecondViewController {  
            secondViewController.delegate = self  
        }  
    }  
  
    // required method of our custom DataEnteredDelegate protocol  
    func userDidEnterInformation(info: String) {  
        label.text = info  
        navigationController?.popViewControllerAnimated(true)  
    }  
}
```

## Objective-C

```
@interface FirstViewController : UIViewController <DataEnteredDelegate>
@property (weak, nonatomic) IBOutlet UILabel *label;
@end

@implementation FirstViewController
- (void)viewDidLoad {
    [super viewDidLoad];
}
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    SecondViewController *secondViewController = segue.destinationViewController;
    secondViewController.delegate = self;
}

    ... (NSString *)info {
_label.text = info
    [self.navigationController popViewControllerAnimated:YES];
}
@end
```

注意我们自定义的DataEnteredDelegate协议的使用。

## 第二视图控制器和协议

第二个视图控制器的代码是

### Swift

```
// 用于发送数据回传的协议
protocol DataEnteredDelegate: class {
    func userDidEnterInformation(info: String)
}

class SecondViewController: UIViewController {

    // 将其设为弱引用变量，以避免产生强引用循环
    weak var delegate: DataEnteredDelegate?

    @IBOutlet weak var textField: UITextField!

    @IBAction func sendTextBackButton(sender: AnyObject) {
        // 调用实现了我们代理协议的类（第一个视图控制器）上的此方法
        delegate?.userDidEnterInformation(textField.text ?? "")
    }
}
```

## Objective-C

```
@protocol DataEnteredDelegate <NSObject>
-(void)userDidEnterInformation:(NSString *)info;
@end

@interface SecondViewController : UIViewController
@property (nonatomic) id <DataEnteredDelegate> delegate;
@property (weak, nonatomic) IBOutlet UITextField *textField;
@end

@implementation SecondViewController
- (void)viewDidLoad {

```

## Objective-C

```
@interface FirstViewController : UIViewController <DataEnteredDelegate>
@property (weak, nonatomic) IBOutlet UILabel *label;
@end

@implementation FirstViewController
- (void)viewDidLoad {
    [super viewDidLoad];
}
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    SecondViewController *secondViewController = segue.destinationViewController;
    secondViewController.delegate = self;
}

    ... (NSString *)info {
_label.text = info
    [self.navigationController popViewControllerAnimated:YES];
}
@end
```

Note the use of our custom DataEnteredDelegate protocol.

## Second View Controller and Protocol

The code for the second view controller is

### Swift

```
// protocol used for sending data back
protocol DataEnteredDelegate: class {
    func userDidEnterInformation(info: String)
}

class SecondViewController: UIViewController {

    // making this a weak variable so that it won't create a strong reference cycle
    weak var delegate: DataEnteredDelegate?

    @IBOutlet weak var textField: UITextField!

    @IBAction func sendTextBackButton(sender: AnyObject) {
        // call this method on whichever class implements our delegate protocol (the first view
        // controller)
        delegate?.userDidEnterInformation(textField.text ?? "")
    }
}
```

## Objective-C

```
@protocol DataEnteredDelegate <NSObject>
-(void)userDidEnterInformation:(NSString *)info;
@end

@interface SecondViewController : UIViewController
@property (nonatomic) id <DataEnteredDelegate> delegate;
@property (weak, nonatomic) IBOutlet UITextField *textField;
@end

@implementation SecondViewController
- (void)viewDidLoad {
```

```

    [super viewDidLoad];
}

- (IBAction) sendTextBackButton:(id)sender{
    [_delegate userDidEnterInformation:textField.text];
}
@end

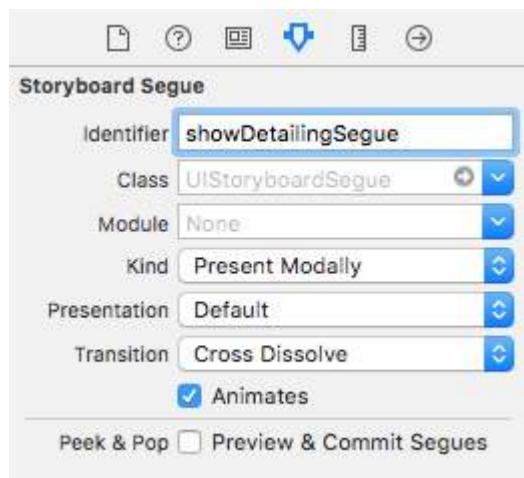
```

请注意，protocol 位于视图控制器类之外。

## 第63.2节：使用Segue（向前传递数据）

要使用segue从当前视图控制器向下一个新视图控制器（而非之前的视图控制器）传递数据，首先在相关的故事板中创建一个带有标识符的segue。重写当前视图控制器的prepareForSegue方法。在该方法内通过标识符检查刚创建的segue。将目标视图控制器进行类型转换，并通过设置向下转换后的视图控制器的属性来传递数据。

为segue设置标识符：



segue可以通过编程方式执行，也可以通过在故事板中按住ctrl拖动到目标视图控制器设置按钮动作事件来执行。你也可以在视图控制器中根据需要使用segue标识符以编程方式调用segue：

### Objective-C

```

- (void)showDetail {
    [self executeSegue, 标识符为:@"showDetailingSegue", 发送者为self];
}

```

### Swift

```

func showDetail() {
    self.performSegue(withIdentifier: "showDetailingSegue", sender: self)
}

```

你可以在重写的prepareForSegue方法中配置segue的负载。在目标视图控制器加载之前，可以设置所需的属性。

### Objective-C

```

- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    if([segue.identifier isEqualToString:@"showDetailingSegue"]){

```

```

    [super viewDidLoad];
}

- (IBAction) sendTextBackButton:(id)sender{
    [_delegate userDidEnterInformation:textField.text];
}
@end

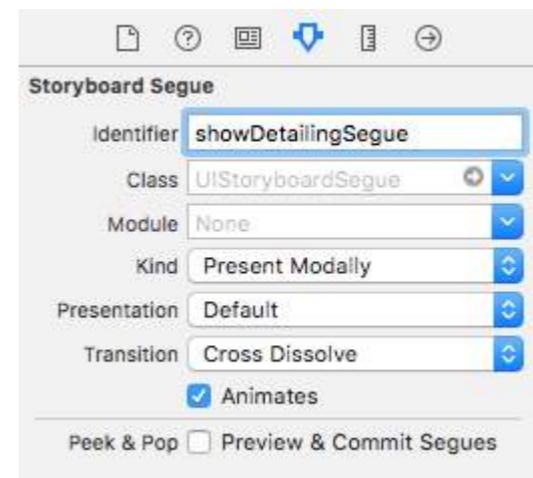
```

Note that the protocol is outside of the View Controller class.

## Section 63.2: Using Segues (passing data forward)

To pass data from the current view controller to the next new view controller (not a previous view controller) using segues, first create a segue with an identifier in the relevant storyboard. Override your current view controller's prepareForSegue method. Inside the method check for the segue you just created by its identifier. Cast the destination view controller and pass data to it by setting properties on the downcast view controller.

Setting an identifier for a segue:



Segues can be performed programmatically or using button action event set in the storyboard by ctrl+drag to destination view controller. You can call for a segue programmatically, when needed, using segue identifier in the view controller:

### Objective-C

```

- (void)showDetail {
    [self performSegueWithIdentifier:@"showDetailingSegue" sender:self];
}

```

### Swift

```

func showDetail() {
    self.performSegue(withIdentifier: "showDetailingSegue", sender: self)
}

```

You can configure segue payload in the overridden version of prepareForSegue method. You can set required properties before destination view controller is loaded.

### Objective-C

```

- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    if([segue.identifier isEqualToString:@"showDetailingSegue"]){

```

```
DetailViewController *controller = (DetailViewController *)segue.destinationViewController;
    controller.isDetailingEnabled = YES;
}
}
```

## Swift

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if segue.identifier == "showDetailingSegue" {
        let controller = segue.destinationViewController as! DetailViewController
        controller.isDetailingEnabled = true
    }
}
```

DetailViewController 是第二个视图控制器的名称，isDetailingEnabled 是该视图控制器中的一个公共变量。

进一步扩展这个模式，你可以将 DetailViewController 上的公共方法视为伪初始化器，用于帮助初始化任何必需的变量。这将自我说明需要在 DetailViewController 上设置的变量，而无需阅读其源代码。这也是设置默认值的一个方便位置。

## Objective-C

```
- (void)initVC:(BOOL *)isDetailingEnabled {
    self.isDetailingEnabled = isDetailingEnabled
}
```

## Swift

```
func initVC(isDetailingEnabled: Bool) {
    self.isDetailingEnabled = isDetailingEnabled
}
```

## 第63.3节：使用unwind传递数据回传

与允许你将数据“向前”从当前视图控制器传递到目标视图控制器的segue相反：

(VC1) -> (VC2)

使用“unwind”你可以做相反的操作，将数据从目标或当前视图控制器传递到其呈现的视图控制器：

(VC1) <- (VC2)

**注意：**请注意，使用unwind允许你先传递数据，随后当前视图控制器 (VC2) 将被释放。

操作方法如下：

首先，你需要在呈现视图控制器 (VC1) 中添加以下声明，VC1是我们想要传递数据的视图控制器：

```
@IBAction func unwindToPresentingViewController(segue: UIStoryboardSegue)
```

重要的是使用前缀unwind，这会“告知”Xcode这是一个unwind方法，使你可以在故事板中使用它。

```
DetailViewController *controller = (DetailViewController *)segue.destinationViewController;
    controller.isDetailingEnabled = YES;
}
}
```

## Swift

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if segue.identifier == "showDetailingSegue" {
        let controller = segue.destinationViewController as! DetailViewController
        controller.isDetailingEnabled = true
    }
}
```

DetailViewController 是名称的第二个视图控制器 and isDetailingEnabled 是该视图控制器中的一个公共变量。

To expand on this pattern, you can treat a public method on DetailViewController as a pseudo initializer, to help initialize any required variables. This will self document variables that need to be set on DetailViewController without having to read through its source code. It's also a handy place to put defaults.

## Objective-C

```
- (void)initVC:(BOOL *)isDetailingEnabled {
    self.isDetailingEnabled = isDetailingEnabled
}
```

## Swift

```
func initVC(isDetailingEnabled: Bool) {
    self.isDetailingEnabled = isDetailingEnabled
}
```

## Section 63.3: Passing data backwards using unwind to segue

In contrast to segue that lets you pass data "forward" from current view controller to destination view controller:

(VC1) -> (VC2)

Using "unwind" you can do the opposite, pass data from the destination or current view controller to its presenting view controller:

(VC1) <- (VC2)

**NOTE:** Pay attention that using unwind lets you pass the data first and afterwards the current view controller (VC2) will get deallocated.

Here's how to do it:

First, you will need to add the following declaration at the presenting view controller (VC1) which is the view controller that we want to pass the data to:

```
@IBAction func unwindToPresentingViewController(segue: UIStoryboardSegue)
```

The important thing is to use the prefix unwind, this "informs" Xcode that this is an unwind method giving you the option to use it in storyboard as well.

之后你需要实现该方法，它看起来几乎和实际的segue一样：

```
@IBAction func unwindToPresentingViewController(segue:UIStoryboardSegue)
{
    if segue.identifier == "YourCustomIdentifier"
    {
        if let VC2 = segue.sourceViewController as? VC2
        {
            // 在这里写你的自定义代码以访问VC2类成员
        }
    }
}
```

现在你有两种方式来调用unwind：

1. 你可以“硬编码”调用：self.performSegueWithIdentifier("YourCustomIdentifier", sender: self)，这将在你调用performSegueWithIdentifier时执行unwind。
2. 你也可以通过storyboard将unwind方法链接到你的“Exit”对象：按住ctrl拖动你想要的按钮要调用“Exit”对象的unwind方法：



释放后，您将有选项选择自定义的unwind方法：



## 第63.4节：使用闭包传递数据（传回数据）

您可以不用使用代理模式，该模式将实现分散在UIViewController类的各个部分，  
也可以使用闭包来传递数据。假设您使用的是UIStoryboardSegue，  
在prepareForSegue方法中，您可以一步轻松设置新的控制器

```
final class DestinationViewController: UIViewController {
    var onCompletion: ((success: Bool) -> ())?

    @IBAction func someButtonTapped(sender: AnyObject?) {
        onCompletion?(success: true)
    }
}

final class MyViewController: UIViewController {
    override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
        guard let destinationController = segue.destinationViewController as?
            DestinationViewController else { return }
    }
}
```

Afterwards you will need to implement the method, it looks almost the same as an actual segue:

```
@IBAction func unwindToPresentingViewController(segue:UIStoryboardSegue)
{
    if segue.identifier == "YourCustomIdentifier"
    {
        if let VC2 = segue.sourceViewController as? VC2
        {
            // Your custom code in here to access VC2 class member
        }
    }
}
```

Now you have 2 options to invoke the unwind calls:

1. You can "hard code" invoke the: `self.performSegueWithIdentifier("YourCustomIdentifier", sender: self)` which will do the unwind for you whenever you will `performSegueWithIdentifier`.
2. You can link the unwind method using the storyboard to your "Exit" object: `ctrl + drag` the button you want to invoke the unwind method, to the "Exit" object:



Release and you will have the option to choose your custom unwind method:



## Section 63.4: Passing data using closures (passing data back)

Instead of using the **delegate pattern**, that split the implementation in various part of the `UIViewController` class, you can even use closures to pass data back and forward. By assuming that you're using the `UIStoryboardSegue`, in the `prepareForSegue` method you can easily setup the new controller in one step

```
final class DestinationViewController: UIViewController {
    var onCompletion: ((success: Bool) -> ())?

    @IBAction func someButtonTapped(sender: AnyObject?) {
        onCompletion?(success: true)
    }
}

final class MyViewController: UIViewController {
    override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
        guard let destinationController = segue.destinationViewController as?
            DestinationViewController else { return }
    }
}
```

```

destinationController.onCompletion = { success in
    // 当调用 `someButtonTapped(_:)` 时将执行此代码
    print(success)
}
}

```

这是一个使用示例，建议在Swift中使用，Objective-C的block语法不太容易使代码更易读

## 第63.5节：使用回调闭包（block）传递数据回调

这个话题是iOS开发中的经典问题，其解决方案多种多样，正如其他示例已经展示的那样。在这个示例中，我将展示另一个日常常用的方法：通过适配本页上的代理模式示例，将数据传递改为使用闭包回调！

这种方法优于代理模式的一点是，不需要将设置代码分散在两个不同的地方（  
看看本页的代理示例，`prepareForSegue`, `userDidEnterInformation`），而是将它们  
集中在一起（只在`prepareForSegue`中，我会展示）。

### 从第二个视图控制器开始

我们必须弄清楚如何使用回调，然后才能编写它，这就是为什么我们从第二个视图控制器开始，因为那里是我们使用回调的地方：当我们获得新的文本输入时，我们调用回调，**使用回调的参数作为将数据传回第一个视图控制器的媒介**，注意我说的是使用回调的参数，这一点非常重要，初学者（就像我当初一样）总是忽视这一点，不知道从哪里开始正确编写回调闭包。

所以在这种情况下，我们知道回调只接受一个参数：文本，类型是`String`，先声明它并作为属性，因为我们需要从第一个视图控制器赋值

我只是注释掉了所有代理部分并保留它们以便比较

```

class SecondViewController: UIViewController {

    //weak var delegate: DataEnteredDelegate? = nil
    var callback: ((String?) -> ())?

    @IBOutlet weak var textField: UITextField!

    @IBAction func sendTextBackButton(sender: AnyObject) {

        //delegate?.userDidEnterInformation(textField.text!)
        callback?(input.text)

        self.navigationController?.popViewControllerAnimated(true)
    }
}

```

### 完成第一个视图控制器

你所要做的就是传递回调闭包，就完成了，闭包会为我们完成后续工作，因为我们已经在第二个视图控制器中设置好了

看看它是如何让我们的代码相比委托模式更简洁的

```

//不再需要 DataEnteredDelegate
class FirstViewController: UIViewController {

```

```

destinationController.onCompletion = { success in
    // this will be executed when `someButtonTapped(_:)` will be called
    print(success)
}
}

```

This is an example of use and it's better to use on Swift, Objective-C block's syntax is not so easy to make the code more readable

## Section 63.5: Using callback closure(block) passing data back

this topic is a classical issue in iOS development, and its solution is various as other example already shown. In this example I'll show another daily common use one: passing data using closure by adapting delegate pattern example on this page into callback closure!

one thing this method is superior to delegate pattern is instead of split the setting up code in two different place( look at delegate example on this page, `prepareForSegue`, `userDidEnterInformation` ) rather gathering them together( only in `prepareForSegue`, I'll show it )

### Start from Second View Controller

we must figure out how to use callback, then can we write it, this is why we start from second view controller since it's where we use callback: when we got the new text input, we call our callback, **using callback's parameter as a medium to passing data back to first ViewController**, notice that I said using callback's parameter, this is very important, novices(as I was) always overlook this and don't know where to start to write callback closure properly

so in this case, we know that our callback only take one parameter: text and its type is `String`, let's declare it and make it property since we need populate from our first view controller

I just comment all the delegate part and keep it for comparing

```

class SecondViewController: UIViewController {

    //weak var delegate: DataEnteredDelegate? = nil
    var callback: ((String?) -> ())?

    @IBOutlet weak var textField: UITextField!

    @IBAction func sendTextBackButton(sender: AnyObject) {

        //delegate?.userDidEnterInformation(textField.text!)
        callback?(input.text)

        self.navigationController?.popViewControllerAnimated(true)
    }
}

```

### Finish first view controller

all you have to do is passing callback closure, and we are done, closure will do the future work for us since we already set it up in second view controller

look how it make our code shorter compared to the delegate pattern

```

//no more DataEnteredDelegate
class FirstViewController: UIViewController {

```

```

@IBOutlet weak var label: UILabel!

override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "showSecondViewController" {
        let secondViewController = segue.destinationViewController as! SecondViewController
        //secondViewController.delegate = self
    }
}

// 我们自定义的 DataEnteredDelegate 协议的必需方法
//func userDidEnterInformation(info: String) {
//    label.text = info
//}

}

```

最后，也许你们中有人会困惑，为什么我们只单向传递数据（这里是闭包），从第一个视图控制器传到第二个视图控制器，而没有直接从第二个视图控制器传回，这怎么能算作一种通信工具呢？也许你真的应该运行它并自己验证，我只想说，这是参数，是回调闭包的参数，用来传回数据的！

## 第63.6节：通过赋值属性（向前传递数据）

你可以在推入或展示下一个视图控制器之前，直接通过赋值下一个视图控制器的属性来传递数据。

```

class FirstViewController: UIViewController {

    func openSecondViewController() {

        // 这里我们初始化 SecondViewController 并将 id 属性设置为 492
        let secondViewController = SecondViewController()
        secondViewController.id = 492

        // 一旦赋值完成，我们现在推送或展示视图控制器
        present(secondViewController, animated: true, completion: nil)
    }

}

class SecondViewController: UIViewController {

    var id: Int?

    override func viewDidLoad() {
        super.viewDidLoad()

        // 这里我们解包了 id，并将从上一个视图控制器获取数据。
        if let id = id {
            print("Id 已设置: \(id)")
        }
    }

}

```

```

@IBOutlet weak var label: UILabel!

override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "showSecondViewController" {
        let secondViewController = segue.destinationViewController as! SecondViewController
        //secondViewController.delegate = self
        secondViewController.callback = { text in self.label.text = text }
    }
}

// required method of our custom DataEnteredDelegate protocol
//func userDidEnterInformation(info: String) {
//    label.text = info
//}

}

```

and in the last, maybe someone of you will confused by the looking that we only passing the data(closure in this case) only in one way, from first view controller to second, no directly coming back from second view controller, how can we consider it as a communicating tool? maybe you really should run it and prove it yourself, all I will say it's **parameter**, it's **callback closure's parameter** that passing data back!

## Section 63.6: By assigning property (Passing data forward)

You can pass data directly by assigning the property of the next view controller before you push or present it.

```

class FirstViewController: UIViewController {

    func openSecondViewController() {

        // Here we initialize SecondViewController and set the id property to 492
        let secondViewController = SecondViewController()
        secondViewController.id = 492

        // Once it was assign we now push or present the view controller
        present(secondViewController, animated: true, completion: nil)
    }

}

class SecondViewController: UIViewController {

    var id: Int?

    override func viewDidLoad() {
        super.viewDidLoad()

        // Here we unwrapped the id and will get the data from the previous view controller.
        if let id = id {
            print("Id was set: \(id)")
        }
    }

}

```

# 第64章：管理键盘

## 第64.1节：创建自定义应用内键盘



这是一个基础的应用内键盘。相同的方法可以用来制作几乎任何键盘布局。主要需要完成的事项如下：

- 在 .xib 文件中创建键盘布局，该文件的所有者是一个继承自 `UIView` 的 Swift 或 Objective-C 类。
- 告诉 `UITextField` 使用自定义键盘。
- 使用代理在键盘和主视图控制器之间进行通信。

### 创建 .xib 键盘布局文件

- 在 Xcode 中，依次选择 **File > New > File... > iOS > User Interface > View** 来创建 .xib 文件。
- 我将文件命名为 `Keyboard.xib`
- 添加所需的按钮。
- 使用自动布局约束，这样无论键盘大小如何，按钮都会相应调整大小。
- 将文件所有者 (File's Owner) (而非根视图) 设置为 `Keyboard` 类。这是一个常见错误来源。你将在下一步创建此类。详见末尾注释。

### 创建 .swift UIView 子类键盘文件

- 在 Xcode 中，依次选择 **File > New > File... > iOS > Source > Cocoa Touch Class** 来创建 Swift 或 Objective-C 类。为新建类选择 `UIView` 作为父类。
- 我称我的为 `Keyboard.swift` (Objective-C 中的 `Keyboard` 类)

# Chapter 64: Managing the Keyboard

## Section 64.1: Create a custom in-app keyboard



This is a basic in-app keyboard. The same method could be used to make just about any keyboard layout. Here are the main things that need to be done:

- Create the keyboard layout in an .xib file, whose owner is a Swift or Objective-C class that is a `UIView` subclass.
- Tell the `UITextField` to use the custom keyboard.
- Use a delegate to communicate between the keyboard and the main view controller.

### Create the .xib keyboard layout file

- In Xcode go to **File > New > File... > iOS > User Interface > View** to create the .xib file.
- I called mine `Keyboard.xib`
- Add the buttons that you need.
- Use auto layout constraints so that no matter what size the keyboard is, the buttons will resize accordingly.
- Set the File's Owner (not the root view) to be the `Keyboard` class. This is a common source of error. You'll create this class in the next step. See the note at the end.

### Create the .swift UIView subclass keyboard file

- In Xcode go to **File > New > File... > iOS > Source > Cocoa Touch Class** to create the Swift or Objective-C class. Choose `UIView` as a superclass for newly created class
- I called mine `Keyboard.swift` (Keyboard class in Objective-C)

- 为 Swift 添加以下代码：

```

import UIKit

// 视图控制器将采用此协议（代理）
// 因此必须包含 keyWasTapped 方法
@protocol KeyboardDelegate: class {
    func keyWasTapped(character: String)
}

class Keyboard: UIView {

    // 该变量将被设置为视图控制器,
    // 以便键盘可以向视图控制器发送消息。
    weak var delegate: KeyboardDelegate?

    // MARK: 键盘初始化

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
    }

    initializeSubviews()
}

override init(frame: CGRect) {
    super.init(frame: frame)
    initializeSubviews()
}

func initializeSubviews() {
    let xibFileName = "Keyboard" // 不包含 xib 扩展名
    let view = NSBundle.mainBundle().loadNibNamed(xibFileName, owner: self, options: nil)[0] as! UIView
    self.addSubview(view)
    view.frame = self.bounds
}

// MARK: 来自 .xib 文件的按钮操作

@IBAction func keyTapped(sender: UIButton) {
    // 当按钮被点击时, 将该信息发送给
    // 代理 (即视图控制器)
    self.delegate?.keyWasTapped(sender.titleLabel!.text!) // 也可以选择发送一个
tag 值
}

```

- 为 Objective-C 添加以下代码：

#### Keyboard.h 文件

```

#import <UIKit/UIKit.h>

// 视图控制器将采用此协议（代理）
// 因此必须包含 keyWasTapped 方法
@protocol KeyboardDelegate<NSObject>
- (void)keyWasTapped:(NSString *)character;
@end

@interface Keyboard : UIView

```

- Add the following code for Swift:

```

import UIKit

// The view controller will adopt this protocol (delegate)
// and thus must contain the keyWasTapped method
@protocol KeyboardDelegate: class {
    func keyWasTapped(character: String)
}

class Keyboard: UIView {

    // This variable will be set as the view controller so that
    // the keyboard can send messages to the view controller.
    weak var delegate: KeyboardDelegate?

    // MARK: keyboard initialization

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        initializeSubviews()
    }

    override init(frame: CGRect) {
        super.init(frame: frame)
        initializeSubviews()
    }

    func initializeSubviews() {
        let xibFileName = "Keyboard" // xib extention not included
        let view = NSBundle.mainBundle().loadNibNamed(xibFileName, owner: self, options: nil)[0] as! UIView
        self.addSubview(view)
        view.frame = self.bounds
    }

    // MARK: Button actions from .xib file

    @IBAction func keyTapped(sender: UIButton) {
        // When a button is tapped, send that information to the
        // delegate (ie, the view controller)
        self.delegate?.keyWasTapped(sender.titleLabel!.text!) // could alternatively send a
tag value
    }
}

```

- Add the following code for Objective-C:

#### Keyboard.h File

```

#import <UIKit/UIKit.h>

// The view controller will adopt this protocol (delegate)
// and thus must contain the keyWasTapped method
@protocol KeyboardDelegate<NSObject>
- (void)keyWasTapped:(NSString *)character;
@end

@interface Keyboard : UIView

```

```
@property (nonatomic, weak) id<KeyboardDelegate> delegate;
@end
```

## Keyboard.m 文件

```
#import "Keyboard.h"

@implementation Keyboard

- (id)initWithCoder:(NSCoder *)aDecoder {
    self = [super initWithCoder:aDecoder];
    [self initializeSubviews];
    return self;
}

- (id)initWithFrame:(CGRect)frame {
    self = [super initWithFrame:frame];
    [self initializeSubviews];
    return self;
}

- (void)initializeSubviews {
    NSString *xibName = @"Keyboard"; // 不包含 xib 扩展名
    UIView *view = [[[NSBundle mainBundle] loadNibNamed:xibName owner:self options:nil] firstObject];
    [self addSubview:view];
    view.frame = self.bounds;
}

// MARK: 来自 .xib 文件的按钮操作

-(IBAction)keyTapped:(UIButton *)sender {
    // 当按钮被点击时，将该信息发送给
    // 代理（即视图控制器）
    [self.delegate keyWasTapped:sender.titleLabel.text]; // 也可以选择发送标签值
}

@end
```

- 从按钮拖动控件操作到.xib文件中的按钮回调，连接到Swift或Objective-C所有者中的@IBAction方法，以便全部连接起来。
- 注意协议和代理代码。参见[this answer](#)，了解关于代理工作原理的简单解释。

## 设置视图控制器

- 在主故事板中添加一个UITextField，并通过IBOutlet连接到视图控制器。命名为textField。
- 视图控制器的Swift代码如下：

```
import UIKit

class ViewController: UIViewController, KeyboardDelegate {

    @IBOutlet weak var textField: UITextField!

    override func viewDidLoad() {
```

```
@property (nonatomic, weak) id<KeyboardDelegate> delegate;
@end
```

## Keyboard.m File

```
#import "Keyboard.h"

@implementation Keyboard

- (id)initWithCoder:(NSCoder *)aDecoder {
    self = [super initWithCoder:aDecoder];
    [self initializeSubviews];
    return self;
}

- (id)initWithFrame:(CGRect)frame {
    self = [super initWithFrame:frame];
    [self initializeSubviews];
    return self;
}

- (void)initializeSubviews {
    NSString *xibName = @"Keyboard"; // xib extention not included
    UIView *view = [[[NSBundle mainBundle] loadNibNamed:xibName owner:self options:nil] firstObject];
    [self addSubview:view];
    view.frame = self.bounds;
}

// MARK: Button actions from .xib file

-(IBAction)keyTapped:(UIButton *)sender {
    // When a button is tapped, send that information to the
    // delegate (ie, the view controller)
    [self.delegate keyWasTapped:sender.titleLabel.text]; // could alternatively send a tag
    value
}

@end
```

- Control drag actions from the buttons to button callback in the .xib file to the `@IBAction` method in the Swift or Objective-C owner to hook them all up.
- Note that the protocol and delegate code. See [this answer](#) for a simple explanation about how delegates work.

## Set up the View Controller

- Add a `UITextField` to your main storyboard and connect it to your view controller with an `IBOutlet`. Call it `textField`.
- Use the following code for the View Controller in Swift:

```
import UIKit

class ViewController: UIViewController, KeyboardDelegate {

    @IBOutlet weak var textField: UITextField!

    override func viewDidLoad() {
```

```

super.viewDidLoad()

// 初始化自定义键盘
let keyboardView = Keyboard(frame: CGRect(x: 0, y: 0, width: 0, height: 300))
keyboardView.delegate = self // 视图控制器将在每次按键被点击时收到键盘的通知

// 用自定义键盘替换系统键盘
textField.inputView = keyboardView
}

// 键盘代理协议的必需方法
func keyWasTapped(character: String) {
textField.insertText(character)
}
}

```

- 请使用以下代码（Objective-C）：

#### .h 文件

```

#import <UIKit/UIKit.h>

@interface ViewController : UIViewController

@end

```

#### .m 文件

```

#import "ViewController.h"
#import "Keyboard.h"

@interface ViewController ()<KeyboardDelegate>

@property (nonatomic, weak) IBOutlet UITextField *textField;

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // 在视图加载后执行任何额外的设置，通常来自nib文件。

    // 初始化自定义键盘
    Keyboard *keyboardView = [[Keyboard alloc] initWithFrame:CGRectMake(0, 0, 0, 300)];    keyboardView.d
elegate = self; // 视图控制器将在每次按键被点击时收到键盘的通知

    // 用自定义键盘替换系统键盘
    self.textField.inputView = keyboardView;
}

- (void)keyWasTapped:(NSString *)character {
    [self.textField insertText:character];
}

@end

```

```
super.viewDidLoad()
```

```

// initialize custom keyboard
let keyboardView = Keyboard(frame: CGRect(x: 0, y: 0, width: 0, height: 300))
keyboardView.delegate = self // the view controller will be notified by the
keyboard whenever a key is tapped

// replace system keyboard with custom keyboard
textField.inputView = keyboardView
}

// required method for keyboard delegate protocol
func keyWasTapped(character: String) {
    textField.insertText(character)
}
}

```

- Use the following code for Objective-C:

#### .h File

```

#import <UIKit/UIKit.h>

@interface ViewController : UIViewController

@end

```

#### .m File

```

#import "ViewController.h"
#import "Keyboard.h"

@interface ViewController ()<KeyboardDelegate>

@property (nonatomic, weak) IBOutlet UITextField *textField;

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.

    // initialize custom keyboard
    Keyboard *keyboardView = [[Keyboard alloc] initWithFrame:CGRectMake(0, 0, 0, 300)];
    keyboardView.delegate = self; // the view controller will be notified by the keyboard
    whenever a key is tapped

    // replace system keyboard with custom keyboard
    self.textField.inputView = keyboardView;
}

- (void)keyWasTapped:(NSString *)character {
    [self.textField insertText:character];
}

@end

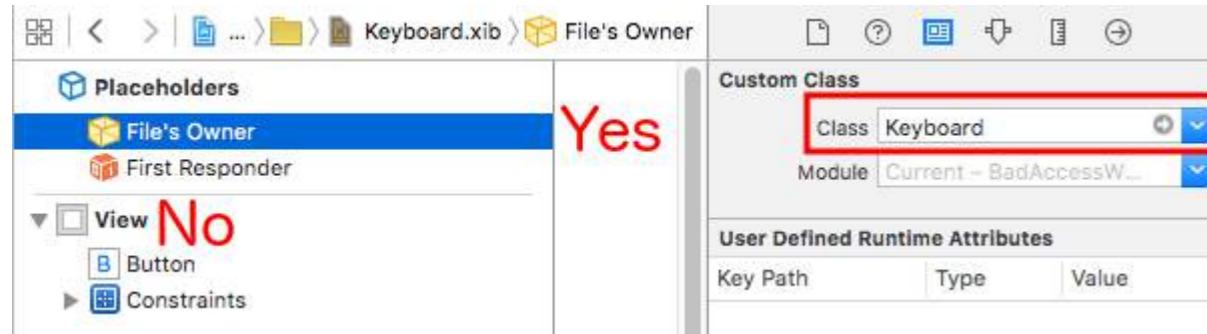
```

- 注意视图控制器采用了我们上面定义的KeyboardDelegate协议。

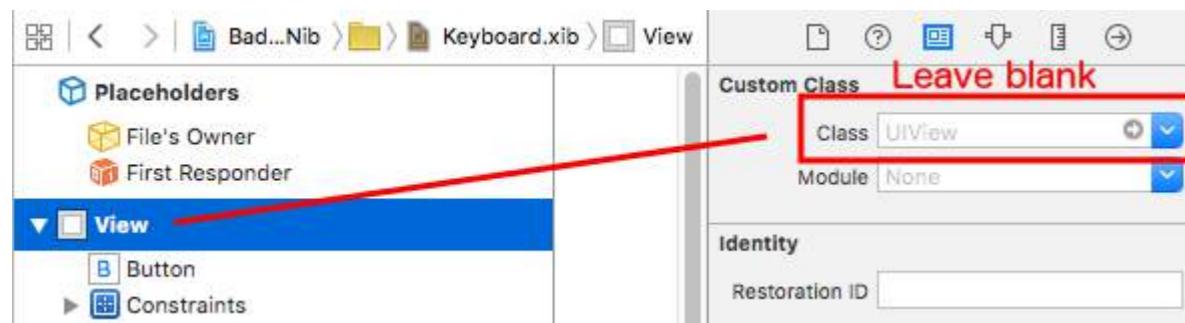
## 常见错误

如果出现EXC\_BAD\_ACCESS错误，可能是因为你将视图的自定义类设置为Keyboard而不是为nib文件的所有者设置。

选择Keyboard.nib，然后选择文件所有者（File's Owner）。



确保根视图的自定义类为空。



## 注意事项

此示例最初来自这个Stack Overflow回答。

## 第64.2节：通过点击视图来隐藏键盘

如果你想通过点击键盘外部来隐藏键盘，可以使用这个小技巧（仅适用于Objective-C）：

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

    // 点击文本框外部时收起键盘
    UITapGestureRecognizer *tapGestureRecognizer = [[UITapGestureRecognizer alloc]
initWithTarget:self.view action:@selector(endEditing:)];
    [tapGestureRecognizer setCancelsTouchesInView:NO];
    [self.view addGestureRecognizer:tapGestureRecognizer];
}
```

对于 Swift 来说，代码会稍微多一点：

```
override func viewDidLoad() {
    super.viewDidLoad()

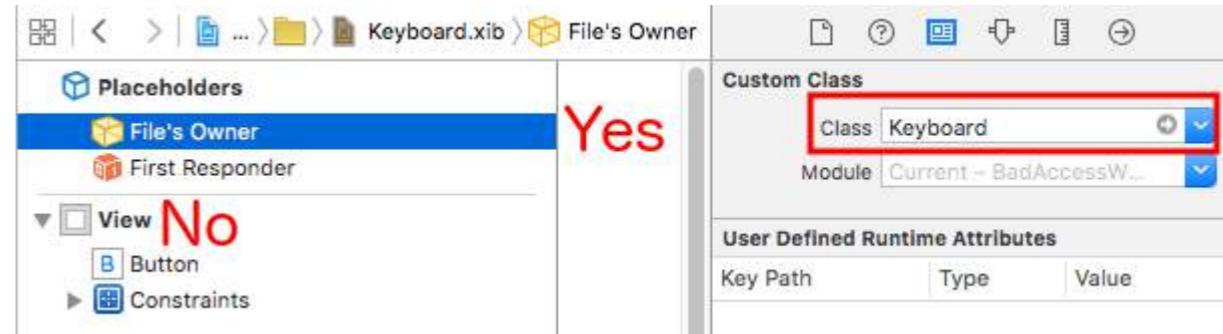
    // 点击文本框外部时收起键盘
    let tapGestureRecognizer: UITapGestureRecognizer = UITapGestureRecognizer(target: self, action:
#selector(YourVCName.dismissKeyboard))
    view.addGestureRecognizer(tapGestureRecognizer)
```

- Note that the view controller adopts the KeyboardDelegate protocol that we defined above.

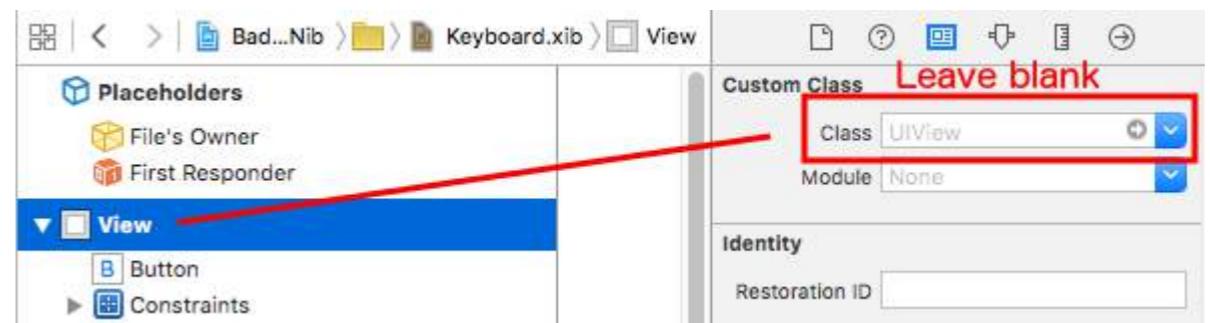
## Common error

If you are getting an EXC\_BAD\_ACCESS error, it is probably because you set the view's custom class as Keyboard rather than do this for the nib File's Owner.

Select Keyboard.nib and then choose File's Owner.



Make sure that the custom class for the root view is blank.



## Notes

This example comes originally from [this Stack Overflow answer](#).

## Section 64.2: Dismiss a keyboard with tap on view

If you want to hide a keyboard by tap outside of it, it's possible to use this hacky trick (works only with Objective-C):

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

    // dismiss keyboard when tap outside a text field
    UITapGestureRecognizer *tapGestureRecognizer = [[UITapGestureRecognizer alloc]
initWithTarget:self.view action:@selector(endEditing:)];
    [tapGestureRecognizer setCancelsTouchesInView:NO];
    [self.view addGestureRecognizer:tapGestureRecognizer];
}
```

for Swift there will be a bit more code:

```
override func viewDidLoad() {
    super.viewDidLoad()

    // dismiss keyboard when tap outside a text field
    let tapGestureRecognizer: UITapGestureRecognizer = UITapGestureRecognizer(target: self, action:
#selector(YourVCName.dismissKeyboard))
    view.addGestureRecognizer(tapGestureRecognizer)
```

```

}

// 当识别到点击时调用此函数。
func dismissKeyboard() {
    // 使视图（或其嵌入的某个文本框）放弃第一响应者状态。
    view.endEditing(true)
}

```

另一个 Swift 3/iOS 10 示例

```

class vc: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        // 在视图加载后执行任何额外的设置，通常来自nib文件。

        txtSomeField.delegate = self
    }

    extension vc: UITextFieldDelegate {
        //当用户在键盘外点击时，隐藏任何文本框的键盘。
        override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
            self.view.endEditing(true) //隐藏键盘
        }
    }
}

```

## 第64.3节：使用单例+代理管理键盘

刚开始管理键盘时，我会在每个视图控制器中使用单独的通知。

通知方法（使用NSNotification）：

```

class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        NSNotificationCenter.defaultCenter().addObserver(self, selector:
#selector(ViewController.keyboardNotification(_:)), name: UIKeyboardWillChangeFrameNotification,
object: nil)
    }

    func keyboardNotification(notification:NSNotification) {
        guard let userInfo = notification.userInfo else { return }

        let endFrame = (userInfo[UIKeyboardFrameEndUserInfoKey] as? NSValue)?.CGRectValue()
        let duration: NSTimeInterval = (userInfo[UIKeyboardAnimationDurationUserInfoKey] as?
NSNumber)?.doubleValue ?? 0
        let animationCurveRawNSN = userInfo[UIKeyboardAnimationCurveUserInfoKey] as? NSNumber
        let animationCurveRaw = animationCurveRawNSN?.unsignedLongValue ??
        UIViewAnimationOptions.CurveEaseOut.rawValue
        let animationCurve: UIViewAnimationOptions = UIViewAnimationOptions(rawValue:
animationCurveRaw)

        if endFrame?.origin.y >= UIScreen.mainScreen().bounds.size.height {
            lowerViewBottomConstraint.constant = 0
        } else {
            lowerViewBottomConstraint.constant = endFrame?.size.height ?? 0.0
        }
        view.animateConstraintWithDuration(duration, delay: NSTimeInterval(0), options:

```

```

}

//Calls this function when the tap is recognized.
func dismissKeyboard() {
    //Causes the view (or one of its embedded text fields) to resign the first responder status.
    view.endEditing(true)
}

```

Another Swift 3/iOS 10 example

```

class vc: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.

        txtSomeField.delegate = self
    }

    extension vc: UITextFieldDelegate {
        //Hide the keyboard for any text field when the UI is touched outside of the keyboard.
        override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
            self.view.endEditing(true) //Hide the keyboard
        }
    }
}

```

## Section 64.3: Managing the Keyboard Using a Singleton + Delegate

When I first started managing the keyboard I would use separate Notifications in each ViewController.

Notification Method (Using NSNotification):

```

class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        NSNotificationCenter.defaultCenter().addObserver(self, selector:
#selector(ViewController.keyboardNotification(_:)), name: UIKeyboardWillChangeFrameNotification,
object: nil)
    }

    func keyboardNotification(notification:NSNotification) {
        guard let userInfo = notification.userInfo else { return }

        let endFrame = (userInfo[UIKeyboardFrameEndUserInfoKey] as? NSValue)?.CGRectValue()
        let duration: NSTimeInterval = (userInfo[UIKeyboardAnimationDurationUserInfoKey] as?
NSNumber)?.doubleValue ?? 0
        let animationCurveRawNSN = userInfo[UIKeyboardAnimationCurveUserInfoKey] as? NSNumber
        let animationCurveRaw = animationCurveRawNSN?.unsignedLongValue ??
        UIViewAnimationOptions.CurveEaseOut.rawValue
        let animationCurve: UIViewAnimationOptions = UIViewAnimationOptions(rawValue:
animationCurveRaw)

        if endFrame?.origin.y >= UIScreen.mainScreen().bounds.size.height {
            lowerViewBottomConstraint.constant = 0
        } else {
            lowerViewBottomConstraint.constant = endFrame?.size.height ?? 0.0
        }
        view.animateConstraintWithDuration(duration, delay: NSTimeInterval(0), options:

```

```

animationCurve, 完成 : nil)
}
}

```

我的问题是，我自己为每个视图控制器都反复编写这段代码。经过一些尝试，我发现使用单例+代理模式让我能够重用大量代码，并将所有键盘管理集中组织在一个地方！

单例+代理方法：

```

protocol KeyboardManagerDelegate: class {
    func keyboardWillChangeFrame(endFrame: CGRect?, duration: TimeInterval, animationCurve:
    UIViewAnimationOptions)
}

class KeyboardManager {

    weak var delegate: KeyboardManagerDelegate?

    class var sharedInstance: KeyboardManager {
        struct Singleton {
            static let instance = KeyboardManager()
        }
        return Singleton.instance
    }

    init() {
        NSNotificationCenter.defaultCenter().addObserver(self, selector:
#selector(KeyboardManager.keyboardWillChangeFrameNotification(_:)), name:
UIKeyboardWillChangeFrameNotification, object: nil)
    }

    @objc func keyboardWillChangeFrameNotification(notification: NSNotification) {
        guard let userInfo = notification.userInfo else { return }

        let endFrame = (userInfo[UIKeyboardFrameEndUserInfoKey] as? NSValue)?.CGRectValue()
        let duration: TimeInterval = (userInfo[UIKeyboardAnimationDurationUserInfoKey] as?
NSNumber)?.doubleValue ?? 0
        let animationCurveRawNSN = userInfo[UIKeyboardAnimationCurveUserInfoKey] as? NSNumber
        let animationCurveRaw = animationCurveRawNSN?.unsignedLongValue ??
UIViewAnimationOptions.CurveEaseOut.rawValue
        let animationCurve: UIViewAnimationOptions = UIViewAnimationOptions(rawValue:
animationCurveRaw)

        delegate?.keyboardWillChangeFrame(endFrame, duration: duration, animationCurve:
animationCurve)
    }
}

```

现在，当我想从一个视图控制器管理键盘时，我所需要做的就是将代理设置为该视图控制器并实现任何代理方法。

```

class ViewController: UIViewController {
    override func viewDidAppear(animated: Bool) {
        super.viewDidAppear(animated)
        KeyboardManager.sharedInstance.delegate = self
    }

    // MARK: - Keyboard Manager
}

```

```

animationCurve, completion: nil)
}
}

```

My problem was that I found myself writing this code again and again for every single ViewController. After experimenting a bit I found using a Singleton + Delegate pattern allowed me to reuse a bunch of code and organize all of the Keyboard Management in a single place!

Singleton + Delegate Method:

```

protocol KeyboardManagerDelegate: class {
    func keyboardWillChangeFrame(endFrame: CGRect?, duration: TimeInterval, animationCurve:
    UIViewAnimationOptions)
}

class KeyboardManager {

    weak var delegate: KeyboardManagerDelegate?

    class var sharedInstance: KeyboardManager {
        struct Singleton {
            static let instance = KeyboardManager()
        }
        return Singleton.instance
    }

    init() {
        NSNotificationCenter.defaultCenter().addObserver(self, selector:
#selector(KeyboardManager.keyboardWillChangeFrameNotification(_:)), name:
UIKeyboardWillChangeFrameNotification, object: nil)
    }

    @objc func keyboardWillChangeFrameNotification(notification: NSNotification) {
        guard let userInfo = notification.userInfo else { return }

        let endFrame = (userInfo[UIKeyboardFrameEndUserInfoKey] as? NSValue)?.CGRectValue()
        let duration: TimeInterval = (userInfo[UIKeyboardAnimationDurationUserInfoKey] as?
NSNumber)?.doubleValue ?? 0
        let animationCurveRawNSN = userInfo[UIKeyboardAnimationCurveUserInfoKey] as? NSNumber
        let animationCurveRaw = animationCurveRawNSN?.unsignedLongValue ??
UIViewAnimationOptions.CurveEaseOut.rawValue
        let animationCurve: UIViewAnimationOptions = UIViewAnimationOptions(rawValue:
animationCurveRaw)

        delegate?.keyboardWillChangeFrame(endFrame, duration: duration, animationCurve:
animationCurve)
    }
}

```

Now when I want to manage the keyboard from a ViewController all I need to do is set the delegate to that ViewController and implement any delegate methods.

```

class ViewController: UIViewController {
    override func viewDidAppear(animated: Bool) {
        super.viewDidAppear(animated)
        KeyboardManager.sharedInstance.delegate = self
    }

    // MARK: - Keyboard Manager
}

```

```

extension ViewController: KeyboardManagerDelegate {
    func keyboardWillChangeFrame(endFrame: CGRect?, duration: NSTimeInterval, animationCurve: UIViewAnimationOptions) {
        if endFrame?.origin.y >= UIScreen.mainScreen().bounds.size.height {
            lowerViewBottomConstraint.constant = 0
        } else {
            lowerViewBottomConstraint.constant = (endFrame?.size.height ?? 0.0)
        }
        view.animateConstraintWithDuration(duration, delay: NSTimeInterval(0), options: animationCurve, completion: nil)
    }
}

```

此方法也非常可定制！假设我们想为UIKeyboardWillHideNotification添加功能。  
这和向我们的KeyboardManagerDelegate添加一个方法一样简单。

KeyboardManagerDelegate中关于UIKeyboardWillHideNotification的方法：

```

protocol KeyboardManagerDelegate: class {
    func keyboardWillChangeFrame(endFrame: CGRect?, duration: NSTimeInterval, animationCurve: UIViewAnimationOptions)
    func keyboardWillHide(notificationUserInfo: [NSObject: AnyObject])
}

class KeyboardManager {
    init() {
        NSNotificationCenter.defaultCenter().addObserver(self, selector: #selector(KeyboardManager.keyboardWillChangeFrameNotification(_:)), name: UIKeyboardWillChangeFrameNotification, object: nil)
        NSNotificationCenter.defaultCenter().addObserver(self, selector: #selector(KeyboardManager.keyboardWillHide(_:)), name: UIKeyboardWillHideNotification, object: nil)
    }

    func keyboardWillHide(notification:NSNotification) {
        guard let userInfo = notification.userInfo else { return }
        delegate?.keyboardWillHide(userInfo)
    }
}

```

假设我们只想在一个视图控制器中实现func keyboardWillHide(notificationUserInfo: [NSObject: AnyObject])方法。我们也可以将此方法设为可选。

**类型别名** KeyboardManagerDelegate = 协议<KeyboardManagerModel, KeyboardManagerConfigureable>

```

协议 KeyboardManagerModel: class {
    func keyboardWillChangeFrame(endFrame: CGRect?, duration: NSTimeInterval, animationCurve: UIViewAnimationOptions)
}

@objc 协议 KeyboardManagerConfigureable {
    可选方法 keyboardWillHide(userInfo: [NSObject: AnyObject])
}

```

\*注意这种模式有助于避免过度使用@objc。更多详情请参见<http://www.jessesquires.com/avoiding-objc-in-swift/> !

总之，我发现使用单例+代理来管理键盘比使用通知更高效且更易用。

```

extension ViewController: KeyboardManagerDelegate {
    func keyboardWillChangeFrame(endFrame: CGRect?, duration: NSTimeInterval, animationCurve: UIViewAnimationOptions) {
        if endFrame?.origin.y >= UIScreen.mainScreen().bounds.size.height {
            lowerViewBottomConstraint.constant = 0
        } else {
            lowerViewBottomConstraint.constant = (endFrame?.size.height ?? 0.0)
        }
        view.animateConstraintWithDuration(duration, delay: NSTimeInterval(0), options: animationCurve, completion: nil)
    }
}

```

This method is very customizable too! Say we want to add functionality for UIKeyboardWillHideNotification. This is as easy as adding a method to our KeyboardManagerDelegate.

KeyboardManagerDelegate with UIKeyboardWillHideNotification:

```

protocol KeyboardManagerDelegate: class {
    func keyboardWillChangeFrame(endFrame: CGRect?, duration: NSTimeInterval, animationCurve: UIViewAnimationOptions)
    func keyboardWillHide(notificationUserInfo: [NSObject: AnyObject])
}

class KeyboardManager {
    init() {
        NSNotificationCenter.defaultCenter().addObserver(self, selector: #selector(KeyboardManager.keyboardWillChangeFrameNotification(_:)), name: UIKeyboardWillChangeFrameNotification, object: nil)
        NSNotificationCenter.defaultCenter().addObserver(self, selector: #selector(KeyboardManager.keyboardWillHide(_:)), name: UIKeyboardWillHideNotification, object: nil)
    }

    func keyboardWillHide(notification:NSNotification) {
        guard let userInfo = notification.userInfo else { return }
        delegate?.keyboardWillHide(userInfo)
    }
}

```

Say we only want to implement **func keyboardWillHide(notificationUserInfo: [NSObject: AnyObject])** in one ViewController. We can also make this method optional.

```

typealias KeyboardManagerDelegate = protocol<KeyboardManagerModel, KeyboardManagerConfigureable>

protocol KeyboardManagerModel: class {
    func keyboardWillChangeFrame(endFrame: CGRect?, duration: NSTimeInterval, animationCurve: UIViewAnimationOptions)
}

@objc protocol KeyboardManagerConfigureable {
    optional func keyboardWillHide(userInfo: [NSObject: AnyObject])
}

```

\*Note this pattern helps avoid overuse of **@objc**. See <http://www.jessesquires.com/avoiding-objc-in-swift/> for more details!

In summary, I've found using a Singleton + Delegate to manage the keyboard is both more efficient and easier to use than using Notifications

## 第64.4节：键盘出现时视图上下移动

注意：这仅适用于iOS提供的内置键盘

SWIFT :

为了让UIViewController的视图在显示时增加其frame的原点，隐藏时减少，向你的类中添加以下函数：

```
func keyboardWillShow(notification: NSNotification) {  
  
    if let keyboardSize = (notification.userInfo?[UIKeyboardFrameBeginUserInfoKey] as?  
NSValue)?.cgRectValue {  
        if self.view.frame.origin.y == 0{  
            self.view.frame.origin.y -= keyboardSize.height  
        }  
    }  
  
}  
  
func keyboardWillHide(notification: NSNotification) {  
    if let keyboardSize = (notification.userInfo?[UIKeyboardFrameBeginUserInfoKey] as?  
NSValue)?.cgRectValue {  
        if self.view.frame.origin.y != 0{  
            self.view.frame.origin.y += keyboardSize.height  
        }  
    }  
}
```

在你的类的viewDidLoad()方法中，添加以下观察者：

```
NotificationCenter.default.addObserver(self, selector: #selector(Login.keyboardWillShow), name:  
NSNotification.Name.UIKeyboardWillShow, object: nil)  
NotificationCenter.default.addObserver(self, selector: #selector(Login.keyboardWillHide), name:  
NSNotification.Name.UIKeyboardWillHide, object: nil)
```

这将适用于任何屏幕尺寸，使用键盘的高度属性。

### OBJECTIVE-C :

要在Objective-C中做同样的事情，可以使用以下代码：

```
- (void)viewWillAppear:(BOOL)animated {  
    [super viewWillAppear:animated];  
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(keyboardWillShow:)  
name:UIKeyboardWillShowNotification object:nil];  
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(keyboardWillHide:)  
name:UIKeyboardWillHideNotification object:nil];  
}  
  
- (void)viewWillDisappear:(BOOL)animated {  
    [super viewWillDisappear:animated];  
    [[NSNotificationCenter defaultCenter] removeObserver:self name:UIKeyboardWillShowNotification  
object:nil];  
    [[NSNotificationCenter defaultCenter] removeObserver:self name:UIKeyboardWillHideNotification  
object:nil];  
}
```

## Section 64.4: Moving view up or down when keyboard is present

Note: This only works for the built-in keyboard provided by iOS

SWIFT:

In order for the view of a **UIViewController** to increase the origin of the frame when it is presented and decrease it when it is hidden, add the following functions to your class:

```
func keyboardWillShow(notification: NSNotification) {  
  
    if let keyboardSize = (notification.userInfo?[UIKeyboardFrameBeginUserInfoKey] as?  
NSValue)?.cgRectValue {  
        if self.view.frame.origin.y == 0{  
            self.view.frame.origin.y -= keyboardSize.height  
        }  
    }  
  
}  
  
func keyboardWillHide(notification: NSNotification) {  
    if let keyboardSize = (notification.userInfo?[UIKeyboardFrameBeginUserInfoKey] as?  
NSValue)?.cgRectValue {  
        if self.view.frame.origin.y != 0{  
            self.view.frame.origin.y += keyboardSize.height  
        }  
    }  
}
```

And in the viewDidLoad() method of your class, add the following observers:

```
NotificationCenter.default.addObserver(self, selector: #selector(Login.keyboardWillShow), name:  
NSNotification.Name.UIKeyboardWillShow, object: nil)  
NotificationCenter.default.addObserver(self, selector: #selector(Login.keyboardWillHide), name:  
NSNotification.Name.UIKeyboardWillHide, object: nil)
```

And this will work for any screen size, using the height property of the keyboard.

### OBJECTIVE-C :

To do the same thing in Objective-C, this code can be used:

```
- (void)viewWillAppear:(BOOL)animated {  
    [super viewWillAppear:animated];  
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(keyboardWillShow:)  
name:UIKeyboardWillShowNotification object:nil];  
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(keyboardWillHide:)  
name:UIKeyboardWillHideNotification object:nil];  
}  
  
- (void)viewWillDisappear:(BOOL)animated {  
    [super viewWillDisappear:animated];  
    [[NSNotificationCenter defaultCenter] removeObserver:self name:UIKeyboardWillShowNotification  
object:nil];  
    [[NSNotificationCenter defaultCenter] removeObserver:self name:UIKeyboardWillHideNotification  
object:nil];  
}
```

```

- (void)keyboardWillShow:(NSNotification *)notification
{
    CGSize keyboardSize = [[[notification userInfo] objectForKey:UIKeyboardFrameBeginUserInfoKey] CGRectValue].size;

    [UIView animateWithDuration:0.3 animations:^{
        CGRect f = self.view.frame;
        f.origin.y = -keyboardSize.height;
        self.view.frame = f;
    }];
}

-(void)keyboardWillHide:(NSNotification *)notification
{
    [UIView animateWithDuration:0.3 animations:^{
        CGRect f = self.view.frame;
        f.origin.y = 0.0f;
        self.view.frame = f;
    }];
}

```

## 第64.5节：显示键盘时滚动UIScrollView/UITableView

这里有几种可用的方法：

- 您可以订阅键盘出现事件通知并手动更改偏移量：

```

//Swift 2.0+
override func viewDidLoad() {
    super.viewDidLoad()

    NSNotificationCenter.defaultCenter().addObserver(self, selector:
#selector(YourVCClassName.keyboardWillShow(_:)), name: UIKeyboardWillShowNotification, object: nil)
    NSNotificationCenter.defaultCenter().addObserver(self, selector:
#selector(YourVCClassName.keyboardWillHide(_:)), name: UIKeyboardWillHideNotification, object: nil)
}

func keyboardWillShow(notification: NSNotification) {
    if let userInfo = notification.userInfo {
        if let keyboardHeight = userInfo[UIKeyboardFrameEndUserInfoKey]?CGRectValue.size.height {
            tableView.contentInset = UIEdgeInsetsMake(0, 0, keyboardHeight, 0)
        }
    }
}

func keyboardWillHide(notification: NSNotification) {
    tableView.contentInset = UIEdgeInsetsMake(0, 0, 0, 0)
}

//Objective-C
- (void)viewDidLoad {
    [super viewDidLoad];

    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(keyboardWillShow:)
name:UIKeyboardWillShowNotification object:nil];
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(keyboardWillHide:)
name:UIKeyboardWillHideNotification object:nil];
}

```

```

- (void)keyboardWillShow:(NSNotification *)notification
{
    CGSize keyboardSize = [[[notification userInfo] objectForKey:UIKeyboardFrameBeginUserInfoKey] CGRectValue].size;

    [UIView animateWithDuration:0.3 animations:^{
        CGRect f = self.view.frame;
        f.origin.y = -keyboardSize.height;
        self.view.frame = f;
    }];
}

-(void)keyboardWillHide:(NSNotification *)notification
{
    [UIView animateWithDuration:0.3 animations:^{
        CGRect f = self.view.frame;
        f.origin.y = 0.0f;
        self.view.frame = f;
    }];
}

```

## Section 64.5: Scrolling a UIScrollView/UITableView When Displaying the Keyboard

There are few approaches available there:

1. You can subscribe for keyboard appearance events notifications and change offset manually:

```

//Swift 2.0+
override func viewDidLoad() {
    super.viewDidLoad()

    NSNotificationCenter.defaultCenter().addObserver(self, selector:
#selector(YourVCClassName.keyboardWillShow(_:)), name: UIKeyboardWillShowNotification, object: nil)
    NSNotificationCenter.defaultCenter().addObserver(self, selector:
#selector(YourVCClassName.keyboardWillHide(_:)), name: UIKeyboardWillHideNotification, object: nil)
}

func keyboardWillShow(notification: NSNotification) {
    if let userInfo = notification.userInfo {
        if let keyboardHeight = userInfo[UIKeyboardFrameEndUserInfoKey]?CGRectValue.size.height {
            tableView.contentInset = UIEdgeInsetsMake(0, 0, keyboardHeight, 0)
        }
    }
}

func keyboardWillHide(notification: NSNotification) {
    tableView.contentInset = UIEdgeInsetsMake(0, 0, 0, 0)
}

//Objective-C
- (void)viewDidLoad {
    [super viewDidLoad];

    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(keyboardWillShow:)
name:UIKeyboardWillShowNotification object:nil];
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(keyboardWillHide:)
name:UIKeyboardWillHideNotification object:nil];
}

```

```
}

- (void)keyboardWillShow:(NSNotification *)notification {
    NSDictionary *userInfo = [notification userInfo];
    if (userInfo) {
        CGRect keyboardEndFrame;
        [[userInfo objectForKey:UIKeyboardFrameEndUserInfoKey] getValue:&keyboardEndFrame];
        tableView.contentInset = UIEdgeInsetsMake(0, 0, keyboardEndFrame.size.height, 0);
    }
}

- (void)keyboardWillHide:(NSNotification *)notification {
    tableView.contentInset = UIEdgeInsetsMake(0, 0, 0, 0);
}
```

2.或者使用现成的解决方案，如 TPKeyboardAvoidingTableView 或 TPKeyboardAvoidingScrollView

<https://github.com/michaeltyson/TPKeyboardAvoiding>

```
}

- (void)keyboardWillShow:(NSNotification *)notification {
    NSDictionary *userInfo = [notification userInfo];
    if (userInfo) {
        CGRect keyboardEndFrame;
        [[userInfo objectForKey:UIKeyboardFrameEndUserInfoKey] getValue:&keyboardEndFrame];
        tableView.contentInset = UIEdgeInsetsMake(0, 0, keyboardEndFrame.size.height, 0);
    }
}

- (void)keyboardWillHide:(NSNotification *)notification {
    tableView.contentInset = UIEdgeInsetsMake(0, 0, 0, 0);
}
```

2. Or use ready-made solutions like TPKeyboardAvoidingTableView or TPKeyboardAvoidingScrollView

<https://github.com/michaeltyson/TPKeyboardAvoiding>

# 第65章：检查网络连接

## 第65.1节：创建可达性监听器

苹果的Reachability类会定期检查网络状态并提醒观察者状态变化。

```
Reachability *internetReachability = [Reachability reachabilityForInternetConnection];
[internetReachability startNotifier];
```

## 第65.2节：添加网络变化观察者

Reachability使用NSNotification消息在网络状态变化时提醒观察者。你的类需要成为观察者。

```
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(reachabilityChanged:)
name:kReachabilityChangedNotification object:nil];
```

在类的其他地方，实现方法签名

```
- (void)reachabilityChanged:(NSNotification *)note {
    //响应网络变化的代码
}
```

## 第65.3节：网络不可用时提醒

```
- (void)reachabilityChanged:(NSNotification *)note {
    Reachability* reachability = [note object];
    NetworkStatus netStatus = [reachability currentReachabilityStatus];

    if (netStatus == NotReachable) {
        NSLog(@"网络不可用");
    }
}
```

## 第65.4节：当连接变为WIFI或蜂窝网络时发出警报

```
- (void)reachabilityChanged:(NSNotification *)note {
    Reachability* reachability = [note object];
    NetworkStatus netStatus = [reachability currentReachabilityStatus];

    switch (netStatus) {
        case NotReachable:
            NSLog(@"网络不可用");
            break;
        case ReachableViaWWAN:
            NSLog(@"网络为蜂窝网络");
            break;
        case ReachableViaWiFi:
            NSLog(@"网络为WIFI");
            break;
    }
}
```

# Chapter 65: Checking for Network Connectivity

## Section 65.1: Creating a Reachability listener

Apple's [Reachability](#) class periodically checks the network status and alerts observers to changes.

```
Reachability *internetReachability = [Reachability reachabilityForInternetConnection];
[internetReachability startNotifier];
```

## Section 65.2: Add observer to network changes

Reachability uses [NSNotification](#) messages to alert observers when the network state has changed. Your class will need to become an observer.

```
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(reachabilityChanged:)
name:kReachabilityChangedNotification object:nil];
```

Elsewhere in your class, implement method signature

```
- (void)reachabilityChanged:(NSNotification *)note {
    //code which reacts to network changes
}
```

## Section 65.3: Alert when network becomes unavailable

```
- (void)reachabilityChanged:(NSNotification *)note {
    Reachability* reachability = [note object];
    NetworkStatus netStatus = [reachability currentReachabilityStatus];

    if (netStatus == NotReachable) {
        NSLog(@"Network unavailable");
    }
}
```

## Section 65.4: Alert when connection becomes a WIFI or cellular network

```
- (void)reachabilityChanged:(NSNotification *)note {
    Reachability* reachability = [note object];
    NetworkStatus netStatus = [reachability currentReachabilityStatus];

    switch (netStatus) {
        case NotReachable:
            NSLog(@"Network unavailable");
            break;
        case ReachableViaWWAN:
            NSLog(@"Network is cellular");
            break;
        case ReachableViaWiFi:
            NSLog(@"Network is WIFI");
            break;
    }
}
```

## 第65.5节：验证是否已连接到网络

### Swift

```
import SystemConfiguration

/// 该类有助于在处理互联网网络连接时实现代码重用。
class NetworkHelper {

    /**
     验证设备是否连接到互联网网络。
     - 返回值： 如果连接到任何互联网网络则返回true，如果未连接到任何互联网网络则返回false。
     */
    class func isConnectedToNetwork() -> Bool {
        var zeroAddress = sockaddr_in()

        zeroAddress.sin_len = UInt8(sizeofValue(zeroAddress))
        zeroAddress.sin_family = sa_family_t(AF_INET)

        let defaultRouteReachability = withUnsafePointer(&zeroAddress) {
            SCNetworkReachabilityCreateWithAddress(nil, UnsafePointer($0))
        }

        var flags = SCNetworkReachabilityFlags()

        if !SCNetworkReachabilityGetFlags(defaultRouteReachability!, &flags) {
            return false
        }

        let isReachable = (flags.rawValue & UInt32(kSCNetworkFlagsReachable)) != 0
        let needsConnection = (flags.rawValue & UInt32(kSCNetworkFlagsConnectionRequired)) != 0

        return (isReachable && !needsConnection)
    }

    if NetworkHelper.isConnectedToNetwork() {
        // 已连接到网络
    }
}
```

### Objective-C :

我们可以通过几行代码检查网络连接情况，如下所示：

```
-(BOOL)isConnectedToNetwork
{
    Reachability *networkReachability = [Reachability reachabilityForInternetConnection];
    NetworkStatus networkStatus = [networkReachability currentReachabilityStatus];
    if (networkStatus == NotReachable)
    {
        NSLog(@"没有网络连接");
        return false;
    } else
    {
        NSLog(@"有网络连接");
        return true;
    }
}
```

## Section 65.5: Verify if is connected to network

### Swift

```
import SystemConfiguration

/// Class helps to code reuse in handling internet network connections.
class NetworkHelper {

    /**
     Verify if the device is connected to internet network.
     - returns: true if is connected to any internet network, false if is not
     connected to any internet network.
     */
    class func isConnectedToNetwork() -> Bool {
        var zeroAddress = sockaddr_in()

        zeroAddress.sin_len = UInt8(sizeofValue(zeroAddress))
        zeroAddress.sin_family = sa_family_t(AF_INET)

        let defaultRouteReachability = withUnsafePointer(&zeroAddress) {
            SCNetworkReachabilityCreateWithAddress(nil, UnsafePointer($0))
        }

        var flags = SCNetworkReachabilityFlags()

        if !SCNetworkReachabilityGetFlags(defaultRouteReachability!, &flags) {
            return false
        }

        let isReachable = (flags.rawValue & UInt32(kSCNetworkFlagsReachable)) != 0
        let needsConnection = (flags.rawValue & UInt32(kSCNetworkFlagsConnectionRequired)) != 0

        return (isReachable && !needsConnection)
    }

    if NetworkHelper.isConnectedToNetwork() {
        // Is connected to network
    }
}
```

### Objective-C:

we can check network connectivity within few lines of code as:

```
- (BOOL)isConnectedToNetwork
{
    Reachability *networkReachability = [Reachability reachabilityForInternetConnection];
    NetworkStatus networkStatus = [networkReachability currentReachabilityStatus];
    if (networkStatus == NotReachable)
    {
        NSLog(@"There IS NO internet connection");
        return false;
    } else
    {
        NSLog(@"There IS internet connection");
        return true;
    }
}
```



# 第66章：辅助功能

iOS中的辅助功能允许听力障碍和视力障碍用户通过支持VoiceOver、语音控制、黑底白字、单声道音频、语音转文字等多种功能来访问iOS及您的应用程序。为iOS应用提供辅助功能意味着让应用对所有人都可用。

## 第66.1节：使视图可访问

将您的UIView子类标记为可访问元素，以便VoiceOver能够识别。

```
myView.isAccessibilityElement = YES;
```

确保视图能够朗读有意义的标签、值和提示。苹果在Accessibility Programming Guide中提供了如何选择良好描述的更多细节。

## 第66.2节：无障碍框架

无障碍框架被VoiceOver用于触摸的命中测试、绘制VoiceOver光标，以及计算当用户双击屏幕时在聚焦元素中模拟点击的位置。请注意，框架是以屏幕坐标为单位的！

```
myElement.accessibilityFrame = frameInScreenCoordinates;
```

如果您的元素或屏幕布局经常变化，考虑重写-accessibilityFrame以始终提供最新的矩形。计算滚动视图子视图的相对于屏幕的框架可能容易出错且繁琐。iOS 10引入了一个新API来简化此操作：accessibilityFrameInContainerSpace。

## 第66.3节：布局变化

在许多情况下，单个屏幕内的内容会更新为新的或不同的内容。例如，想象一个表单，根据用户对前一个问题的回答显示额外选项。在这种情况下，“布局变化”通知允许您宣布变化或聚焦到新元素。此通知接受与屏幕变化通知相同的参数。

```
UIAccessibilityPostNotification(UIAccessibilityLayoutChangedNotification, firstElement);
```

## 第66.4节：无障碍容器

VoiceOver 可以导航 iOS 上的许多应用程序，因为大多数 UIKit 类都实现了 UIAccessibilityProtocol。不使用 UIView 表示屏幕元素的功能，包括利用 Core Graphics 或 Metal 进行绘图的应用程序，必须为这些元素描述无障碍信息。从 iOS 8.0 开始，可以通过为包含不可访问元素的 UIView 分配一个属性来实现：

```
myInaccessibleContainerView.accessibilityElements = @[elements, that, should, be, accessible];
```

数组中的每个对象可以是 UIAccessibilityElement 的实例或任何其他遵循 UIAccessibilityProtocol 的类。子元素应按用户应导航的顺序返回。作为应用程序作者，您可以使用无障碍容器来覆盖 VoiceOver 滑动导航的默认从左上到右下的顺序。鉴于 UIView 实现了 UIAccessibilityProtocol，您可以在同一子无障碍元素数组中组合 UIAccessibilityElement 和 UIView 的实例。请注意，如果您手动分配元素，则无需实现任何动态无障碍协议方法，尽管您

# Chapter 66: Accessibility

Accessibility in iOS allows users with hearing disabilities and visual impairments to access iOS and your application by supporting various features like VoiceOver, Voice Control, White on Black, Mono Audio, Speech to Text and so on. Providing Accessibility in the iOS app means making the app usable for everyone.

## Section 66.1: Make a View Accessible

Mark your `UIView` subclass as an accessible element so that it is visible to VoiceOver.

```
myView.isAccessibilityElement = YES;
```

Ensure that the view speaks a meaningful label, value, and hint. Apple provides more details on how to choose good descriptions in the [Accessibility Programming Guide](#).

## Section 66.2: Accessibility Frame

The accessibility frame is used by VoiceOver for hit testing touches, drawing the VoiceOver cursor, and calculating where in the focused element to simulate a tap when the user double-taps the screen. Note that the frame is in screen coordinates!

```
myElement.accessibilityFrame = frameInScreenCoordinates;
```

If your elements or screen layouts change often, consider overriding `-accessibilityFrame` to always provide an up-to-date rect. Calculating the screen-relative frame of scroll view subviews can be error-prone and tedious. iOS 10 introduces a new API to make this easier: `accessibilityFrameInContainerSpace`.

## Section 66.3: Layout Change

In many cases, content within a single screen will update with new or different content. For example, imagine a form that reveals additional options based on the user's answer to a previous question. In this case, a "layout change" notification lets you either announce the change or focus on a new element. This notification accepts the same parameters as the screen change notification.

```
UIAccessibilityPostNotification(UIAccessibilityLayoutChangedNotification, firstElement);
```

## Section 66.4: Accessibility Container

VoiceOver can navigate many apps on iOS because most UIKit classes implement `UIAccessibilityProtocol`. Features that don't represent onscreen elements using `UIView`, including apps that leverage Core Graphics or Metal to perform drawing, must describe these elements for accessibility. As of iOS 8.0, this can be done by assigning a property on the `UIView` containing inaccessible elements:

```
myInaccessibleContainerView.accessibilityElements = @[elements, that, should, be, accessible];
```

Each object in the array can be an instance of `UIAccessibilityElement` or any other class that adheres to `UIAccessibilityProtocol`. The child elements should be returned in the order the user should navigate them. As an application author, you can use accessibility containers to override the default top-left to bottom-right ordering of VoiceOver swipe navigation. Given that `UIView` implements `UIAccessibilityProtocol`, you can combine instances of `UIAccessibilityElement` and `UIView` in the same array of child accessibility elements. Note that if you assign elements manually, you do not need to implement any dynamic accessibility protocol methods, though you

可能需要发出屏幕更改通知，以便 VoiceOver 能检测到这些元素。

## 第 66.5 节：隐藏元素

大多数 UIKit 类，包括 UIView，都遵循 UIAccessibilityProtocol 并默认返回正确的值。很容易理所当然地认为设置为隐藏的 UIView 也会从无障碍层级中消失，VoiceOver 不会导航它们。虽然这种默认行为通常足够，但有时视图会存在于视图层级中，但不可见或不可导航。例如，一组按钮可能被另一个视图覆盖，使其对有视力的用户不可见。然而，VoiceOver 仍会尝试导航它们，因为它们在技术上并未被 UIKit 隐藏，因此仍存在于无障碍层级中。在这种情况下，您必须提示 VoiceOver 父视图不可访问。您可以通过在视图离开屏幕时显式将其隐藏来从 UIKit 中隐藏视图：

```
myViewFullOfButtons.hidden = YES;
```

或者，您可以保持父视图可见，仅将其子视图从无障碍层级中隐藏：

```
myViewFullOfButtons.accessibilityElementsHidden = YES;
```

临时视图是你想要在无障碍层级中隐藏元素但仍让用户可见的另一种方式。例如，当你按下音量按钮时弹出的视图对有视力的用户是可见的，但它不像普通警报那样需要引起注意。你不会希望 VoiceOver 打断用户，将光标从他们正在做的事情上移开去宣布新的音量，尤其是调整音量已经通过点击声提供了听觉反馈。在这种情况下，你会想使用 accessibilityElementsHidden 来隐藏该视图。

## 第66.6节：屏幕变化

VoiceOver 大多数时候表现出色，轻松朗读充满内容的屏幕并直观地跟随用户。可惜，没有通用的解决方案是完美的。有时只有你，作为应用开发者，知道 VoiceOver 应该聚焦在哪里以获得最佳用户体验。幸运的是，VoiceOver 会监听系统无障碍通知以获取焦点所在的线索。要手动移动 VoiceOver 光标，请发布无障碍屏幕变化通知：

```
UIAccessibilityPostNotification(UIAccessibilityScreenChangedNotification, firstElement);
```

发布此通知时，一系列简短的音调会通知用户发生了变化。第二个参数可以是下一个要聚焦的元素，也可以是宣布变化的字符串。只有在没有此通知时 VoiceOver 体验很差且没有其他解决方法时，才发布屏幕变化通知。移动 VoiceOver 光标就像戳视力用户的屏幕一样，这样引导可能令人烦恼和迷失方向。

## 第66.7节：公告

公告用于提醒用户无需任何交互的事件，例如“屏幕已锁定”或“加载完成。”使用更具体的公告来通知用户屏幕变化或较小的布局变化。

```
UIAccessibilityPostNotification(UIAccessibilityAnnouncementNotification, @"事件发生了！");
```

## 第66.8节：元素排序

VoiceOver 从左上角导航到右下角，无论视图层次结构如何。这通常是内容的呈现方式

may need to issue a screen change notification for the elements to be detected by VoiceOver.

## Section 66.5: Hiding Elements

Most UIKit classes, including UIView, adhere to UIAccessibilityProtocol and return correct values by default. It's easy to take for granted that a `UIView` set to hidden is also absent from the accessibility hierarchy and won't be navigated by VoiceOver. While this default behavior is usually sufficient, there are times where a view will be present in the view hierarchy but not visible or navigable. For example, a collection of buttons may be overlapped by another view, rendering them invisible to a sighted user. VoiceOver, however, will still try to navigate them since they are technically not hidden from UIKit and therefore are still present in the accessibility hierarchy. In such cases, you must hint to VoiceOver that the parent view isn't accessible. You can do this by explicitly hiding the view from UIKit by setting hidden when the view goes offscreen:

```
myViewFullOfButtons.hidden = YES;
```

Alternatively, you can leave the parent view visible and simply hide its children from the accessibility hierarchy:

```
myViewFullOfButtons.accessibilityElementsHidden = YES;
```

Temporary views are another place you'll want to hide elements from the accessibility hierarchy while leaving them visible to users. For example, the view that pops up when you hit the volume button is visible to sighted users but doesn't demand attention the way a normal alert does. You wouldn't want VoiceOver to interrupt the user and move the cursor from away from whatever they were doing to announce the new volume, especially given that adjusting volume already provides auditory feedback through the clicking sound it makes. In cases like this, you'll want to hide the view using accessibilityElementsHidden.

## Section 66.6: Screen Change

VoiceOver works great most of the time, breezily reading aloud screens full of content and intuitively following the user. Alas, no general solution is perfect. Sometimes only you, the app developer, know where VoiceOver should be focused for an optimal user experience. Fortunately, VoiceOver listens to system accessibility notifications for clues about where focus belongs. To move the VoiceOver cursor manually, post an accessibility screen changed notification:

```
UIAccessibilityPostNotification(UIAccessibilityScreenChangedNotification, firstElement);
```

When this notification is posted, a short series of tones notify users of the change. The second parameter can be either the next element to focus or a string announcing the change. Only post a screen change notification if the VoiceOver experience is poor without it and no other workaround exists. Moving the VoiceOver cursor is like poking at a sighted user's screen. It can be annoying and disorienting to be led around that way.

## Section 66.7: Announcement

Announcements are useful for alerting users to events that don't require any interaction, such as "screen locked" or "finished loading." Use a more specific announcement to notify users of screen changes or more minor layout changes.

```
UIAccessibilityPostNotification(UIAccessibilityAnnouncementNotification, @"The thing happened!");
```

## Section 66.8: Ordering Elements

VoiceOver navigates from top-left to bottom-right, irrespective of the view hierarchy. This is usually how content is

由于视力正常的用户倾向于以“F形模式”扫描屏幕，因此在从左到右的语言中进行排列。

VoiceOver 用户期望以与普通用户相同的方式进行导航。可预测性和一致性对无障碍性非常重要。请避免进行“改进”默认行为的自定义（例如，将标签栏放在滑动顺序的最前面）。话虽如此，如果您收到反馈说应用中元素的顺序令人意外，有几种方法可以改善体验。

如果 VoiceOver 应该依次朗读视图的子视图但没有这样做，您可能需要提示 VoiceOver 这些包含在单个视图中的元素是相关的。您可以通过设置 `shouldGroupAccessibilityChildren` 来实现：

```
myView.shouldGroupAccessibilityChildren = YES;
```

为了支持跨多个容器或包含非 UIKit 渲染界面的复杂导航结构，考虑在父视图上实现容器协议。

## 第 66.9 节：模态视图

模态视图会完全吸引用户的注意力，直到任务完成。iOS 通过在模态视图（如警报或弹出窗口）可见时使所有其他内容变暗并禁用，向用户明确这一点。实现自定义模态界面的应用需要通过设置

`accessibilityViewIsModal` 来提示 VoiceOver 该视图应获得用户的全部注意力。请注意，此属性应仅设置在包含模态内容的视图上，而不是模态视图内包含的元素上。

```
myModalView.accessibilityViewIsModal = YES;
```

将视图标记为模态会促使 VoiceOver 忽略同级视图。如果在设置此属性后，您发现 VoiceOver 仍然导航到应用中的其他元素，请尝试在模态消失之前隐藏有问题的视图。

arranged in left-to-right languages since sighted individuals tend to scan the screen in an “F-shaped pattern”. VoiceOver users will expect to navigate the same way as typical users. Predictability and consistency are very important to accessibility. Please refrain from making customizations that “improve” on default behavior (eg. ordering the tab bar first in the swipe order). That said, if you have received feedback that the order of elements in your app is surprising, there are a couple of ways you can improve the experience.

If VoiceOver should read a view’s subviews one after the next but is not, you may need to hint to VoiceOver that the elements contained within a single view are related. You can do this by setting `shouldGroupAccessibilityChildren`:

```
myView.shouldGroupAccessibilityChildren = YES;
```

To support complex navigation structures that span multiple containers or include interfaces rendered without UIKit, consider implementing the container protocol on the parent view.

## Section 66.9: Modal View

Modal views completely capture the user’s attention until a task is complete. iOS clarifies this to users by dimming and disabling all other content when a modal view, such as an alert or popover, is visible. An app that implements a custom modal interface needs to hint to VoiceOver that this view deserves the user’s undivided attention by setting `accessibilityViewIsModal`. Note that this property should only be set on the view containing modal content, not elements contained within a modal view.

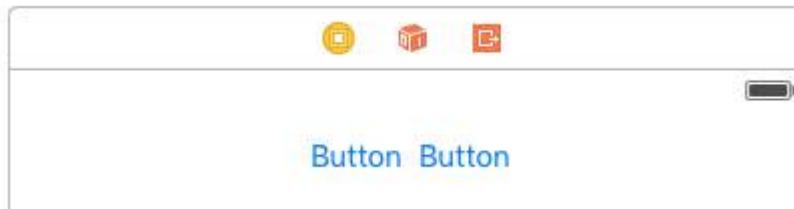
```
myModalView.accessibilityViewIsModal = YES;
```

Tagging a view as modal encourages VoiceOver to ignore sibling views. If, after setting this property, you find that VoiceOver still navigates other elements in your app, try hiding problem views until the modal dismisses.

# 第67章：自动布局

自动布局根据对视图施加的约束，动态计算视图层级中所有视图的大小和位置。来源

## 第67.1节：均匀分布视图



通常希望两个视图并排放置，并在其父视图中居中。Stack Overflow 上常见的答案是将这两个视图嵌入一个UIView中，并使该UIView居中。但这既不必要也不推荐。

摘自UILayoutGuide文档：

向视图层级中添加虚拟视图会带来多种成本。首先，是创建和维护该视图本身的成本。其次，虚拟视图是视图层级的完整成员，这意味着它会增加层级执行每项任务的开销。最糟糕的是，隐藏的虚拟视图可能会拦截原本发送给其他视图的消息，导致非常难以发现的问题。

您可以使用UILayoutGuide来实现此目的，而不是将按钮添加到不必要的UIView中。UILayoutGuide本质上是一个可以与自动布局交互的矩形空间。您可以在按钮的左右两侧放置UILayoutGuide，并设置它们的宽度相等。这样可以使按钮居中。以下是代码示例：

### 视觉格式语言样式

```
view.addSubview(button1)
view.addSubview(button2)

let leftSpace = UILayoutGuide()
view.addLayoutGuide(leftSpace)

let rightSpace = UILayoutGuide()
view.addLayoutGuide(rightSpace)

let views = [
    "leftSpace" : leftSpace,
    "button1" : button1,
    "button2" : button2,
    "rightSpace" : rightSpace
]

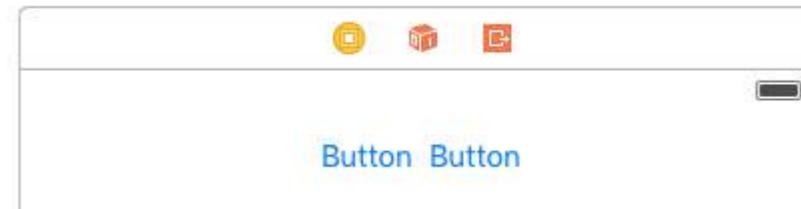
// 将按钮和布局引导水平排列成一行。
// 将布局引导放在两端。
NSLayoutConstraint.activateConstraints(NSLayoutConstraint.constraintsWithVisualFormat("H:[leftSpace][button1]-[button2][rightSpace]"), options: [], metrics: nil, views: views)

// 现在设置布局引导的宽度相等，这样按钮左右的空间将相等
leftSpace.widthAnchor.constraintEqualToAnchor(rightSpace.widthAnchor).active = true
```

# Chapter 67: Auto Layout

Auto Layout dynamically calculates the size and position of all the views in your view hierarchy, based on constraints placed on those views. [Source](#)

## Section 67.1: Space Views Evenly



It is common to want two views to be side by side, centered in their superview. The common answer given on Stack Overflow is to embed these two views in a `UIView` and center the `UIView`. This is not necessary or recommended. From the [UILayoutGuide](#) docs:

There are a number of costs associated with adding dummy views to your view hierarchy. First, there is the cost of creating and maintaining the view itself. Second, the dummy view is a full member of the view hierarchy, which means that it adds overhead to every task the hierarchy performs. Worst of all, the invisible dummy view can intercept messages that are intended for other views, causing problems that are very difficult to find.

You can use `UILayoutGuide` to do this, instead of adding the buttons into an unnecessary `UIView`. A `UILayoutGuide` is essentially a rectangular space that can interact with Auto Layout. You put a `UILayoutGuide` on the left and right sides of the buttons and set their widths to be equal. This will center the buttons. Here is how to do it in code:

### Visual Format Language style

```
view.addSubview(button1)
view.addSubview(button2)

let leftSpace = UILayoutGuide()
view.addLayoutGuide(leftSpace)

let rightSpace = UILayoutGuide()
view.addLayoutGuide(rightSpace)

let views = [
    "leftSpace" : leftSpace,
    "button1" : button1,
    "button2" : button2,
    "rightSpace" : rightSpace
]

// Lay the buttons and layout guides out horizontally in a line.
// Put the layout guides on each end.
NSLayoutConstraint.activateConstraints(NSLayoutConstraint.constraintsWithVisualFormat("H:[leftSpace][button1]-[button2][rightSpace]"), options: [], metrics: nil, views: views)

// Now set the layout guides widths equal, so that the space on the
// left and the right of the buttons will be equal
leftSpace.widthAnchor.constraintEqualToAnchor(rightSpace.widthAnchor).active = true
```

## 锚点样式

```
let leadingSpace = UILayoutGuide()  
let trailingSpace = UILayoutGuide()  
view.addLayoutGuide(leadingSpace)  
view.addLayoutGuide(trailingSpace)  
  
leadingSpace.widthAnchor.constraintEqualToAnchor(trailingSpace.widthAnchor).active = true  
  
leadingSpace.leadingAnchor.constraintEqualToAnchor(view.leadingAnchor).active = true  
leadingSpace.trailingAnchor.constraintEqualToAnchor(button1.leadingAnchor).active = true  
  
trailingSpace.leadingAnchor.constraintEqualToAnchor(button2.trailingAnchor).active = true  
trailingSpace.trailingAnchor.constraintEqualToAnchor(view.trailingAnchor).active = true
```

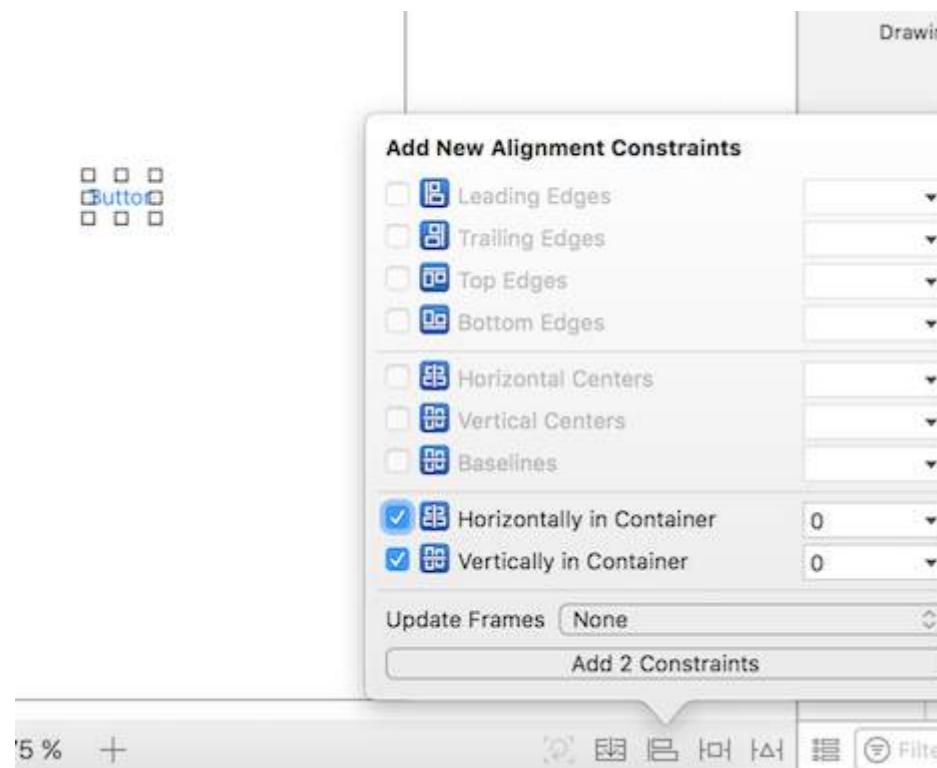
你还需要为此添加垂直约束，但这将使按钮在视图中居中，而无需添加任何“虚拟”视图！这将节省系统在显示那些“虚拟”视图时浪费的CPU时间。这个示例使用按钮，但你可以将按钮替换为任何你想要添加约束的视图。

如果你支持的是iOS 8或更早版本，创建此布局的最简单方法是添加隐藏的虚拟视图。在iOS 9中你可以用布局指南替代虚拟视图。

注意：Interface Builder尚不支持布局指南（Xcode 7.2.1）。所以如果你想使用它们，必须在代码中创建约束。[Source](#)

## 第67.2节：居中约束

在**storyboard**上选择你的按钮（或你想居中的任何视图）。然后点击右下角的对齐按钮。选择**Horizontally in Container**和**Vertically in Container**。点击“添加2个约束”。



如果它还没有完全居中，你可能需要再做一件事。点击底部栏中“Embed In Stack”按钮左边第二个的“更新框架”按钮。

## Anchor Style

```
let leadingSpace = UILayoutGuide()  
let trailingSpace = UILayoutGuide()  
view.addLayoutGuide(leadingSpace)  
view.addLayoutGuide(trailingSpace)  
  
leadingSpace.widthAnchor.constraintEqualToAnchor(trailingSpace.widthAnchor).active = true  
  
leadingSpace.leadingAnchor.constraintEqualToAnchor(view.leadingAnchor).active = true  
leadingSpace.trailingAnchor.constraintEqualToAnchor(button1.leadingAnchor).active = true  
  
trailingSpace.leadingAnchor.constraintEqualToAnchor(button2.trailingAnchor).active = true  
trailingSpace.trailingAnchor.constraintEqualToAnchor(view.trailingAnchor).active = true
```

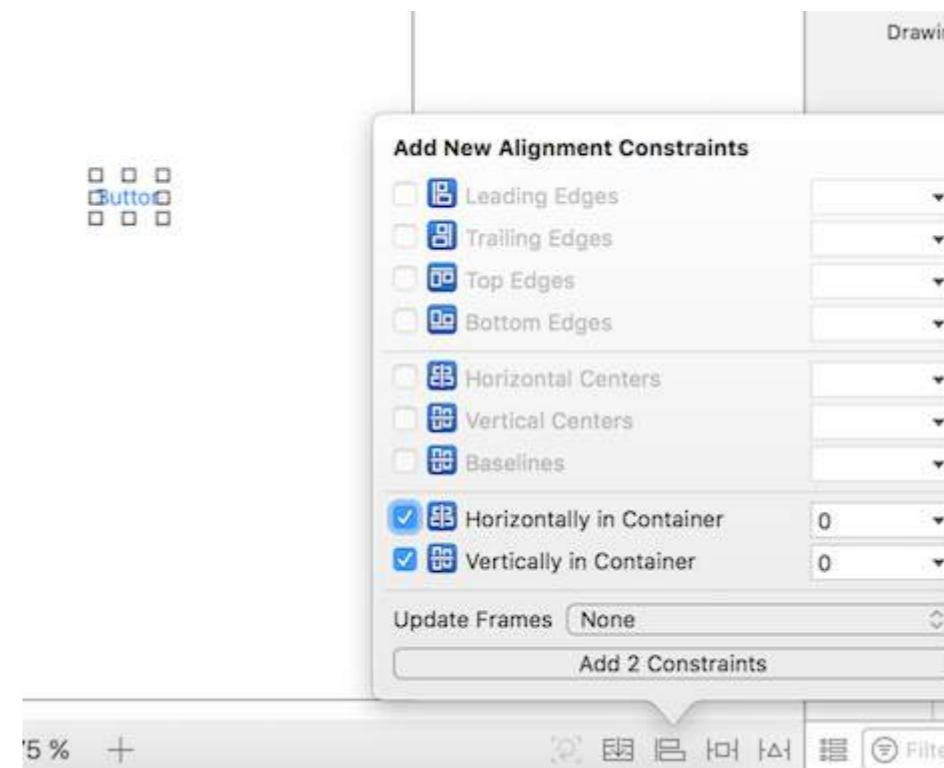
You will need to add vertical constraints to this as well, but this will center the buttons in the view without adding any "dummy" views! This will save the system from wasting CPU time on displaying those "dummy" views. This example uses buttons, but you can swap buttons out for any view you want to put constraints on.

If you are supporting iOS 8 or earlier the easiest way to create this layout is to add hidden dummy views. With iOS 9 you can replace dummy views with layout guides.

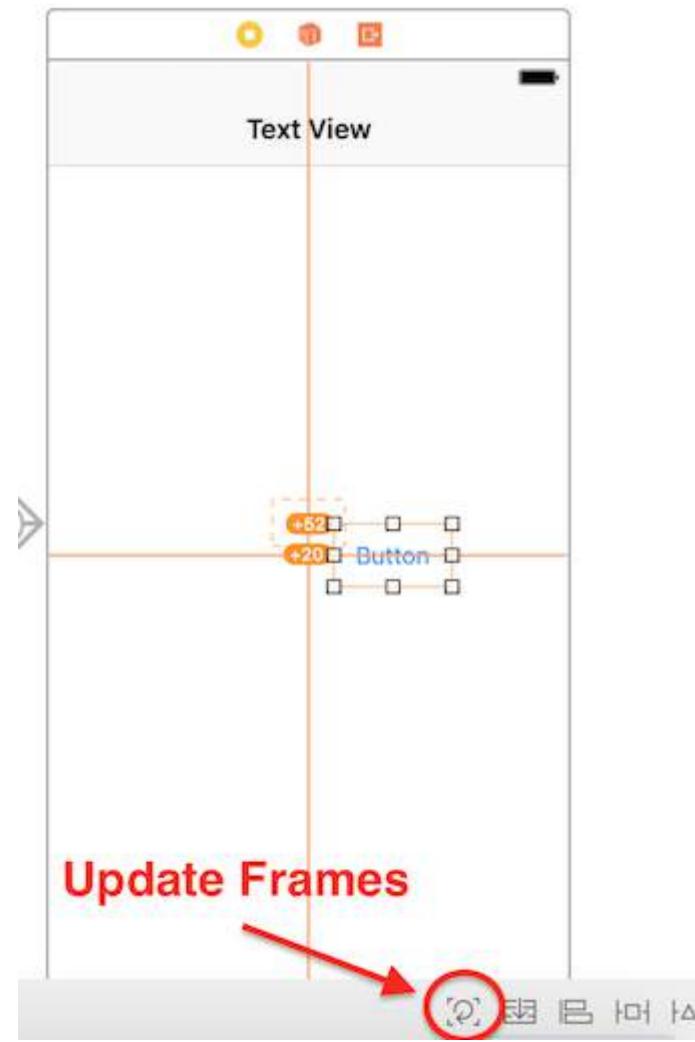
Note: Interface Builder does not support layout guides yet (Xcode 7.2.1). So if you want to use them you must create your constraints in code. [Source](#).

## Section 67.2: Center Constraints

Select your button (or whatever view you want to center) on the **Storyboard**. Then click the align button on the bottom right. Select **Horizontally in Container** and **Vertically in Container**. Click "Add 2 Constraints".



If it wasn't perfectly centered already, you may need to do one more thing. Click the "Update Frames" button that is two to the left of the "Embed In Stack" button on the bottom bar.

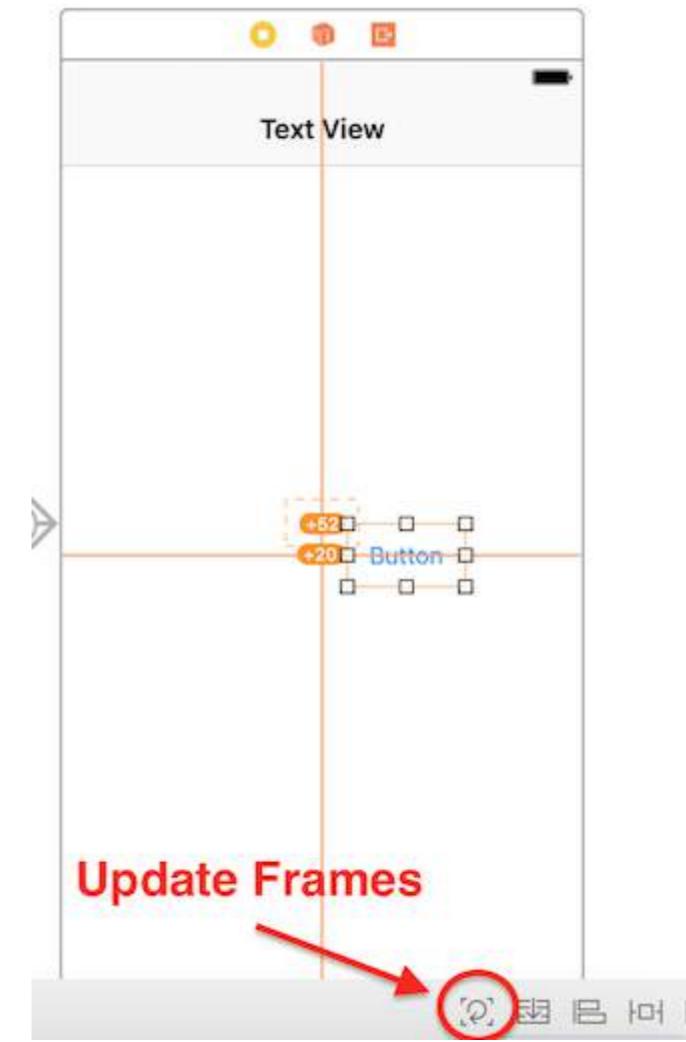
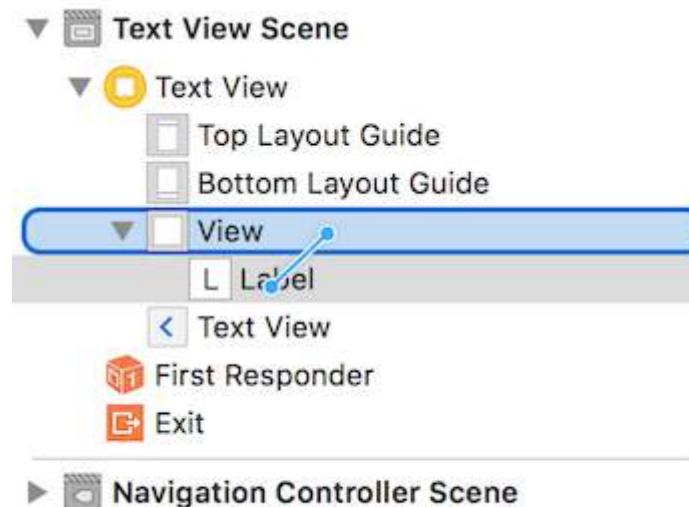


你也可以通过同时按下键盘快捷键来“按需更新框架”  
 $\text{⌘} + \text{⌥} + =$  (Command + Option 和等号) 在选择视图后，这可能节省一些时间。

现在当你运行你的应用时，无论使用什么设备尺寸，它都应该是居中的。

使用界面构建器居中视图的另一种方法是按住控制键点击并拖动。假设你想在视图中居中一个UILabel。通过点击左下角的侧边栏按钮打开故事板中的文档大纲。按住

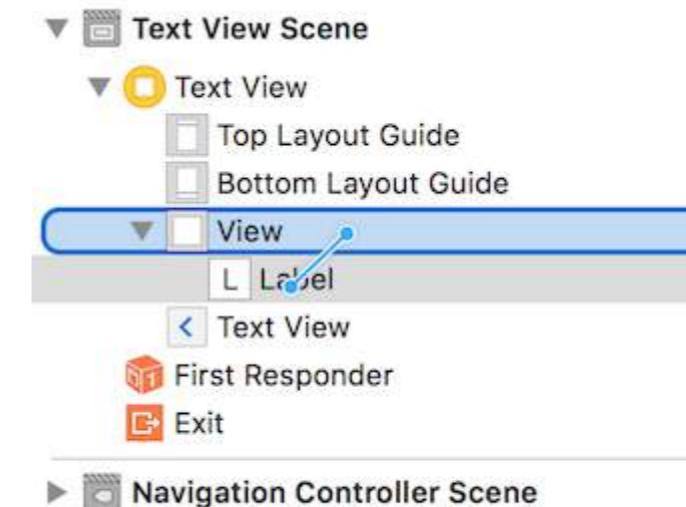
`ctrl` (控制键)，从标签拖动到视图时，应该会出现一条蓝线：



You can also "update frames as necessary" by pressing together  $\text{⌘} + \text{⌥} + =$  (Command + Option and equals) after selecting the view, this might save some time.

Now when you run your app it should be centered, no matter what device size you are using.

Another way to center views using Interface Builder is by control-click-dragging. Say you want to center a `UILabel` in a view. Open the Document Outline in your storyboard by clicking the sidebar button at the bottom left. Click and drag from the label to the view while holding `ctrl` (control), and a blue line should appear:



松开后，会出现一个约束选项菜单：



选择“在容器中水平居中”和“在容器中垂直居中”。根据需要更新框架，就完成了！一个居中的标签。

或者，你也可以通过编程方式添加约束。创建约束并将其添加到所需的UI元素和视图中，示例如下，我们创建一个按钮并将其水平和垂直居中对齐到其父视图：

## Objective-C

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    UIButton *yourButton = [[UIButton alloc] initWithFrame:CGRectMake(0, 0, 100, 18)];
    [yourButton setTitle:@"Button" forState:UIControlStateNormal];

    [self.view addConstraint:[NSLayoutConstraint constraintWithItem:yourButton
attribute:NSLayoutAttributeCenterY relatedBy:NSLayoutRelationEqual toItem:self.view
attribute:NSLayoutAttributeCenterY multiplier:1 constant:0]]; //垂直居中对齐到父视图

    [self.view addConstraint:[NSLayoutConstraint constraintWithItem:yourButton
attribute:NSLayoutAttributeCenterX relatedBy:NSLayoutRelationEqual toItem:self.view
attribute:NSLayoutAttributeCenterX multiplier:1 constant:0]]; //水平居中对齐到父视图

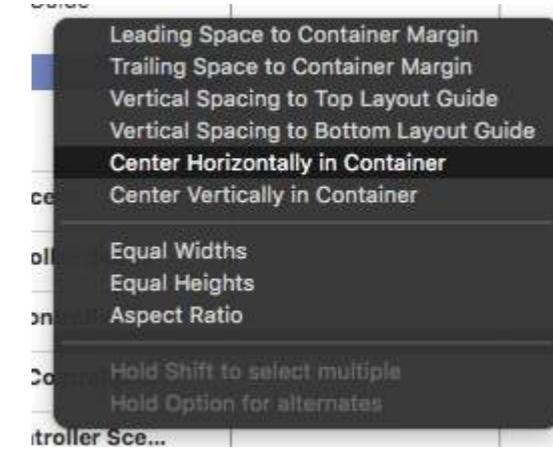
    [self.view addSubview:yourButton]; //将按钮添加到父视图
}
```

## Swift

```
override func viewDidLoad()
{
    super.viewDidLoad()
    let yourButton: UIButton = UIButton(frame: CGRect(x: 0, y: 0, width: 100, height: 18))
    yourButton.setTitle("Button", forState: .Normal)

    let centerVertically = NSLayoutConstraint(item: yourButton,
                                              attribute: .CenterX,
                                              relatedBy: .Equal,
                                              toItem: view,
                                              attribute: .CenterX,
                                              multiplier: 1.0,
                                              constant: 0.0)
```

Upon release, a menu of constraint options will appear:



Select "Center Horizontally in Container" and "Center Vertically in Container". Update frames as necessary, and voila! A centered label.

Alternatively, you can add the constraints programmatically. Create the constraints and add them to the desired UI elements and views as the following example describes, where we create a button and align it in the center, horizontally and vertically to its superview:

## Objective-C

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    UIButton *yourButton = [[UIButton alloc] initWithFrame:CGRectMake(0, 0, 100, 18)];
    [yourButton setTitle:@"Button" forState:UIControlStateNormal];

    [self.view addConstraint:[NSLayoutConstraint constraintWithItem:yourButton
attribute:NSLayoutAttributeCenterY relatedBy:NSLayoutRelationEqual toItem:self.view
attribute:NSLayoutAttributeCenterY multiplier:1 constant:0]]; //Align vertically center to superView

    [self.view addConstraint:[NSLayoutConstraint constraintWithItem:yourButton
attribute:NSLayoutAttributeCenterX relatedBy:NSLayoutRelationEqual toItem:self.view
attribute:NSLayoutAttributeCenterX multiplier:1 constant:0]]; //Align horizontally center to superView

    [self.view addSubview:yourButton]; //Add button to superView
}
```

## Swift

```
override func viewDidLoad()
{
    super.viewDidLoad()
    let yourButton: UIButton = UIButton(frame: CGRect(x: 0, y: 0, width: 100, height: 18))
    yourButton.setTitle("Button", forState: .Normal)

    let centerVertically = NSLayoutConstraint(item: yourButton,
                                              attribute: .CenterX,
                                              relatedBy: .Equal,
                                              toItem: view,
                                              attribute: .CenterX,
                                              multiplier: 1.0,
                                              constant: 0.0)
```

```

let centerHorizontally = NSLayoutConstraint(item: yourButton,
    attribute: .CenterY,
    relatedBy: .Equal,
    toItem: view,
    attribute: .CenterY,
    multiplier: 1.0,
    constant: 0.0)
NSLayoutConstraint.activateConstraints([centerVertically, centerHorizontally])
}

```

## 第67.3节：通过编程设置约束

样板代码示例

```

override func viewDidLoad() {
    super.viewDidLoad()

    let myView = UIView()
    myView.backgroundColor = UIColor.blueColor()
    myView.translatesAutoresizingMaskIntoConstraints = false
    view.addSubview(myView)

    // 在此添加约束代码
    // ...
}

```

在下面的示例中，Anchor 风格是优选方法，优于NSLayoutConstraint风格，但它仅在 iOS 9 及以上可用，因此如果你需要支持 iOS 8，则仍应使用NSLayoutConstraint风格。

### 固定

#### 锚点样式

```

let margins = view.layoutMarginsGuide
myView.leadingAnchor.constraintEqualToAnchor(margins.leadingAnchor, constant: 20).active = true

```

- 除了leadingAnchor，还有trailingAnchor、topAnchor和bottomAnchor。

#### NSLayoutConstraint 风格

```

NSLayoutConstraint(item: myView, attribute: NSLayoutAttribute.Leading, relatedBy:
NSLayoutRelation.Equal, toItem: 视图, attribute: NSLayoutAttribute.LeadingMargin, multiplier: 1.0,
constant: 20.0).active = true

```

- 除了.Leading，还有.Trailing、.Top和.Bottom。
- 除了.LeadingMargin，还有.TrailingMargin、.TopMargin和.BottomMargin。

#### 视觉格式语言样式

```

NSLayoutConstraint.constraintsWithVisualFormat("H:|-20-[myViewKey]", options: [], metrics: nil,
views: ["myViewKey": myView])

```

#### 宽度和高度

#### 锚点样式

```

myView.widthAnchor.constraintEqualToAnchor(nil, constant: 200).active = true
myView.heightAnchor.constraintEqualToAnchor(nil, constant: 100).active = true

```

```

let centerHorizontally = NSLayoutConstraint(item: yourButton,
    attribute: .CenterY,
    relatedBy: .Equal,
    toItem: view,
    attribute: .CenterY,
    multiplier: 1.0,
    constant: 0.0)
NSLayoutConstraint.activateConstraints([centerVertically, centerHorizontally])
}

```

## Section 67.3: Setting constraints programmatically

Boilerplate code example

```

override func viewDidLoad() {
    super.viewDidLoad()

    let myView = UIView()
    myView.backgroundColor = UIColor.blueColor()
    myView.translatesAutoresizingMaskIntoConstraints = false
    view.addSubview(myView)

    // Add constraints code here
    // ...
}

```

In the examples below the Anchor Style is the preferred method over NSLayoutConstraint Style, however it is only available from iOS 9, so if you are supporting iOS 8 then you should still use NSLayoutConstraint Style.

### Pinning

#### Anchor Style

```

let margins = view.layoutMarginsGuide
myView.leadingAnchor.constraintEqualToAnchor(margins.leadingAnchor, constant: 20).active = true

```

- In addition to leadingAnchor, there is also trailingAnchor, topAnchor, and bottomAnchor.

#### NSLayoutConstraint Style

```

NSLayoutConstraint(item: myView, attribute: NSLayoutAttribute.Leading, relatedBy:
NSLayoutRelation.Equal, toItem: view, attribute: NSLayoutAttribute.LeadingMargin, multiplier: 1.0,
constant: 20.0).active = true

```

- In addition to .Leading there is also .Trailing, .Top, and .Bottom.
- In addition to .LeadingMargin there is also .TrailingMargin, .TopMargin, and .BottomMargin.

#### Visual Format Language style

```

NSLayoutConstraint.constraintsWithVisualFormat("H:|-20-[myViewKey]", options: [], metrics: nil,
views: ["myViewKey": myView])

```

#### Width and Height

#### Anchor Style

```

myView.widthAnchor.constraintEqualToAnchor(nil, constant: 200).active = true
myView.heightAnchor.constraintEqualToAnchor(nil, constant: 100).active = true

```

## NSLayoutConstraint 风格

```
NSLayoutConstraint(item: myView, attribute: NSLayoutAttribute.Width, relatedBy:  
NSLayoutRelation.Equal, toItem: nil, attribute: NSLayoutAttribute.NotAnAttribute, multiplier: 1,  
constant: 200).active = true  
NSLayoutConstraint(item: myView, attribute: NSLayoutAttribute.Height, relatedBy:  
NSLayoutRelation.Equal, toItem: nil, attribute: NSLayoutAttribute.NotAnAttribute, multiplier: 1,  
constant: 100).active = true
```

### 视觉格式语言样式

```
NSLayoutConstraint.constraintsWithVisualFormat("H:[myViewKey(200)]", options: [], metrics: nil,  
views: ["myViewKey": myView])  
NSLayoutConstraint.constraintsWithVisualFormat("V:[myViewKey(100)]", options: [], metrics: nil,  
views: ["myViewKey": myView])
```

### 容器中居中

#### 锚点样式

```
myView.centerXAnchor.constraintEqualToAnchor(view.centerXAnchor).active = true  
myView.centerYAnchor.constraintEqualToAnchor(view.centerYAnchor).active = true
```

## NSLayoutConstraint 风格

```
NSLayoutConstraint(item: myView, attribute: NSLayoutAttribute.CenterX, relatedBy:  
NSLayoutRelation.Equal, toItem: view, attribute: NSLayoutAttribute.CenterX, multiplier: 1,  
constant: 0).active = true  
NSLayoutConstraint(item: myView, attribute: NSLayoutAttribute.CenterY, relatedBy:  
NSLayoutRelation.Equal, toItem: view, attribute: NSLayoutAttribute.CenterY, multiplier: 1,  
constant: 0).active = true
```

### 视觉格式语言样式

```
NSLayoutConstraint.constraintsWithVisualFormat("V:[viewKey]-(<=0)-[myViewKey]", options:  
NSLayoutFormatOptions.AlignAllCenterX, metrics: nil, views: ["myViewKey": myView, "viewKey": view])  
NSLayoutConstraint.constraintsWithVisualFormat("H:[viewKey]-(<=0)-[myViewKey]", options:  
NSLayoutFormatOptions.AlignAllCenterY, metrics: nil, views: ["myViewKey": myView, "viewKey": view])
```

## 第67.4节：UILabel及其父视图大小根据UILabel中的文本调整

### 逐步指南：

#### 步骤1：设置UIView的约束

- 1.左边距 2) 顶部 3) 右边距 (相对于主视图)

## NSLayoutConstraint Style

```
NSLayoutConstraint(item: myView, attribute: NSLayoutAttribute.Width, relatedBy:  
NSLayoutRelation.Equal, toItem: nil, attribute: NSLayoutAttribute.NotAnAttribute, multiplier: 1,  
constant: 200).active = true  
NSLayoutConstraint(item: myView, attribute: NSLayoutAttribute.Height, relatedBy:  
NSLayoutRelation.Equal, toItem: nil, attribute: NSLayoutAttribute.NotAnAttribute, multiplier: 1,  
constant: 100).active = true
```

### Visual Format Language style

```
NSLayoutConstraint.constraintsWithVisualFormat("H:[myViewKey(200)]", options: [], metrics: nil,  
views: ["myViewKey": myView])  
NSLayoutConstraint.constraintsWithVisualFormat("V:[myViewKey(100)]", options: [], metrics: nil,  
views: ["myViewKey": myView])
```

### Center in container

#### Anchor Style

```
myView.centerXAnchor.constraintEqualToAnchor(view.centerXAnchor).active = true  
myView.centerYAnchor.constraintEqualToAnchor(view.centerYAnchor).active = true
```

## NSLayoutConstraint Style

```
NSLayoutConstraint(item: myView, attribute: NSLayoutAttribute.CenterX, relatedBy:  
NSLayoutRelation.Equal, toItem: view, attribute: NSLayoutAttribute.CenterX, multiplier: 1,  
constant: 0).active = true  
NSLayoutConstraint(item: myView, attribute: NSLayoutAttribute.CenterY, relatedBy:  
NSLayoutRelation.Equal, toItem: view, attribute: NSLayoutAttribute.CenterY, multiplier: 1,  
constant: 0).active = true
```

### Visual Format Language style

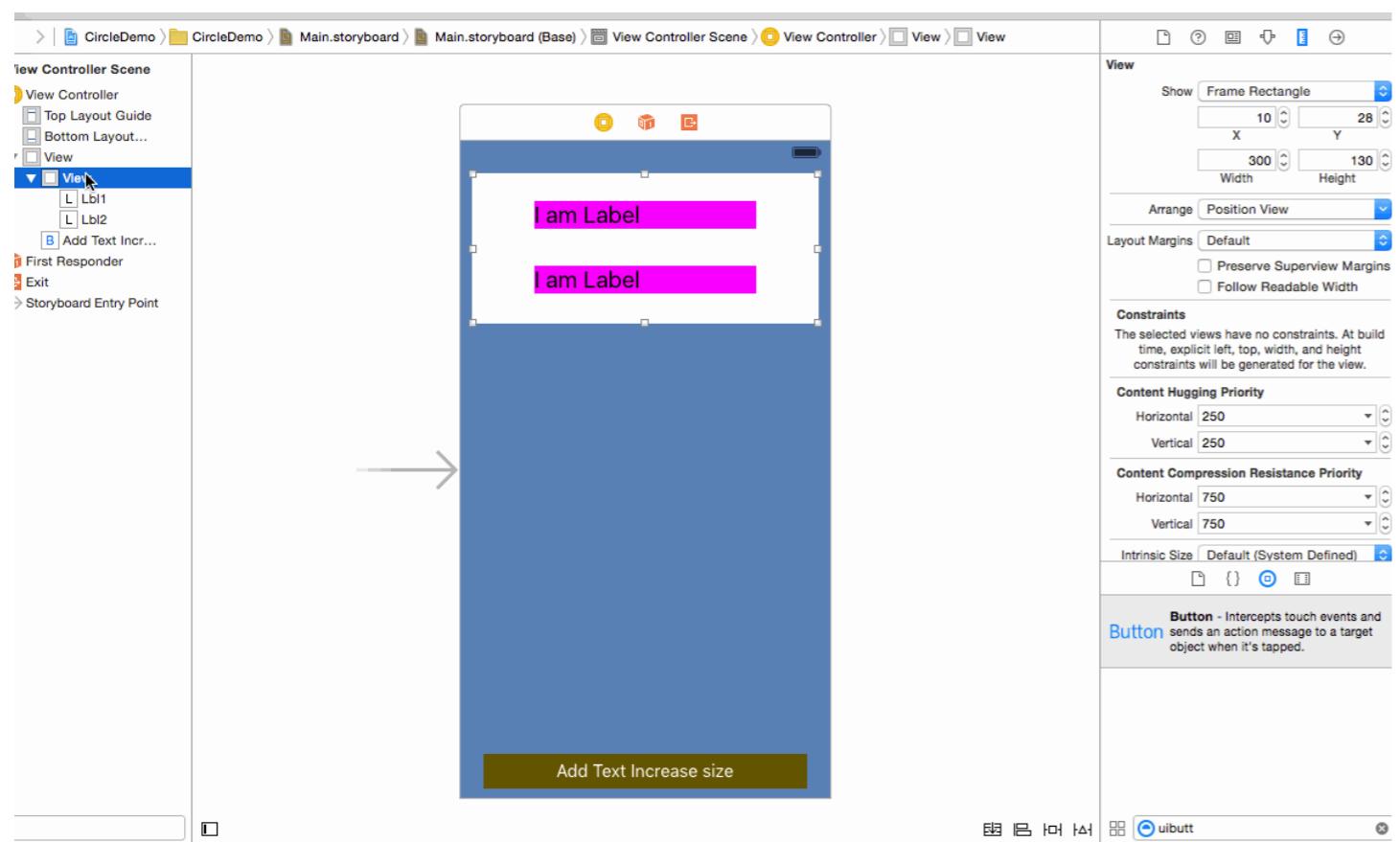
```
NSLayoutConstraint.constraintsWithVisualFormat("V:[viewKey]-(<=0)-[myViewKey]", options:  
NSLayoutFormatOptions.AlignAllCenterX, metrics: nil, views: ["myViewKey": myView, "viewKey": view])  
NSLayoutConstraint.constraintsWithVisualFormat("H:[viewKey]-(<=0)-[myViewKey]", options:  
NSLayoutFormatOptions.AlignAllCenterY, metrics: nil, views: ["myViewKey": myView, "viewKey": view])
```

## Section 67.4: UILabel & Parentview size According to Text in UILabel

### Step by Step Guide:

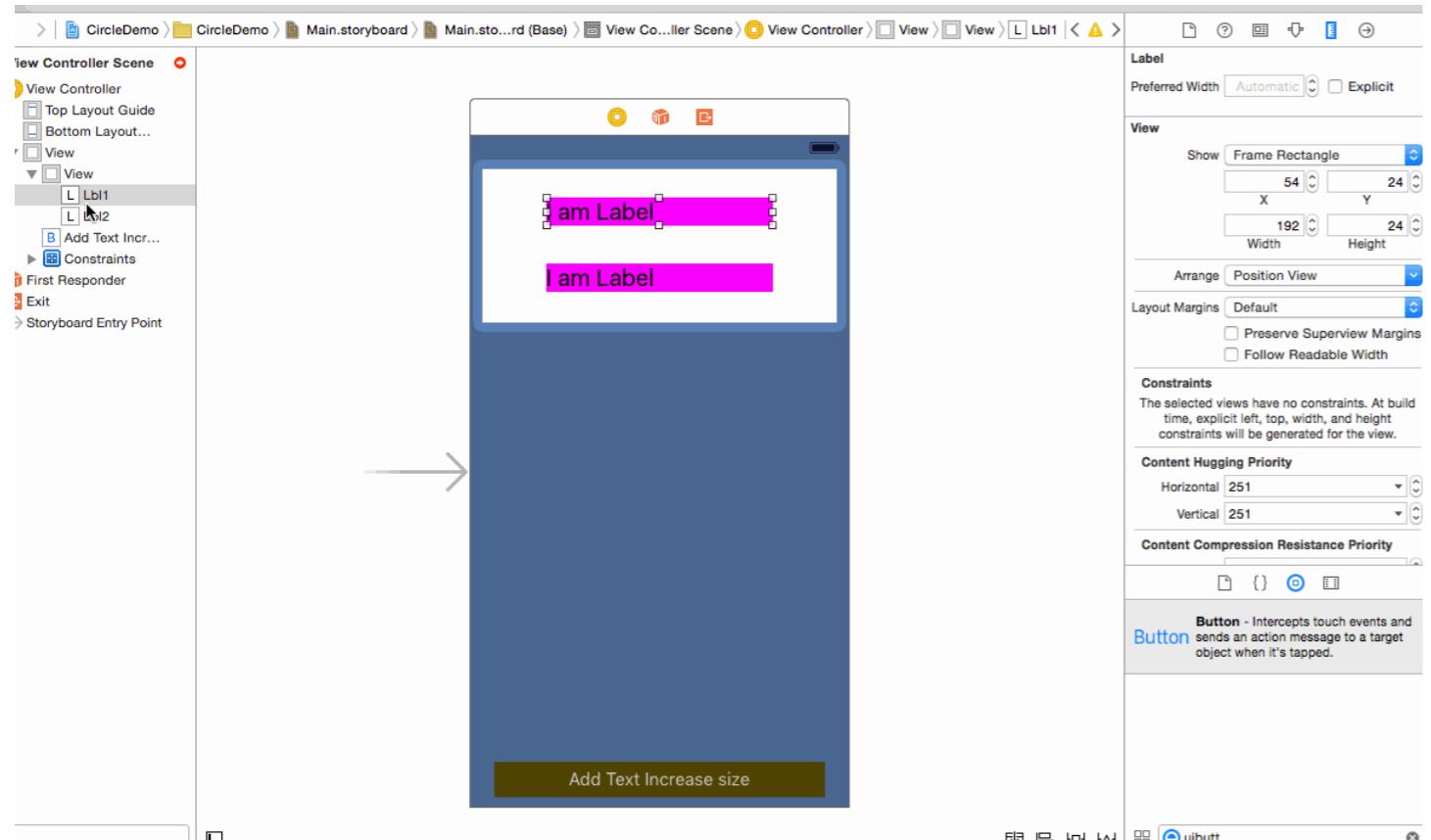
#### Step 1: Set constraint to UIView

1. Leading. 2) Top. 3) Trailing. (From mainview)



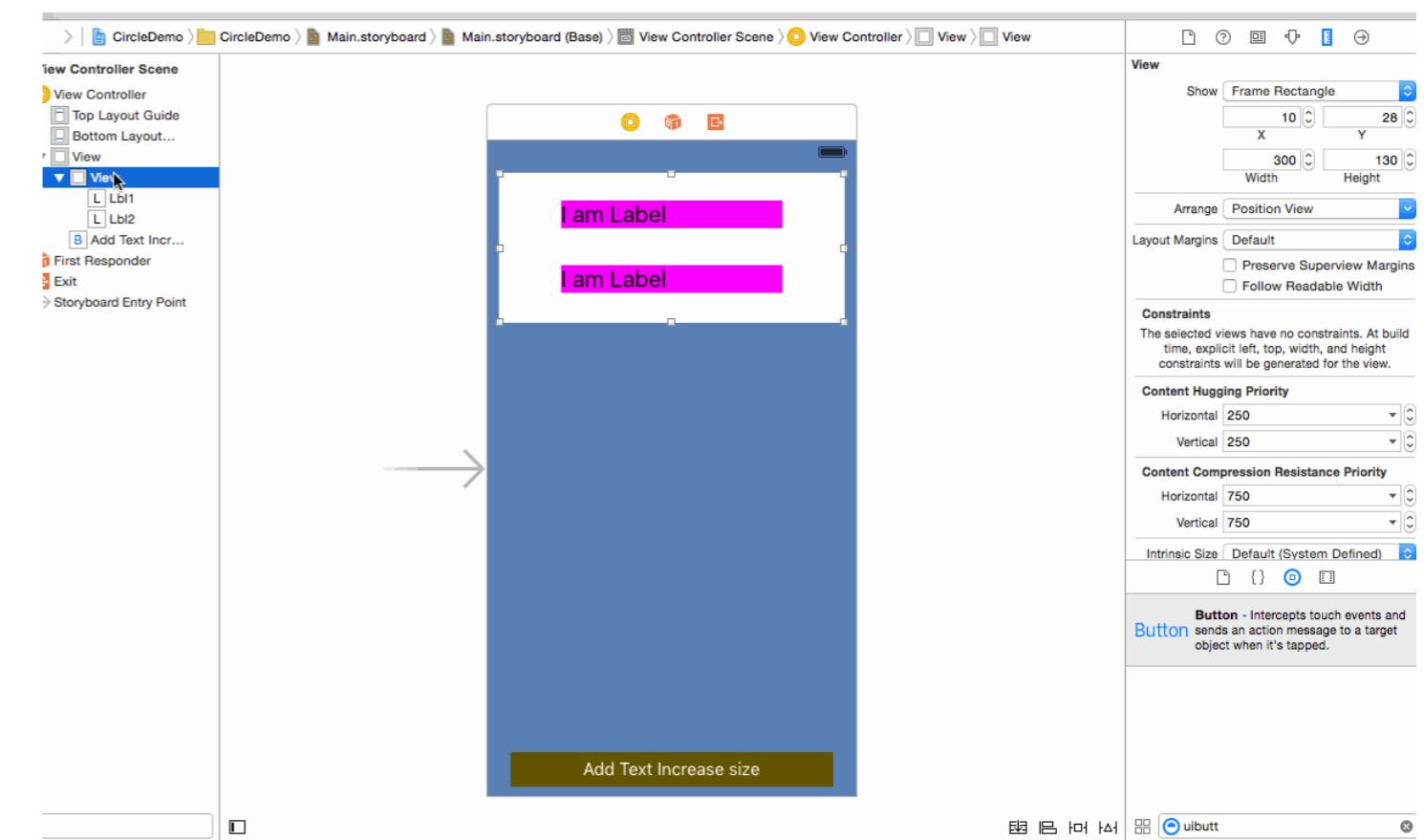
## 步骤2：设置标签1的约束

1.左边距 2) 顶部 3) 右边距 (相对于其父视图)



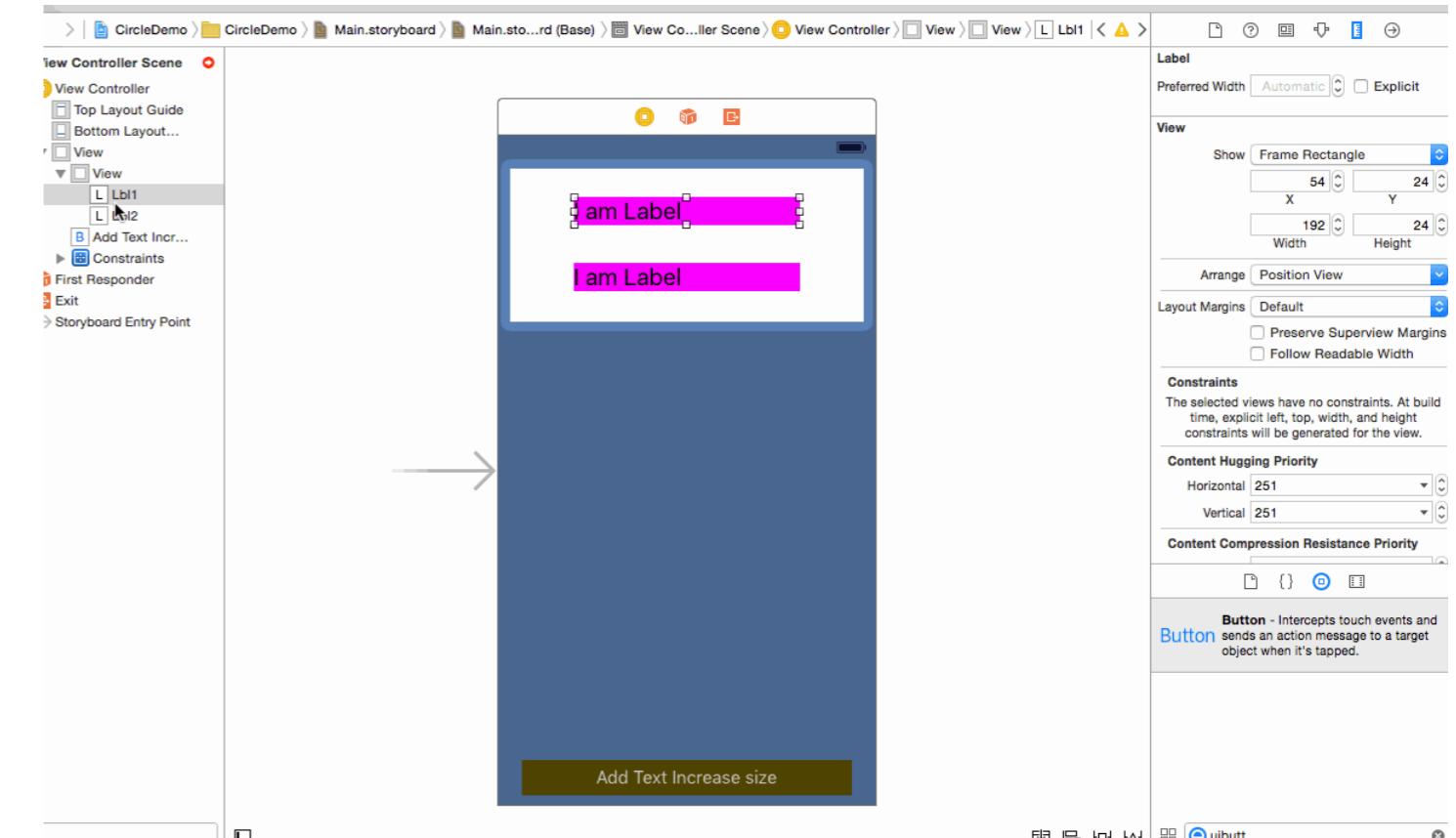
## 步骤3：设置标签2的约束

1.左边距 2) 顶部 3) 右边距 (相对于其父视图)



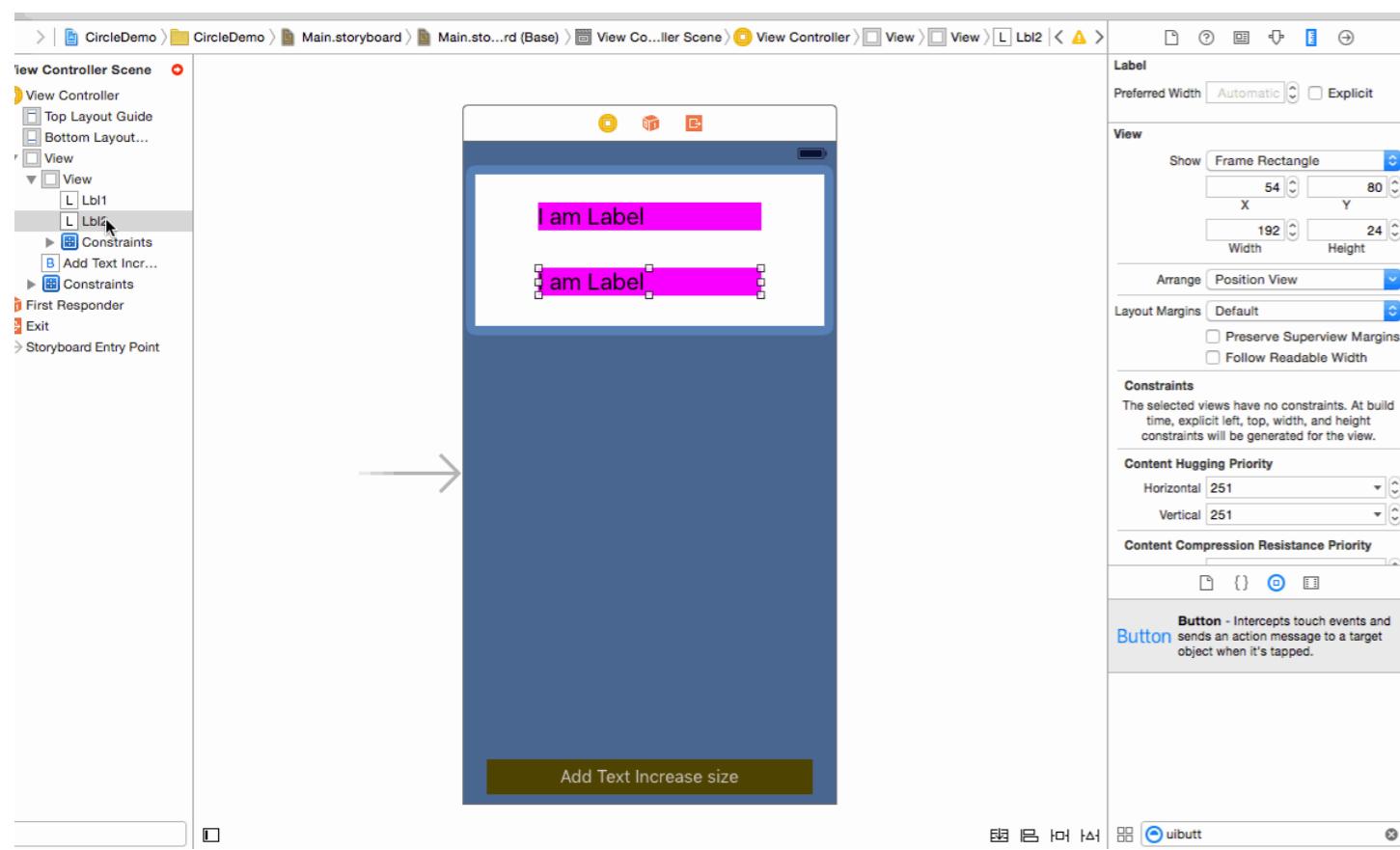
## Step 2: Set constrain to Label 1

1. Leading 2) Top 3) Trailing (From its superview)

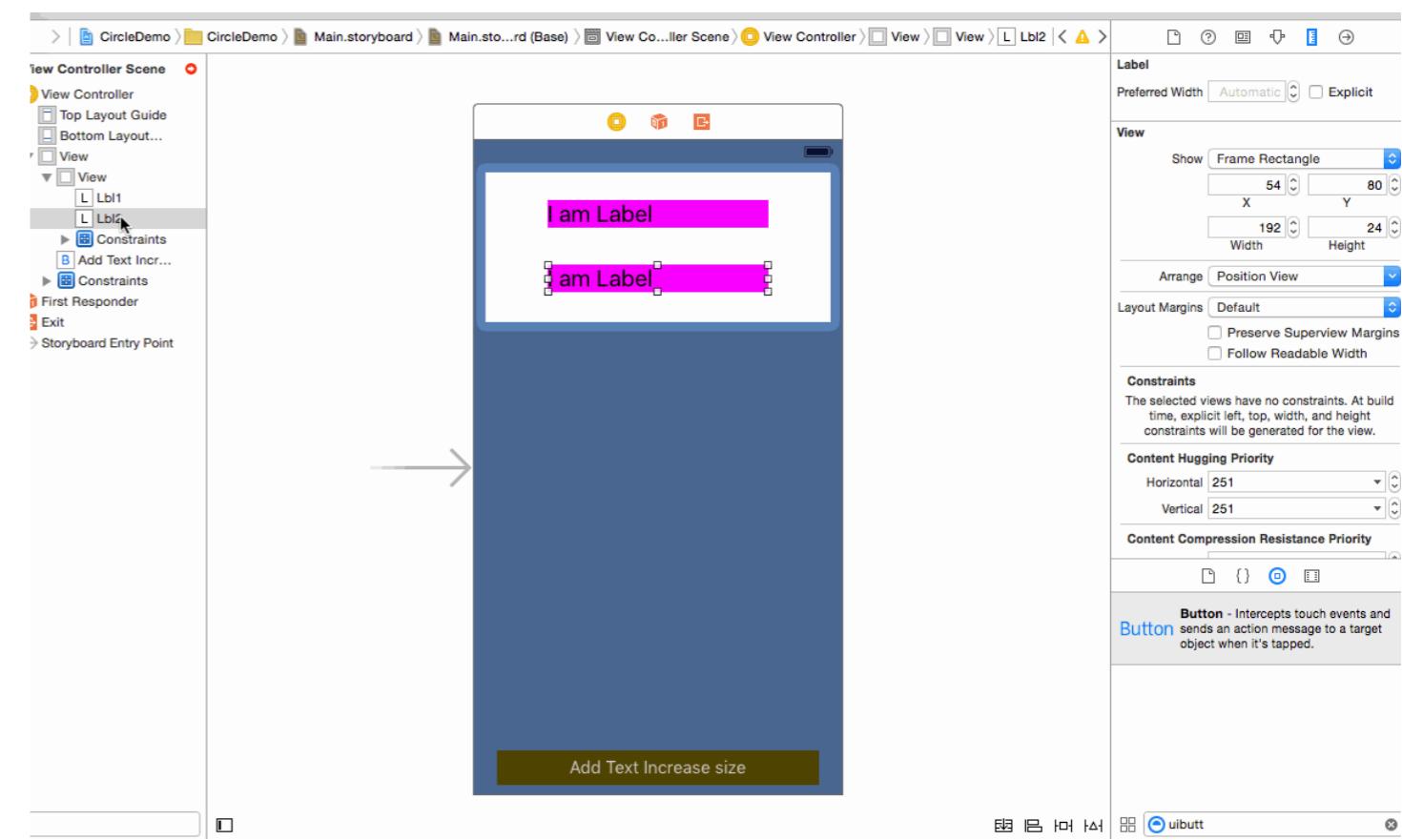


## Step 3: Set constraint to Label 2

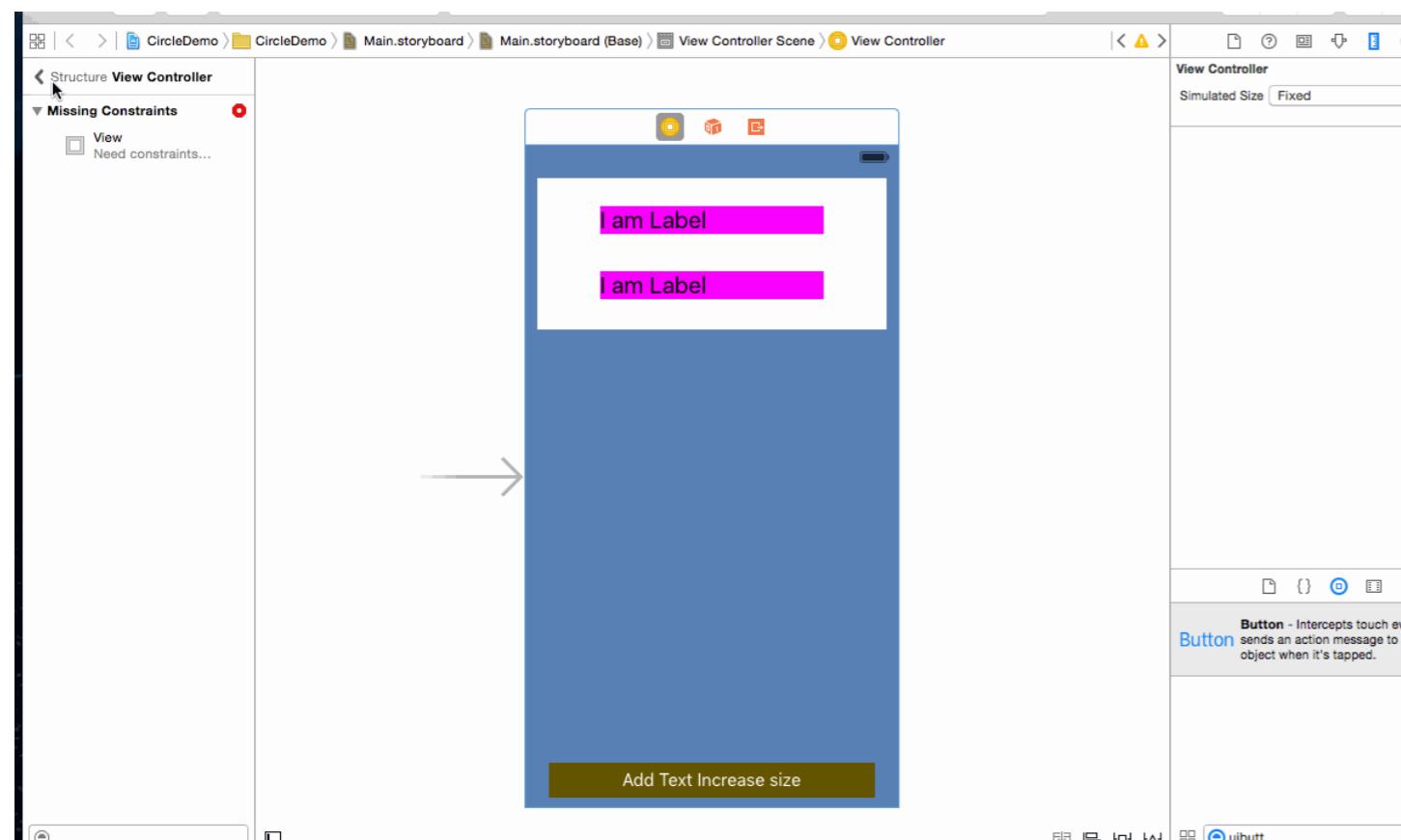
1. Leading 2) Top 3) Trailing (From its superview)



步骤4：最棘手的部分 从UIView底部给UILabel设置约束。



Step 4: Most tricky give a bottom to UILabel from UIView .



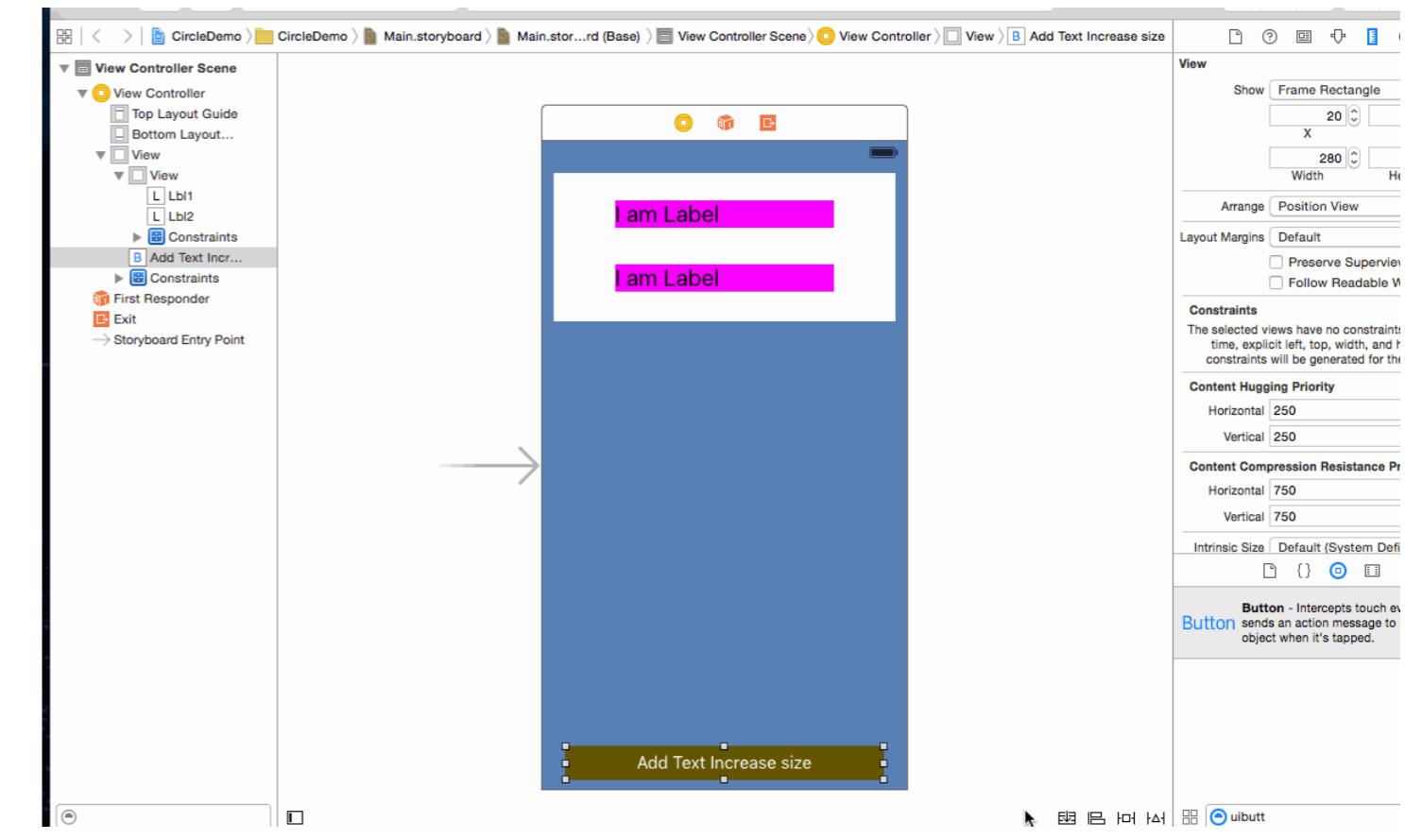
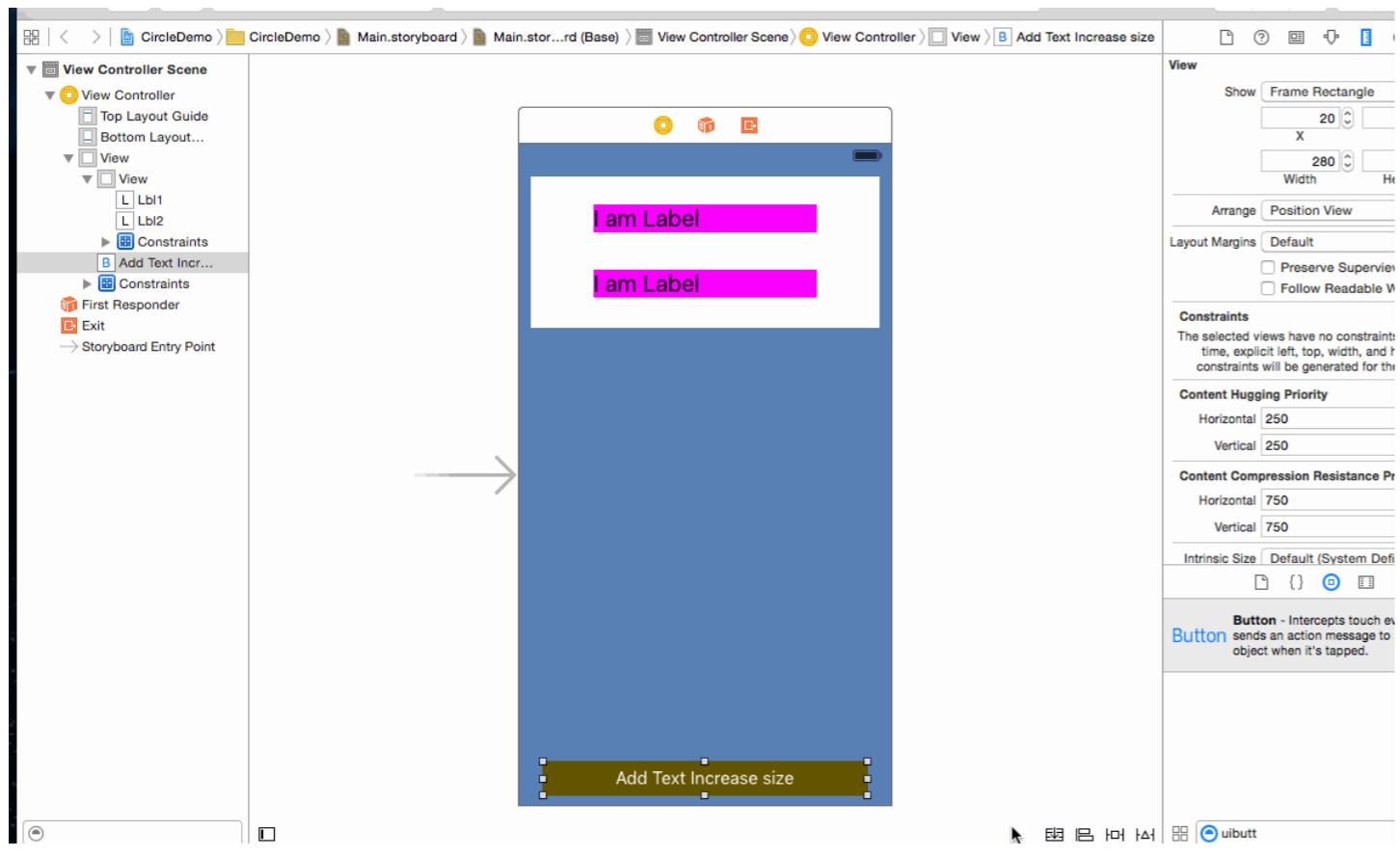
步骤5：(可选) 为UIButton设置约束

1. 左侧
2. 底部
3. 右侧
4. 固定高度 (相对于主视图)



Step 5: (Optional) Set constrain to UIButton

1. Leading
- 2) Bottom
- 3) Trailing
- 4) Fixed Height (From mainview)



输出：

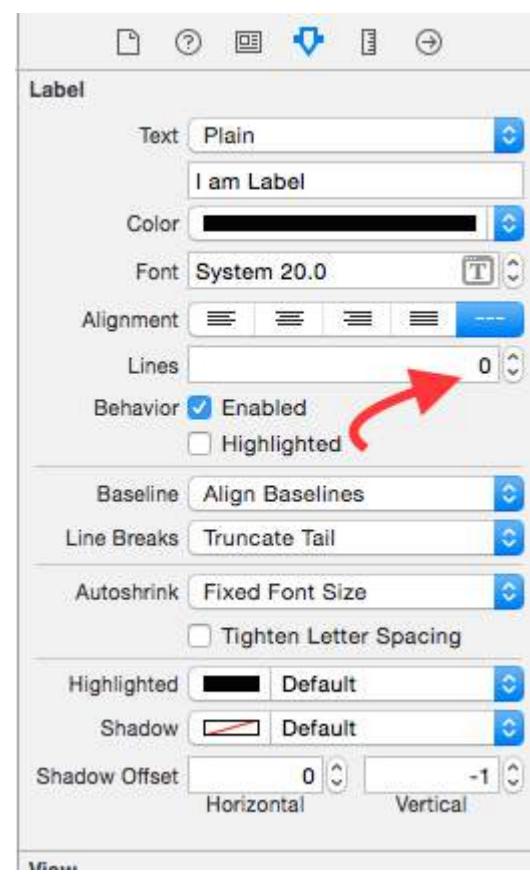
Output:



注意：确保你已在标签属性中将行数设置为0。



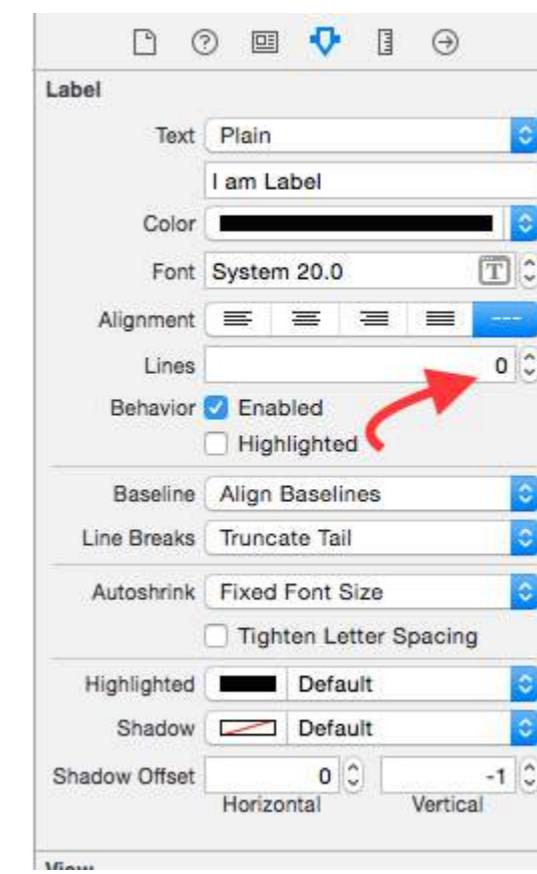
**Note:** Make sure you have set Number of lines =0 in Label property.



希望这些信息足以理解根据UILabel高度自动调整UIView大小以及根据文本自动调整UILabel大小。

## 第67.5节：UILabel的固有尺寸

我们需要创建一个视图，该视图包含一个图像前缀和文本。文本长度可变。我们需要实现的效果是图像+文本始终居中于父视图。



I hope this info enough to understand Autoresize UIView according to UILabel's height and Autoresize UILabel According to text.

## Section 67.5: UILabel Intrinsic Size

We have to create a view which will have a image prefix to a text. text could be of variable length. We have to achieve a result where in Image + text is always in center of a parent view.

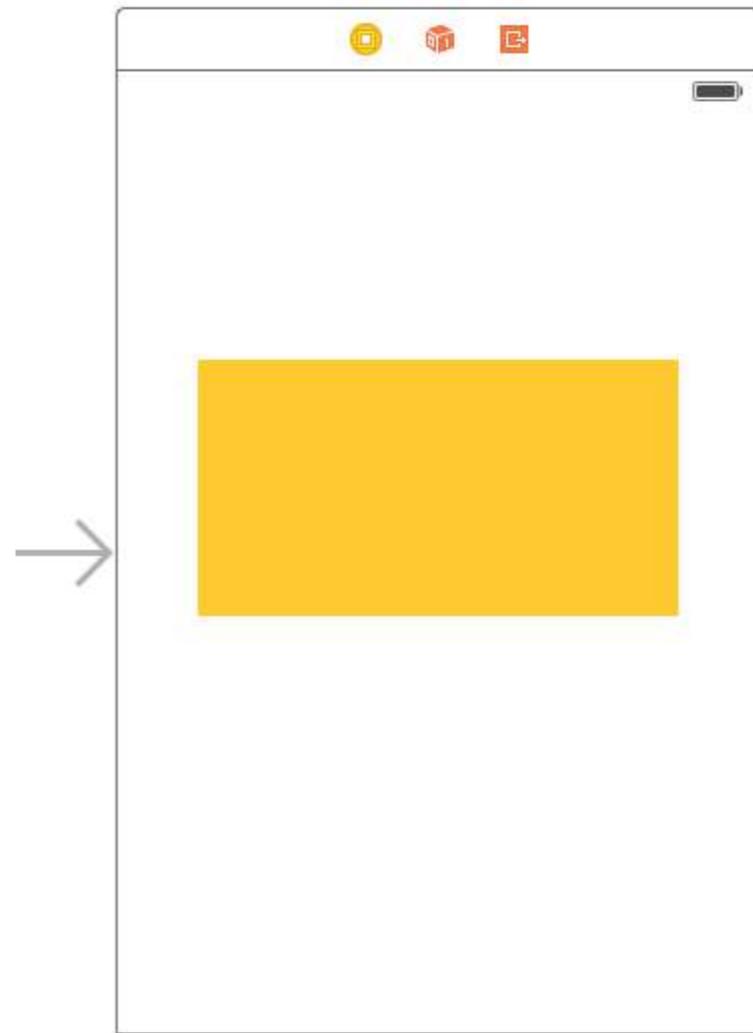
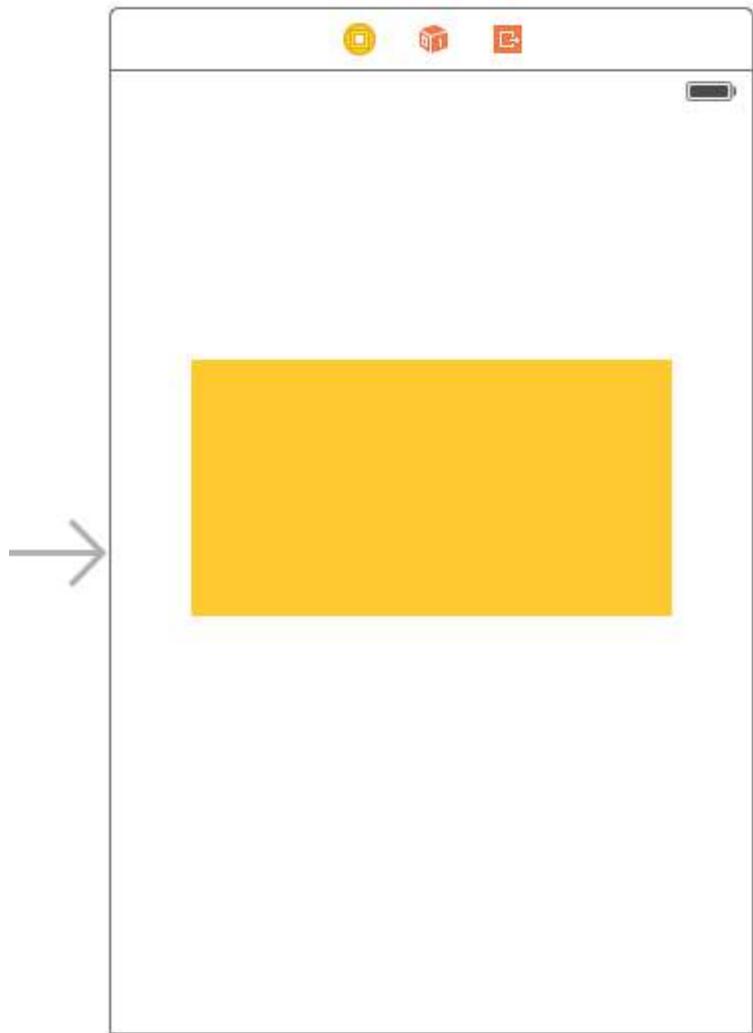


---

步骤 1：首先创建一个单视图项目，命名为你选择的名称，然后打开故事板的第一个视图。拖动一个大小合适的视图，并将其背景颜色设置为黄色。我已将我的视图控制器调整为 3.5”。最终的视图应类似如下所示

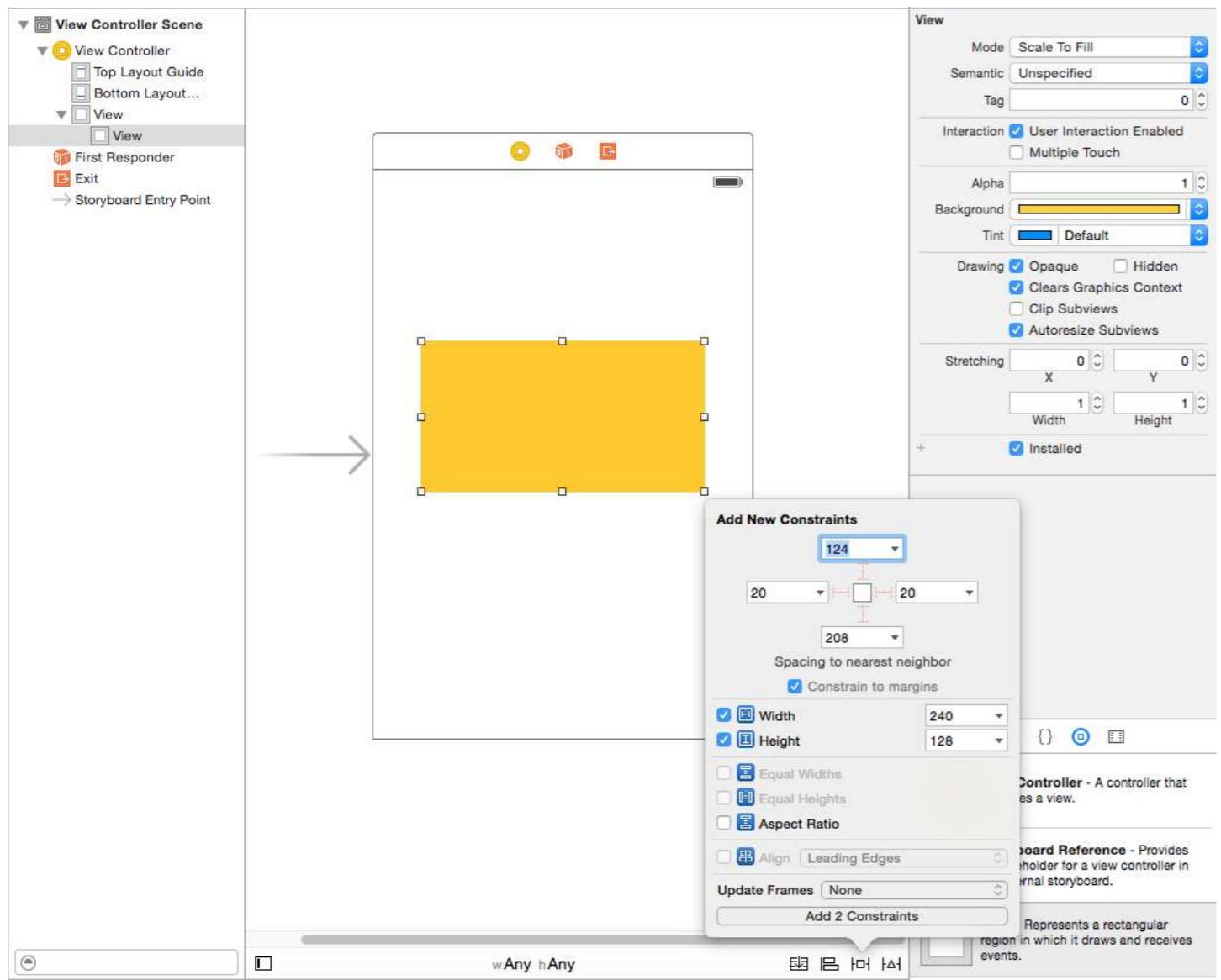
---

**Step 1:** First create a single view project and name it something of your choice and open the story board fist view.Drag a view with some reasonable size and set its background color to yellow.I have resized my viewcontroller to 3.5".The resultant view should look some thing like this

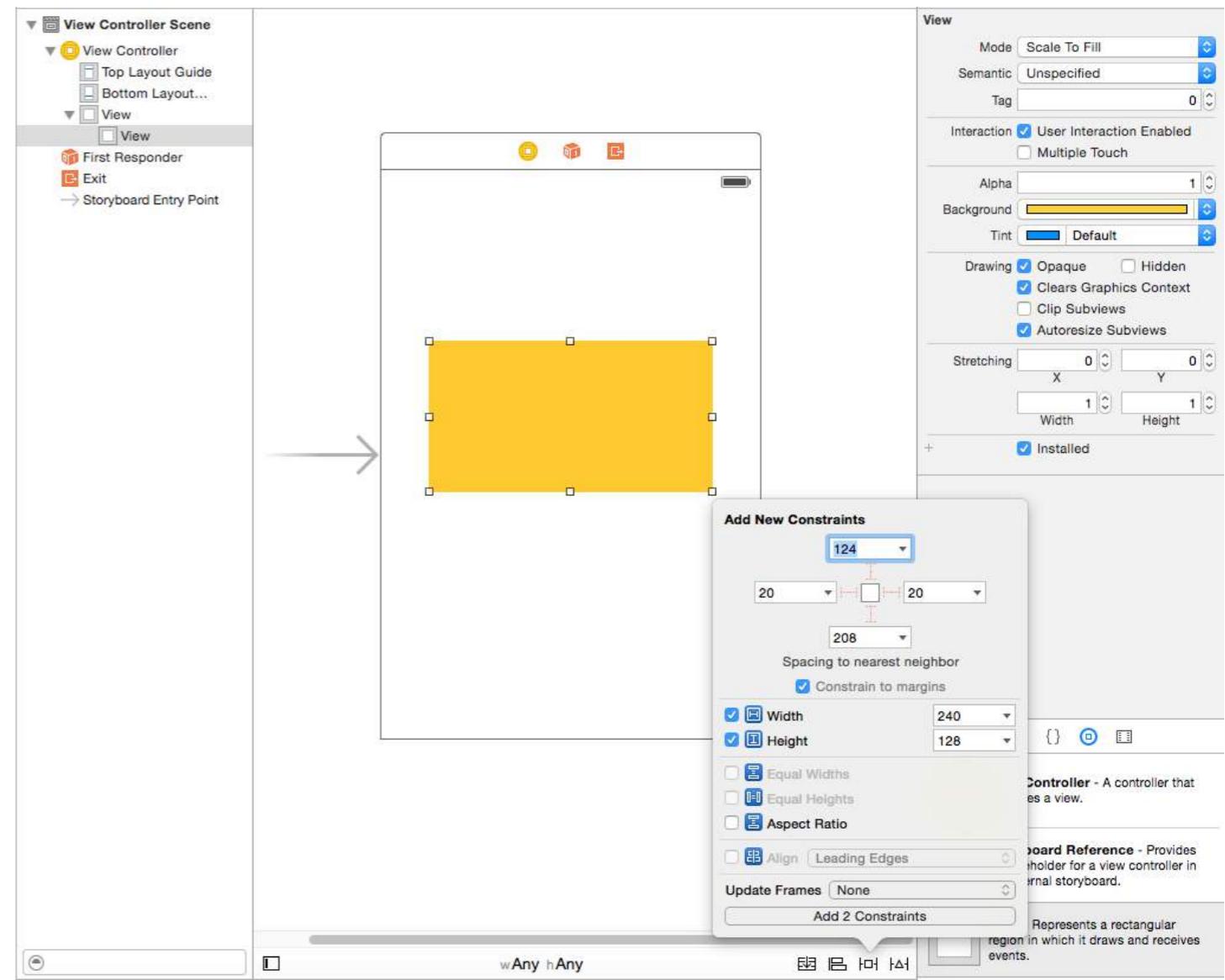


步骤 2：现在我们将为黄色视图添加约束。首先添加宽度和高度约束（等一下，我们不是说视图宽度是动态的吗？好的，我们稍后会回到这个问题）。根据下图添加以下约束，不用在意宽度的具体数值，任何数值都可以，只要足够大以便我们能正确添加自动布局。

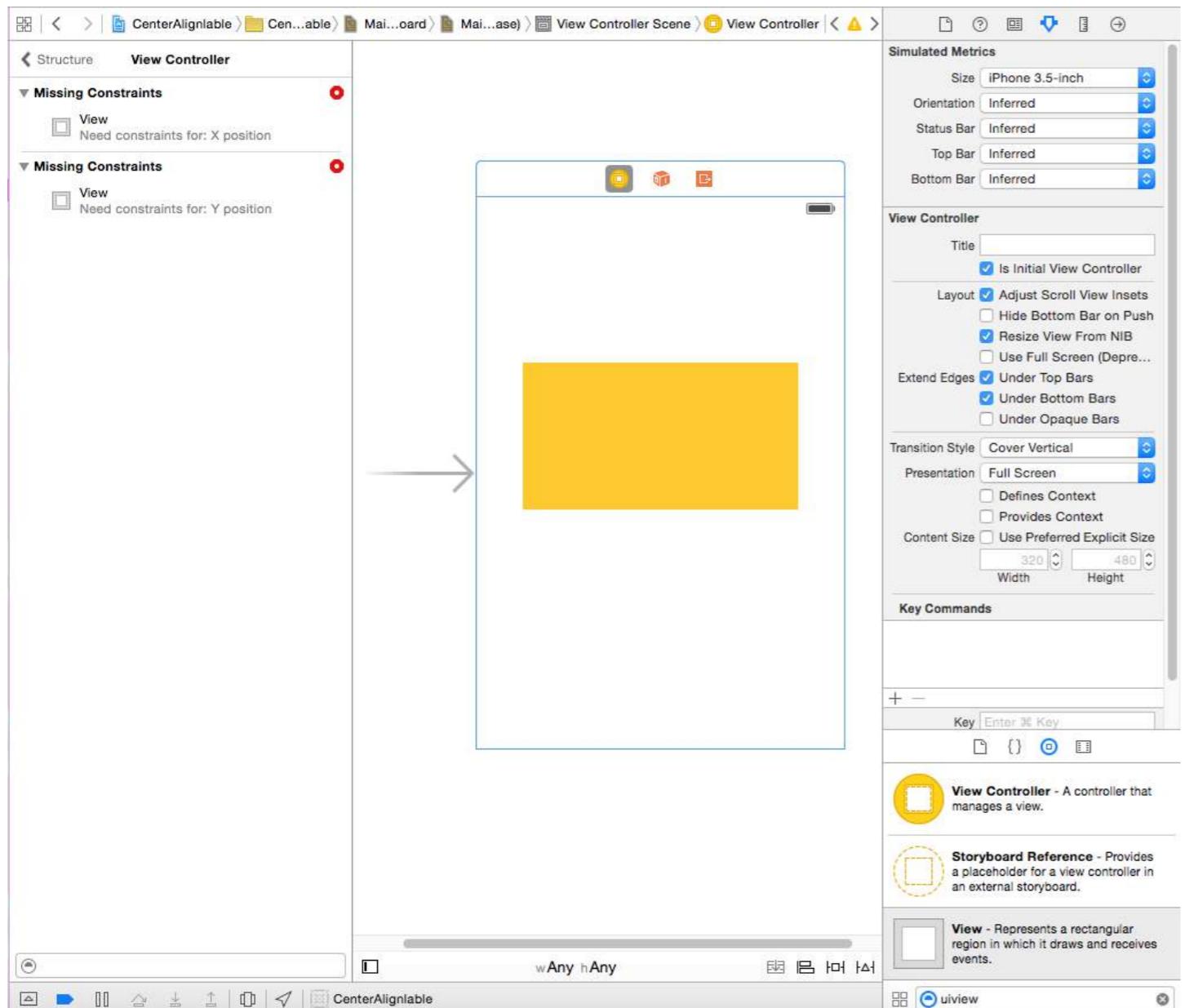
**Step 2:** Now we will add constraints to the yellow view .To begin with we will add width and height constraints (Wait a minute didn't we say that view will have dynamic width? Ok we will get back to it later) Add the following constraints as per the image below do not bother with width value any value will do just fine for width just keep it large enough so that we can add autolayouts properly.



添加这两个约束后，你会看到 XCode 出现如下图所示的错误，让我们来看一下并理解它们。



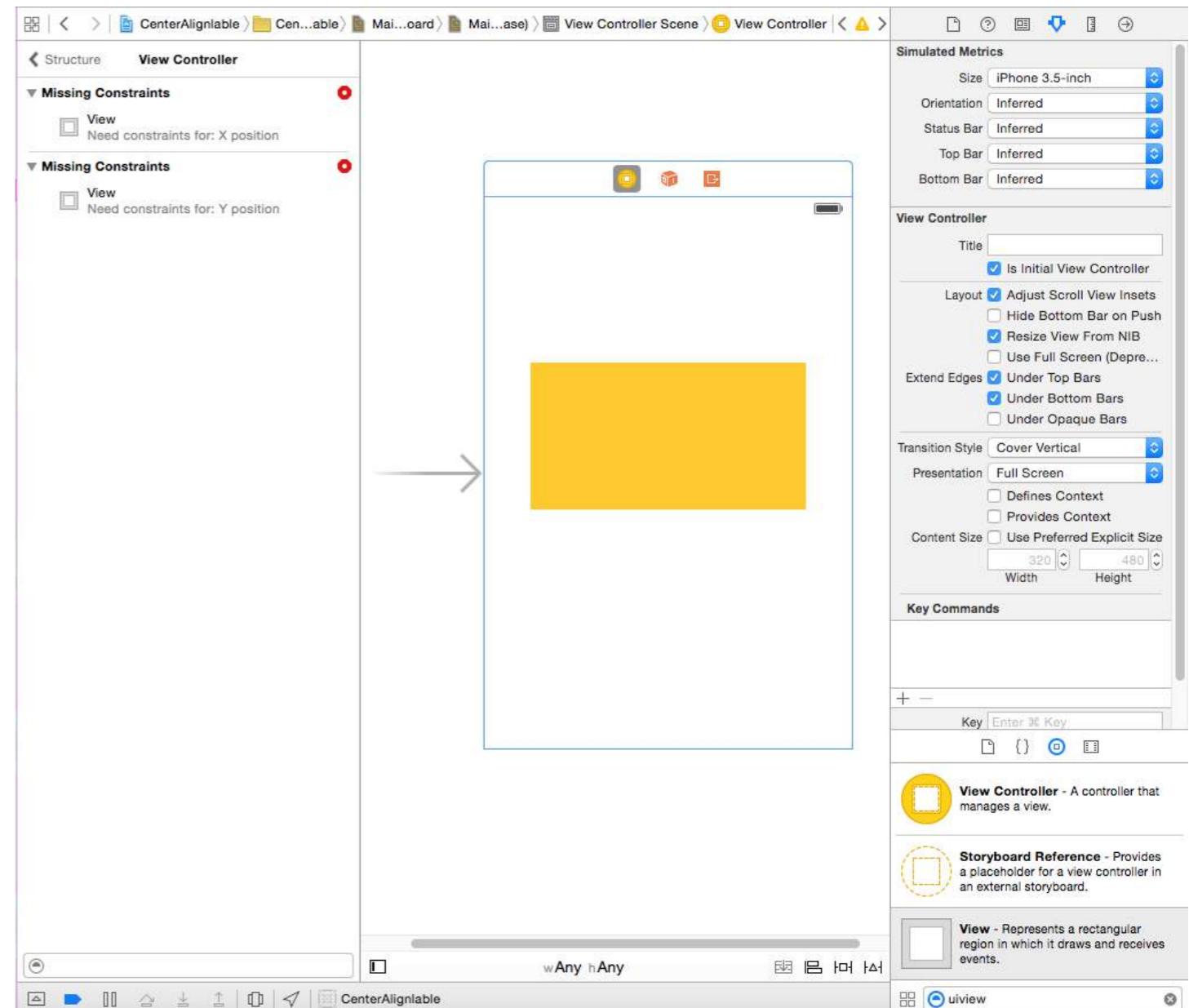
After adding these two constraints you will see that XCode is giving you errors as in below image lets see them and understand them.



我们有两个错误（红色表示错误）。如上所述，让我们重新审视歧义部分。

**缺失约束：需要约束 : X 位置 :-** 如上所述，我们已经给视图设置了宽度和高度，因此它的“边界（BOUNDS）”已定义，但我们没有给出它的起点，因此它的“框架（FRAME）”未定义。自动布局无法确定黄色视图的 X 位置。

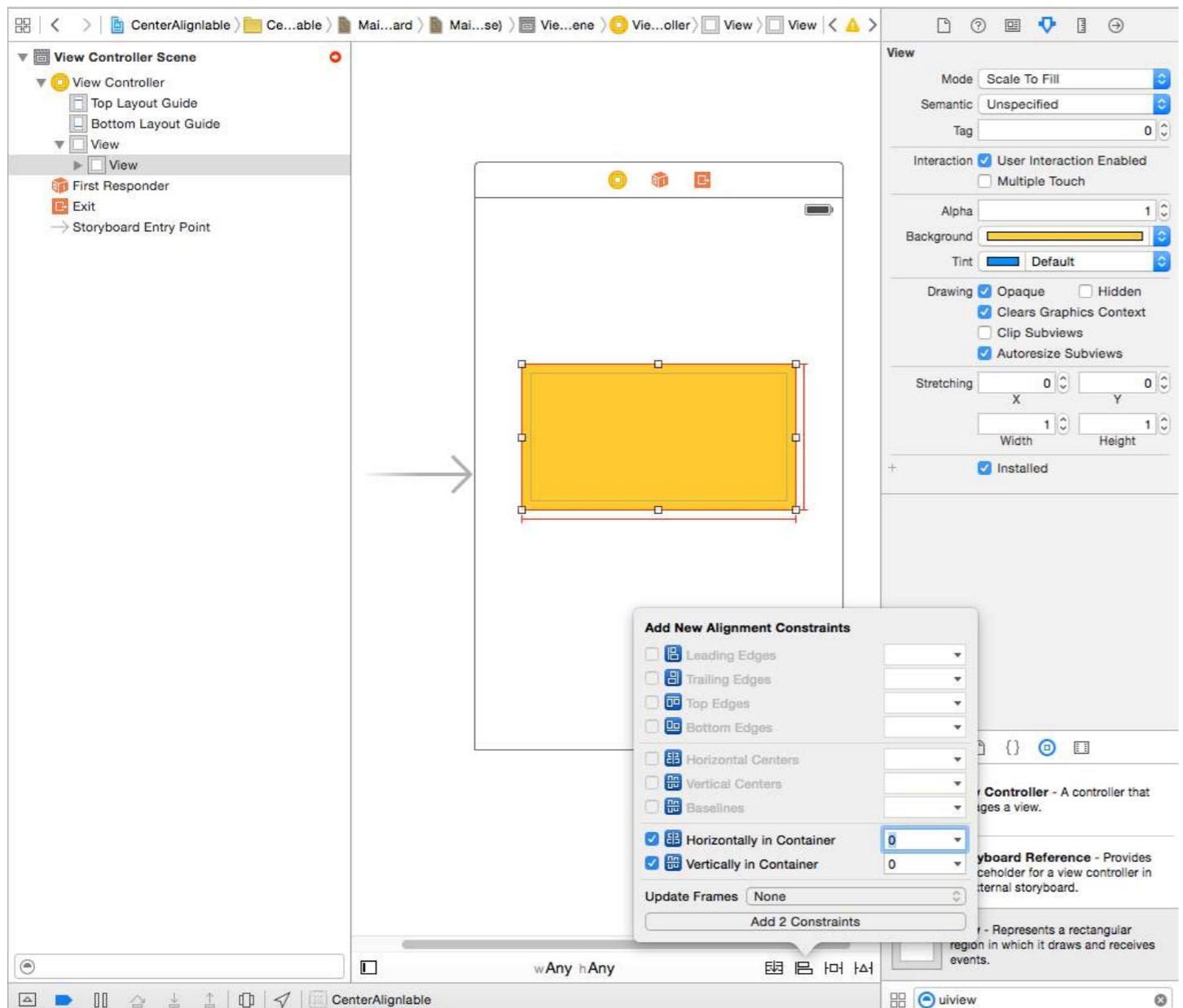
**缺失约束：需要约束 : Y 位置 :-** 如上所述，我们已经给视图设置了宽度和高度，因此它的“边界（BOUNDS）”已定义，但我们没有给出它的起点，因此它的“框架（FRAME）”未定义。自动布局无法确定黄色视图的 Y 位置。为了解决这个问题，我们必须给自动布局一些东西来确定 X 和 Y。由于我们不能直接设置框架，我们将采用自动布局的方式。根据下图添加以下约束，我稍后会解释。



We have two errors (red means error) As discussed above lets revisit the ambiguity part

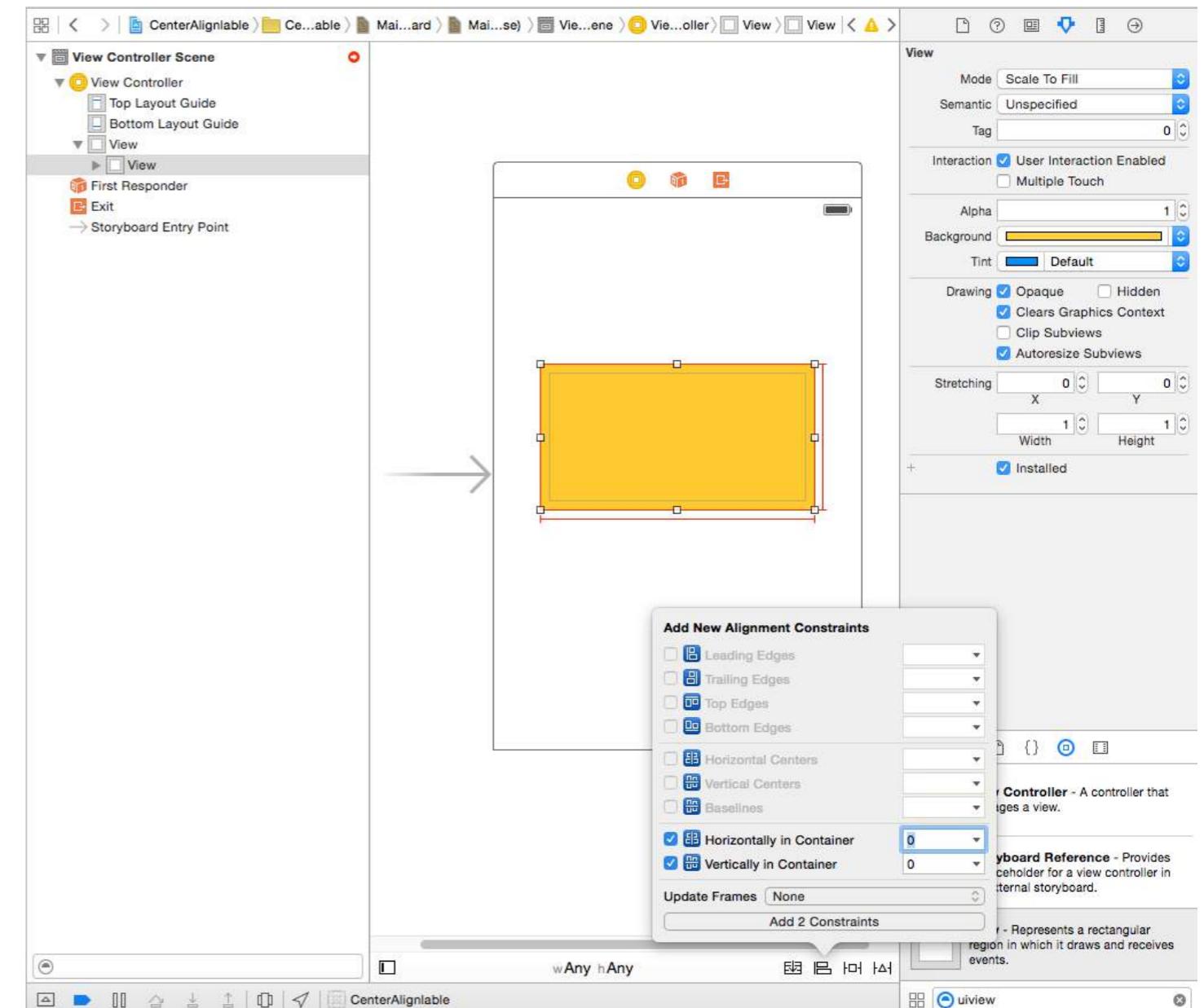
**Missing Constraints :** Need constraints for : X position :- As discussed above we have given the view a width and a height so its “BOUNDS” is defined but we have not given its origin so its “FRAME” is not defined. Autolayout is not able to determine what will be the X position of our yellow view

**Missing Constraints :** Need constraints for : Y position :- As discussed above we have given the view a width and a height so its “BOUNDS” is defined but we have not given its origin so its “FRAME” is not defined. Autolayout is not able to determine what will be the Y position of our yellow view To solve this we have to give autolayout some thing to resole X and Y. Since we cannot set frames we will do it autolayout way.Add following constraints as per the image given below I will explain it later



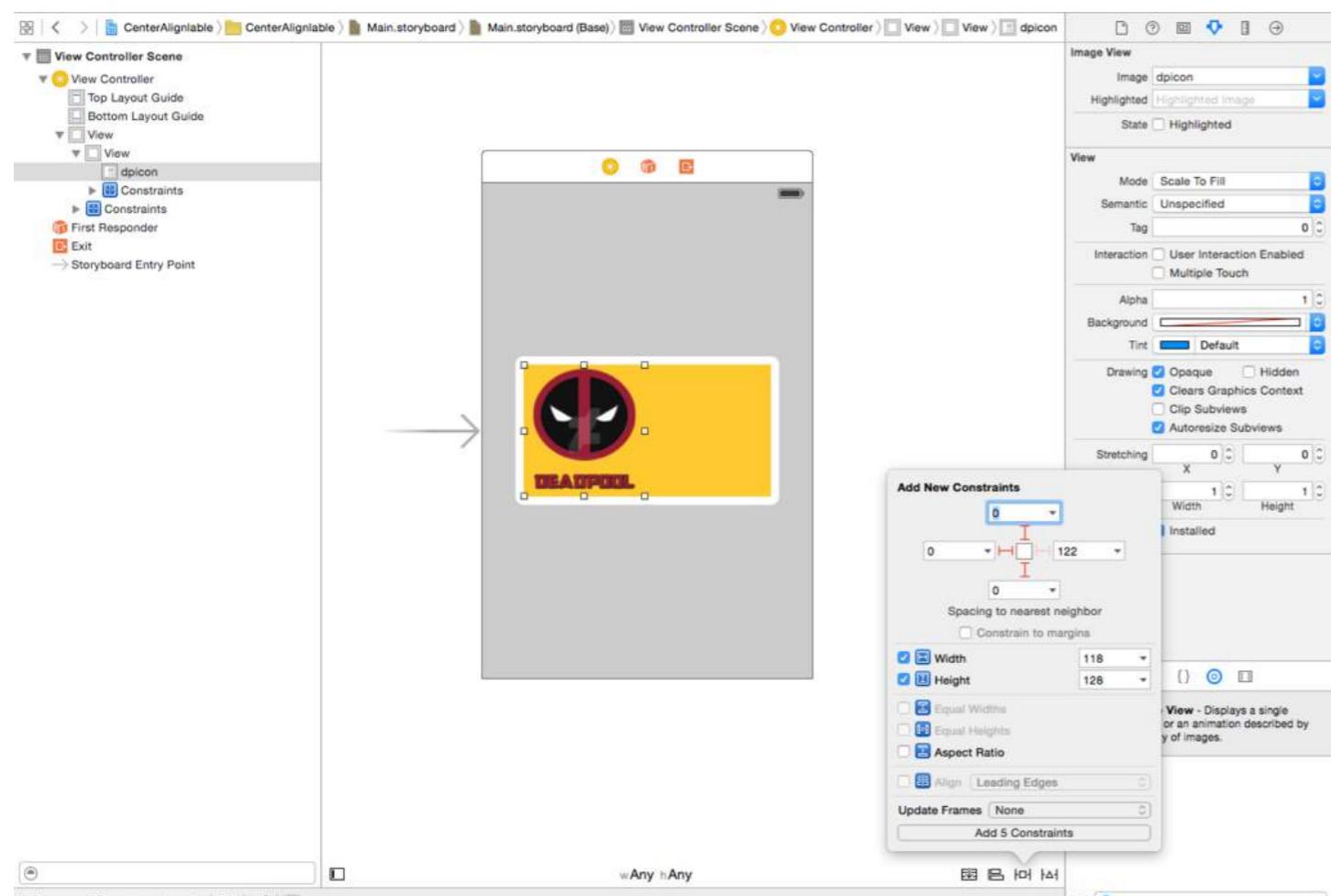
我们所做的是，添加了一个“垂直居中”和“水平居中”，这些约束告诉自动布局我们的黄色视图将始终水平居中：因此X被确定，垂直约束同理，Y也被确定。（你可能需要调整框架）。

步骤3：到现在为止，我们的基础黄色视图已经准备好了。我们将添加前缀图片作为黄色视图的子视图，并设置以下约束。你可以选择任何你喜欢的图片。



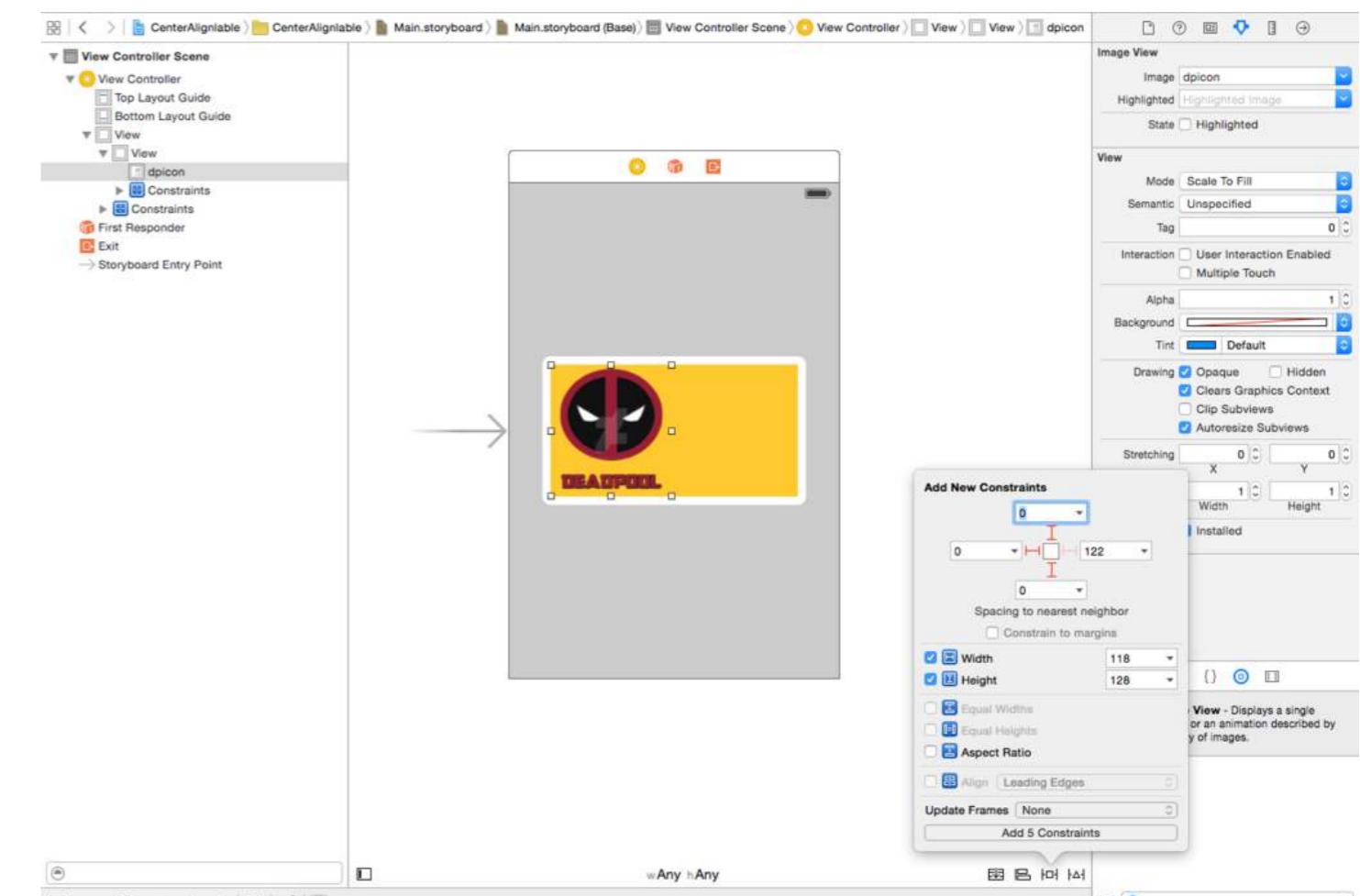
What we have done is, We have added a “Vertical Center” and “Horizontal Center” these constrain tell autolayout that our yellow view will always be in center Horizontally: so X is determined same is with vertical constraint and Y is determined.(you might have to adjust frame).

**Step 3:** By now our base yellow view is ready. We will add the prefix image as subview of our yellow view with following constraints. You can choose any image of your choice.



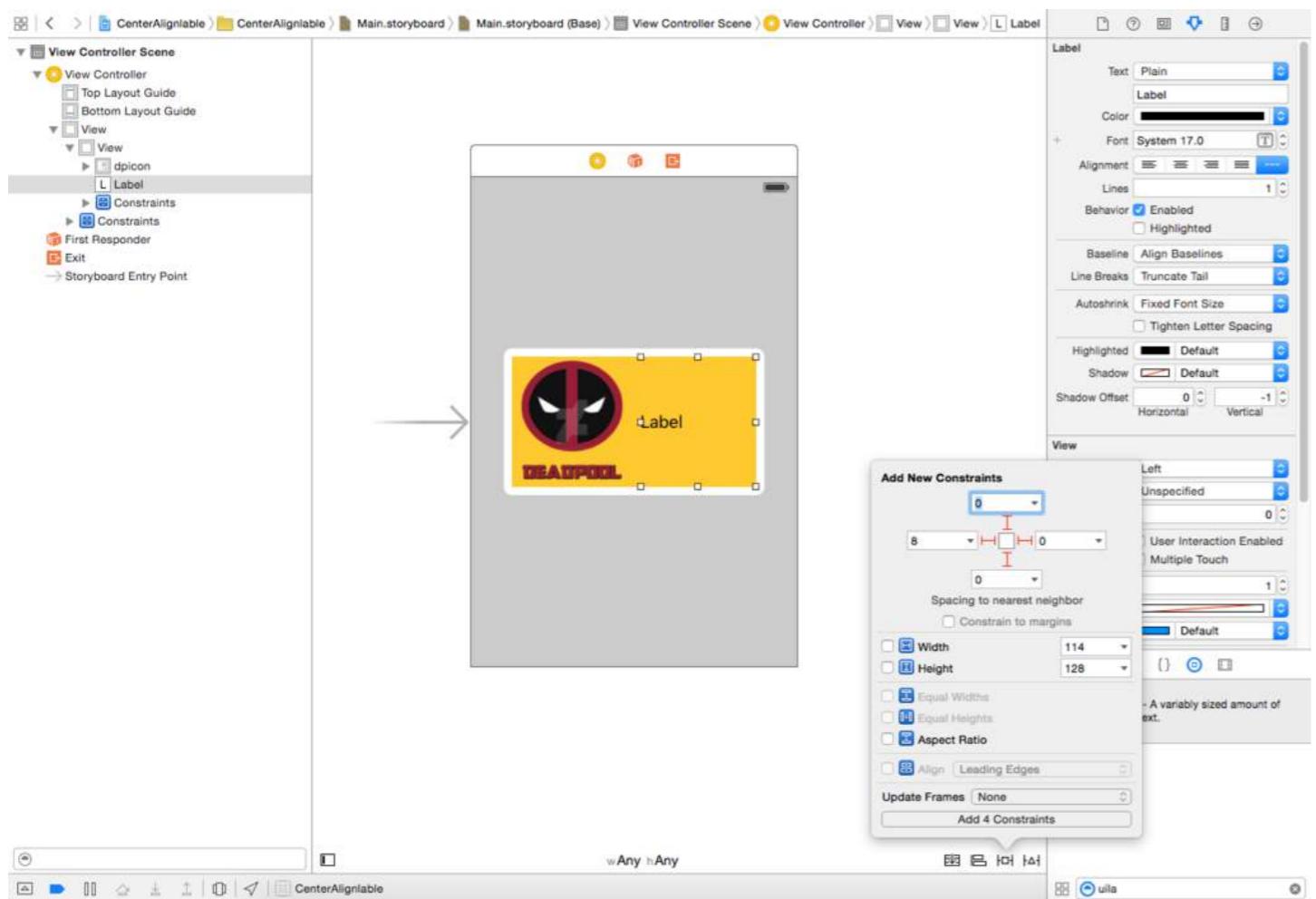
由于我们为前缀图片设定了固定尺寸，因此该图片视图的宽度和高度也是固定的。添加约束后，继续下一步。

**步骤4：**添加一个UILabel作为黄色视图的子视图，并添加以下约束



Since we have fixed dimension for our prefix image we will have fixed width height for this imageview. Add the constraints and proceed to next step.

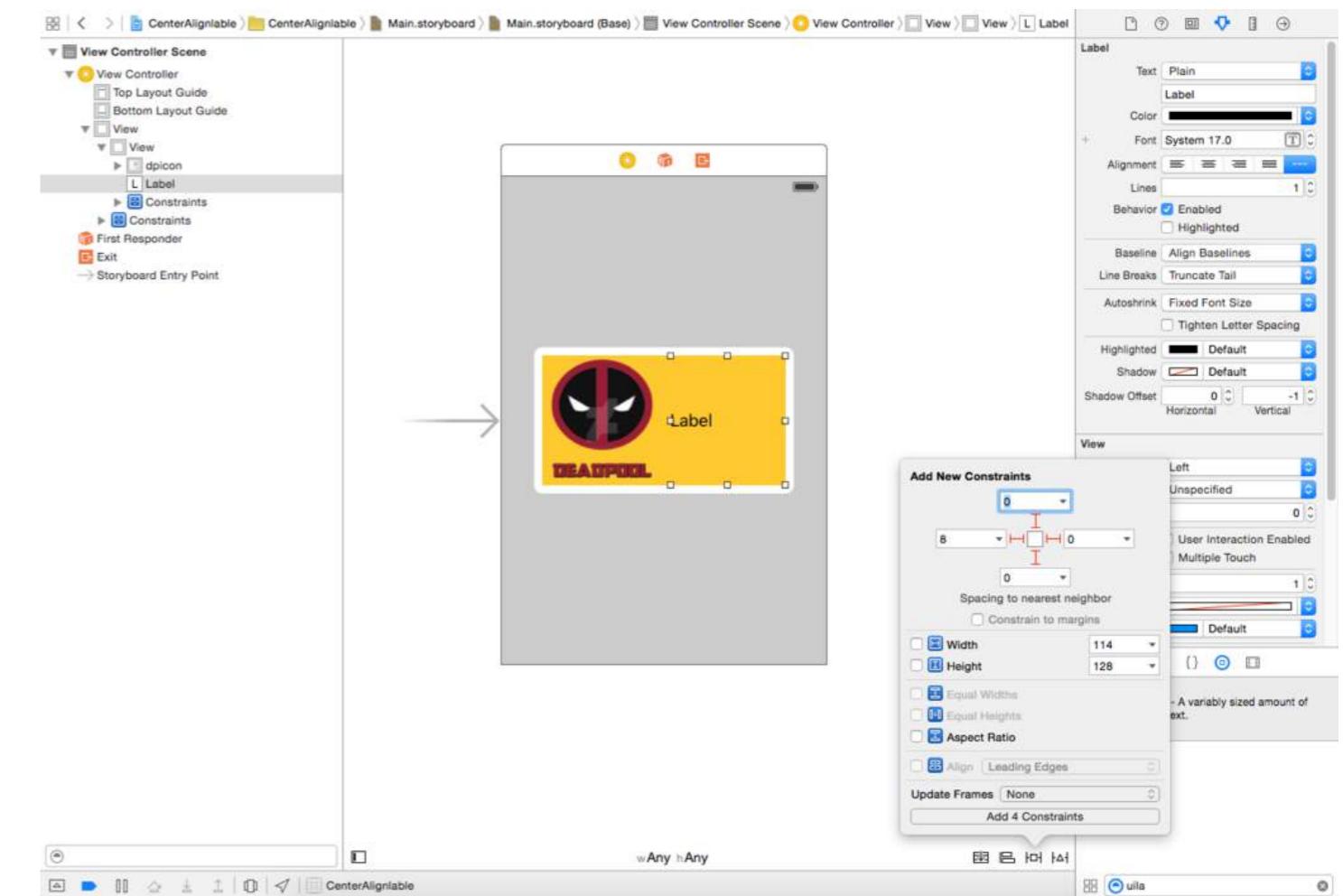
**Step4:** Add a UILabel as the sub view of our yellow view and add following constraints



如你所见，我只给UILabel设置了相对约束。它距离前缀图片8个点，距离黄色视图的顶部、尾部和底部均为0。由于我们希望宽度是动态的，所以不会设置宽度或高度约束。

问：为什么现在没有出现任何错误，我们没有设置宽度和高度？答：只有当自动布局无法解析渲染视图所必需的内容（如高度、宽度或位置）时，才会出现错误或警告。由于我们的标签相对于黄色视图和前缀图片，其框架已明确定义，自动布局能够计算出UILabel的框架。

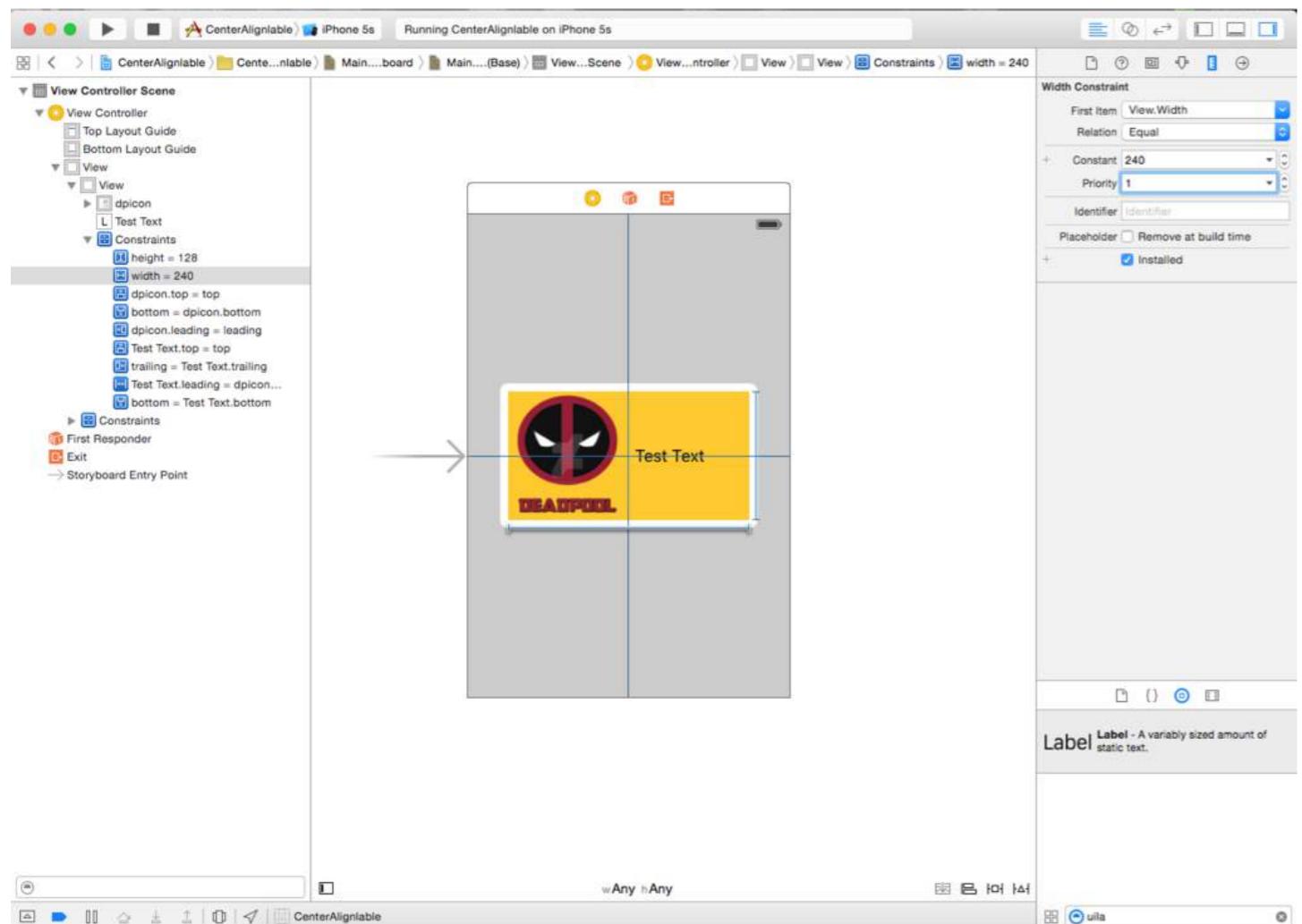
步骤5：现在回想一下，我们之前给黄色视图设置了固定视图，但我们希望它根据UILabel的文本动态变化。因此，我们将修改黄色视图的宽度约束。黄色视图的宽度约束对于解决歧义是必要的，但我们希望在运行时根据UILabel的内容覆盖它。所以我们选择黄色视图，进入尺寸检查器，将宽度约束的优先级降低到1，以便它被覆盖。请参见下图。



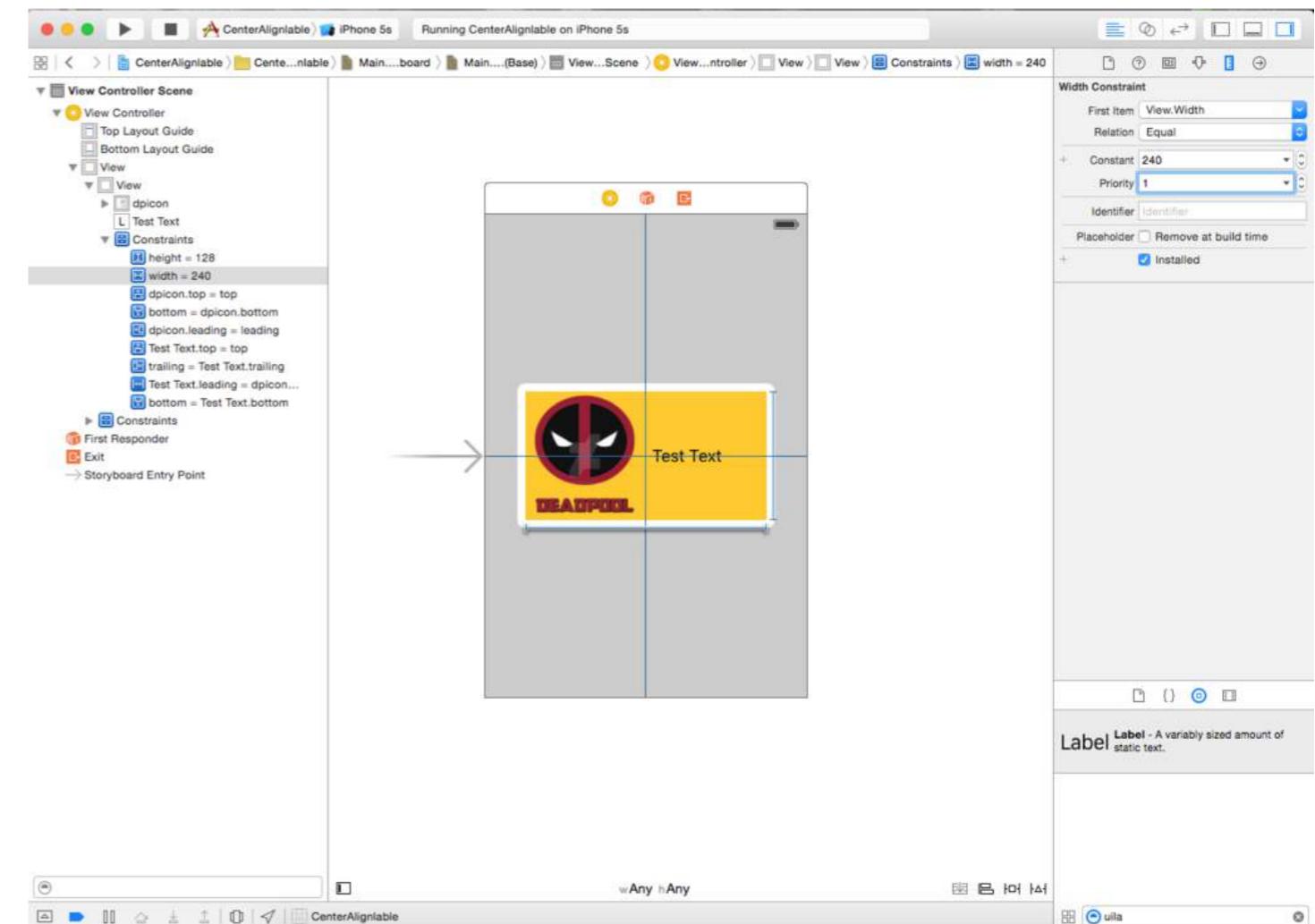
As you can see i have given only relative constraints to our UILabel.Its 8 points from prefix image and 0,0,0 top trailing and bottom from yellow view.Since we want the width to be dynamic we will not give width or height constraints.

Q: Why we are not getting any errors now , we have not given any width and height? Ans:- We get error or warning only when auto layout is not able to resolve any thing which is must to render a view on screen.Be it height width or origin.Since our label is relative to yellow view and prefix image and their frames is well defined autolayout is able to calculate the frame of our Label.

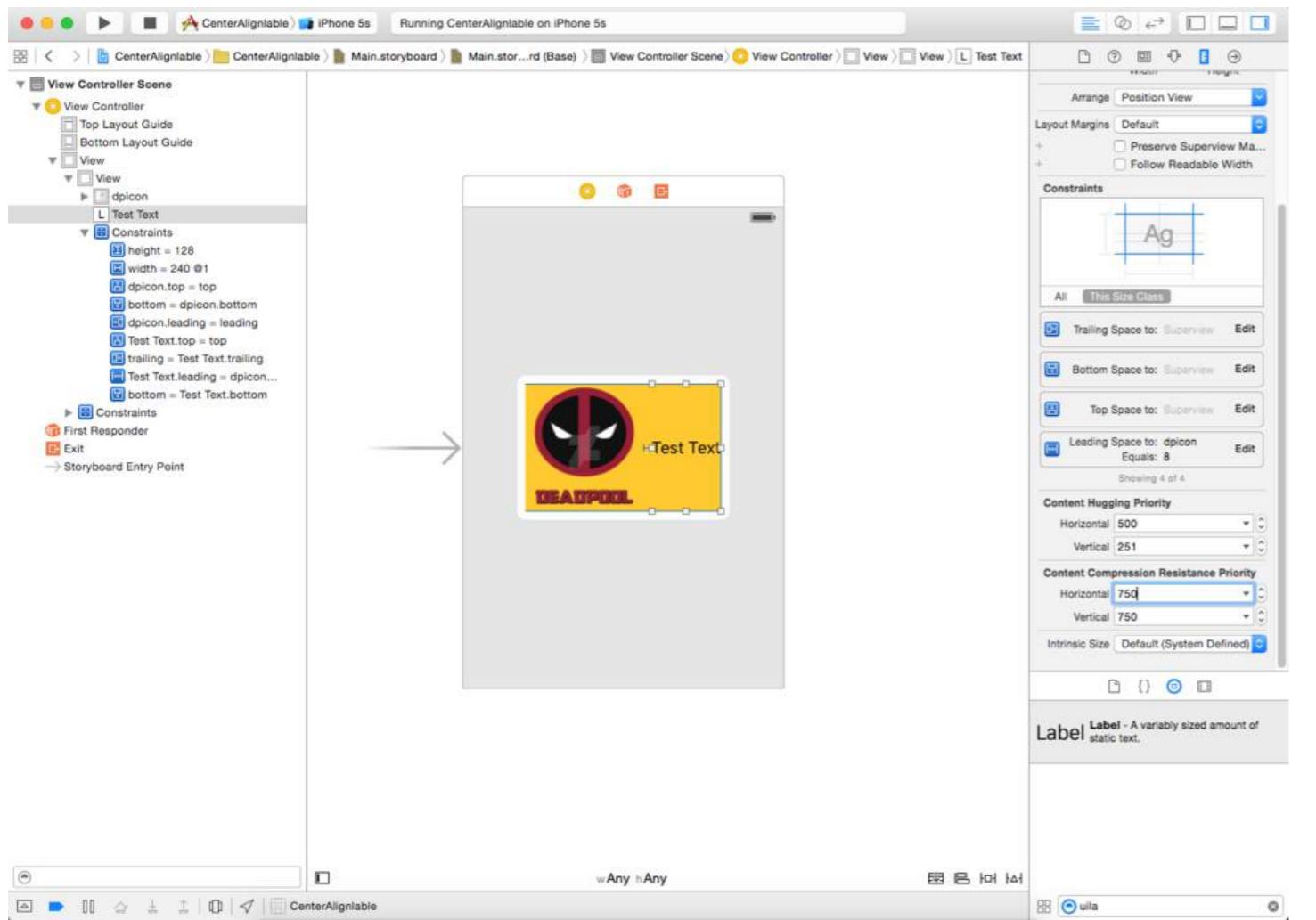
**Step 5:** Now if we recall we will realize that we have given fixed view to out yellow view but we want it to be dynamic dependent upon the text of our label.So We will modify our Width Constraint of yellow view.Width of yellow view is necessary to resolve ambiguity but we want it to be overruled at runtime based upon the content of UILabel. So we will select our yellow view and go to Size inspector and reduce the priority of width constraint to 1 so that it will be over ruled. Follow the image given below.



步骤6： 我们希望UILabel根据文本内容扩展并推动黄色视图。因此，我们降低了黄色视图宽度约束的优先级。现在我们将提高UILabel的文本压缩抗拒优先级。我们也希望视图缩小，所以会提高UILabel的内容拥抱优先级。请参见下图。

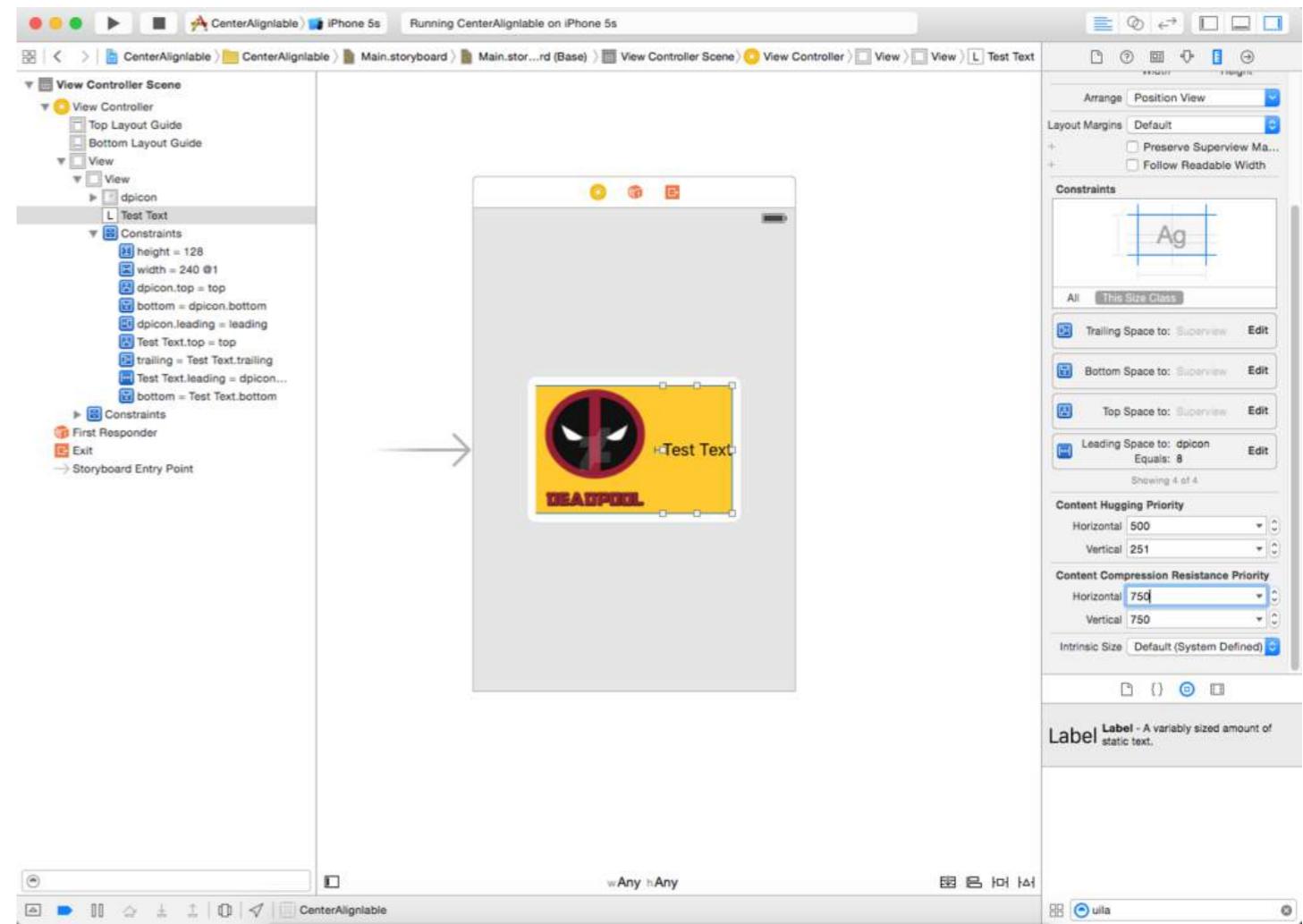


**Step 6:** We want our UILabel to expand according to text and push our yellow view. So we have reduced the priority of yellow view width. Now we will increase the priority of text compression resistance of our UILabel. We want our view to reduce as well so we will increase the priority of content hugging of UILabel. Follow the image below



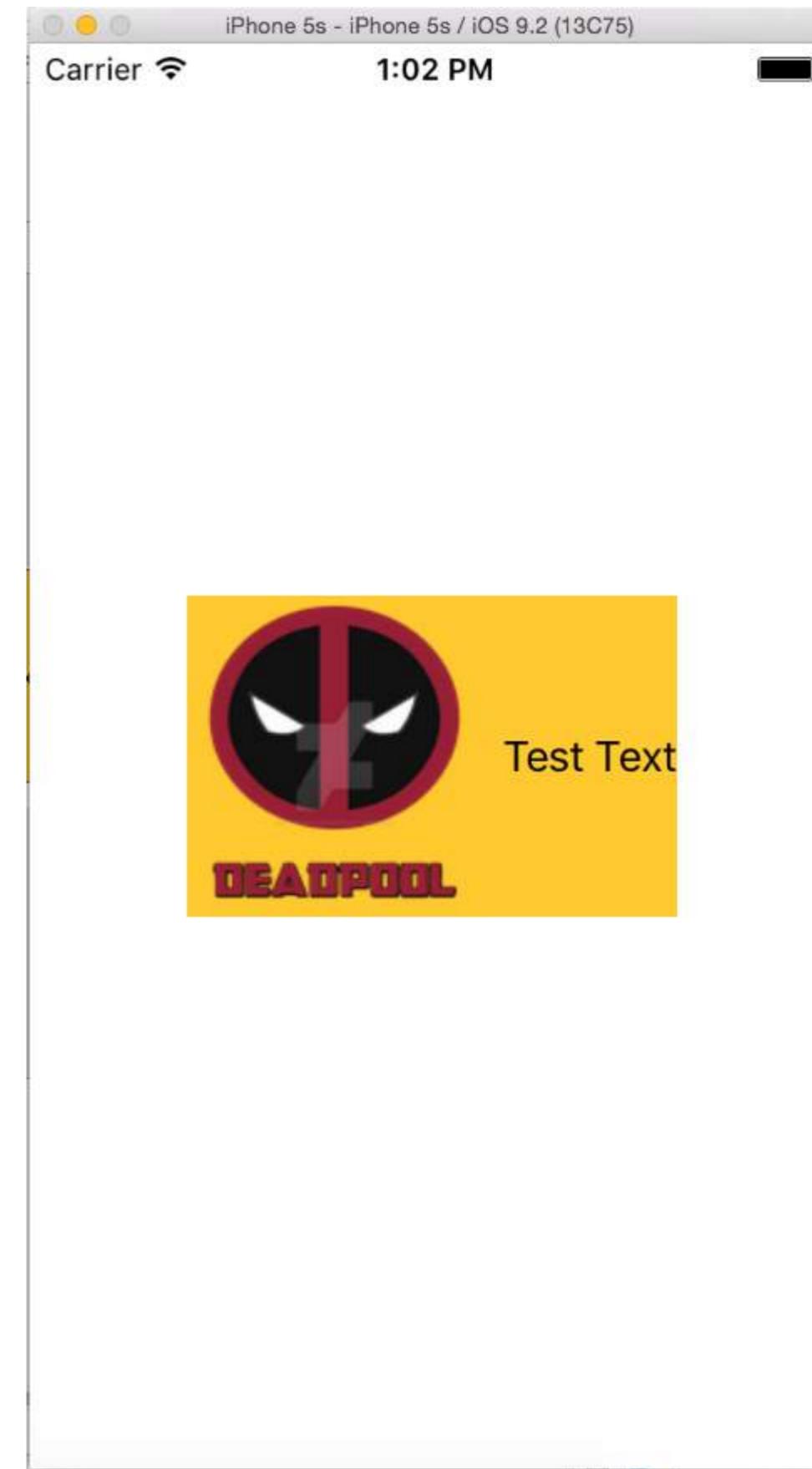
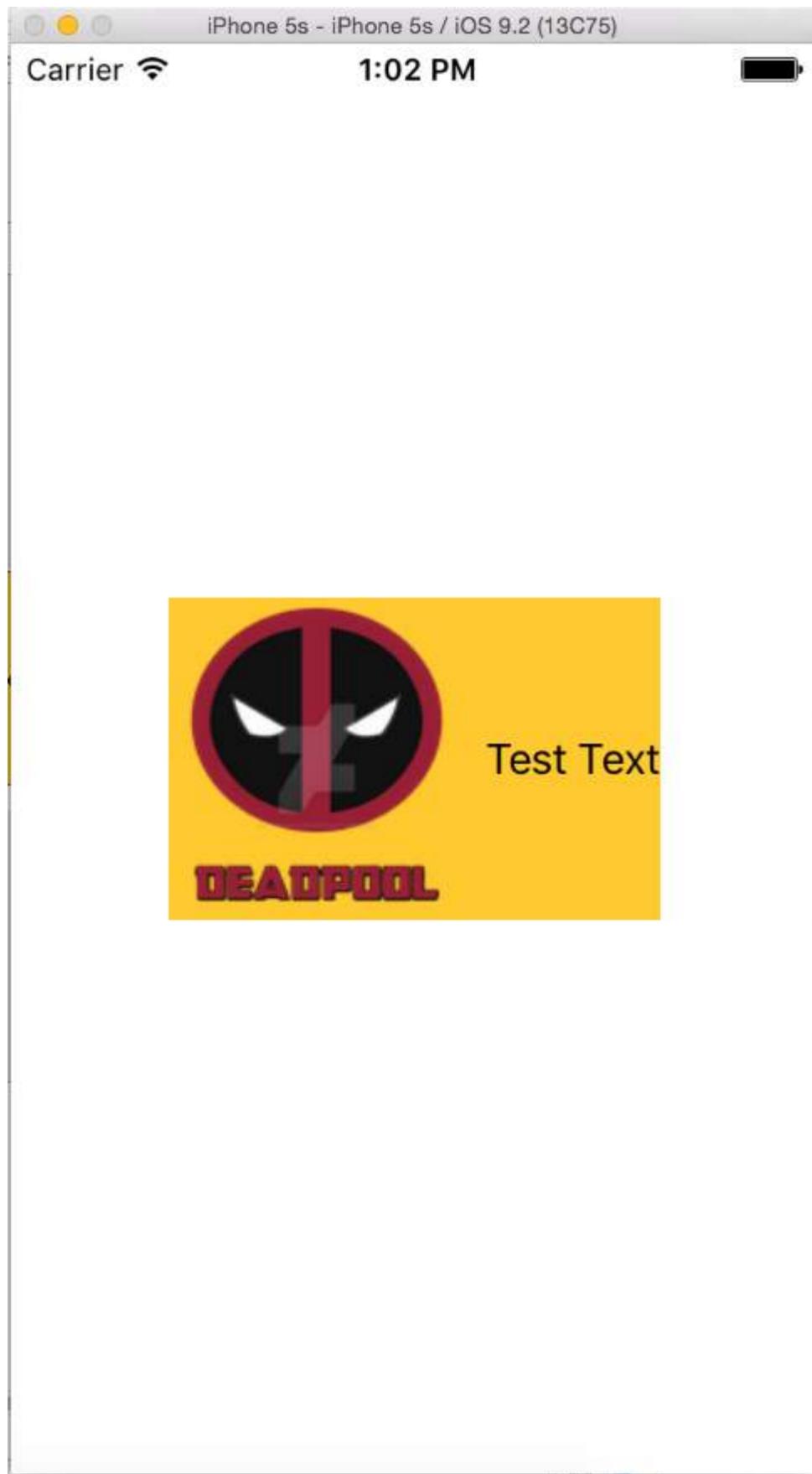
如您所见，我们已将内容拥抱优先级提高到500，压缩抗拒优先级提高到751，这将成功覆盖宽度约束's 1 优先级。

现在构建并运行，您将看到如下内容。



As you can see we have increased content hugging priority to 500 and compression resistance priority to 751 which will successfully over rule the width constraint's 1 priority.

Now build and run you will see some thing as following.



## 第67.6节：视觉格式语言基础：代码中的约束！

HVFL是一种旨在以简单快捷的方式约束UI元素的语言。通常，VFL相比Interface Builder中的传统UI定制具有优势，因为它更易读、易访问且更简洁。

## Section 67.6: Visual Format Language Basics: Constraints in Code!

HVFL is a language designed to constrain UI elements in a simple and quick fashion. Generally, VFL has an advantage over traditional UI customization in the Interface Builder because it's much more readable, accessible and compact.

以下是一个VFL示例，其中三个UIView从左到右被约束，填充了superView的宽度，包含一个aGradeView

```
"H:|[bgView][aGradeView(40)][bGradeView(40)]|"
```

我们可以在两个轴向上约束UI对象，水平和垂直。

每行VFL总是以H:或V:开头。如果两者都没有，默认选项是H:

接下来，我们有一个管道符号。|这个符号，或者称为管道，指的是超级视图。如果你仔细观察上面VFL代码的片段，你会注意到有两个这样的管道符号。

这表示超级视图的两个水平端点，外左和外右边界。

接下来你会看到一些方括号，在第一组方括号内，我们有bgView。当我们看到方括号时，它指的是一个UI元素，现在你可能会想我们如何建立名称和实际UI元素之间的链接，也许是一个outlet？

我会在文章末尾讲解这个。

如果你看第二对方括号[aGradeView(50)]，我们里面还有一些括号包裹着，当括号存在时，它定义了宽度/高度，取决于轴向，在这个例子中是宽度为50像素。

第一组方括号[bgView]没有明确定义宽度，意味着它会尽可能地展开。

好了，基础内容就到这里，更多高级内容将在另一个例子中讲解。

例如：



```
// 1. 创建视图
UIView *blueView = [[UIView alloc] init];
blueView.backgroundColor = [UIColor blueColor];
[self.view addSubview:blueView];

UIView *redView = [[UIView alloc] init];
redView.backgroundColor = [UIColor redColor];
[self.view addSubview:redView];
```

Here's an example of VFL, in which three UIViews are constrained from left to right, filling up superView.width, with aGradeView

```
"H:|[bgView][aGradeView(40)][bGradeView(40)]|"
```

There are two axes in which we can constrain UI Objects to, Horizontally and Vertically.

Each line of VFL always begins with H: or V:. If neither are present, the default option is H:

Moving on, we have a pipeline. | This symbol, or the pipe, refers to the superview. If you take a closer look at the snippet of VFL code above, you'd notice two of these pipelines.

This signifies the two horizontal ends of the superview, the outerleft and outerright boundaries.

Next up you'll see some square brackets, within the first set of square brackets, we have bgView. When we've got square brackets, it's referring to a UI element, now you might wonder how we establish a link between the name and the actual UI element, an outlet perhaps?

I'll cover that at the end of the post.

If you take a look at the second pair of square brackets [aGradeView(50)], we have some parentheses encapsulated within as well, when that is present, it defines the width/height depending on the axes, which in this case is 50 pixels in width.

The first square brackets [bgView] did not have a width explicitly defined, meaning that it'll span out as far as possible.

Alright, that's it for the basics, more on the advanced stuff in another example.

for example:



```
// 1. create views
UIView *blueView = [[UIView alloc] init];
blueView.backgroundColor = [UIColor blueColor];
[self.view addSubview:blueView];

UIView *redView = [[UIView alloc] init];
redView.backgroundColor = [UIColor redColor];
[self.view addSubview:redView];
```

```

// 2. 禁止自动调整大小
blueView.translatesAutoresizingMaskIntoConstraints = NO;
redView.translatesAutoresizingMaskIntoConstraints = NO;

// 3. 创建约束
// 水平
NSArray *blueH = [NSLayoutConstraint constraintsWithVisualFormat:@"H:|-20-[blueView]-20-|"
options:NSLayoutFormatAlignAllLeft metrics:nil views:@{@"blueView" : blueView}];
[self.view addConstraints:blueH];

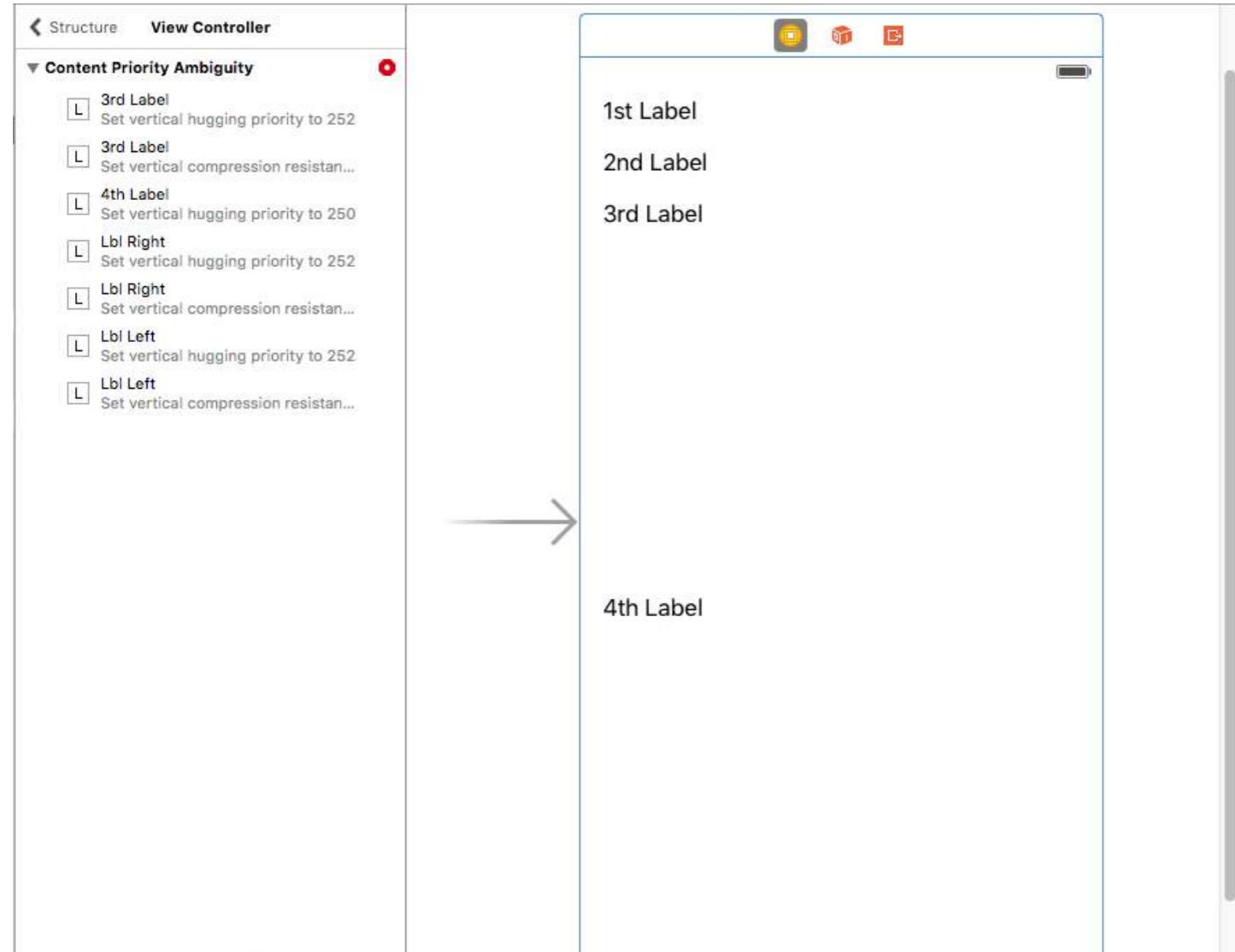
// 垂直
NSArray *blueVandRedV = [NSLayoutConstraint constraintsWithVisualFormat:@"V:|-20-"
[blueView(50)]-20-[redView==blueView]" options:NSLayoutFormatAlignAllTrailing metrics:nil
views:@{@"blueView" : blueView, @"redView" : redView}];
[self.view addConstraints:blueVandRedV];

NSLayoutConstraint *redW = [NSLayoutConstraint constraintWithItem:redView
attribute:NSLayoutAttributeWidth relatedBy:NSLayoutRelationEqual toItem:blueView
attribute:NSLayoutAttributeWidth multiplier:0.5 constant:0];
[self.view addConstraint:redW];

```

## 第67.7节：解决UILabel优先级冲突

问题: 当你在一个视图中使用多个标签时, 可能会收到一个警告:



我们如何解决这个警告?

```

// 2. forbid Autoresizing
blueView.translatesAutoresizingMaskIntoConstraints = NO;
redView.translatesAutoresizingMaskIntoConstraints = NO;

// 3. make constraints
// horizontal
NSArray *blueH = [NSLayoutConstraint constraintsWithVisualFormat:@"H:|-20-[blueView]-20-|"
options:NSLayoutFormatAlignAllLeft metrics:nil views:@{@"blueView" : blueView}];
[self.view addConstraints:blueH];

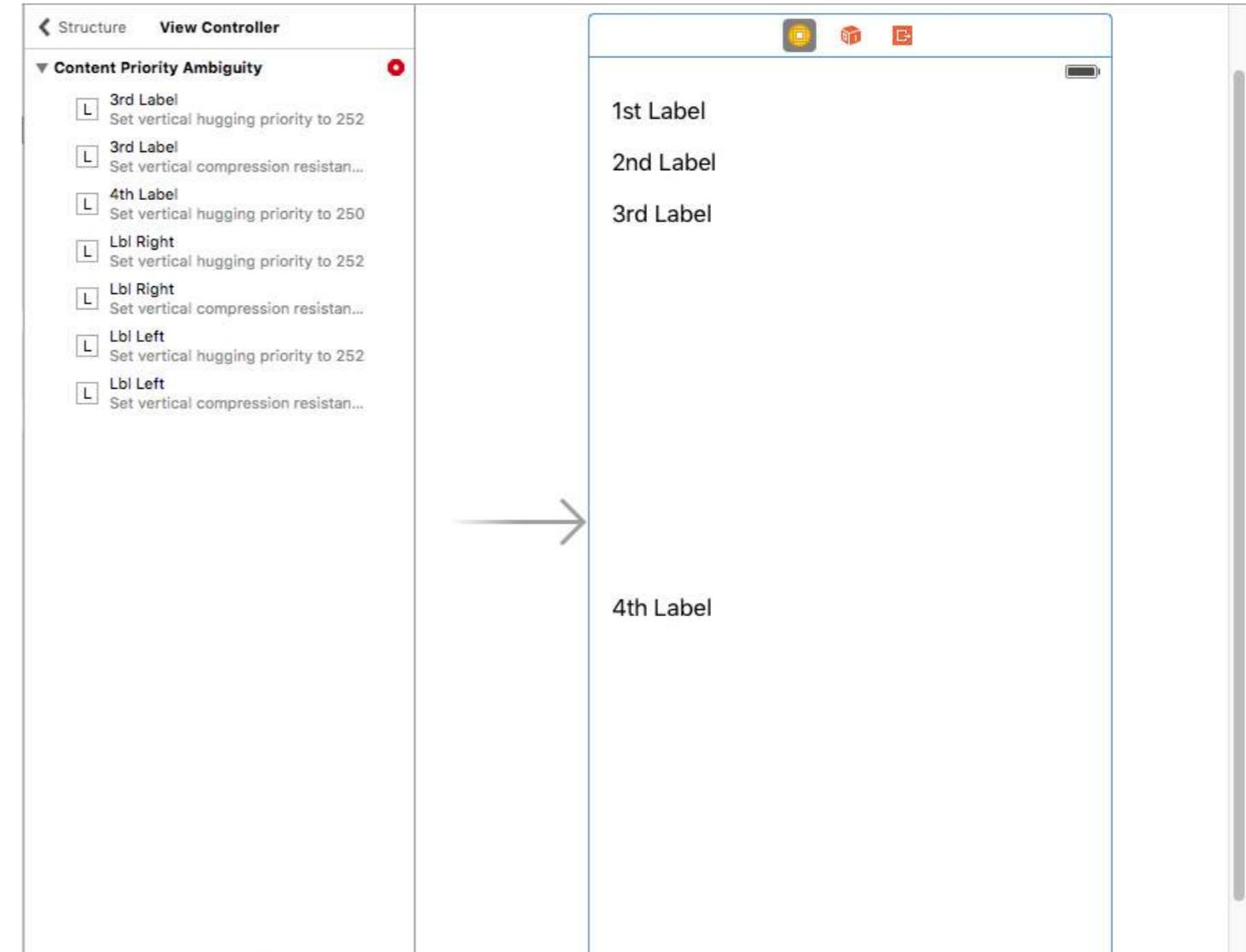
// vertical
NSArray *blueVandRedV = [NSLayoutConstraint constraintsWithVisualFormat:@"V:|-20-"
[blueView(50)]-20-[redView==blueView]" options:NSLayoutFormatAlignAllTrailing metrics:nil
views:@{@"blueView" : blueView, @"redView" : redView}];
[self.view addConstraints:blueVandRedV];

NSLayoutConstraint *redW = [NSLayoutConstraint constraintWithItem:redView
attribute:NSLayoutAttributeWidth relatedBy:NSLayoutRelationEqual toItem:blueView
attribute:NSLayoutAttributeWidth multiplier:0.5 constant:0];
[self.view addConstraint:redW];

```

## Section 67.7: Resolve UILabel Priority Conflict

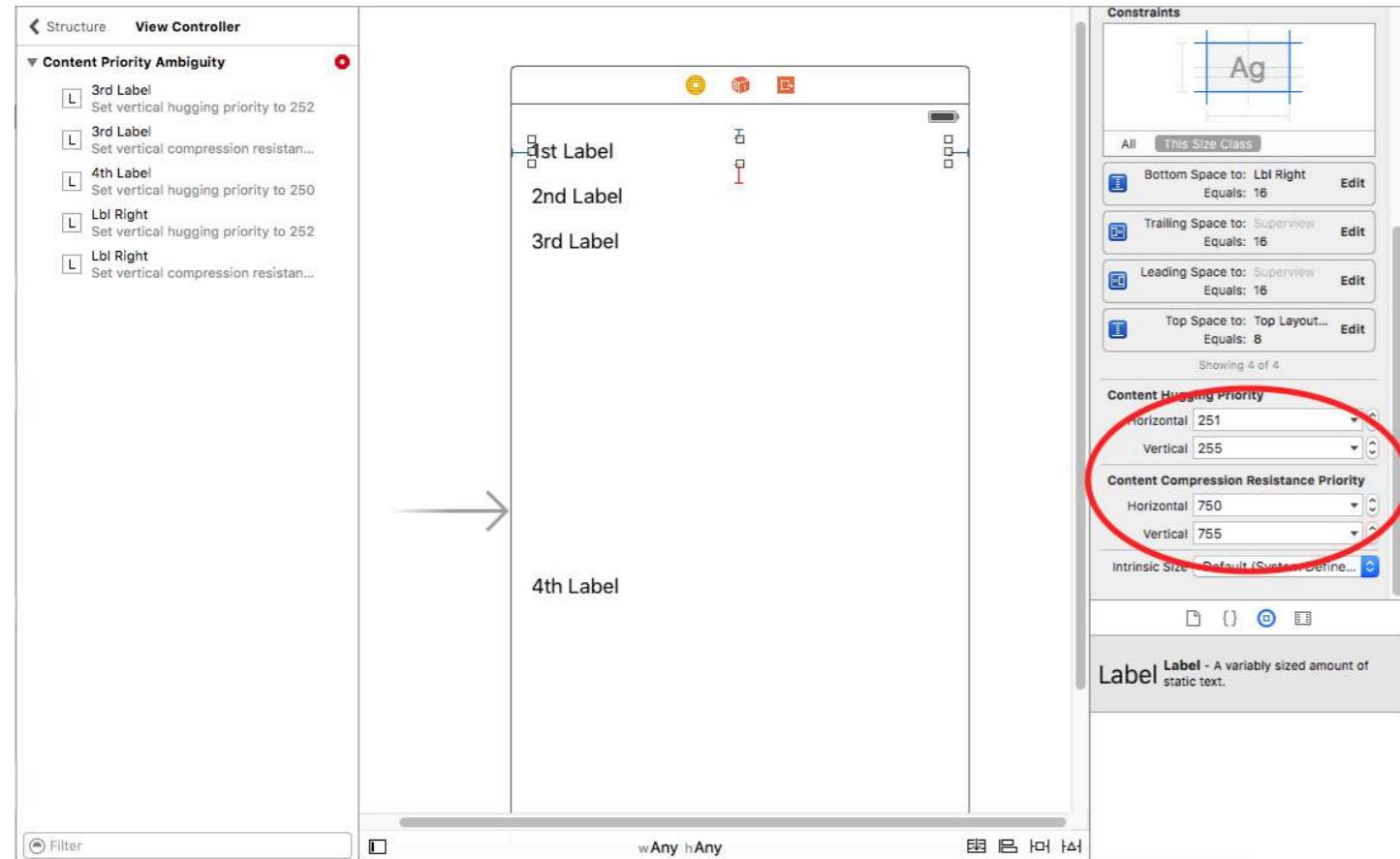
**Problem:** When you use many labels inside a view, you maybe get a **warning**:



How can we fix this **warning**?

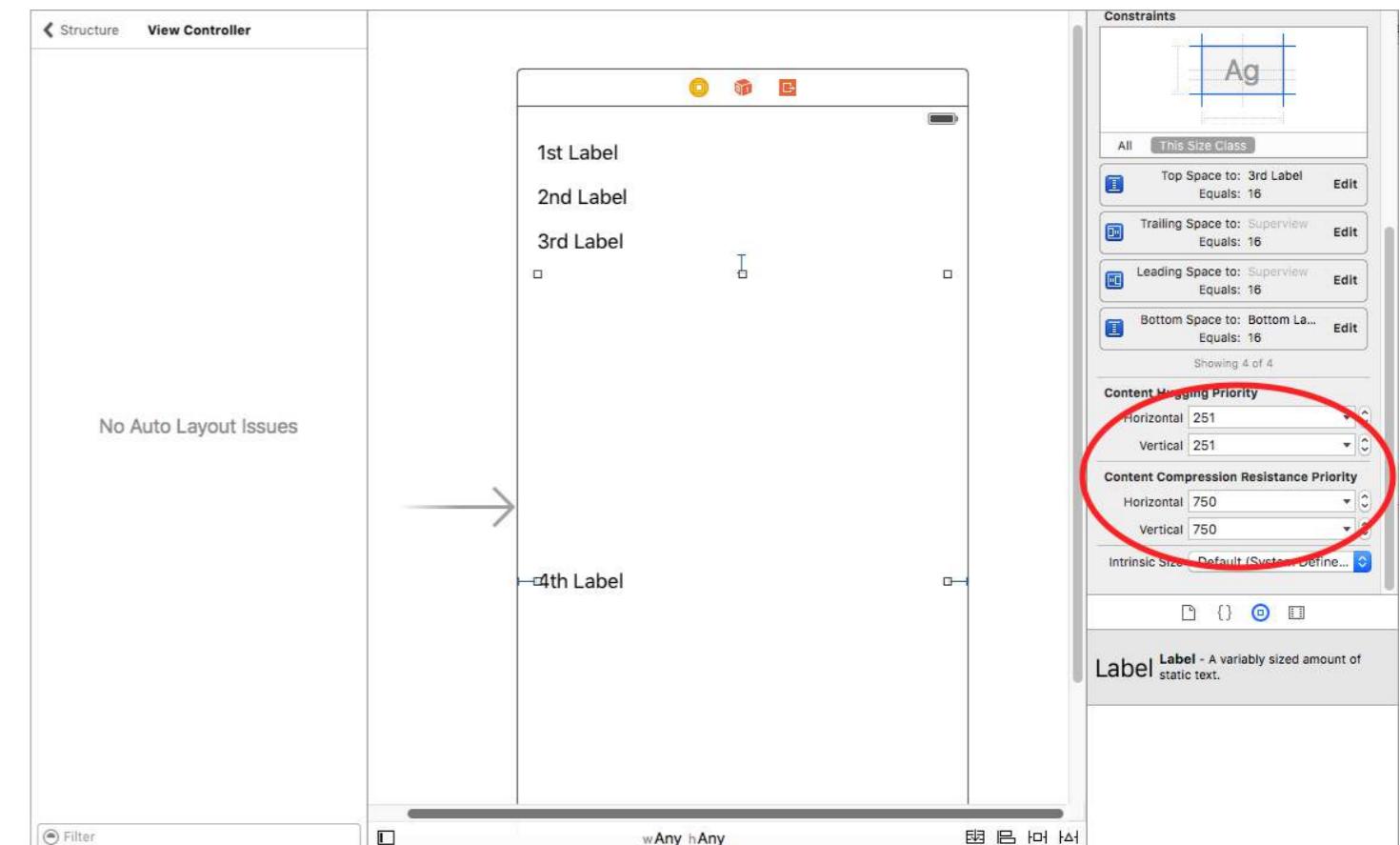
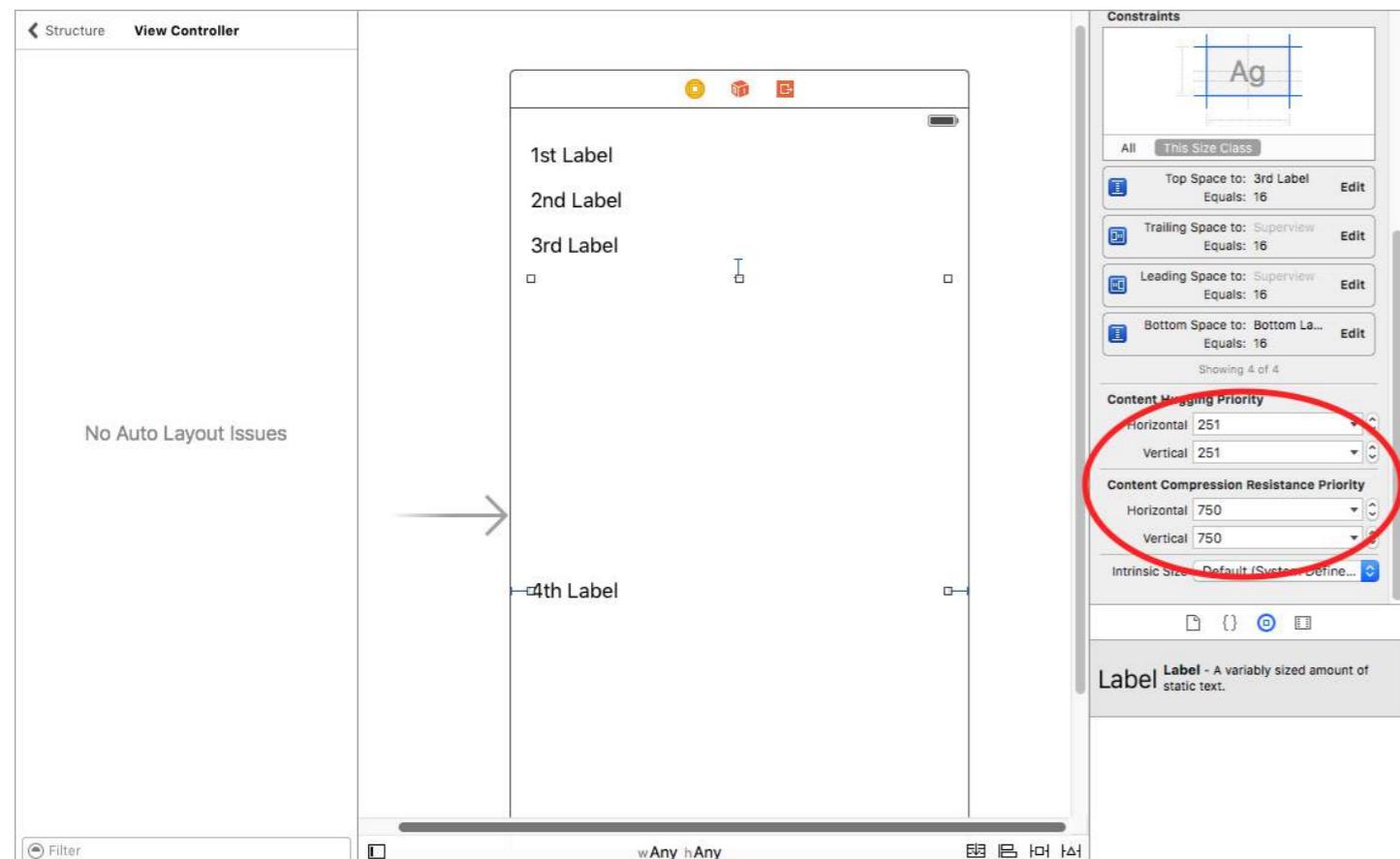
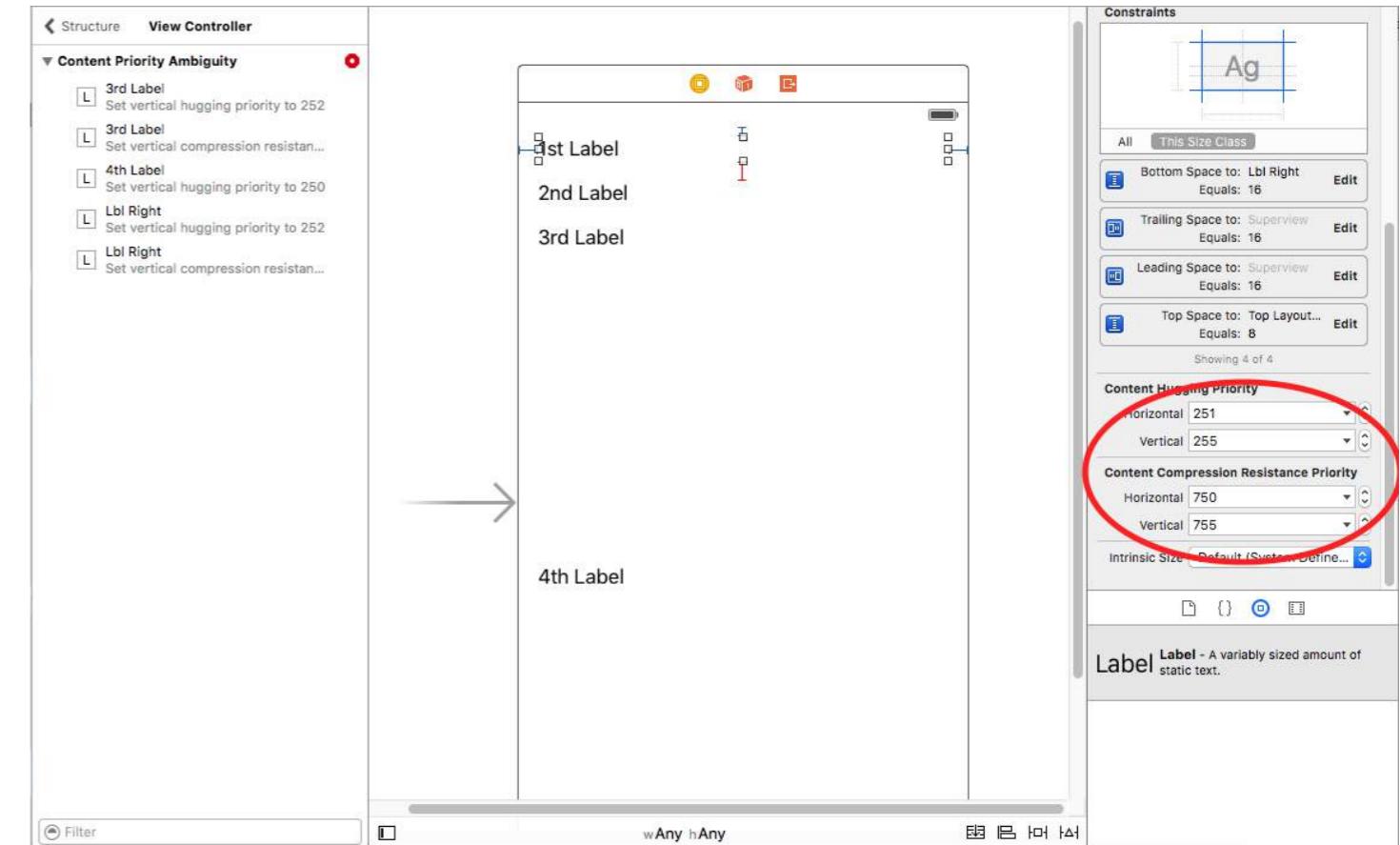
解决方案：我们按顺序计算并设置优先级。优先级必须与标签不同。这意味着重要的内容将获得更高的优先级。例如，在我的情况下，我为标签设置的垂直优先级如下：

我为第一个标签设置了最高优先级，为第四个标签设置了最低优先级。



**Solution:** We calculate and set the priorities in order. The priorities must be different from labels. It means which is important will get higher priority. For example, in my case, I set the vertical priorities for my labels look like this:

I set the highest priority for 1st label and the lowest for 4th label.



在一个视图控制器中，我认为你很难看到这些优先级的效果。然而，在UITableViewCell加上估算单元格高度时，这一点非常明显。

希望这能帮到你。

## 第67.8节：如何使用自动布局进行动画

没有自动布局时，动画是通过随时间改变视图的框架来实现的。使用自动布局时，约束决定视图的框架，因此你必须动画化约束。这种间接方式使动画更难以直观展示。

以下是使用自动布局进行动画的方法：

- 在创建后通过周期性调用 (CADisplayLink、`dispatch_source_t`、`dispatch_after`、`NSTimer`) 改变约束的常量。然后调用`layoutIfNeeded`来更新约束。示例：

**Objective-C :**

```
self.someConstraint.constant = 10.0;
[UIView animateWithDuration:0.25 animations:^{
    [self.view layoutIfNeeded];
}];
```

**Swift :**

```
self.someConstraint.constant = 10.0
UIView.animate(withDuration: 0.25, animations: self.view.layoutIfNeeded)
```

- 在动画块内更改约束 并调用 `[view layoutIfNeeded]`。这会在动画过程中插值两个位置之间，忽略动画期间的约束。

```
[UIView animateWithDuration:0.5 animations:^{
    [view layoutIfNeeded];
}]
```

- 更改约束的优先级。这比添加和移除约束的CPU消耗更低。

- 移除所有约束并使用自动调整大小掩码。对于后者，您必须设置 `view.translatesAutoresizingMaskIntoConstraints = YES`。

- 使用不会干扰预期动画的约束。

- 使用容器视图。使用约束定位父视图。然后添加一个带有约束的子视图，这些约束不会与动画冲突，例如：相对于父视图的中心。这将部分约束卸载到父视图，因此它们不会与子视图中的动画冲突。

- 对图层进行动画而不是视图。图层变换不会触发自动布局。

```
CABasicAnimation* ba = [CABasicAnimation animationWithKeyPath:@"transform"];
ba.autoreverses = YES;
ba.duration = 0.3;
ba.toValue = [NSValue valueWithCATransform3D:CATransform3DMakeScale(1.1, 1.1, 1)];
[v.layer addAnimation:ba forKey:nil];
```

- 重写 `layoutSubviews`。调用 `[super layoutSubviews]` 并微调约束。

- 在 `viewDidLayoutSubviews` 中更改框架。自动布局应用于 `layoutSubviews`，因此完成后，

In a ViewController, I think you're hard to see the effect of those priorities. However, it's very clearly with UITableViewCell + estimate cell height.

Hope this help.

## Section 67.8: How to animate with Auto Layout

Without Auto Layout, animation is accomplished changing a view's frame over time. With Auto Layout, the constraints dictate the view frame, so you have to animate the constraints instead. This indirection makes animation harder to visualize.

Here are the ways to animate with Auto Layout:

- Change the constant of the constraint after creation using periodic calls (`CADisplayLink`, `dispatch_source_t`, `dispatch_after`, `NSTimer`). Then call `layoutIfNeeded` to update the constraint. Example:

**Objective-C :**

```
self.someConstraint.constant = 10.0;
[UIView animateWithDuration:0.25 animations:^{
    [self.view layoutIfNeeded];
}];
```

**Swift:**

```
self.someConstraint.constant = 10.0
UIView.animate(withDuration: 0.25, animations: self.view.layoutIfNeeded)
```

- Change the constraints and call `[view layoutIfNeeded]` inside an animation block. This interpolates between the two positions ignoring constraints during the animation.

```
[UIView animateWithDuration:0.5 animations:^{
    [view layoutIfNeeded];
}]
```

- Change the priority of the constraints. This is less CPU intensive than adding and removing constraints.

- Remove all constraints and use autosizing masks. For the later, you have to set `view.translatesAutoresizingMaskIntoConstraints = YES`.

- Use constraints that don't interfere with the intended animation.

- Use a container view. Position the superview using constraints. Then add a subview with constraints that don't fight the animation, eg: a center relative to the superview. This unloads part of the constraints to the superview, so they don't fight the animation in the subview.

- Animate layers instead views. Layer transforms don't trigger the Auto Layout.

```
CABasicAnimation* ba = [CABasicAnimation animationWithKeyPath:@"transform"];
ba.autoreverses = YES;
ba.duration = 0.3;
ba.toValue = [NSValue valueWithCATransform3D:CATransform3DMakeScale(1.1, 1.1, 1)];
[v.layer addAnimation:ba forKey:nil];
```

- Override `layoutSubviews`. Call `[super layoutSubviews]` and fine tune the constraints.

- Change the frame in `viewDidLayoutSubviews`. Auto Layout is applied in `layoutSubviews`, so once done,

在 `viewDidLayoutSubviews` 中更改它。

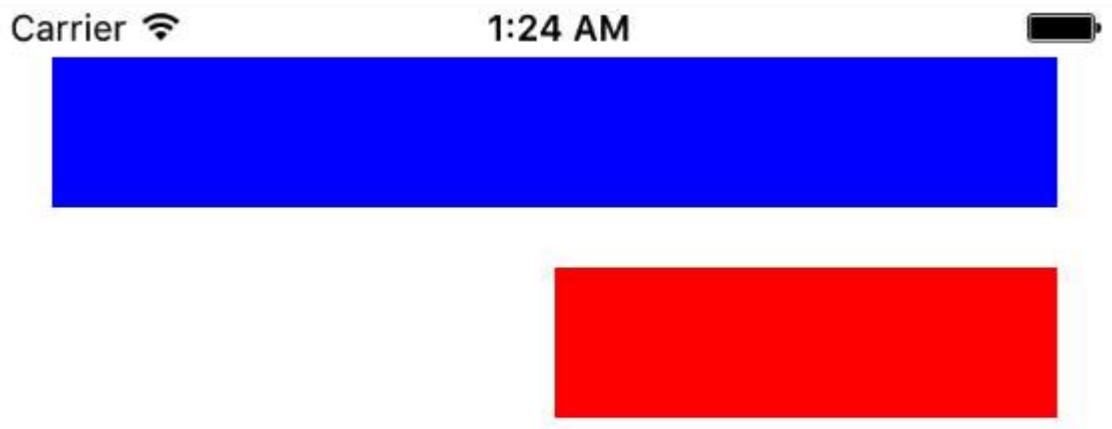
10. 放弃自动布局，手动设置视图。你可以通过重写 `layoutSubviews/layout` 而不调用父类'的实现来实现。

快速提示：如果动画视图的父视图没有被插值（即动画从开始状态跳到结束状态），请在动画视图的最深层父视图中调用 `layoutIfNeeded()`（换句话说，就是不受动画影响的视图）。我不太清楚为什么这样做有效。

## 第67.9节：NSLayoutConstraint：代码中的约束！

当我们在开发框架时，如果约束不太复杂，最好使用界面构建器或代码中的 `NSLayoutConstraint` 来使代码更简洁，而不是引入 `Masonry` 或 `SnapKit`。

例如：



- Objective-C

```
// 1. 创建视图
UIView *blueView = [[UIView alloc] init];
blueView.backgroundColor = [UIColor blueColor];
[self.view addSubview:blueView];

UIView *redView = [[UIView alloc] init];
redView.backgroundColor = [UIColor redColor];
[self.view addSubview:redView];

// 2. 禁止自动调整大小
blueView.translatesAutoresizingMaskIntoConstraints = NO;
redView.translatesAutoresizingMaskIntoConstraints = NO;

// 3. 创建约束
// 3.1 blueView
NSLayoutConstraint *blueLeft = [NSLayoutConstraint constraintWithItem:blueView
attribute:NSLayoutAttributeLeft relatedBy:NSLayoutRelationEqual toItem:self.view
attribute:NSLayoutAttributeLeft multiplier:1 constant:20];
[self.view addConstraint:blueLeft];

NSLayoutConstraint *blueTop = [NSLayoutConstraint constraintWithItem:blueView
attribute:NSLayoutAttributeTop relatedBy:NSLayoutRelationEqual toItem:self.view
attribute:NSLayoutAttributeTop multiplier:1 constant:20];
[self.view addConstraint:blueTop];
```

change it in `viewDidLayoutSubviews`.

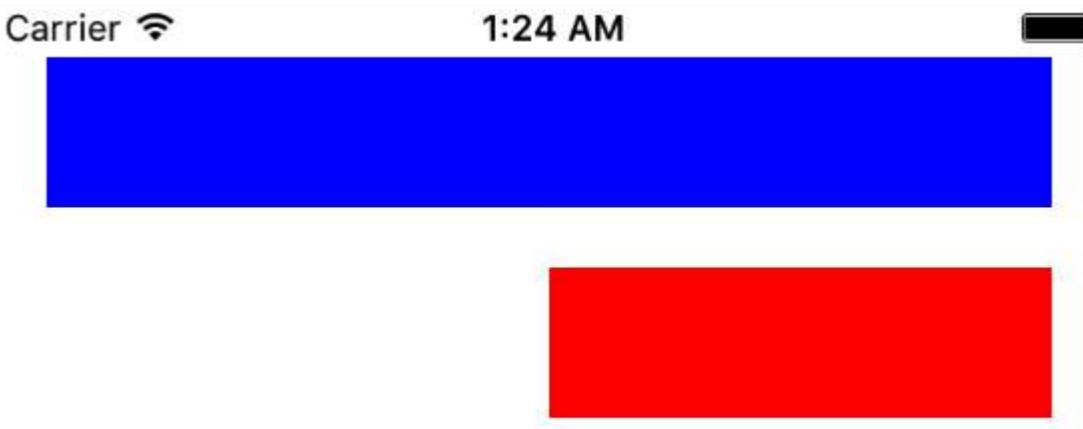
10. **Opt out from Auto Layout** and set views manually. You can do this overriding `layoutSubviews/layout` without calling the super class's implementation.

Quick tip: if the parent of the animated view is not being interpolated (that is, the animation jumps from beginning to end state), call `layoutIfNeeded()` in the deepest view that is the parent of the view that is animated (in other words, that is not affected by the animation). I don't know exactly why this works.

## Section 67.9: NSLayoutConstraint: Constraints in code!

When we are working on a framework, if the constraints are not too complex, we'd better use Interface Builder or `NSLayoutConstraint` in code to make it smaller enough, instead of import `Masonry` or `SnapKit`.

for example:



- Objective-C

```
// 1. create views
UIView *blueView = [[UIView alloc] init];
blueView.backgroundColor = [UIColor blueColor];
[self.view addSubview:blueView];

UIView *redView = [[UIView alloc] init];
redView.backgroundColor = [UIColor redColor];
[self.view addSubview:redView];

// 2. forbid Autoresizing
blueView.translatesAutoresizingMaskIntoConstraints = NO;
redView.translatesAutoresizingMaskIntoConstraints = NO;

// 3. make constraints
// 3.1 blueView
NSLayoutConstraint *blueLeft = [NSLayoutConstraint constraintWithItem:blueView
attribute:NSLayoutAttributeLeft relatedBy:NSLayoutRelationEqual toItem:self.view
attribute:NSLayoutAttributeLeft multiplier:1 constant:20];
[self.view addConstraint:blueLeft];

NSLayoutConstraint *blueTop = [NSLayoutConstraint constraintWithItem:blueView
attribute:NSLayoutAttributeTop relatedBy:NSLayoutRelationEqual toItem:self.view
attribute:NSLayoutAttributeTop multiplier:1 constant:20];
[self.view addConstraint:blueTop];
```

```

    NSLayoutConstraint *blueRight = [NSLayoutConstraint constraintWithItem:blueView
attribute:NSLayoutAttributeRight relatedBy:NSLayoutRelationEqual toItem:self.view
attribute:NSLayoutAttributeRight multiplier:1 constant:-20];
    [self.view addConstraint:blueRight];

    NSLayoutConstraint *blueHeight = [NSLayoutConstraint constraintWithItem:blueView
attribute:NSLayoutAttributeHeight relatedBy:NSLayoutRelationEqual toItem:nil
attribute:NSLayoutAttributeNotAnAttribute multiplier:1 constant:50];
    [self.view addConstraint:blueHeight];

// 3.2 redView
    NSLayoutConstraint *redTop = [NSLayoutConstraint constraintWithItem:redView
attribute:NSLayoutAttributeTop relatedBy:NSLayoutRelationEqual toItem:blueView
attribute:NSLayoutAttributeBottom multiplier:1 constant:20];
    [self.view addConstraint:redTop];

    NSLayoutConstraint *redRight = [NSLayoutConstraint constraintWithItem:redView
attribute:NSLayoutAttributeRight relatedBy:NSLayoutRelationEqual toItem:self.view
attribute:NSLayoutAttributeRight multiplier:1 constant:-20];
    [self.view 添加约束:redRight];

    NSLayoutConstraint *redHeight = [NSLayoutConstraint 约束项:redView
属性:NSLayoutAttributeHeight 关系:NSLayoutRelationEqual 目标项:blueView
属性:NSLayoutAttributeHeight 乘数:1 常数:0];
    [self.view 添加约束:redHeight];

    NSLayoutConstraint *redWidth = [NSLayoutConstraint 约束项:redView
属性:NSLayoutAttributeWidth 关系:NSLayoutRelationEqual 目标项:blueView
属性:NSLayoutAttributeWidth 乘数:0.5 常数:0];
    [self.view 添加约束:redWidth];

```

## 第67.10节：比例布局

创建的约束为

```

NSLayoutConstraint(item: myView, attribute: NSLayoutAttribute.Leading, relatedBy:
NSLayoutRelation.Equal, 目标项: view, 属性: NSLayoutAttribute.LeadingMargin, 乘数: 1.0,
常数: 20.0)

```

或者，从数学角度看：

view.属性 \* 乘数 + 常数 (1)

您可以使用乘数来为不同的尺寸因子创建比例布局。

示例：

青绿色视图 (V1) 是一个宽度与父视图宽度成比例的正方形，比例为1:1.1

灰色正方形 (V2) 是V1的子视图。底部间距通过常量设置为60，尾部间距通过乘数设置为1.125且常量为0

尾部间距按比例设置，底部间距设置为常量。

```

    NSLayoutConstraint *blueRight = [NSLayoutConstraint constraintWithItem:blueView
attribute:NSLayoutAttributeRight relatedBy:NSLayoutRelationEqual toItem:self.view
attribute:NSLayoutAttributeRight multiplier:1 constant:-20];
    [self.view addConstraint:blueRight];

    NSLayoutConstraint *blueHeight = [NSLayoutConstraint constraintWithItem:blueView
attribute:NSLayoutAttributeHeight relatedBy:NSLayoutRelationEqual toItem:nil
attribute:NSLayoutAttributeNotAnAttribute multiplier:1 constant:50];
    [self.view addConstraint:blueHeight];

// 3.2 redView
    NSLayoutConstraint *redTop = [NSLayoutConstraint constraintWithItem:redView
attribute:NSLayoutAttributeTop relatedBy:NSLayoutRelationEqual toItem:blueView
attribute:NSLayoutAttributeBottom multiplier:1 constant:20];
    [self.view addConstraint:redTop];

    NSLayoutConstraint *redRight = [NSLayoutConstraint constraintWithItem:redView
attribute:NSLayoutAttributeRight relatedBy:NSLayoutRelationEqual toItem:self.view
attribute:NSLayoutAttributeRight multiplier:1 constant:-20];
    [self.view addConstraint:redRight];

    NSLayoutConstraint *redHeight = [NSLayoutConstraint constraintWithItem:redView
attribute:NSLayoutAttributeHeight relatedBy:NSLayoutRelationEqual toItem:blueView
attribute:NSLayoutAttributeHeight multiplier:1 constant:0];
    [self.view addConstraint:redHeight];

    NSLayoutConstraint *redWidth = [NSLayoutConstraint constraintWithItem:redView
attribute:NSLayoutAttributeWidth relatedBy:NSLayoutRelationEqual toItem:blueView
attribute:NSLayoutAttributeWidth multiplier:0.5 constant:0];
    [self.view addConstraint:redWidth];

```

## Section 67.10: Proportional Layout

Constraint created as

```

NSLayoutConstraint(item: myView, attribute: NSLayoutAttribute.Leading, relatedBy:
NSLayoutRelation.Equal, toItem: view, attribute: NSLayoutAttribute.LeadingMargin, multiplier: 1.0,
constant: 20.0)

```

or, from math point of view:

view.attribute \* multiplier + constant (1)

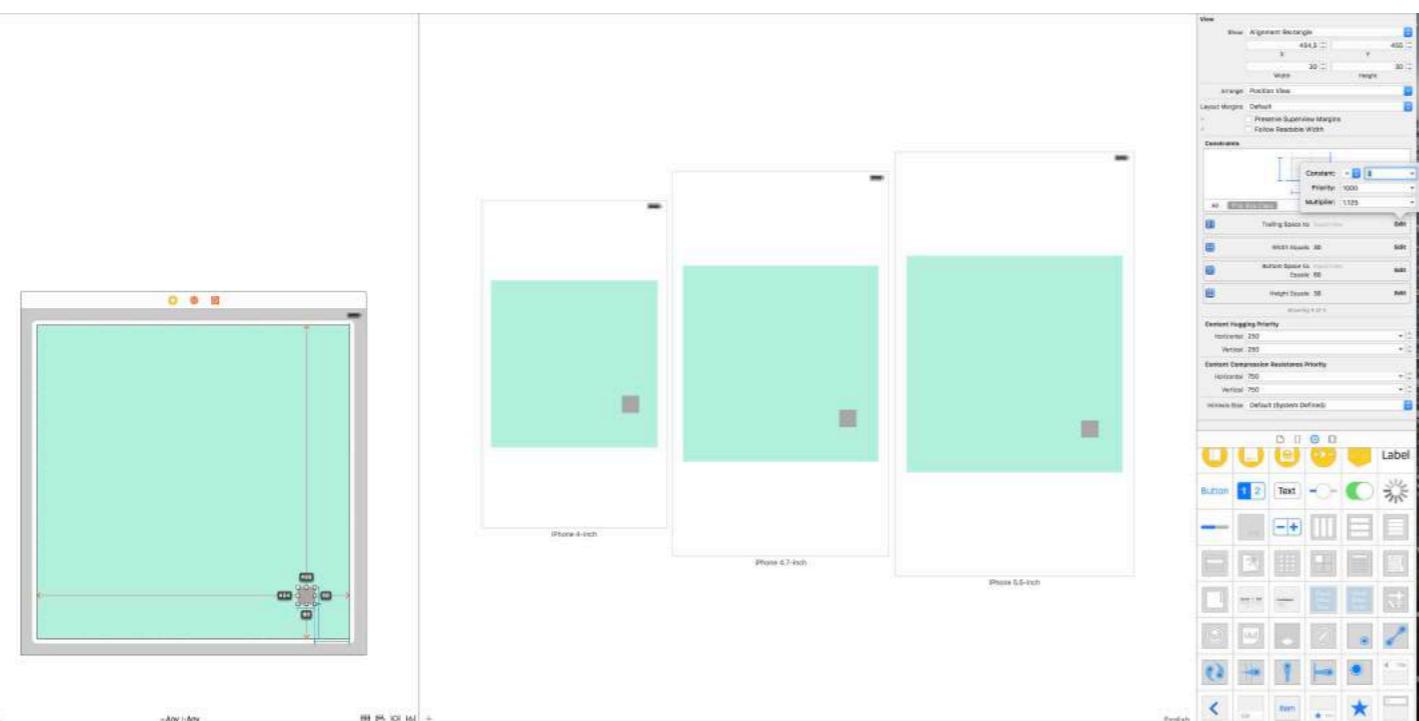
You can use multiplier to create proportional layout for different size factor.

Example:

Turquoise View (V1) is a square with width proportional superview width with ratio 1:1.1

Gary square(V2) is a subview of V1. Bottom space set by constant = 60, Trailing space set by multiplier = 1.125 and constant = 0

Trailing space set proportionally, bottom space set as a constant.



注意：如果view.attribute等于0（例如前导间距），约束公式（1）将等于0。你需要更改约束的第二项或将约束设置为相对于边距，以使view.attribute不等于0。

## 第67.11节：Auto Layout与非Auto Layout的混合使用

有时你可能想对Auto Layout由UIKit自身完成的计算执行一些额外操作。

**示例：**当你有一个带有maskLayer的UIView时，你可能需要在Auto Layout更改UIView的frame后立即更新maskLayer

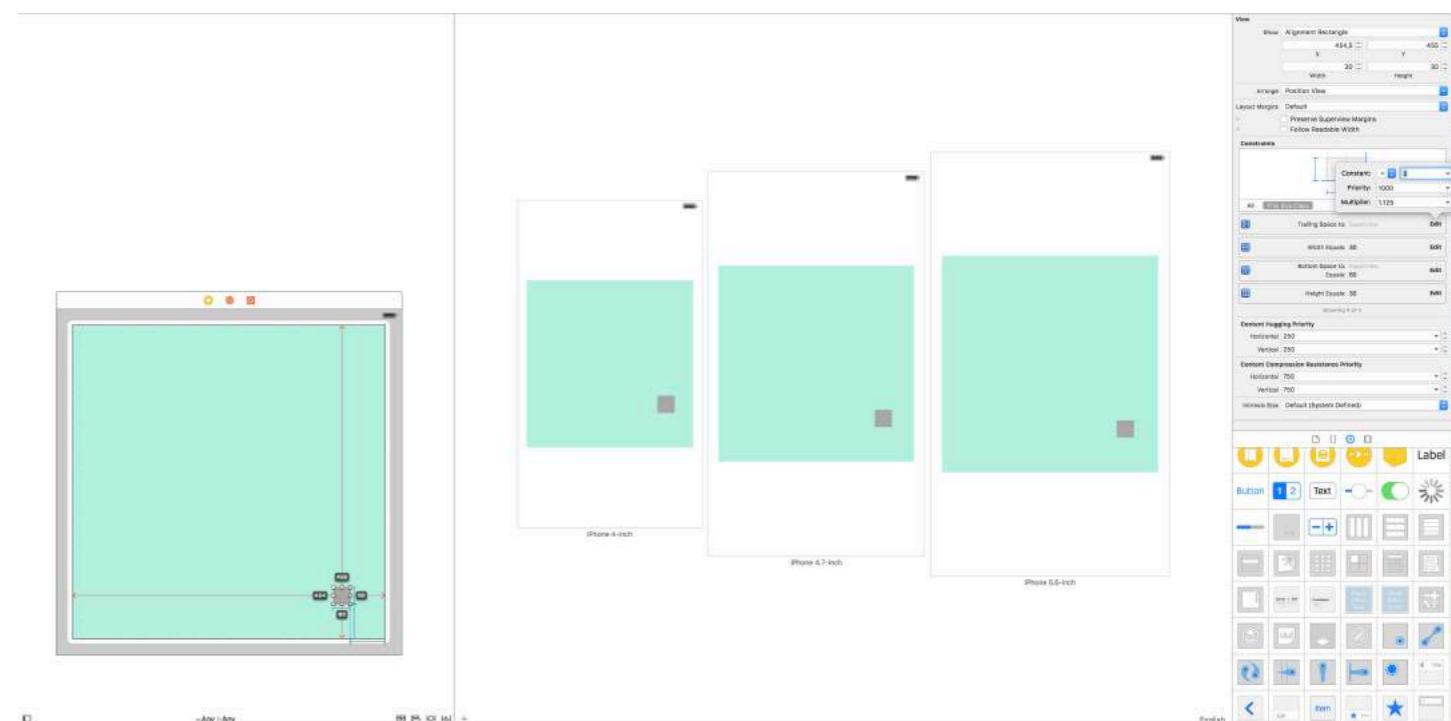
```
// CustomView.m
- (void)layoutSubviews {
    [super layoutSubviews];
    // 现在你可以假设Auto Layout已经完成了它的工作
    // 你可以在计算中使用视图的frame
    CALayer maskLayer = self.maskLayer;
    maskLayer.bounds = self.bounds;
    ...
}
```

或者如果你想对ViewController中的Auto Layout采取一些额外操作

```
- (void)viewDidLayoutSubviews {
    [super viewDidLayoutSubviews];
    // 现在你可以假设所有子视图的位置和大小都已正确设置
    self.customView.frame = self.containerView.frame;
}
```

## 第67.12节：如何使用自动布局

自动布局用于排列视图，使其在任何设备和方向上都能良好显示。约束是告诉布局如何进行的规则，包括固定边缘、居中和设置大小等内容。



Note: if view.attribute is equal 0 (for example leading space), constraint formula (1), will be equal 0. You need to change second item of constraint or set constraint relative to margin, in order to view.attribute != 0.

## Section 67.11: Mixed usage of Auto Layout with non-Auto Layout

Sometimes you may want to perform some additional actions to **Auto Layout** calculations done by UIKit itself.

**Example:** when you have a **UIView** that has a **maskLayer**, you may need to update **maskLayer** as soon as **Auto Layout** changes **UIView's frame**

```
// CustomView.m
- (void)layoutSubviews {
    [super layoutSubviews];
    // now you can assume Auto Layout did its job
    // you can use view's frame in your calculations
    CALayer maskLayer = self.maskLayer;
    maskLayer.bounds = self.bounds;
    ...
}
```

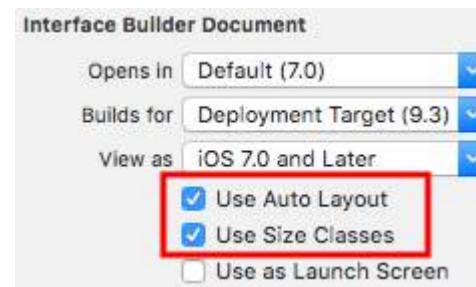
or if you want to take some additional action to **Auto Layout** in **ViewController**

```
- (void)viewDidLayoutSubviews {
    [super viewDidLayoutSubviews];
    // now you can assume all your subviews are positioned/resized correctly
    self.customView.frame = self.containerView.frame;
}
```

## Section 67.12: How to use Auto Layout

Auto layout is used to arrange views so that they look good on any device and orientation. Constraints are the rules that tell how everything should be laid down. They include pinning edges, centering, and setting sizes, among other things.

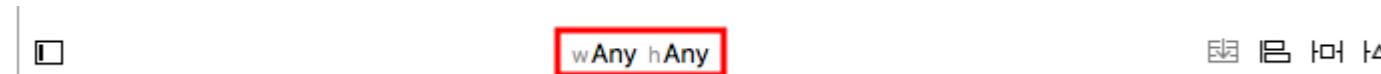
自动布局默认启用，但你可以再次确认。点击项目导航中的Main.storyboard，然后显示文件检查器。确保勾选了自动布局和尺寸类：



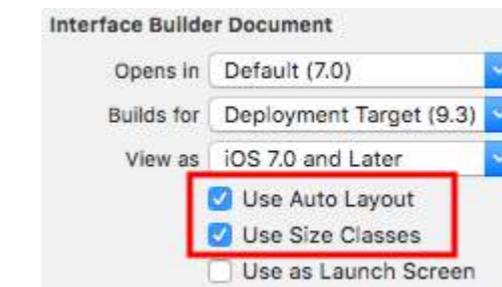
自动布局约束可以在界面构建器或代码中设置。在界面构建器中，你可以在右下角找到自动布局工具。点击它们会显示设置视图约束的不同选项。



如果你希望针对不同设备尺寸或方向设置不同的约束，可以在底部中间的wAny hAny尺寸类选项中进行设置。



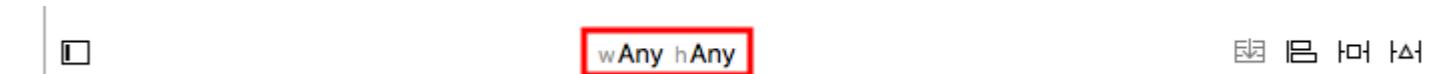
Auto layout is enabled by default, but you can double check this. If you click *Main.storyboard* in the Project Navigator and then show the File inspector. Make sure that Auto Layout and Size Classes are checked:



Auto layout constraints can be set in the Interface Builder or in code. In the Interface Builder you find the Auto Layout tools at the bottom right. Clicking them will reveal different options for setting the constraints on a view.



If you wish to have different constraints for different device sizes or orientations, you can set them in wAny hAny Size Class options found in the bottom middle.



# 第68章：MKMapView

## 第68.1节：更改地图类型

有5种不同类型 ([MKMapType](#))，[MKMapView](#) 可以显示。

版本 ≥ iPhone OS 3

**.standard**

显示所有道路位置和部分道路名称的街道地图。

**Swift 2**

```
mapView.mapType = .Standard
```

**Swift 3**

```
mapView.mapType = .standard
```

**Objective-C**

```
_mapView.mapType = MKMapTypeStandard;
```

# Chapter 68: MKMapView

## Section 68.1: Change map-type

There are 5 different types ([MKMapType](#)), [MKMapView](#) can display.

Version ≥ iPhone OS 3

**.standard**

Displays a street map that shows the position of all roads and some road names.

**Swift 2**

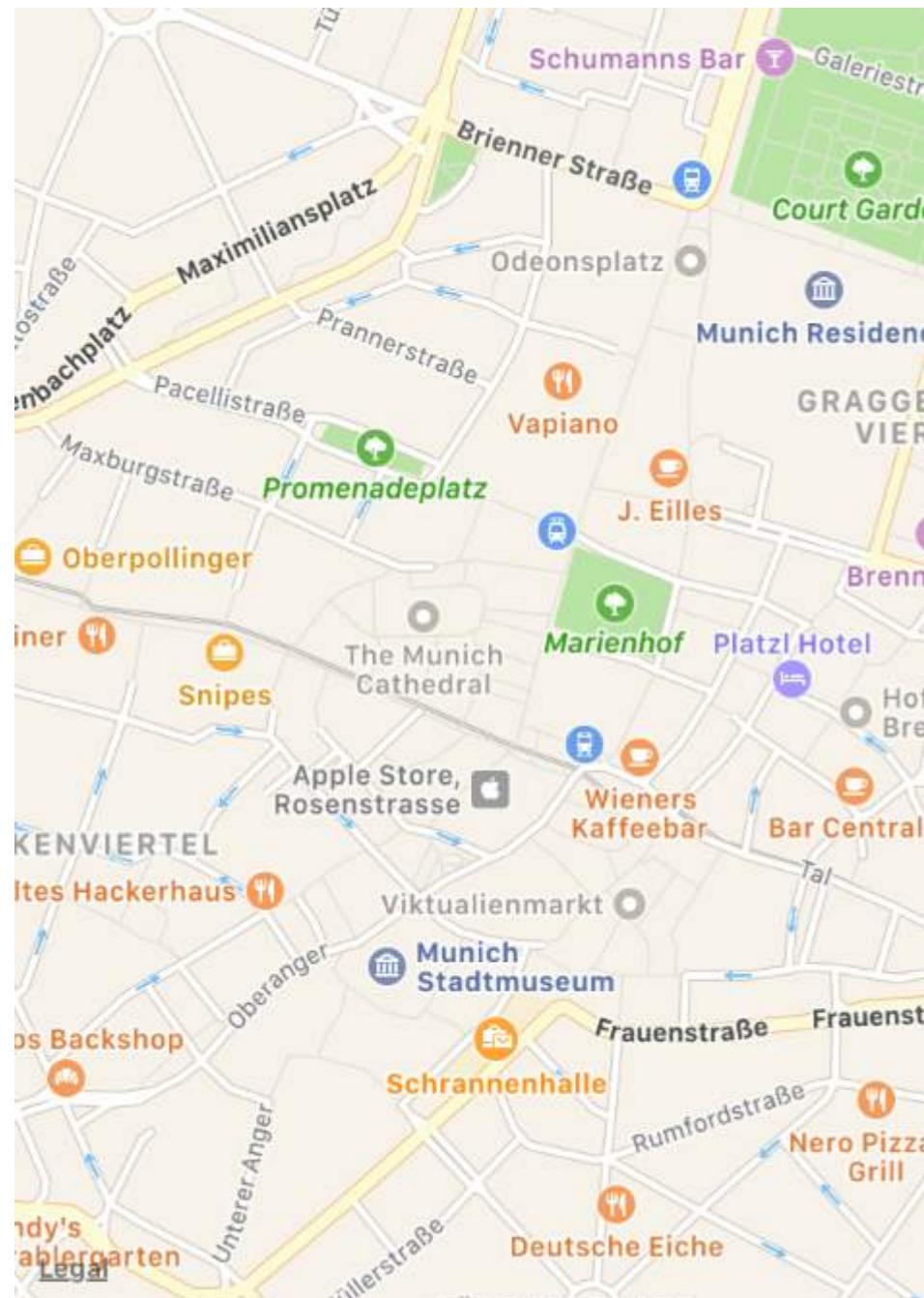
```
mapView.mapType = .Standard
```

**Swift 3**

```
mapView.mapType = .standard
```

**Objective-C**

```
_mapView.mapType = MKMapTypeStandard;
```



版本 ≥ iPhone OS 3

#### .satellite

显示该区域的卫星影像。

#### Swift 2

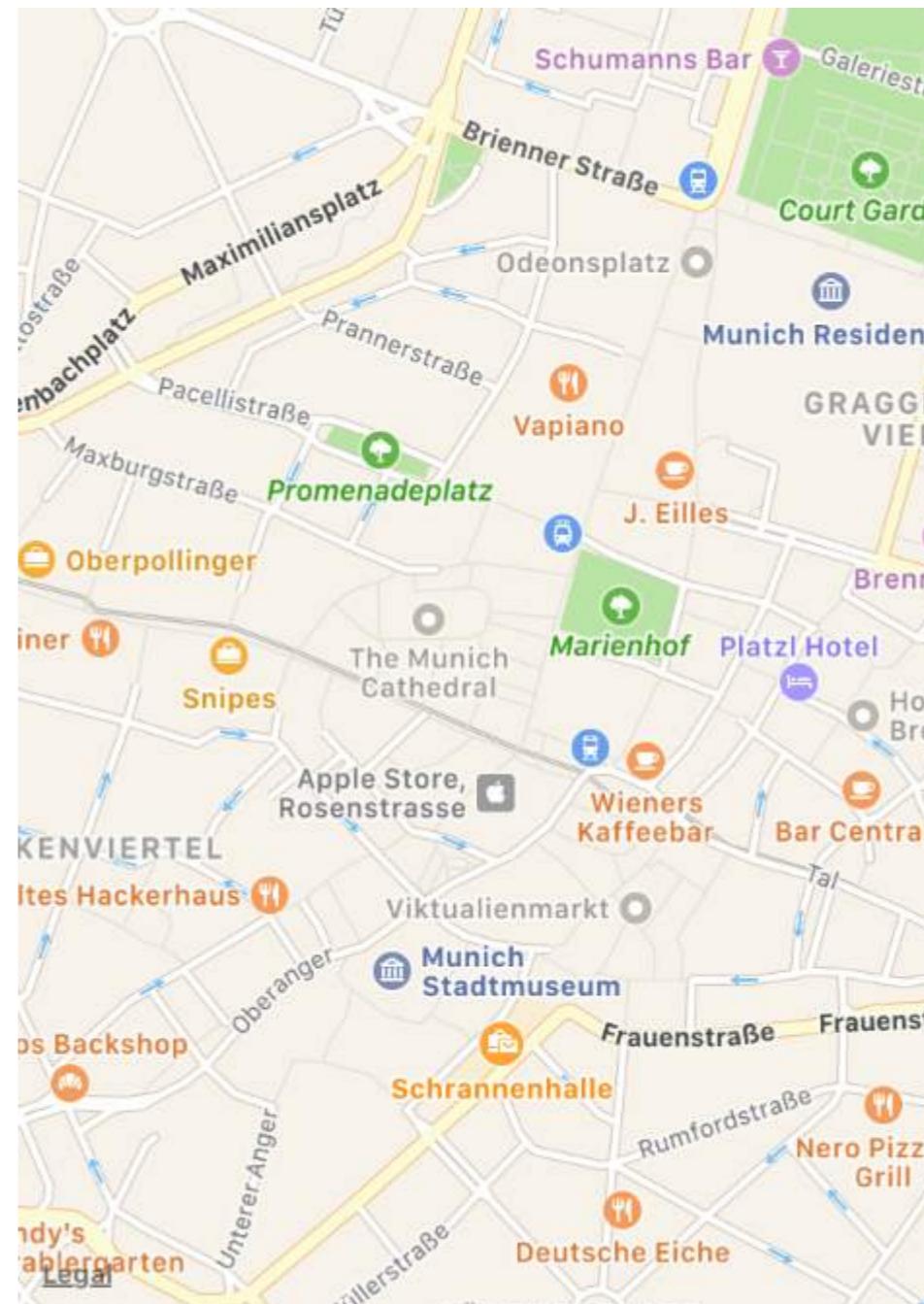
```
mapView.mapType = .卫星
```

#### Swift 3

```
mapView.mapType = .卫星
```

#### Objective-C

```
_mapView.mapType = MKMapTypeSatellite;
```



Version ≥ iPhone OS 3

#### .satellite

Displays satellite imagery of the area.

#### Swift 2

```
mapView.mapType = .Satellite
```

#### Swift 3

```
mapView.mapType = .satellite
```

#### Objective-C

```
_mapView.mapType = MKMapTypeSatellite;
```



版本 ≥ iOS 9

#### .卫星飞越

显示带有飞越数据的区域卫星图像（如有）。

#### Swift 2

```
mapView.mapType = .卫星飞越
```

#### Swift 3

```
mapView.mapType = .卫星飞越
```

#### Objective-C

```
_mapView.mapType = MKMapTypeSatelliteFlyover;
```

版本 ≥ iPhone OS 3

#### .混合

显示该区域的卫星图像，并叠加道路及道路名称信息。



Version ≥ iOS 9

#### .satelliteFlyover

Displays a satellite image of the area with flyover data where available.

#### Swift 2

```
mapView.mapType = .SatelliteFlyover
```

#### Swift 3

```
mapView.mapType = .satelliteFlyover
```

#### Objective-C

```
_mapView.mapType = MKMapTypeSatelliteFlyover;
```

Version ≥ iPhone OS 3

#### .hybrid

Displays a satellite image of the area with road and road name information layered on top.

## Swift 2

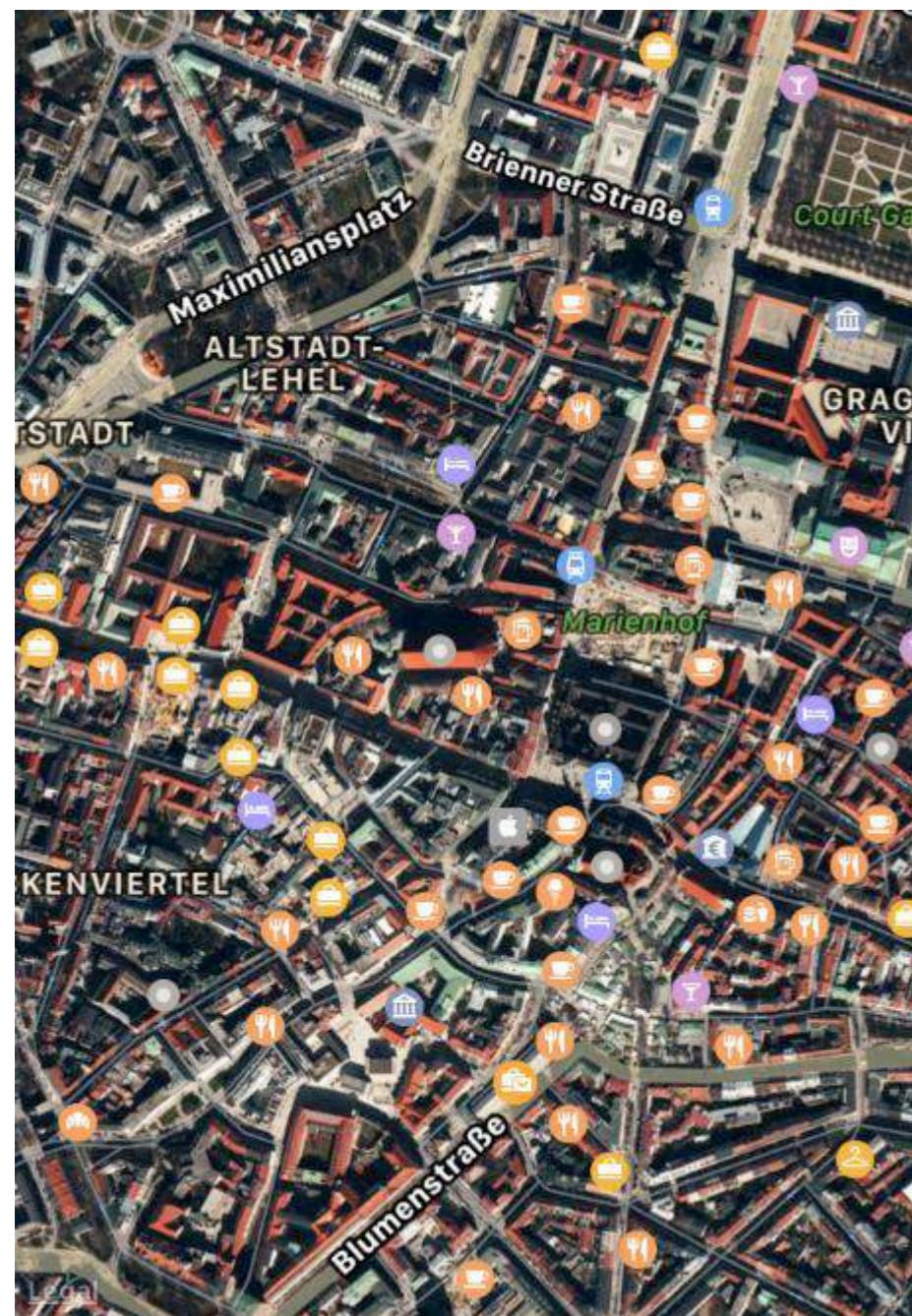
```
mapView.mapType = .Hybrid
```

## Swift 3

```
mapView.mapType = .hybrid
```

## Objective-C

```
_mapView.mapType = MKMapTypeHybrid;
```



版本 ≥ iOS 9

### .hybridFlyover

显示带有飞越数据的混合卫星图像（如有）。

## Swift 2

```
mapView.mapType = .HybridFlyover
```

## Swift 2

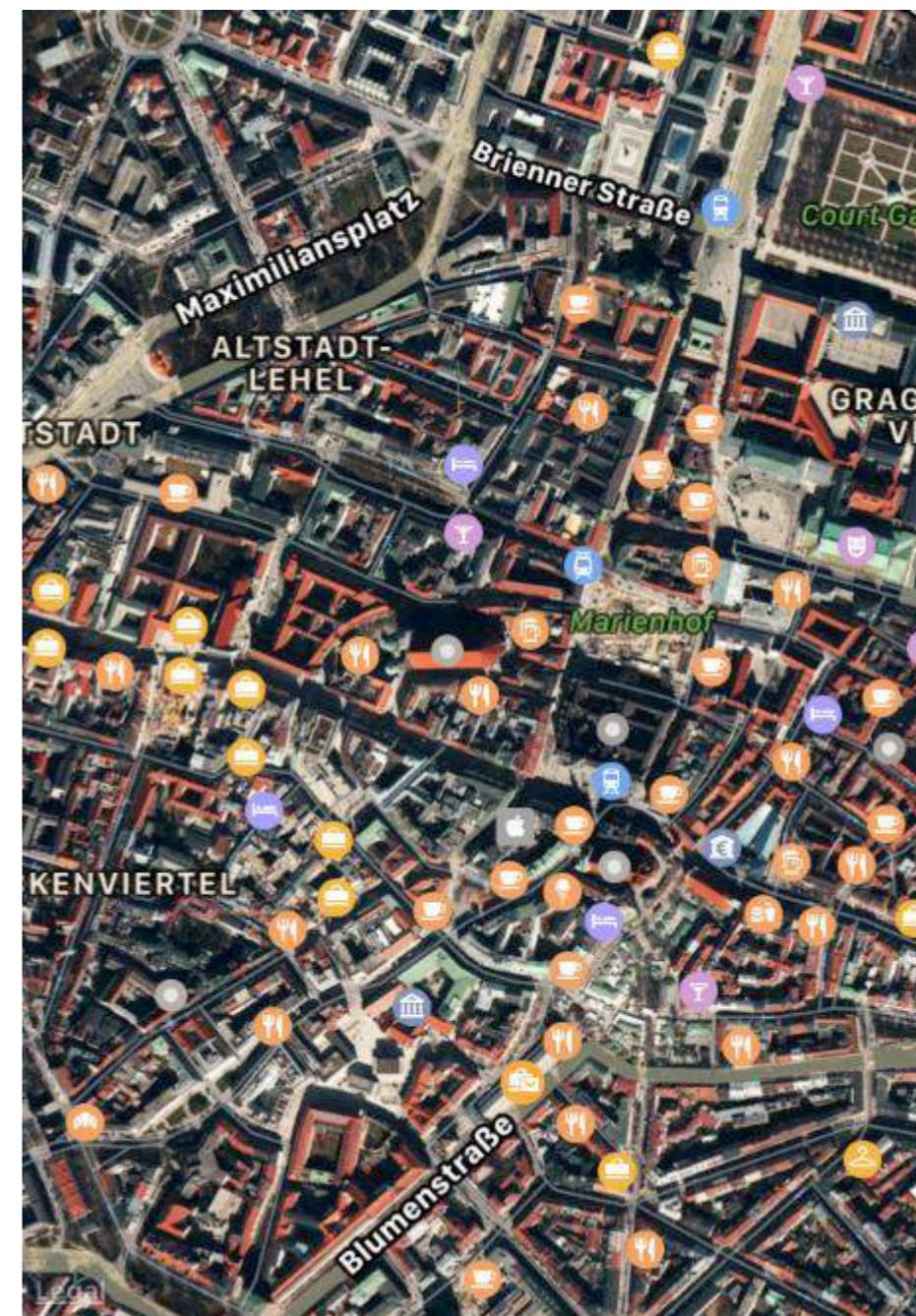
```
mapView.mapType = .Hybrid
```

## Swift 3

```
mapView.mapType = .hybrid
```

## Objective-C

```
_mapView.mapType = MKMapTypeHybrid;
```



Version ≥ iOS 9

### .hybridFlyover

Displays a hybrid satellite image with flyover data where available.

## Swift 2

```
mapView.mapType = .HybridFlyover
```

### Swift 3

```
mapView.mapType = .hybridFlyover
```

### Objective-C

```
_mapView.mapType = MKMapTypeHybridFlyover;
```

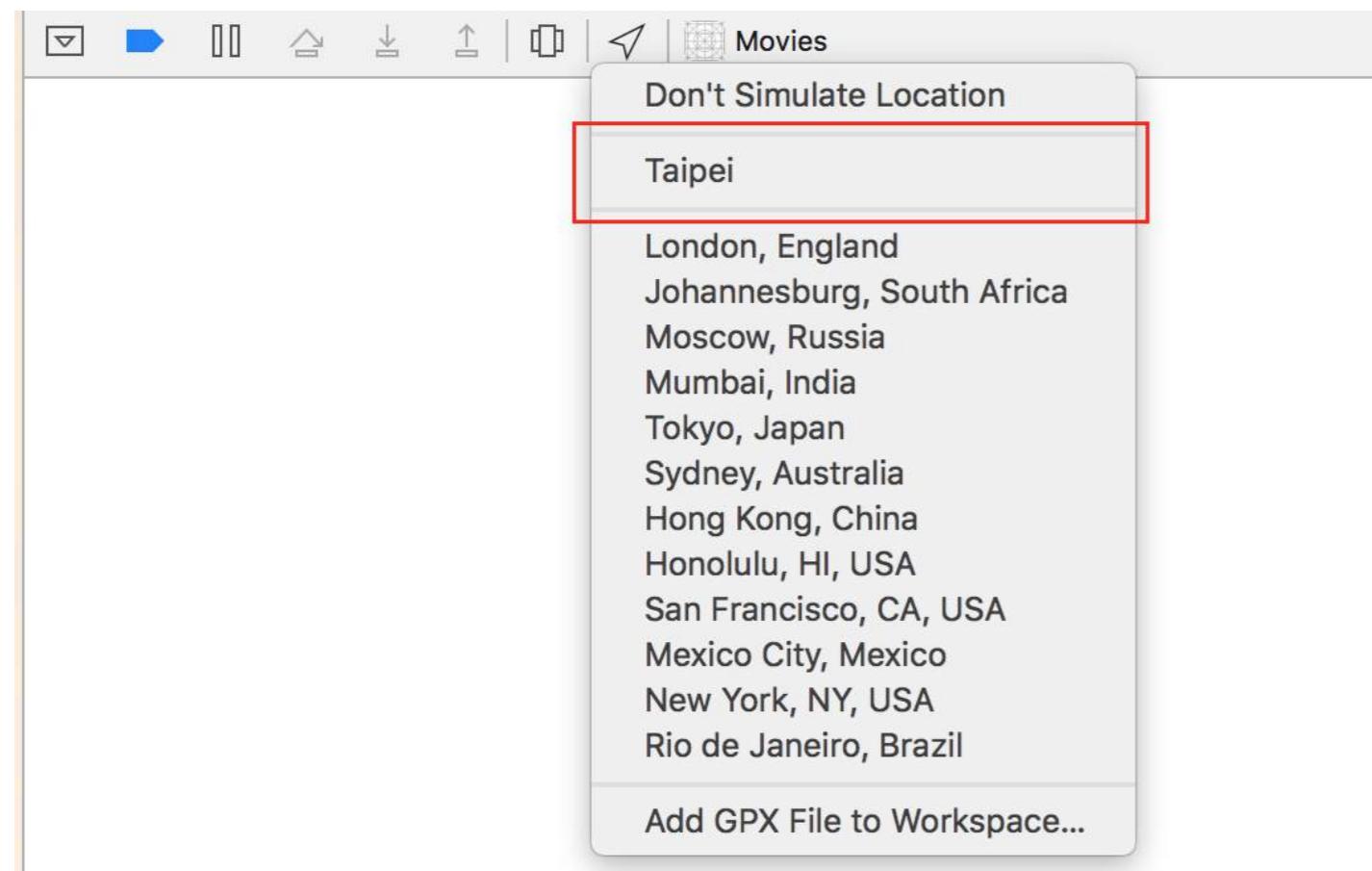
## 第68.2节：模拟自定义位置

步骤 1：在 Xcode 中：文件 -> 新建 -> 文件 -> 资源 -> GPX 文件 -> 下一步 -> 给 GPX 文件命名（本例中为台北）-> 创建

步骤 2：编辑 GPX 文件

```
<?xml version="1.0"?>
<gpx version="1.1" creator="Xcode">
    <wpt lat="25.041865" lon="121.551361"> // 编辑纬度和经度
        <name>台北</name> // 编辑地点名称
        <time>2014-09-24T14:55:37Z</time>
    </wpt>
</gpx>
```

步骤 3：当模拟器运行时：



您可以重复此过程以创建多个位置。

## 第 68.3 节：设置地图缩放/区域

为了设置某个缩放级别，比如我们想以用户位置为中心，半径为2公里的区域进行缩放。然后，我们使用以下代码

### Swift 3

```
mapView.mapType = .hybridFlyover
```

### Objective-C

```
_mapView.mapType = MKMapTypeHybridFlyover;
```

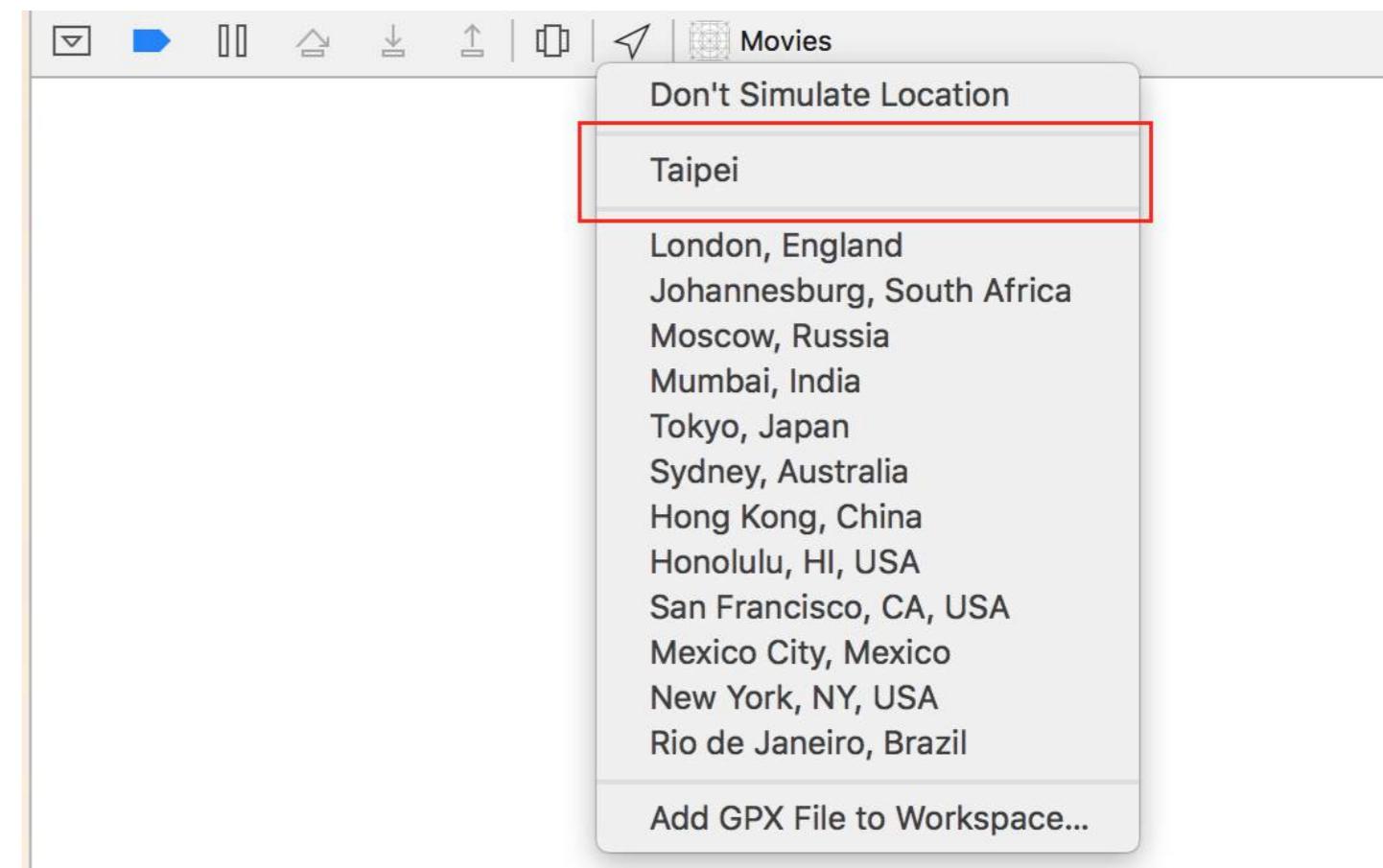
## Section 68.2: Simulate a custom location

Step 1: In Xcode: File -> New -> File -> Resource -> GPX File -> Next -> Give the GPX file a name(It's Taipei in this example) -> Create

Step 2: Edit the GPX file

```
<?xml version="1.0"?>
<gpx version="1.1" creator="Xcode">
    <wpt lat="25.041865" lon="121.551361"> // Edit the latitude and longitude
        <name>Taipei</name> // Edit the name of the location
        <time>2014-09-24T14:55:37Z</time>
    </wpt>
</gpx>
```

Step 3: When the simulator is running:



You can repeat this process to create multiple locations.

## Section 68.3: Set Zoom/Region for Map

For setting some zoom level, let say we want to zoom user's location with user location as center and 2km of area as radius. Then, we use following code

```

MKUserLocation *userLocation = _mapView.userLocation;
MKCoordinateRegion region = MKCoordinateRegionMakeWithDistance (userLocation.location.coordinate,
2000, 2000);
[_mapView setRegion:region animated:NO];

```

## 第68.4节：使用

MKLocalSearch实现本地搜索

MKLocalSearch允许用户使用自然语言字符串（如“健身房”）进行位置搜索。一旦搜索完成，该类会返回指定区域内与搜索字符串匹配的位置列表。

搜索结果以MKLocalSearchResponse对象中的MKMapItem形式呈现。

让我们通过示例来尝试

```

MKLocalSearchRequest *request =
    [[MKLocalSearchRequest alloc] init];//初始化搜索请求
request.naturalLanguageQuery = @"健身房"; // 添加查询
request.region = _mapView.region; // 设置区域
MKLocalSearch *search =
    [[MKLocalSearch alloc] initWithRequest:request];//开始搜索

[search startWithCompletionHandler:^(MKLocalSearchResponse
    *response, NSError *error)
{
    if (response.mapItems.count == 0)
        NSLog(@"无匹配项");
    else
        for (MKMapItem *item in response.mapItems)
    {
        NSLog(@"名称 = %@", item.name);
        NSLog(@"电话 = %@", item.phoneNumber);
    }
}];

```

## 第68.5节：OpenStreetMap 瓦片覆盖层

在某些情况下，您可能不想使用苹果提供的默认地图。

您可以向您的MapView添加一个覆盖层，该覆盖层包含来自OpenStreetMap的自定义瓦片。

假设self.mapView是您已经添加到ViewController中的MKMapView。

首先，您的ViewController需要遵守协议MKMapViewDelegate。

```
class MyViewController: UIViewController, MKMapViewDelegate
```

然后你必须将ViewController设置为MapView的代理

```
mapView.delegate = self
```

接下来，您需要为地图配置覆盖层。您需要一个URL模板。该URL在所有切片服务器上应类似于此，即使您将地图数据离线存储也应如此：

<http://tile.openstreetmap.org/{z}/{x}/{y}.png>

```
let urlTemplate = "http://tile.openstreetmap.org/{z}/{x}/{y}.png"
```

```

MKUserLocation *userLocation = _mapView.userLocation;
MKCoordinateRegion region = MKCoordinateRegionMakeWithDistance (userLocation.location.coordinate,
2000, 2000);
[_mapView setRegion:region animated:NO];

```

## Section 68.4: Local search implementation using MKLocalSearch

MKLocalSearch allows users to search for location using natural language strings like "gym". Once the search get completed, the class returns a list of locations within a specified region that match the search string.

Search results are in form of MKMapItem within MKLocalSearchResponse object.

lets try by example

```

MKLocalSearchRequest *request =
    [[MKLocalSearchRequest alloc] init];//initialising search request
request.naturalLanguageQuery = @"Gym"; // adding query
request.region = _mapView.region; //setting region
MKLocalSearch *search =
    [[MKLocalSearch alloc] initWithRequest:request];//initiate search

[search startWithCompletionHandler:^(MKLocalSearchResponse
    *response, NSError *error)
{
    if (response.mapItems.count == 0)
        NSLog(@"No Matches");
    else
        for (MKMapItem *item in response.mapItems)
    {
        NSLog(@"name = %@", item.name);
        NSLog(@"Phone = %@", item.phoneNumber);
    }
}];

```

## Section 68.5: OpenStreetMap Tile-Overlay

In some cases, you might not want to use the default maps, Apple provides.

You can add an overlay to your mapView that contains custom tiles for example from [OpenStreetMap](http://tile.openstreetmap.org/{z}/{x}/{y}.png).

Let's assume, `self.mapView` is your MKMapView that you have already added to your ViewController.

At first, your ViewController needs to conform to the protocol MKMapViewDelegate.

```
class MyViewController: UIViewController, MKMapViewDelegate
```

Then you have to set the ViewController as delegate of mapView

```
mapView.delegate = self
```

Next, you configure the overlay for the map. You'll need an URL-template for this. The URL should be similar to this on all tile-servers and even if you would store the map-data offline:

<http://tile.openstreetmap.org/{z}/{x}/{y}.png>

```
let urlTemplate = "http://tile.openstreetmap.org/{z}/{x}/{y}.png"
```

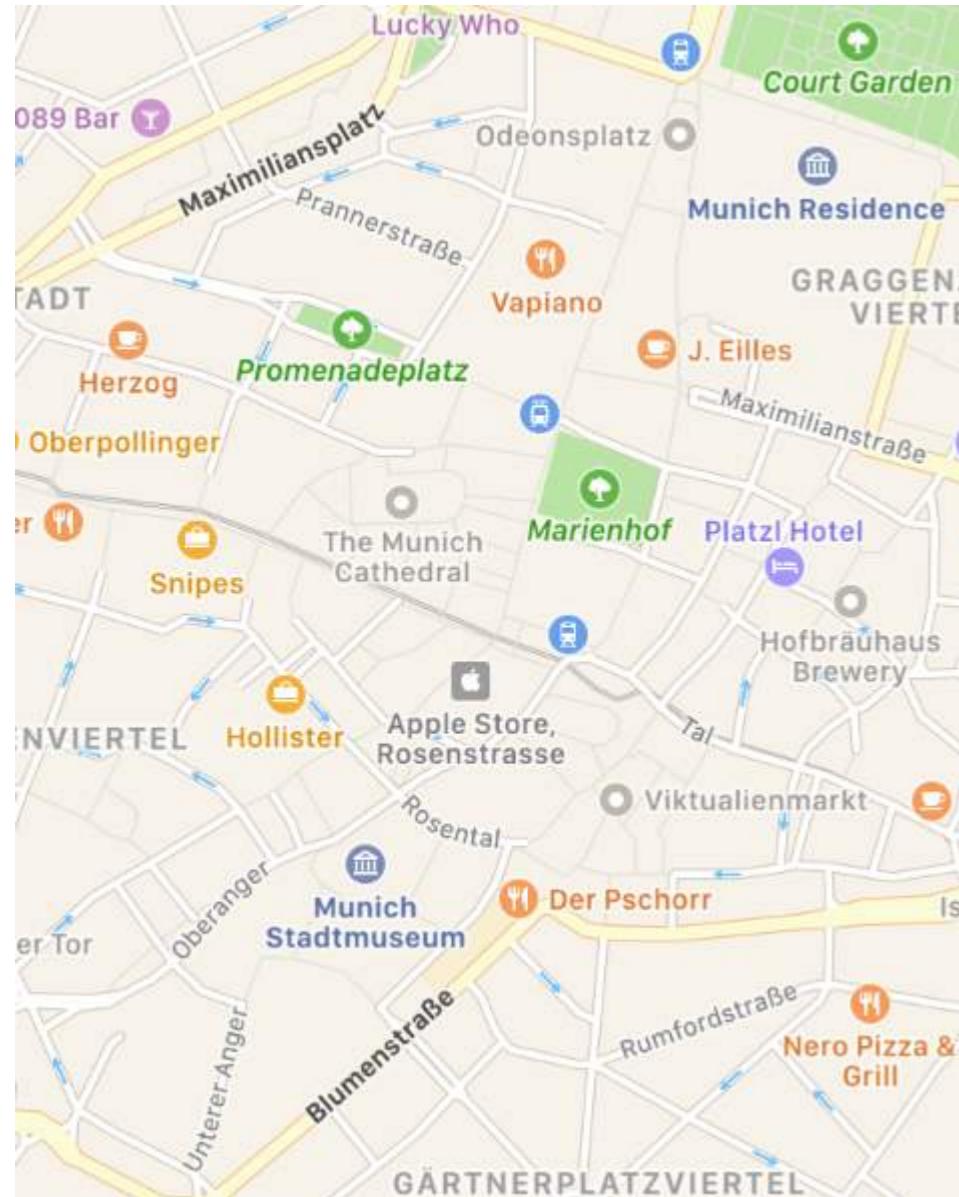
```
let overlay = MKTileOverlay(urlTemplate: urlTemplate)
overlay.canReplaceMapContent = true
```

配置好覆盖层后，您必须将其添加到您的mapView中。

```
mapView.add(overlay, level: .aboveLabels)
```

要使用自定义地图，建议将level设置为`.aboveLabels`。否则，默认标签将会显示在您的自定义地图上。如果您想看到默认标签，可以在这里选择`.aboveRoads`。

如果您现在运行项目，会发现地图仍然显示默认地图：



这是因为我们还没有告诉MapView如何渲染覆盖层。这就是为什么你之前必须设置代理的原因。现在你可以在视图控制器中添加`func mapView(_ mapView: MKMapView, rendererFor overlay: MKOverlay) -> MKOverlayRenderer`方法：

```
func mapView(_ mapView: MKMapView, rendererFor overlay: MKOverlay) -> MKOverlayRenderer {
    if overlay is MKTileOverlay {
        let renderer = MKTileOverlayRenderer(overlay: overlay)
        return renderer
    } else {
        return MKTileOverlayRenderer()
    }
}
```

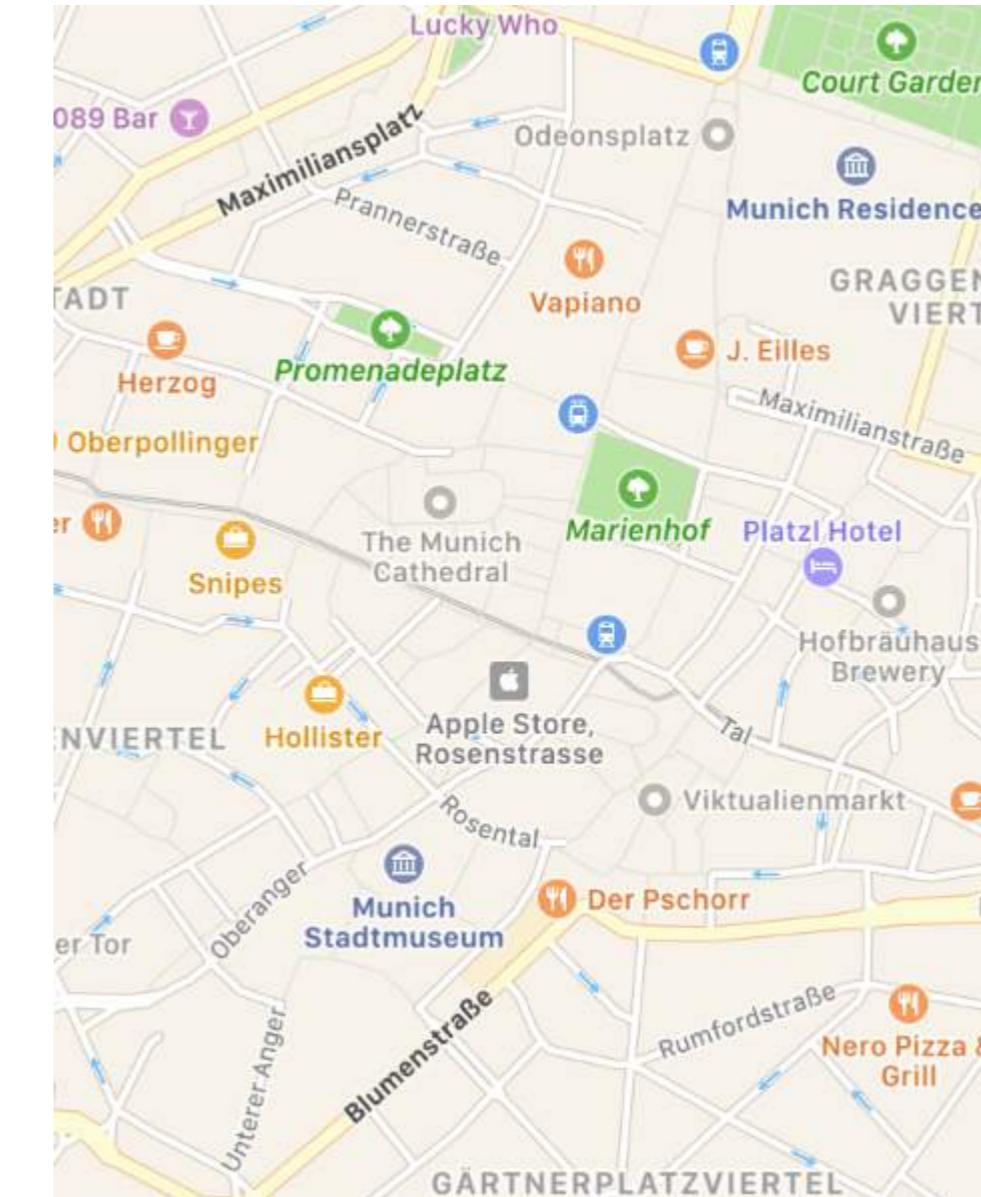
```
let overlay = MKTileOverlay(urlTemplate: urlTemplate)
overlay.canReplaceMapContent = true
```

After you configured the overlay, you must add it to your mapView.

```
mapView.add(overlay, level: .aboveLabels)
```

To use custom maps, it is recommended to use `.aboveLabels` for level. Otherwise, the default labels would be visible on your custom map. If you want to see the default labels, you can choose `.aboveRoads` here.

If you would run your project now, you would recognize, that your map would still show the default map:



That's because we haven't told the mapView yet, how to render the overlay. This is the reason, why you had to set the delegate before. Now you can add `func mapView(_ mapView: MKMapView, rendererFor overlay: MKOverlay) -> MKOverlayRenderer` to your view controller:

```
func mapView(_ mapView: MKMapView, rendererFor overlay: MKOverlay) -> MKOverlayRenderer {
    if overlay is MKTileOverlay {
        let renderer = MKTileOverlayRenderer(overlay: overlay)
        return renderer
    } else {
        return MKTileOverlayRenderer()
    }
}
```

}

这将返回正确的MKOverlayRenderer给你的MapView。如果你现在运行项目，你应该会看到这样的地图：



}

This will return the correct MKOverlayRenderer to your mapView. If you run your project now, you should see a map like this:



如果你想显示另一张地图，只需更改URL模板。OSM Wiki中有一个tile-servers列表。

## 第68.6节：滚动到坐标和缩放级别

当你向用户显示一个位置时，你可能希望MKMapView以缩放级别显示一个坐标，而不是设置一个区域来显示。此功能默认未实现，因此你需要扩展MKMapView，添加从coordinate和zoom-level到MKCoordinateRegion的复杂计算方法。

```
let MERCATOR_OFFSET = 268435456.0
let MERCATOR_RADIUS = 85445659.44705395
let DEGREES = 180.0

public extension MKMapView {

    //MARK: 地图转换方法

    private func longitudeToPixelSpaceX(longitude:Double) -> Double {
        return round(MERCATOR_OFFSET + MERCATOR_RADIUS * longitude * M_PI / DEGREES)
    }
}
```

If you want to display another map, you just have to change the URL-template. There is a [list of tile-servers](#) in the OSM Wiki.

## Section 68.6: Scroll to coordinate and zoom-level

When you show a location to your users, you might want the MKMapView to display a coordinate at a zoom-level instead of setting a region to show. This functionality is not implemented by default, so you need to extend MKMapView with a methods that do the complex calculation from a coordinate and zoom-level to a MKCoordinateRegion.

```
let MERCATOR_OFFSET = 268435456.0
let MERCATOR_RADIUS = 85445659.44705395
let DEGREES = 180.0

public extension MKMapView {

    //MARK: Map Conversion Methods

    private func longitudeToPixelSpaceX(longitude:Double) -> Double {
        return round(MERCATOR_OFFSET + MERCATOR_RADIUS * longitude * M_PI / DEGREES)
    }
}
```

```

}

private func latitudeToPixelSpaceY(latitude:Double) -> Double{
    return round(MERCATOR_OFFSET - MERCATOR_RADIUS * log((1 + sin(latitude * M_PI / DEGREES)) /
(1 - sin(latitude * M_PI / DEGREES))) / 2.0)
}

private func pixelSpaceXToLongitude(pixelX:Double) -> Double{
    return ((round(pixelX) - MERCATOR_OFFSET) / MERCATOR_RADIUS) * DEGREES / M_PI
}

private func pixelSpaceYToLatitude(pixelY:Double) -> Double{
    return (M_PI / 2.0 - 2.0 * atan(exp((round(pixelY) - MERCATOR_OFFSET) / MERCATOR_RADIUS))) *
DEGREES / M_PI
}

private func coordinateSpanWithCenterCoordinate(centerCoordinate:CLLocationCoordinate2D,
zoomLevel:Double) -> MKCoordinateSpan{
    // 将中心坐标转换为像素空间
    let centerPixelX = longitudeToPixelSpaceX(longitude: centerCoordinate.longitude)
    let centerPixelY = latitudeToPixelSpaceY(latitude: centerCoordinate.latitude)
    print(centerCoordinate)
    // 根据缩放级别确定缩放值
    let zoomExponent:Double = 20.0 - zoomLevel
    let zoomScale:Double = pow(2.0, zoomExponent)
    // 按像素空间缩放地图的大小
    let mapSizeInPixels = self.bounds.size
    let scaledMapWidth = Double(mapSizeInPixels.width) * zoomScale
    let scaledMapHeight = Double(mapSizeInPixels.height) * zoomScale
    // 计算左上角像素的位置
    let topLeftPixelX = centerPixelX - (scaledMapWidth / 2.0)
    let topLeftPixelY = centerPixelY - (scaledMapHeight / 2.0)
    // 计算左右经度的差值
    let minLng = pixelSpaceXToLongitude(pixelX: topLeftPixelX)
    let maxLng = pixelSpaceXToLongitude(pixelX: topLeftPixelX + scaledMapWidth)
    let longitudeDelta = maxLng - minLng
    let minLat = pixelSpaceYToLatitude(pixelY: topLeftPixelY)
    let maxLat = pixelSpaceYToLatitude(pixelY: topLeftPixelY + scaledMapHeight)
    let latitudeDelta = -1.0 * (maxLat - minLat)
    return MKCoordinateSpan(latitudeDelta: latitudeDelta, longitudeDelta: longitudeDelta)
}

/**
将 `MKMapView` 的中心设置为带有自定义缩放级别的 `CLLocationCoordinate2D`。
无需手动设置区域。:-)

```

- 作者 : Mylene Bayan (GitHub)  
`public func setCenter(_ coordinate:CLLocationCoordinate2D, zoomLevel:Double, animated:Bool){`  
 // 将过大的数字限制为28  
 `var zoomLevel = zoomLevel`  
 `zoomLevel = min(zoomLevel, 28)`  
 // 使用缩放级别计算区域  
 `print(coordinate)`  
 `let span = self.coordinateSpanWithCenterCoordinate(centerCoordinate: coordinate, zoomLevel:`  
 `zoomLevel)`  
 `let region = MKCoordinateRegionMake(coordinate, span)`  
 `if region.center.longitude == -180.0000000{`  
 `print("无效区域")`  
`}`  
`else{`  
 `self.setRegion(region, animated: animated)`

```

}

private func latitudeToPixelSpaceY(latitude:Double) -> Double{
    return round(MERCATOR_OFFSET - MERCATOR_RADIUS * log((1 + sin(latitude * M_PI / DEGREES)) /
(1 - sin(latitude * M_PI / DEGREES))) / 2.0)
}

private func pixelSpaceXToLongitude(pixelX:Double) -> Double{
    return ((round(pixelX) - MERCATOR_OFFSET) / MERCATOR_RADIUS) * DEGREES / M_PI
}

private func pixelSpaceYToLatitude(pixelY:Double) -> Double{
    return (M_PI / 2.0 - 2.0 * atan(exp((round(pixelY) - MERCATOR_OFFSET) / MERCATOR_RADIUS))) *
DEGREES / M_PI
}

private func coordinateSpanWithCenterCoordinate(centerCoordinate:CLLocationCoordinate2D,
zoomLevel:Double) -> MKCoordinateSpan{
    // convert center coordiate to pixel space
    let centerPixelX = longitudeToPixelSpaceX(longitude: centerCoordinate.longitude)
    let centerPixelY = latitudeToPixelSpaceY(latitude: centerCoordinate.latitude)
    print(centerCoordinate)
    // determine the scale value from the zoom level
    let zoomExponent:Double = 20.0 - zoomLevel
    let zoomScale:Double = pow(2.0, zoomExponent)
    // scale the map's size in pixel space
    let mapSizeInPixels = self.bounds.size
    let scaledMapWidth = Double(mapSizeInPixels.width) * zoomScale
    let scaledMapHeight = Double(mapSizeInPixels.height) * zoomScale
    // figure out the position of the top-left pixel
    let topLeftPixelX = centerPixelX - (scaledMapWidth / 2.0)
    let topLeftPixelY = centerPixelY - (scaledMapHeight / 2.0)
    // find delta between left and right longitudes
    let minLng = pixelSpaceXToLongitude(pixelX: topLeftPixelX)
    let maxLng = pixelSpaceXToLongitude(pixelX: topLeftPixelX + scaledMapWidth)
    let longitudeDelta = maxLng - minLng
    let minLat = pixelSpaceYToLatitude(pixelY: topLeftPixelY)
    let maxLat = pixelSpaceYToLatitude(pixelY: topLeftPixelY + scaledMapHeight)
    let latitudeDelta = -1.0 * (maxLat - minLat)
    return MKCoordinateSpan(latitudeDelta: latitudeDelta, longitudeDelta: longitudeDelta)
}

/**
Sets the center of the `MKMapView` to a `CLLocationCoordinate2D` with a custom zoom-level.
There is no nee to set a region manually. :-)

```

- 作者 : Mylene Bayan (on GitHub)  
`public func setCenter(_ coordinate:CLLocationCoordinate2D, zoomLevel:Double, animated:Bool){`  
 // clamp large numbers to 28  
 `var zoomLevel = zoomLevel`  
 `zoomLevel = min(zoomLevel, 28)`  
 // use the zoom level to compute the region  
 `print(coordinate)`  
 `let span = self.coordinateSpanWithCenterCoordinate(centerCoordinate: coordinate, zoomLevel:`  
 `zoomLevel)`  
 `let region = MKCoordinateRegionMake(coordinate, span)`  
 `if region.center.longitude == -180.0000000{`  
 `print("Invalid Region")`  
`}`  
`else{`  
 `self.setRegion(region, animated: animated)`

```
    }
}
```

(原始的 Swift 2 版本由 [Mylene Bayan](#) 编写，可在 [GitHub](#) 上找到)

实现此 `extension` 后，您可以按如下方式设置中心坐标：

```
let centerCoordinate = CLLocationCoordinate2DMake(48.136315, 11.5752901) // 纬度, 经度
mapView?.setCenter(centerCoordinate, zoomLevel: 15, animated: true)
```

`zoomLevel` 是一个 `Double` 类型的值，通常在 0 到 21 之间（21 是非常高的缩放级别），但允许的最大值可达 28。

## 第68.7节：使用注释

### 获取所有注释

```
//以下方法返回地图上添加的所有注释对象
NSArray *allAnnotations = mapView.annotations;
```

### 获取注释视图

```
for (id<MKAnnotation> annotation in mapView.annotations)
{
    MKAnnotationView* annotationView = [mapView viewForAnnotation:annotation];
    if (annotationView)
    {
        // 对注释视图进行操作
        // 例如更改注释视图的图片
        annotationView.image = [UIImage imageNamed:@"SelectedPin.png"];
    }
}
```

### 移除所有注释

```
[mapView removeAnnotations:mapView.annotations]
```

### 移除单个注释

```
//获取所有注释
NSArray *allAnnotations = self.myMapView.annotations;

if (allAnnotations.count > 0)
{
    //获取第一个标注
    id <MKAnnotation> annotation=[allAnnotations firstObject];

    //移除标注
    [mapView removeAnnotation:annotation];
}
```

```
    }
}
```

(The original Swift 2 version by [Mylene Bayan](#) can be found on [GitHub](#))

After you implemented this `extension`, you can set the center coordinate as following:

```
let centerCoordinate = CLLocationCoordinate2DMake(48.136315, 11.5752901) //latitude, longitude
mapView?.setCenter(centerCoordinate, zoomLevel: 15, animated: true)
```

`zoomLevel` is a `Double` value, usually between 0 and 21 (which is a very high zoom-level), but values up to 28 are allowed.

## Section 68.7: Working With Annotation

### Get All Annotation

```
//following method returns all annotations object added on map
NSArray *allAnnotations = mapView.annotations;
```

### Get Annotation View

```
for (id<MKAnnotation> annotation in mapView.annotations)
{
    MKAnnotationView* annotationView = [mapView viewForAnnotation:annotation];
    if (annotationView)
    {
        // Do something with annotation view
        // for e.g change image of annotation view
        annotationView.image = [UIImage imageNamed:@"SelectedPin.png"];
    }
}
```

### Remove All Annotations

```
[mapView removeAnnotations:mapView.annotations]
```

### Remove Single Annotation

```
//getting all Annotation
NSArray *allAnnotations = self.myMapView.annotations;

if (allAnnotations.count > 0)
{
    //getting first annoation
    id <MKAnnotation> annotation=[allAnnotations firstObject];

    //removing annotation
    [mapView removeAnnotation:annotation];
}
```

## 第68.8节：添加MKMapView

### Swift

## Section 68.8: Add MKMapView

### Swift

```
let mapView = MKMapView(frame: CGRect(x: 0, y: 0, width: 320, height: 500))
```

建议将mapView作为包含它的ViewController的属性存储，因为在更复杂的实现中你可能需要访问它。

#### Objective C

```
self.map = [[MKMapView alloc] initWithFrame:CGRectMake(0, 0, self.view.frame.size.width,  
self.view.frame.size.height)];  
[self.view addSubview:self.map];
```

## 第68.9节：显示用户位置和用户跟踪示例

这将显示用户在地图上的位置

#### Objective-C

```
[self.map setShowsUserLocation:YES];
```

#### Swift

```
self.map?.showsUserLocation = true
```

```
let mapView = MKMapView(frame: CGRect(x: 0, y: 0, width: 320, height: 500))
```

It's recommended to store the mapView as a property of the containing ViewController since you might want to access it in more complex implementations.

#### Objective C

```
self.map = [[MKMapView alloc] initWithFrame:CGRectMake(0, 0, self.view.frame.size.width,  
self.view.frame.size.height)];  
[self.view addSubview:self.map];
```

## Section 68.9: Show UserLocation and UserTracking example

This will show the user location on the map

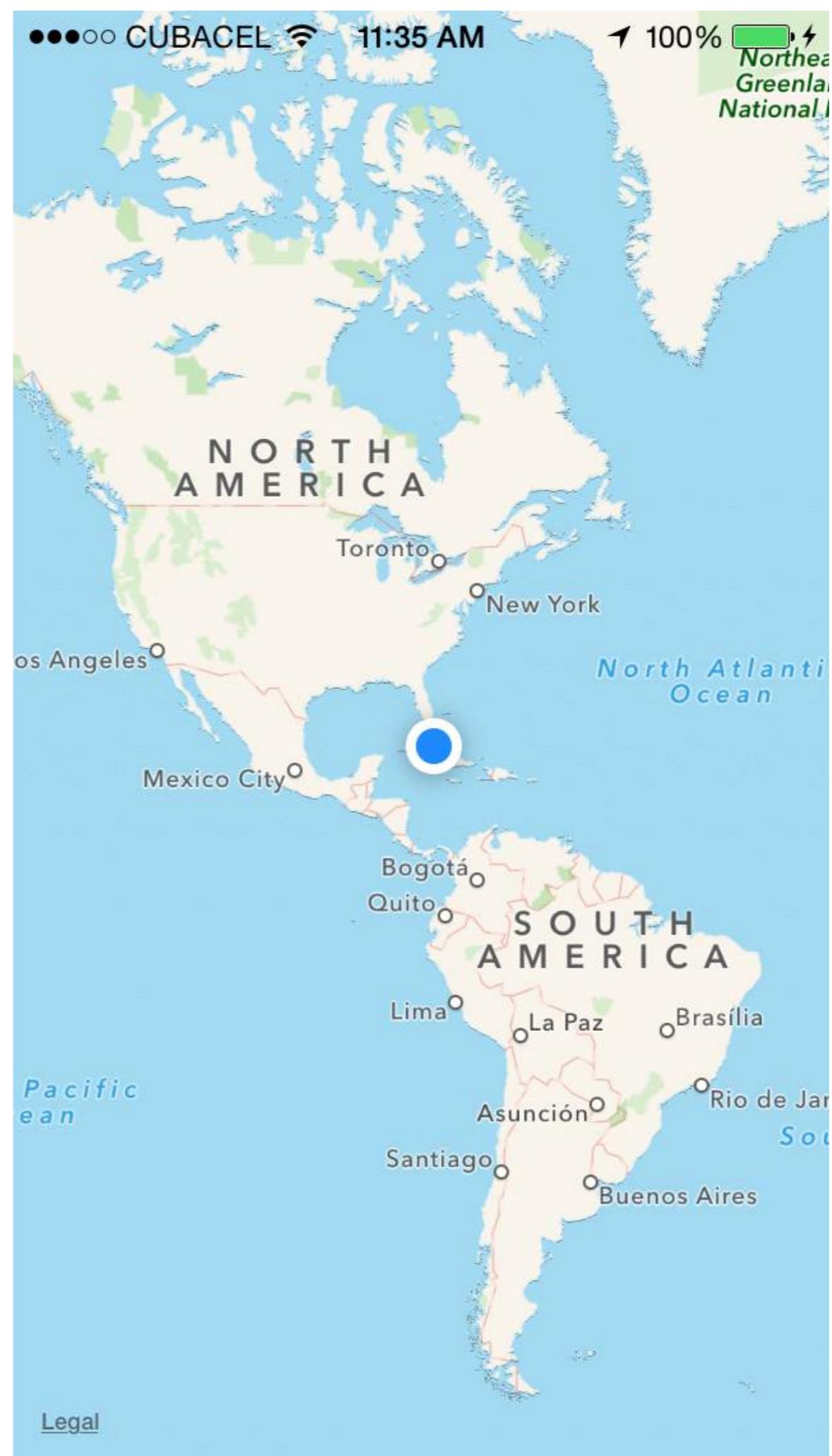
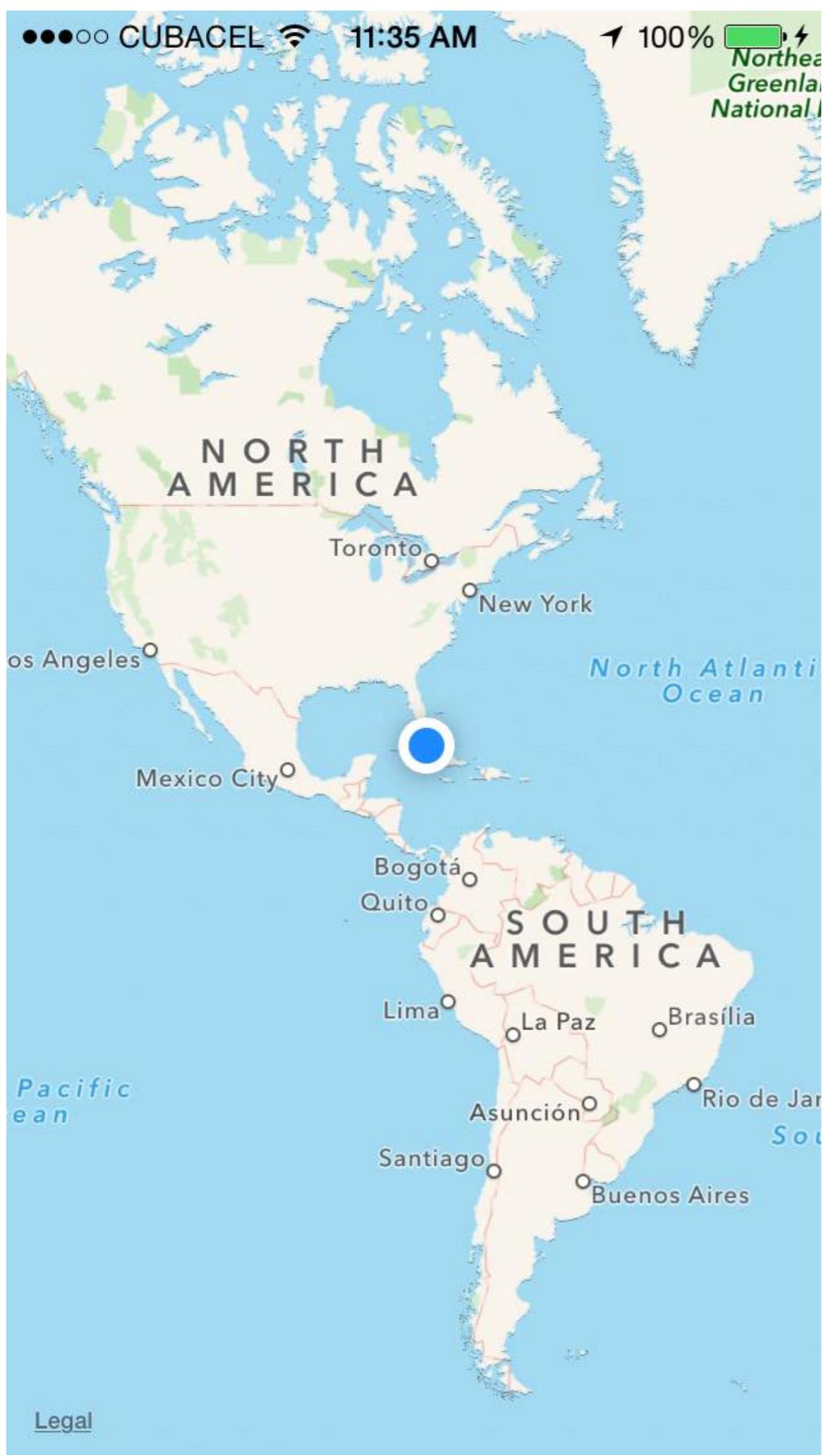
#### Objective-C

```
[self.map setShowsUserLocation:YES];
```

#### Swift

```
self.map?.showsUserLocation = true
```





这将跟踪用户在地图上的位置，并根据情况更新区域

#### Objective-C

```
[self.map setUserTrackingMode:MKUserTrackingModeFollow];
```

#### Swift

```
self.map?.userTrackingMode = .follow
```

## 第68.10节：在地图上添加图钉/点注释

为了在地图上标注某个兴趣点，我们使用图钉注释。现在，首先创建注释对象。

```
MKPointAnnotation *pointAnnotation = [[MKPointAnnotation alloc] init];
```

现在为pointAnnotation提供坐标，如下：

```
CLLocationCoordinate2D coordinate = CLLocationCoordinate2DMake(23.054625, 72.534562);  
pointAnnotation.coordinate = coordinate;
```

现在，为注释提供标题和副标题，

```
pointAnnotation.title = @"XYZ Point";  
pointAnnotation.subtitle = @"Ahmedabad Area";
```

现在，将此注释添加到地图上。

```
[self.mapView addAnnotation:pointAnnotation];
```

耶..万岁..你已经完成了这项工作。你现在可以在给定坐标看到点注释（红色图钉）。

但是现在，如果你想更改图钉的颜色（可用的三种颜色是紫色、红色和绿色），请按照以下步骤操作。

将mapView的代理设置为self，

```
self.mapView.delegate = self;
```

添加 MKMapViewDelegate 实现。现在添加以下方法，

```
- (MKAnnotationView *)mapView:(MKMapView *)mapView viewForAnnotation:(id <MKAnnotation>)annotation  
{  
    // 如果是用户位置，直接返回 nil，因为它有用户位置自己的标注，  
    // 如果你想更改它，则使用此对象；  
    if ([annotation isKindOfClass:[MKUserLocation class]])  
        return nil;  
  
    if ([annotation isKindOfClass:[MKPointAnnotation class]])  
    {  
        // 如果有可复用的标注视图，则使用它  
        MKAnnotationView *pinView = [mapView  
dequeueReusableAnnotationViewWithIdentifier:@"PinAnnotationView"];  
  
        if (!pinView)  
        {  
            // 如果没有可复用的，则创建新的。  
        }  
    }  
}
```

This will track the user location on the map, updating regions according

#### Objective-C

```
[self.map setUserTrackingMode:MKUserTrackingModeFollow];
```

#### Swift

```
self.map?.userTrackingMode = .follow
```

## Section 68.10: Adding Pin/Point Annotation on map

For annotating some point of interest on map, we use pin annotation. Now, start by creating annotation object first.

```
MKPointAnnotation *pointAnnotation = [[MKPointAnnotation alloc] init];
```

Now provide coordinate to pointAnnotation,as

```
CLLocationCoordinate2D coordinate = CLLocationCoordinate2DMake(23.054625, 72.534562);  
pointAnnotation.coordinate = coordinate;
```

Now, provide title and subtitle to annotation,

```
pointAnnotation.title = @"XYZ Point";  
pointAnnotation.subtitle = @"Ahmedabad Area";
```

Now, add this annotation to map.

```
[self.mapView addAnnotation:pointAnnotation];
```

Yeaah.. Hurrah.. you have done the job. You can now see point annotation(red coloured pin) at given coordinate.

But now, what if you want to change color of the pin(3 available colors are - Purple,red and green). Then follow this step.

set mapView's delegate to self,

```
self.mapView.delegate = self;
```

Add MKMapViewDelegate implementation. Now add following method then,

```
- (MKAnnotationView *)mapView:(MKMapView *)mapView viewForAnnotation:(id <MKAnnotation>)annotation  
{  
    // If it's the user location, just return nil, because it have user location's own annotation,  
    // if you want to change that, then use this object;  
    if ([annotation isKindOfClass:[MKUserLocation class]])  
        return nil;  
  
    if ([annotation isKindOfClass:[MKPointAnnotation class]])  
    {  
        // Use dequeued pin if available  
        MKAnnotationView *pinView = [mapView  
dequeueReusableAnnotationViewWithIdentifier:@"PinAnnotationView"];  
  
        if (!pinView)  
        {  
            // If not dequeued, then create new.  
        }  
    }  
}
```

```

pinView = [[MKAnnotationView alloc] initWithAnnotation:annotation
reuseIdentifier:@"PinAnnotationView"];
pinView.canShowCallout = YES;
pinView.image = [UIImage imageNamed:@"abc.png"];
pinView.calloutOffset = CGPointMake(0, 32);
} else {
pinView.annotation = annotation;
}
return pinView;
}
return nil;
}

```

## 第68.11节：调整地图视图的可见区域以显示所有标注

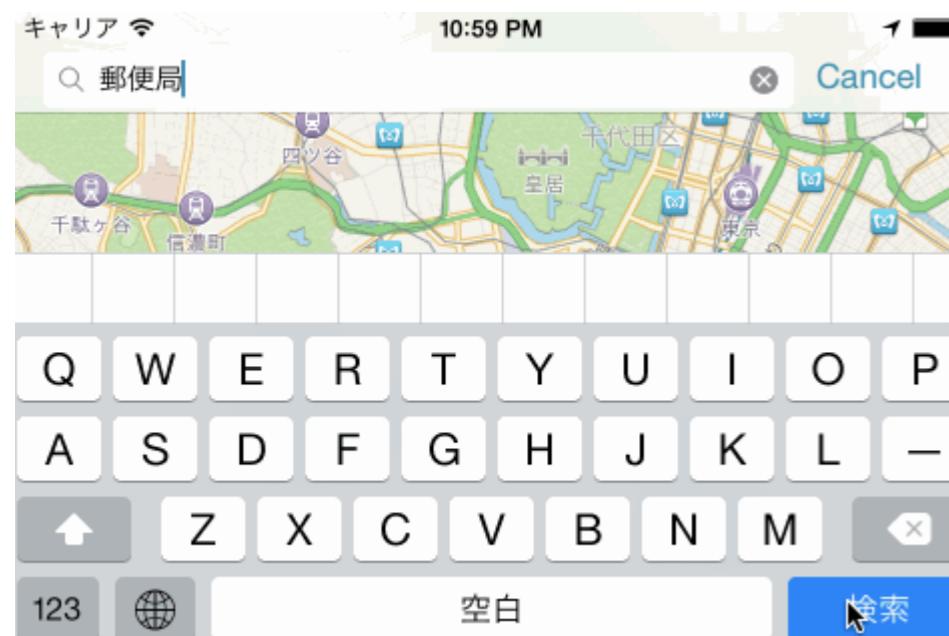
Swift :

```
mapView.showAnnotations(mapView.annotations, animated: true)
```

Objective-C :

```
[mapView showAnnotations:mapView.annotations animated:YES];
```

演示：



```

pinView = [[MKAnnotationView alloc] initWithAnnotation:annotation
reuseIdentifier:@"PinAnnotationView"];
pinView.canShowCallout = YES;
pinView.image = [UIImage imageNamed:@"abc.png"];
pinView.calloutOffset = CGPointMake(0, 32);
} else {
pinView.annotation = annotation;
}
return pinView;
}
return nil;
}

```

## Section 68.11: Adjust the map view's visible rect in order to display all annotations

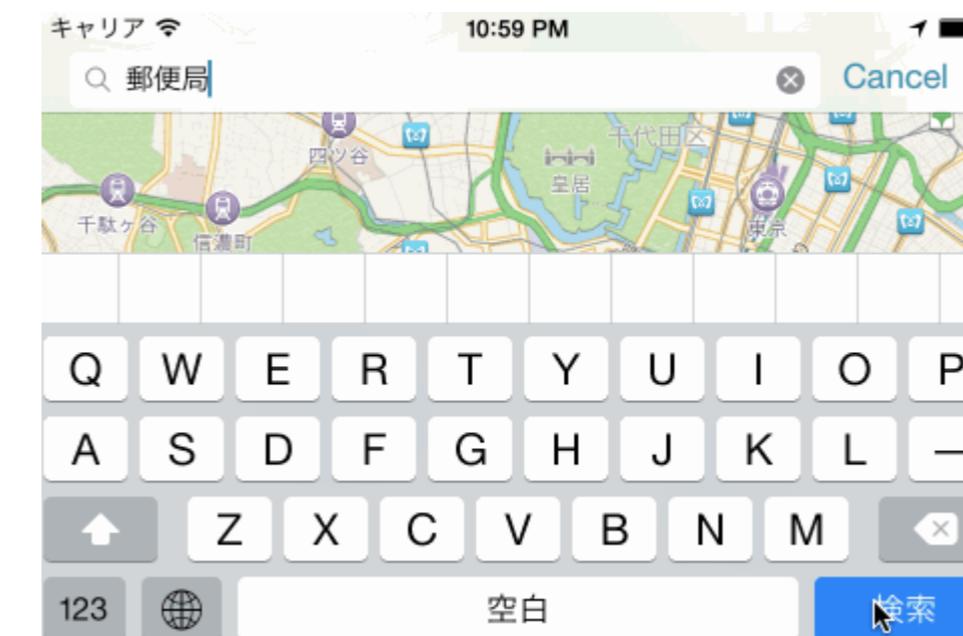
Swift:

```
mapView.showAnnotations(mapView.annotations, animated: true)
```

Objective-C:

```
[mapView showAnnotations:mapView.annotations animated:YES];
```

Demo:



# 第69章：NSArray

这里有一些有用的实用函数/方法，可以作为数组扩展使用，方便开发者通过一行代码执行某些关键的数组操作。

## 第69.1节：将数组转换为json字符串

调用此函数时，参数为类型为“any”的数组。它将返回json字符串。json字符串用于在Swift中作为请求输入参数提交数组到网络服务调用。

```
//-----  
  
let array = [[ "one" : 1], [ "two" : 2], [ "three" : 3], [ "four" : 4]]  
  
let jsonString = convertIntoJSONString(arrayObject: array)  
print("jsonString - \(jsonString)")  
  
//-----  
  
func convertIntoJSONString(arrayObject: [Any]) -> String? {  
  
    do {  
        let jsonData: Data = try JSONSerialization.data(withJSONObject: arrayObject, options:  
        [])  
        if let jsonString = NSString(data: jsonData, encoding: String.Encoding.utf8.rawValue){  
            return jsonString as String  
        }  
    } catch let error as NSError {  
        print("Array convertIntoJSON - \(error.description)")  
    }  
    return nil  
}
```

# Chapter 69: NSArray

Here are some useful utility functions/methods that can be used as with Array extension for ease of developer to perform certain critical operations on array with help of single line code.

## Section 69.1: Convert Array into json string

Call this function with parameter argument as array with type 'any'. It will return you json string. Json string is used to submit array in web service call as request input parameter in Swift.

```
//-----  
  
let array = [[ "one" : 1], [ "two" : 2], [ "three" : 3], [ "four" : 4]]  
  
let jsonString = convertIntoJSONString(arrayObject: array)  
print("jsonString - \(jsonString)")  
  
//-----  
  
func convertIntoJSONString(arrayObject: [Any]) -> String? {  
  
    do {  
        let jsonData: Data = try JSONSerialization.data(withJSONObject: arrayObject, options:  
        [])  
        if let jsonString = NSString(data: jsonData, encoding: String.Encoding.utf8.rawValue){  
            return jsonString as String  
        }  
    } catch let error as NSError {  
        print("Array convertIntoJSON - \(error.description)")  
    }  
    return nil  
}
```

# 第70章：NSAttributedString

## 第70.1节：创建具有自定义字距（字母间距）的字符串

NSAttributedString（及其可变子类NSMutableAttributedString）允许你创建在用户看来外观复杂的字符串。

一个常见的应用是使用它来显示字符串并添加自定义字距/字母间距。

实现方法如下（假设label是一个UILabel），为单词“kerning”设置不同的字距

### Swift

```
var attributedString = NSMutableAttributedString("Apply kerning")
attributedString.addAttribute(attribute: NSKernAttributeName, value: 5, range: NSMakeRange(6, 7))
label.attributedText = attributedString
```

### Objective-C

```
NSMutableAttributedString *attributedString;
attributedString = [[NSMutableAttributedString alloc] initWithString:@"Apply kerning"];
[attributedString addAttribute: NSKernAttributeName value:@5 range:NSMakeRange(6, 7)];
[label setAttributedText:attributedString];
```

## 第70.2节：更改单词或字符串的颜色

### Objective-C

```
UIColor *color = [UIColor redColor];
NSString *textToFind = @"redword";

NSMutableAttributedString *attrsString = [[NSMutableAttributedString alloc]
initWithAttributedString:yourLabel.attributedText];

// 搜索单词出现位置
NSRange range = [yourLabel.text rangeOfString:textToFind];
if (range.location != NSNotFound) {
    [attrsString addAttribute:NSForegroundColorAttributeName value:color range:range];
}

// 设置富文本
yourLabel.attributedText = attrsString;
```

### Swift

```
let color = UIColor.red;
let textToFind = "redword"

let attrsString = NSMutableAttributedString(string:yourlabel.text);

// 搜索单词出现位置
let range = (yourlabel.text! as NSString).range(of: textToFind)
if (range.length > 0) {
    attrsString.addAttribute(NSForegroundColorAttributeName, value:color, range:range)
}
```

# Chapter 70: NSAttributedString

## Section 70.1: Creating a string that has custom kerning (letter spacing)

NSAttributedString（and its mutable sibling NSMutableAttributedString）allows you to create strings that are complex in their appearance to the user.

A common application is to use this to display a string and adding custom kerning / letter-spacing.

This would be achieved as follows (where label is a UILabel), giving a different kerning for the word "kerning"

### Swift

```
var attributedString = NSMutableAttributedString("Apply kerning")
attributedString.addAttribute(attribute: NSKernAttributeName, value: 5, range: NSMakeRange(6, 7))
label.attributedText = attributedString
```

### Objective-C

```
NSMutableAttributedString *attributedString;
attributedString = [[NSMutableAttributedString alloc] initWithString:@"Apply kerning"];
[attributedString addAttribute: NSKernAttributeName value:@5 range:NSMakeRange(6, 7)];
[label setAttributedText:attributedString];
```

## Section 70.2: Change the color of a word or string

### Objective-C

```
UIColor *color = [UIColor redColor];
NSString *textToFind = @"redword";

NSMutableAttributedString *attrsString = [[NSMutableAttributedString alloc]
initWithAttributedString:yourLabel.attributedText];

// search for word occurrence
NSRange range = [yourLabel.text rangeOfString:textToFind];
if (range.location != NSNotFound) {
    [attrsString addAttribute:NSForegroundColorAttributeName value:color range:range];
}

// set attributed text
yourLabel.attributedText = attrsString;
```

### Swift

```
let color = UIColor.red;
let textToFind = "redword"

let attrsString = NSMutableAttributedString(string:yourlabel.text);

// search for word occurrence
let range = (yourlabel.text! as NSString).range(of: textToFind)
if (range.length > 0) {
    attrsString.addAttribute(NSForegroundColorAttributeName, value:color, range:range)
}
```

```
// 设置富文本  
yourlabel.attributedText = attrsString
```

#### 注意:

这里的主要方法是使用NSMutableAttributedString和选择器addAttribute:value:range，利用属性NSForegroundColorAttributeName来改变字符串范围的颜色：

```
NSMutableAttributedString *attrsString = [[NSMutableAttributedString alloc]  
initWithAttributedString:label.attributedText];  
[attrsString addAttribute:NSForegroundColorAttributeName value:color range:range];
```

你也可以使用其他方法来获取范围，例如：NSRegularExpression。

## 第70.3节：移除所有属性

### Objective-C

```
NSMutableAttributedString *mutAttString = @"string goes here";  
NSRange range = NSMakeRange(0, mutAttString.length);  
[mutAttString setAttributes:@{} range:originalRange];
```

根据苹果文档，我们使用setAttributes而不是addAttribute。

### Swift

```
mutAttString.setAttributes([:], range: NSRange(0..
```

## 第70.4节：在Swift中追加属性字符串和加粗文本

```
let someValue : String = "用户输入的内容"  
let text = NSMutableAttributedString(string: "值是: ")  
text.appendAttributedString(NSAttributedString(string: someValue, attributes:  
[UIFontAttributeName:UIFont.boldSystemFontOfSize(UIFont.systemFontSize())]))
```

结果如下所示：

值是: 用户输入的内容

## 第70.5节：创建带有删除线的字符串

### Objective-C

```
NSMutableAttributedString *attributeString = [[NSMutableAttributedString alloc]  
initWithString:@"你的字符串"];  
[attributeString addAttribute:NSStrikethroughStyleAttributeName  
value:@2  
range:NSMakeRange(0, [attributeString length])];
```

### Swift

```
let attributeString: NSMutableAttributedString = NSMutableAttributedString(string: "你的字符串  
这里")
```

```
// set attributed text  
yourlabel.attributedText = attrsString
```

### Note:

The main here is to use a `NSMutableAttributedString` and the selector `addAttribute:value:range` with the attribute `NSForegroundColorAttributeName` to change a color of a string range:

```
NSMutableAttributedString *attrsString = [[NSMutableAttributedString alloc]  
initWithAttributedString:label.attributedText];  
[attrsString addAttribute:NSForegroundColorAttributeName value:color range:range];
```

You could use another way to get the range, for example: `NSRegularExpression`.

## Section 70.3: Removing all attributes

### Objective-C

```
NSMutableAttributedString *mutAttString = @"string goes here";  
NSRange range = NSMakeRange(0, mutAttString.length);  
[mutAttString setAttributes:@{} range:originalRange];
```

As per Apple Documentation we use, `setAttributes` and not `addAttribute`.

### Swift

```
mutAttString.setAttributes([:], range: NSRange(0..
```

## Section 70.4: Appending Attributed Strings and bold text in Swift

```
let someValue : String = "Something the user entered"  
let text = NSMutableAttributedString(string: "The value is: ")  
text.appendAttributedString(NSAttributedString(string: someValue, attributes:  
[UIFontAttributeName:UIFont.boldSystemFontOfSize(UIFont.systemFontSize())]))
```

The result looks like:

The value is: **Something the user entered**

## Section 70.5: Create a string with strikethrough text

### Objective-C

```
NSMutableAttributedString *attributeString = [[NSMutableAttributedString alloc]  
initWithString:@"Your String here"];  
[attributeString addAttribute:NSStrikethroughStyleAttributeName  
value:@2  
range:NSMakeRange(0, [attributeString length])];
```

### Swift

```
let attributeString: NSMutableAttributedString = NSMutableAttributedString(string: "Your String  
here")
```

```
attributeString.addAttribute(NSStrikethroughStyleAttributeName, value: 2, range: NSMakeRange(0, attributeString.length))
```

然后你可以将其添加到你的 UILabel :

```
yourLabel.attributedText = attributeString;
```

```
attributeString.addAttribute(NSStrikethroughStyleAttributeName, value: 2, range: NSMakeRange(0, attributeString.length))
```

Then you can add this to your UILabel:

```
yourLabel.attributedText = attributeString;
```

# 第71章：HTML与NSAttributedString字符串的相互转换

## 第71.1节：Objective C代码实现HTML字符串与NSAttributedString的相互转换

HTML转NSAttributedString的转换代码：

```
//HTML字符串
NSString *htmlString=[[NSString alloc]initWithFormat:@"<!DOCTYPE html><html><body><h1>我的第一个
标题</h1><p>我的第一段.</p></body></html>"];
//使用UTF-8编码将HTML字符串转换为NSAttributedString
NSAttributedString *attributedString = [[NSAttributedString alloc]
initWithData: [htmlString
dataUsingEncoding:NSUTF8StringEncoding]
options: @{ NSDocumentTypeDocumentAttribute:
NSHTMLTextDocumentType }
documentAttributes: nil
error: nil ];
```

NSAttributedString 转换为 HTML：

```
// 用于保存 NSAttributedString 所有属性的字典
NSDictionary *documentAttributes = @{@"NSDocumentTypeDocumentAttribute": NSHTMLTextDocumentType};
// 将带有所有属性的 NSAttributedString 保存为 NSData 实体
NSData *htmlData = [attributedString dataFromRange:NSMakeRange(0, attributedString.length)
documentAttributes:documentAttributes error:NULL];
// 将 NSData 转换为使用 UTF-8 编码的 HTML 字符串
NSString *htmlString = [[NSString alloc] initWithData:htmlData encoding:NSUTF8StringEncoding];
```

# Chapter 71: Convert HTML to NSAttributedString string and vice versa

## Section 71.1: Objective C code to convert HTML string to NSAttributedString and Vice Versa

HTML to NSAttributedString conversion Code:

```
//HTML String
NSString *htmlString=[[NSString alloc]initWithFormat:@"<!DOCTYPE html><html><body><h1>My First
Heading</h1><p>My first paragraph.</p></body></html>"];
//Converting HTML string with UTF-8 encoding to NSAttributedString
NSAttributedString *attributedString = [[NSAttributedString alloc]
initWithData: [htmlString
dataUsingEncoding:NSUTF8StringEncoding]
options: @{ NSDocumentTypeDocumentAttribute:
NSHTMLTextDocumentType }
documentAttributes: nil
error: nil ];
```

NSAttributedString to HTML Conversion:

```
//Dictionary to hold all the attributes of NSAttributedString
NSDictionary *documentAttributes = @{@"NSDocumentTypeDocumentAttribute": NSHTMLTextDocumentType};
//Saving the NSAttributedString with all its attributes as a NSData Entity
NSData *htmlData = [attributedString dataFromRange:NSMakeRange(0, attributedString.length)
documentAttributes:documentAttributes error:NULL];
//Convert the NSData into HTML String with UTF-8 Encoding
NSString *htmlString = [[NSString alloc] initWithData:htmlData encoding:NSUTF8StringEncoding];
```

# 第72章：NSTimer

参数	详情
间隔	等待触发定时器的时间（秒）；或者对于重复定时器，触发之间的时间间隔。
目标	调用 selector 的对象
selector	在 Swift 中，Selector 对象指定要在 target 上调用的方法
重复	如果false，则定时器只触发一次。如果true，则每隔interval秒触发一次定时器。

## 第72.1节：创建定时器

这将创建一个定时器，在5秒后调用self的doSomething方法。

### Swift

```
let timer = NSTimer.scheduledTimerWithTimeInterval(5,
                                                 target: self,
                                                 selector: Selector(doSomething()),
                                                 userInfo: nil,
                                                 repeats: false)
```

### Swift 3

```
let timer = Timer.scheduledTimer(timeInterval: 1,
                                  target: self,
                                  selector: #selector(doSomething()),
                                  userInfo: nil,
                                  repeats: true)
```

### Objective-C

```
NSTimer *timer = [NSTimer scheduledTimerWithTimeInterval:5.0 target:self
selector:@selector(doSomething) userInfo:nil repeats:NO];
```

将 repeats 设置为false/NO表示我们希望定时器只触发一次。如果设置为true/YES，则定时器会每五秒触发一次，直到手动失效。

## 第72.2节：手动触发定时器

### Swift

```
timer.fire()
```

### Objective-C

```
[timer fire];
```

调用fire方法会使NSTimer执行它通常按计划执行的任务。

在非重复定时器中，这将自动使定时器失效。也就是说，在时间间隔结束前调用fire只会导致一次调用。

在重复定时器中，这只会调用动作而不会中断正常的计划。

# Chapter 72: NSTimer

Parameter	Details
interval	The time, in seconds, to wait before firing the timer; or, in repeating timers, the time between firings.
target	The object to call the selector on
selector	In Swift, a Selector object specifying the method to call on the target
repeats	If <code>false</code> , fire the timer only once. If <code>true</code> , fire the timer every <code>interval</code> seconds.

## Section 72.1: Creating a Timer

This will create a timer to call the doSomething method on `self` in 5 seconds.

### Swift

```
let timer = NSTimer.scheduledTimerWithTimeInterval(5,
                                                 target: self,
                                                 selector: Selector(doSomething()),
                                                 userInfo: nil,
                                                 repeats: false)
```

### Swift 3

```
let timer = Timer.scheduledTimer(timeInterval: 1,
                                  target: self,
                                  selector: #selector(doSomething()),
                                  userInfo: nil,
                                  repeats: true)
```

### Objective-C

```
NSTimer *timer = [NSTimer scheduledTimerWithTimeInterval:5.0 target:self
selector:@selector(doSomething) userInfo:nil repeats:NO];
```

Setting repeats to `false`/NO indicates that we want the timer to fire only once. If we set this to `true`/YES, it would fire every five seconds until manually invalidated.

## Section 72.2: Manually firing a timer

### Swift

```
timer.fire()
```

### Objective-C

```
[timer fire];
```

Calling the `fire` method causes an NSTimer to perform the task it would have usually performed on a schedule.

In a **non-repeating timer**, this will automatically invalidate the timer. That is, calling `fire` before the time interval is up will result in only one invocation.

In a **repeating timer**, this will simply invoke the action without interrupting the usual schedule.

## 第72.3节：定时器频率选项

重复定时器事件

Swift

```
class ViewController: UIViewController {  
  
    var timer = NSTimer()  
  
    override func viewDidLoad() {  
        NSTimer.scheduledTimerWithTimeInterval(1.0, target: self, selector:  
            Selector(self.timerMethod()), userInfo: nil, repeats: true)  
    }  
  
    func timerMethod() {  
        print("Timer method called")  
    }  
  
    func endTimer() {  
        timer.invalidate()  
    }  
}
```

Swift 3

```
class ViewController: UIViewController {  
  
    var timer = Timer()  
  
    override func viewDidLoad() {  
        Timer.scheduledTimer(timeInterval: 1.0, target: self, selector:  
            #selector(self.timerMethod()), userInfo: nil, repeats: true)  
    }  
  
    func timerMethod() {  
        print("Timer method called")  
    }  
  
    func endTimer() {  
        timer.invalidate()  
    }  
}
```

如果需要，必须手动使其失效。

Swift

非重复的延迟计时器事件

```
NSTimer.scheduledTimerWithTimeInterval(3.0, 目标: self, 选择器: Selector(self.timerMethod()),  
userInfo: nil, 重复: false)
```

Swift 3

```
Timer.scheduledTimer(timeInterval: 3.0, 目标: self, 选择器: #selector(self.timerMethod()),  
userInfo: nil, 重复: false)
```

计时器将在执行后3秒触发一次。触发后将自动失效。

## Section 72.3: Timer frequency options

Repeated Timer event

Swift

```
class ViewController: UIViewController {  
  
    var timer = NSTimer()  
  
    override func viewDidLoad() {  
        NSTimer.scheduledTimerWithTimeInterval(1.0, target: self, selector:  
            Selector(self.timerMethod()), userInfo: nil, repeats: true)  
    }  
  
    func timerMethod() {  
        print("Timer method called")  
    }  
  
    func endTimer() {  
        timer.invalidate()  
    }  
}
```

Swift 3

```
class ViewController: UIViewController {  
  
    var timer = Timer()  
  
    override func viewDidLoad() {  
        Timer.scheduledTimer(timeInterval: 1.0, target: self, selector:  
            #selector(self.timerMethod()), userInfo: nil, repeats: true)  
    }  
  
    func timerMethod() {  
        print("Timer method called")  
    }  
  
    func endTimer() {  
        timer.invalidate()  
    }  
}
```

Must be invalidated manually if desired.

Swift

Non-repeated delayed Timer event

```
NSTimer.scheduledTimerWithTimeInterval(3.0, target: self, selector: Selector(self.timerMethod()),  
userInfo: nil, repeats: false)
```

Swift 3

```
Timer.scheduledTimer(timeInterval: 3.0, target: self, selector: #selector(self.timerMethod()),  
userInfo: nil, repeats: false)
```

Timer will be fired once, 3 seconds after time of execution. Will be invalidated automatically, once fired.

## 第72.4节：使计时器失效

### Swift

```
timer.invalidate()
```

### Objective-C

```
[timer失效];
```

这将停止计时器触发。必须从创建计时器的线程调用，详见苹果的说明：

您必须从安装计时器的线程发送此消息。如果您从另一个线程发送此消息，计时器关联的输入源可能不会从其运行循环中移除，这可能导致线程无法正常退出。

注意：计时器一旦失效，就无法再次触发同一个失效的计时器。您需要重新初始化失效的计时器并触发fire方法。

## 第72.5节：使用计时器传递数据

如果您想通过定时器触发器传递一些数据，可以使用userInfo参数。

这里是一个简单的方法，简要说明了如何从计时器（Timer）向触发的方法传递数据。

[Swift 3]

```
Timer.scheduledTimer(timeInterval: 1.0, target: self, selector:#selector(iGotCall(sender:)),  
userInfo: ["Name": "i am iOS guy"], repeats:true)
```

[Objective - C]

```
NSTimer* timer = [NSTimer scheduledTimerWithTimeInterval:1.0  
                                target:self  
                           selector:@selector(iGotCall:)  
                         userInfo:@"i am iOS guy" repeats:YES];
```

上面这行代码将["Name": "i am iOS guy"]传递给了userInfo。因此当 iGotCall被调用时，您可以通过以下代码片段获取传递的值。

[Swift 3]

```
func iGotCall(sender: Timer) {  
    print((sender.userInfo)!)  
}
```

[Objective - C]

```
- (void)iGotCall:(NSTimer*)theTimer {  
    NSLog(@"%@", (NSString*)[theTimer userInfo]);  
}
```

## Section 72.4: Invalidating a timer

### Swift

```
timer.invalidate()
```

### Objective-C

```
[timer invalidate];
```

This will stop the timer from firing. **Must be called from the thread the timer was created in**, see [Apple's notes](#):

You must send this message from the thread on which the timer was installed. If you send this message from another thread, the input source associated with the timer may not be removed from its run loop, which could prevent the thread from exiting properly.

**Notes:** Once timer has been invalidated, its impossible to fire same invalidated timer. Instead you need to initialise the invalidated timer again and trigger fire method.

## Section 72.5: Passing of data using Timer

If you want to pass some data with the timer trigger you can do it with the userInfo parameter.

Here is the simple approach that gives brief idea about how you can pass the data to triggered method from the Timer.

[Swift 3]

```
Timer.scheduledTimer(timeInterval: 1.0, target: self, selector:#selector(iGotCall(sender:)),  
userInfo: ["Name": "i am iOS guy"], repeats:true)
```

[Objective - C]

```
NSTimer* timer = [NSTimer scheduledTimerWithTimeInterval:1.0  
                                target:self  
                           selector:@selector(iGotCall:)  
                         userInfo:@"i am iOS guy" repeats:YES];
```

The above line of code passing ["Name": "i am iOS guy"] into the userInfo. So now when the iGotCall get call you can get the passed value as below code snippet.

[Swift 3]

```
func iGotCall(sender: Timer) {  
    print((sender.userInfo)!)  
}
```

[Objective - C]

```
- (void)iGotCall:(NSTimer*)theTimer {  
    NSLog(@"%@", (NSString*)[theTimer userInfo]);  
}
```

# 第73章：NSDate

您可以设置不同类型的日期格式：以下是它们的完整列表。

格式	含义/描述
y	至少包含1位数字的年份。
yy	恰好包含2位数字的年份。
yyy	至少包含3位数字的一年。
yyyy	至少包含4位数字的年份。
M	至少包含1位数字的月份。
MM	至少包含2位数字的月份。
MMM	三字母月份缩写。
MMMM	月份全称。
MMMMM	单字母月份缩写（1月、6月、7月均为'J'）。
d	至少包含一位数字的日期。
dd	至少包含两位数字的日期。
"E", "EE", 或"EEE"	日期名称的三个字母缩写。
EEEE	完整的星期名称。
EEEEEE	星期名称的1个字母缩写。（星期四和星期二都用T表示）
EEEEEEE	星期名称的2个字母缩写。
a	一天中的时段（上午/下午）。
h	基于1-12的小时，至少包含1位数字。
hh	基于1-12的小时，至少包含2位数字。
H	基于0-23的小时，至少包含1位数字。
HH	基于0-23的小时，至少包含2位数字。
m	至少包含1位数字的分钟。
毫米	至少包含2位数字的分钟。
秒	至少包含1位数字的秒。
秒	至少包含2位数字的秒。

还有更多，比如根据时区(z)获取不同时间，获取带有毫秒细节的时间(S)等。

## 第73.1节：NSDateFormatter

将NSDate对象转换为字符串只需3个步骤。

### 1. 创建一个NSDateFormatter对象

Swift

```
let dateFormatter = NSDateFormatter()
```

Swift 3

```
let dateFormatter = DateFormatter()
```

Objective-C

# Chapter 73: NSDate

There are different types of date format that you can set: Here is full list of them.

Format	Meaning/Description	Example1	Example2
y	A year with at least 1 digit.	175 AD → "175"	2016 AD → "2016"
yy	A year with exactly 2 digits.	5 AD → "05"	2016 AD → "16"
yyy	A year with at least 3 digits.	5 AD → "005"	2016 AD → "2016"
yyyy	A year with at least 4 digits.	5 AD → "0005"	2016 AD → "2016"
M	A month with at least 1 digit.	July → "7"	"November" → "11"
MM	A month with at least 2 digits.	July → "07"	"November" → "11"
MMM	Three letter month abbreviation.	July → "Jul"	"November" → "Nov"
MMMM	Full name of month.	July → "July"	"November" → "November"
MMMMM	One letter month abbreviation(Jan,June,July all will have 'J').	July → "J"	"November" → "N"
d	Day with at least one digit.	8 → "8"	29 → "29"
dd	Day with at least two digits.	8 → "08"	29 → "29"
"E", "EE", or"EEE"	3 letter day abbreviation of day name.	Monday → "Mon"	Thursday → "Thu"
EEEE	Full Day name.	Monday → "Monday"	Thursday → "Thursday"
EEEEEE	1 letter day abbreviation of day name.(Thu and Tue will be 'T')	Monday → "M"	Thursday → "T"
EEEEEEE	2 letter day abbreviation of day name.	Monday → "Mo"	Thursday → "Th"
a	Period of day (AM/PM).	10 PM → "PM"	2 AM → "AM"
h	A 1-12 based hour with at least 1 digit.	10 PM → "10"	2 AM → "2"
hh	A 1-12 based hour with at least 2 digits.	10 PM → "10"	2 AM → "02"
H	A 0-23 based hour with at least 1 digit.	10 PM → "14"	2 AM → "2"
HH	A 0-23 based hour with at least 2 digits.	10 PM → "14"	2 AM → "02"
m	A minute with at least 1 digit.	7 → "7"	29 → "29"
mm	A minute with at least 2 digits.	7 → "07"	29 → "29"
s	A second with at least 1 digit.	7 → "7"	29 → "29"
ss	A second with at least 2 digits.	7 → "07"	29 → "29"

There are many more, for getting different time based on zone(z), for getting time with millisecond details(S), etc.

## Section 73.1: NSDateFormatter

Converting an `NSDate` object to string is just 3 steps.

### 1. Create an `NSDateFormatter` object

Swift

```
let dateFormatter = NSDateFormatter()
```

Swift 3

```
let dateFormatter = DateFormatter()
```

Objective-C

```
NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
```

## 2. 设置你想要的日期格式字符串

### Swift

```
dateFormatter.dateFormat = "yyyy-MM-dd 'at' HH:mm"
```

### Objective-C

```
dateFormatter.dateFormat = @"/yyyy-MM-dd 'at' HH:mm";
```

## 3. 获取格式化后的字符串

### Swift

```
let date = NSDate() // 你的 NSDate 对象  
let dateString = dateFormatter.stringFromDate(date)
```

### Swift 3

```
let date = Date() // 你的 NSDate 对象  
let dateString = dateFormatter.stringFromDate(date)
```

### Objective-C

```
NSDate *date = [NSDate date]; // 你的 NSDate 对象  
NSString *dateString = [dateFormatter stringFromDate:date];
```

这将输出类似于：2001-01-02 在 13:00

### 注意

创建一个 `NSDateFormatter` 实例是一个开销较大的操作，因此建议只创建一次并在可能的情况下重用。

## 将日期转换为字符串的有用扩展。

```
extension Date {  
    func toString() -> String {  
        let dateFormatter = DateFormatter()  
        dateFormatter.dateFormat = "MMMM dd yyyy"  
        return dateFormatter.string(from: self)  
    }  
}
```

关于 swift 日期格式化的有用链接 [swifly-getting-human-readable-date-nsdateformatter](#)。

有关构造日期格式，请参见 [date format patterns](#)。

## 第73.2节：日期比较

有4种比较日期的方法：

### Swift

- `isEqualToDate(anotherDate: NSDate) -> Bool`
- `earlierDate(anotherDate: NSDate) -> NSDate`
- `laterDate(anotherDate: NSDate) -> NSDate`
- `compare(anotherDate: NSDate) -> NSComparisonResult`

### Objective-C

```
NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
```

## 2. Set the date format in which you want your string

### Swift

```
dateFormatter.dateFormat = "yyyy-MM-dd 'at' HH:mm"
```

### Objective-C

```
dateFormatter.dateFormat = @"/yyyy-MM-dd 'at' HH:mm";
```

## 3. Get the formatted string

### Swift

```
let date = NSDate() // your NSDate object  
let dateString = dateFormatter.stringFromDate(date)
```

### Swift 3

```
let date = Date() // your NSDate object  
let dateString = dateFormatter.stringFromDate(date)
```

### Objective-C

```
NSDate *date = [NSDate date]; // your NSDate object  
NSString *dateString = [dateFormatter stringFromDate:date];
```

This will give output something like this: 2001-01-02 at 13:00

### Note

Creating an `NSDateFormatter` instance is an expensive operation, so it is recommended to create it once and reuse when possible.

## Useful extension for converting date to string.

```
extension Date {  
    func toString() -> String {  
        let dateFormatter = DateFormatter()  
        dateFormatter.dateFormat = "MMMM dd yyyy"  
        return dateFormatter.string(from: self)  
    }  
}
```

Useful links for swift date-formation [swifly-getting-human-readable-date-nsdateformatter](#).

For constructing date formats see [date format patterns](#).

## Section 73.2: Date Comparison

There are 4 methods for comparing dates:

### Swift

- `isEqualToDate(anotherDate: NSDate) -> Bool`
- `earlierDate(anotherDate: NSDate) -> NSDate`
- `laterDate(anotherDate: NSDate) -> NSDate`
- `compare(anotherDate: NSDate) -> NSComparisonResult`

### Objective-C

- (BOOL)isEqualToDate:(NSDate \*)anotherDate
- (NSDate \*)earlierDate:(NSDate \*)anotherDate
- (NSDate \*)laterDate:(NSDate \*)anotherDate
- (NSComparisonResult)compare:(NSDate \*)anotherDate

假设我们有两个日期：

#### Swift

```
let date1: NSDate = ... // 初始化为 2016年7月7日 00:00:00
let date2: NSDate = ... // 初始化为 2016年7月2日 00:00:00
```

#### Objective-C

```
NSDate *date1 = ... // 初始化为 2016年7月7日 00:00:00
NSDate *date2 = ... // 初始化为 2016年7月2日 00:00:00
```

然后，为了比较它们，我们尝试以下代码：

#### Swift

```
if date1 isEqualToDate(date2) {
    // 返回false，因为两个日期不相等
}

earlierDate: NSDate = date1.earlierDate(date2) // 返回两个日期中较早的一个 (date2)
laterDate: NSDate = date1.laterDate(date2) // 返回两个日期中较晚的一个 (date1)

result: NSComparisonResult = date1.compare(date2)

if result == .OrderedAscending {
    // 如果date1早于date2，则为true
} else if result == .OrderedSame {
    // 如果两个日期相同，则为true
} else if result == .OrderedDescending {
    // 如果date1晚于date2，则为true
}
```

#### Objective-C

```
if ([date1 isEqualToDate:date2]) {
    // 返回false，因为两个日期不相等
}

NSDate *earlierDate = [date1 earlierDate:date2]; // 返回两个日期中较早的日期，这里返回date2
NSDate *laterDate = [date1 laterDate:date2]; // 返回两个日期中较晚的日期，这里返回date1

NSComparisonResult result = [date1 compare:date2];
if (result == NSOrderedAscending) {
    // 失败
    // 如果 date1 早于 date2 会执行这里，在我们的例子中不会执行这里
} else if (result == NSOrderedSame){
    // 失败
    // 如果 date1 与 date2 相同会执行这里，在我们的例子中不会执行这里
} else{ // NSOrderedDescending
    // 成功
    // 如果 date1 晚于 date2 会执行这里，在我们的例子中会执行这里
}
```

- (BOOL)isEqualToDate:(NSDate \*)anotherDate
- (NSDate \*)earlierDate:(NSDate \*)anotherDate
- (NSDate \*)laterDate:(NSDate \*)anotherDate
- (NSComparisonResult)compare:(NSDate \*)anotherDate

Let's say we have 2 dates:

#### Swift

```
let date1: NSDate = ... // initialized as July 7, 2016 00:00:00
let date2: NSDate = ... // initialized as July 2, 2016 00:00:00
```

#### Objective-C

```
NSDate *date1 = ... // initialized as July 7, 2016 00:00:00
NSDate *date2 = ... // initialized as July 2, 2016 00:00:00
```

Then, to compare them, we try this code:

#### Swift

```
if date1 isEqualToDate(date2) {
    // returns false, as both dates aren't equal
}

earlierDate: NSDate = date1.earlierDate(date2) // returns the earlier date of the two (date 2)
laterDate: NSDate = date1.laterDate(date2) // returns the later date of the two (date1)

result: NSComparisonResult = date1.compare(date2)

if result == .OrderedAscending {
    // true if date1 is earlier than date2
} else if result == .OrderedSame {
    // true if the dates are the same
} else if result == .OrderedDescending {
    // true if date1 is later than date1
}
```

#### Objective-C

```
if ([date1 isEqualToDate:date2]) {
    // returns false, as both date are not equal
}

NSDate *earlierDate = [date1 earlierDate:date2]; // returns date which comes earlier from both
date, here it will return date2
NSDate *laterDate = [date1 laterDate:date2]; // returns date which comes later from both date, here
it will return date1

NSComparisonResult result = [date1 compare:date2];
if (result == NSOrderedAscending) {
    // fails
    // comes here if date1 is earlier than date2, in our case it will not come here
} else if (result == NSOrderedSame){
    // fails
    // comes here if date1 is same as date2, in our case it will not come here
} else{ // NSOrderedDescending
    // succeeds
    // comes here if date1 is later than date2, in our case it will come here
}
```

如果你想比较日期并处理秒、周、月和年：

### Swift 3

```
let dateStringUTC = "2016-10-22 12:37:48 +0000"
let dateFormatter = DateFormatter()
dateFormatter.locale = Locale(identifier: "en_US_POSIX")
dateFormatter.dateFormat = "yyyy-MM-dd HH:mm:ss X"
let date = dateFormatter.date(from: dateStringUTC)!

let now = Date()

let formatter = DateComponentsFormatter()
formatter.unitsStyle = .full
formatter.maximumUnitCount = 2
let string = formatter.string(from: date, to: Date())! + " " + NSLocalizedString("ago", comment: "added after elapsed time to say how long before")
```

或者你也可以对每个组件使用这个：

```
// 获取当前日期和时间
let currentDateTime = Date()

// 获取用户的日历
let userCalendar = Calendar.current

// 选择需要的日期和时间组件
let requestedComponents: Set<Calendar.Component> = [
    .year,
    .month,
    .day,
    .hour,
    .minute,
    .second
]

// 获取组件
let dateTimeComponents = userCalendar.dateComponents(requestedComponents, from: currentDateTime)

// 现在组件已经可用
dateTimeComponents.year
dateTimeComponents.month
dateTimeComponents.day
dateTimeComponents.hour
dateTimeComponents.minute
dateTimeComponents.second
```

## 第73.3节：从NSDate获取历史时间（例如：5秒前，2分钟前，3小时前）

这可以用于各种聊天应用、RSS订阅和社交应用中，需要显示带有时间戳的最新动态：

### Objective-C

```
- (NSString *)getHistoricTimeText:(NSDate *)since
{
    NSString *str;
    NSTimeInterval interval = [[NSDate date] timeIntervalSinceDate:since];
    if(interval < 60)
        str = [NSString stringWithFormat:@"%@is ago",(int)interval];
```

If you want to compare dates and handle seconds, weeks, months and years:

### Swift 3

```
let dateStringUTC = "2016-10-22 12:37:48 +0000"
let dateFormatter = DateFormatter()
dateFormatter.locale = Locale(identifier: "en_US_POSIX")
dateFormatter.dateFormat = "yyyy-MM-dd HH:mm:ss X"
let date = dateFormatter.date(from: dateStringUTC)!

let now = Date()

let formatter = DateComponentsFormatter()
formatter.unitsStyle = .full
formatter.maximumUnitCount = 2
let string = formatter.string(from: date, to: Date())! + " " + NSLocalizedString("ago", comment: "added after elapsed time to say how long before")
```

Or you can use this for each component:

```
// get the current date and time
let currentDateTime = Date()

// get the user's calendar
let userCalendar = Calendar.current

// choose which date and time components are needed
let requestedComponents: Set<Calendar.Component> = [
    .year,
    .month,
    .day,
    .hour,
    .minute,
    .second
]

// get the components
let dateTimeComponents = userCalendar.dateComponents(requestedComponents, from: currentDateTime)

// now the components are available
dateTimeComponents.year
dateTimeComponents.month
dateTimeComponents.day
dateTimeComponents.hour
dateTimeComponents.minute
dateTimeComponents.second
```

## Section 73.3: Get Historic Time from NSDate (eg: 5s ago, 2m ago, 3h ago)

This can be used in various chat applications, rss feeds, and social apps where you need to have latest feeds with timestamps:

### Objective-C

```
- (NSString *)getHistoricTimeText:(NSDate *)since
{
    NSString *str;
    NSTimeInterval interval = [[NSDate date] timeIntervalSinceDate:since];
    if(interval < 60)
        str = [NSString stringWithFormat:@"%@is ago",(int)interval];
```

```

else if(interval < 3600)
{
int minutes = interval/60;
    str = [NSString stringWithFormat:@"%@im ago",minutes];
}
else if(interval < 86400)
{
int hours = interval/3600;

    str = [NSString stringWithFormat:@"%@ih ago",hours];
}
else
{
    NSDateFormatter *dateFormater=[[NSDateFormatter alloc]init];
    [dateFormater setLocale:[NSLocale currentLocale]];
    NSString *dateFormat = [NSDateFormatter dateFormatFromTemplate:@"MMM d, YYYY" options:0
locale:[NSLocale currentLocale]];
    [dateFormater setDateFormat:dateFormat];
    str = [dateFormater stringFromDate:since];
}

return str;
}

```

## 第73.4节：获取Unix纪元时间

要获取Unix纪元时间，使用常量 `timeIntervalSince1970`：

### Swift

```
let date = NSDate() // 当前日期
let unixtime = date.timeIntervalSince1970
```

### Objective-C

```
NSDate *date = [NSDate date]; // 当前日期
int unixtime = [date timeIntervalSince1970];
```

## 第73.5节：从JSON日期格式获取NSDate "/Date(1268123281843)"/

在Json.NET 4.5之前，日期使用微软格式写入：" /Date(1198908717056) /"。如果您的服务器以此格式发送日期，您可以使用以下代码将其序列化为NSDate：

### Objective-C

```
(NSDate*) getDateFromJSON:(NSString *)dateString
{
    // 期望日期格式为 "/Date(1268123281843)/"
    int startPos = [dateString rangeOfString:@"]".location + 1;
    int endPos = [dateString rangeOfString:@")"].location;
    NSRange range = NSMakeRange(startPos, endPos - startPos);
    unsigned long long milliseconds = [[dateString substringWithRange:range] longLongValue];
    NSLog(@"%@", milliseconds);
    NSTimeInterval interval = milliseconds / 1000;
    NSDate *date = [NSDate dateWithTimeIntervalSince1970:interval];
    // 如果需要特定格式的NSDate，添加日期格式化代码。
    return date;
}
```

```

else if(interval < 3600)
{
    int minutes = interval/60;
    str = [NSString stringWithFormat:@"%@im ago",minutes];
}
else if(interval < 86400)
{
    int hours = interval/3600;

    str = [NSString stringWithFormat:@"%@ih ago",hours];
}
else
{
    NSDateFormatter *dateFormater=[[NSDateFormatter alloc]init];
    [dateFormater setLocale:[NSLocale currentLocale]];
    NSString *dateFormat = [NSDateFormatter dateFormatFromTemplate:@"MMM d, YYYY" options:0
locale:[NSLocale currentLocale]];
    [dateFormater setDateFormat:dateFormat];
    str = [dateFormater stringFromDate:since];
}

return str;
}

```

## Section 73.4: Get Unix Epoch time

To get [Unix Epoch Time](#), use the constant `timeIntervalSince1970`:

### Swift

```
let date = NSDate() // current date
let unixtime = date.timeIntervalSince1970
```

### Objective-C

```
NSDate *date = [NSDate date]; // current date
int unixtime = [date timeIntervalSince1970];
```

## Section 73.5: Get NSDate from JSON Date format "/Date(1268123281843)"/

Prior to Json.NET 4.5 dates were written using the Microsoft format: "/Date(1198908717056) /". If your server sends date in this format you can use the below code to serialize it to NSDate:

### Objective-C

```
(NSDate*) getDateFromJSON:(NSString *)dateString
{
    // Expect date in this format "/Date(1268123281843)/"
    int startPos = [dateString rangeOfString:@"]".location + 1;
    int endPos = [dateString rangeOfString:@")"].location;
    NSRange range = NSMakeRange(startPos, endPos - startPos);
    unsigned long long milliseconds = [[dateString substringWithRange:range] longLongValue];
    NSLog(@"%@", milliseconds);
    NSTimeInterval interval = milliseconds/1000;
    NSDate *date = [NSDate dateWithTimeIntervalSince1970:interval];
    // add code for date formatter if need NSDate in specific format.
    return date;
}
```

## 第73.6节：获取时间制类型（12小时制或24小时制）

检查当前日期是否包含上午或下午符号

### Objective-C

```
NSDateFormatter *formatter = [[NSDateFormatter alloc] init];
[formatter setLocale:[NSLocale currentLocale]];
[formatter setDateStyle:NSDateFormatterNoStyle];
[formatter setTimeStyle:NSDateFormatterShortStyle];
NSString *dateString = [formatter stringFromDate:[NSDate date]];
NSRange amRange = [dateString rangeOfString:[formatter AMSymbol]];
NSRange pmRange = [dateString rangeOfString:[formatter PMSymbol]];
BOOL is24h = (amRange.location == NSNotFound && pmRange.location == NSNotFound);
```

从`NSDateFormatter`请求时间制类型

### Objective-C

```
NSString *formatStringForHours = [NSDateFormatter dateFormatFromTemplate:@"j" options:0 locale:[NSLocale currentLocale]];
NSRange containsA = [formatStringForHours rangeOfString:@"a"];
BOOL is24h = containsA.location == NSNotFound;
```

这使用了一个名为“j”的特殊日期模板字符串，根据ICU Spec...

[...] 请求该区域设置的首选小时格式 (h、H、K 或 k)，由补充数据中 hours 元素的 preferred 属性决定。 [...] 注意，在传递给 API 的 skeleton 中使用 'j' 是唯一能让 skeleton 请求区域设置首选时间周期类型（12 小时或 24 小时）的方法。

最后一句话很重要。它“是唯一能让 skeleton 请求区域设置首选时间周期类型”的方法。由于`NSDateFormatter`和`NSCalendar`是基于 ICU 库构建的，这里同样适用。

### 参考文献

第二个选项来源于[this answer](#)。

## 第 73.7 节：获取当前日期

获取当前日期非常简单。你只需一行代码即可获得当前日期的 `NSDate` 对象，如下所示：

### Swift

```
var date = NSDate()
```

### Swift 3

```
var date = Date()
```

### Objective-C

```
NSDate *date = [NSDate date];
```

## 第73.8节：获取距离当前日期N秒的`NSDate`对象

从当前日期和时间起的新日期的秒数。使用负值表示当前日期之前的日期。

## Section 73.6: Get time cycle type (12-hour or 24-hour)

Checking whether the current date contains the symbol for AM or PM

### Objective-C

```
NSDateFormatter *formatter = [[NSDateFormatter alloc] init];
[formatter setLocale:[NSLocale currentLocale]];
[formatter setDateStyle:NSDateFormatterNoStyle];
[formatter setTimeStyle:NSDateFormatterShortStyle];
NSString *dateString = [formatter stringFromDate:[NSDate date]];
NSRange amRange = [dateString rangeOfString:[formatter AMSymbol]];
NSRange pmRange = [dateString rangeOfString:[formatter PMSymbol]];
BOOL is24h = (amRange.location == NSNotFound && pmRange.location == NSNotFound);
```

Requesting the time cycle type from `NSDateFormatter`

### Objective-C

```
NSString *formatStringForHours = [NSDateFormatter dateFormatFromTemplate:@"j" options:0
locale:[NSLocale currentLocale]];
NSRange containsA = [formatStringForHours rangeOfString:@"a"];
BOOL is24h = containsA.location == NSNotFound;
```

This uses a special date template string called "j" which according to the [ICU Spec](#) ...

[...] requests the preferred hour format for the locale (h, H, K, or k), as determined by the preferred attribute of the hours element in supplemental data. [...] Note that use of 'j' in a skeleton passed to an API is the only way to have a skeleton request a locale's preferred time cycle type (12-hour or 24-hour).

That last sentence is important. It "is the only way to have a skeleton request a locale's preferred time cycle type". Since `NSDateFormatter` and `NSCalendar` are built on the ICU library, the same holds true here.

### Reference

The second option was derived from [this answer](#).

## Section 73.7: Get Current Date

Getting current date is very easy. You get `NSDate` object of current date in just single line as follows:

### Swift

```
var date = NSDate()
```

### Swift 3

```
var date = Date()
```

### Objective-C

```
NSDate *date = [NSDate date];
```

## Section 73.8: Get NSDate Object N seconds from current date

The number of seconds from the current date and time for the new date. Use a negative value to specify a date before the current date.

为此，我们有一个名为`dateWithTimeIntervalSinceNow(seconds: NSTimeInterval) -> NSDate` (Swift) 或`+ (NSDate*)dateWithTimeIntervalSinceNow:(NSTimeInterval)seconds` (Objective-C) 的方法。

例如，如果你需要一个从当前日期起一周后的日期和一周前的日期，我们可以这样做

### Swift

```
let totalSecondsInWeek:NSTimeInterval = 7 * 24 * 60 * 60;  
//使用负值表示从今天起的前一个日期  
let nextWeek = NSDate().dateWithTimeIntervalSinceNow(totalSecondsInWeek)  
  
//使用正值表示从今天起的未来日期  
let lastWeek = NSDate().dateWithTimeIntervalSinceNow(-totalSecondsInWeek)
```

### Swift 3

```
let totalSecondsInWeek:TimeInterval = 7 * 24 * 60 * 60;  
  
//使用正值加到当前日期  
let nextWeek = Date(timeIntervalSinceNow: totalSecondsInWeek)  
  
//使用负值获取从当前日期起一周前的日期  
let lastWeek = Date(timeIntervalSinceNow: -totalSecondsInWeek)
```

### Objective-C

```
NSTimeInterval totalSecondsInWeek = 7 * 24 * 60 * 60;  
//使用负值表示从今天起的前一个日期  
NSDate *lastWeek = [NSDate dateWithTimeIntervalSinceNow:-totalSecondsInWeek];  
  
//使用正值表示从今天起的未来日期  
NSDate *nextWeek = [NSDate dateWithTimeIntervalSinceNow:totalSecondsInWeek];  
  
NSLog(@"Last Week: %@", lastWeek);  
NSLog(@"当前时间: %@", now);  
NSLog(@"下周时间: %@", nextWeek);
```

## 第73.9节：使用时区从NSDate获取UTC时间偏移量

这里将计算所需时区中当前日期的UTC时间偏移量。

```
+(NSTimeInterval)getUTCOffsetWithCurrentTimeZone:(NSTimeZone *)current forDate:(NSDate *)date {  
    NSTimeZone *utcTimeZone = [NSTimeZone timeZoneWithAbbreviation:@"UTC"];  
    NSInteger currentGMTOffset = [current secondsFromGMTForDate:date];  
    NSInteger gmtOffset = [utcTimeZone secondsFromGMTForDate:date];  
    NSTimeInterval gmtInterval = currentGMTOffset - gmtOffset;  
    return gmtInterval;  
}
```

## 第73.10节：将仅由小时和分钟组成的NSDate转换为完整NSDate

有很多情况是仅从小时和分钟格式创建NSDate，例如：08:12，这个时间作为字符串从服务器返回，您仅用这些值初始化NSDate实例。

这种情况下的缺点是您的NSDate几乎是“裸露”的，您需要做的是

For doing this we have a method named `dateWithTimeIntervalSinceNow(seconds: NSTimeInterval) -> NSDate` (Swift) or `+ (NSDate*)dateWithTimeIntervalSinceNow:(NSTimeInterval)seconds` (Objective-C).

Now for example, if you require a date one week from current date and one week to current date, then we can do it as.

### Swift

```
let totalSecondsInWeek:NSTimeInterval = 7 * 24 * 60 * 60;  
//Using negative value for previous date from today  
let nextWeek = NSDate().dateWithTimeIntervalSinceNow(totalSecondsInWeek)  
  
//Using positive value for future date from today  
let lastWeek = NSDate().dateWithTimeIntervalSinceNow(-totalSecondsInWeek)
```

### Swift 3

```
let totalSecondsInWeek:TimeInterval = 7 * 24 * 60 * 60;  
  
//Using positive value to add to the current date  
let nextWeek = Date(timeIntervalSinceNow: totalSecondsInWeek)  
  
//Using negative value to get date one week from current date  
let lastWeek = Date(timeIntervalSinceNow: -totalSecondsInWeek)
```

### Objective-C

```
NSTimeInterval totalSecondsInWeek = 7 * 24 * 60 * 60;  
//Using negative value for previous date from today  
NSDate *lastWeek = [NSDate dateWithTimeIntervalSinceNow:-totalSecondsInWeek];  
  
//Using positive value for future date from today  
NSDate *nextWeek = [NSDate dateWithTimeIntervalSinceNow:totalSecondsInWeek];  
  
NSLog(@"Last Week: %@", lastWeek);  
NSLog(@"Right Now: %@", now);  
NSLog(@"Next Week: %@", nextWeek);
```

## Section 73.9: UTC Time offset from NSDate with TimeZone

Here this will calculate the UTC time offset from current data in desired timezone.

```
+(NSTimeInterval)getUTCOffsetWithCurrentTimeZone:(NSTimeZone *)current forDate:(NSDate *)date {  
    NSTimeZone *utcTimeZone = [NSTimeZone timeZoneWithAbbreviation:@"UTC"];  
    NSInteger currentGMTOffset = [current secondsFromGMTForDate:date];  
    NSInteger gmtOffset = [utcTimeZone secondsFromGMTForDate:date];  
    NSTimeInterval gmtInterval = currentGMTOffset - gmtOffset;  
    return gmtInterval;  
}
```

## Section 73.10: Convert NSDate that is composed from hour and minute (only) to a full NSDate

There are many cases when one has created an NSDate from only an hour and minute format, i.e: 08:12 that returns from a server as a String and you initiate an NSDate instance by these **values only**.

The downside for this situation is that your NSDate is almost completely "naked" and what you need to do is to

创建：日、月、年、秒和时区，以使该对象能够与其他NSDate类型“协同工作”。

举例来说，假设 hourAndMinute 是由小时和

分钟格式组成的 NSDate 类型：

#### Objective-C

```
NSDateComponents *hourAndMinuteComponents = [calendar components:NSCalendarUnitHour |  
NSCalendarUnitMinute  
fromDate:hourAndMinute];  
NSDateComponents *componentsOfDate = [[NSCalendar currentCalendar] components:NSCalendarUnitDay |  
NSCalendarUnitMonth | NSCalendarUnitYear  
fromDate:[NSDate date]];  
  
NSDateComponents *components = [[NSDateComponents alloc] init];  
[components setDay: componentsOfDate.day];  
[components setMonth: componentsOfDate.month];  
[components setYear: componentsOfDate.year];  
[components setHour: [hourAndMinuteComponents hour]];  
[components setMinute: [hourAndMinuteComponents minute]];  
[components setSecond: 0];  
[calendar setTimezone: [NSTimeZone defaultTimeZone]];  
  
NSDate *yourFullNSDateObject = [calendar dateFromComponents:components];
```

现在你的对象完全不是“裸露”的状态。

create: day, month, year, second and time zone in order to this object to "play along" with other NSDate types.

For the sake of the example let's say that hourAndMinute is the NSDate type that is composed from hour and minute format:

#### Objective-C

```
NSDateComponents *hourAndMinuteComponents = [calendar components:NSCalendarUnitHour |  
NSCalendarUnitMinute  
fromDate:hourAndMinute];  
NSDateComponents *componentsOfDate = [[NSCalendar currentCalendar] components:NSCalendarUnitDay |  
NSCalendarUnitMonth | NSCalendarUnitYear  
fromDate:[NSDate date]];  
  
NSDateComponents *components = [[NSDateComponents alloc] init];  
[components setDay: componentsOfDate.day];  
[components setMonth: componentsOfDate.month];  
[components setYear: componentsOfDate.year];  
[components setHour: [hourAndMinuteComponents hour]];  
[components setMinute: [hourAndMinuteComponents minute]];  
[components setSecond: 0];  
[calendar setTimezone: [NSTimeZone defaultTimeZone]];  
  
NSDate *yourFullNSDateObject = [calendar dateFromComponents:components];
```

Now your object is the total opposite of being "naked".

# 第74章：NSNotificationCenter

参数	详情
名称	要为其注册观察者的通知名称；也就是说，只有具有此名称的通知才会用于将块添加到操作队列中。如果传入nil，通知中心不会使用通知的名称来决定是否将块添加到操作队列中。
obj	观察者希望接收通知的对象；也就是说，只有由此发送者发送的通知才会传递给观察者。如果传入 nil，通知中心不会使用通知的发送者来决定是否将通知传递给观察者。
queue	应将块添加到的操作队列。如果传入 nil，块将在发布线程上同步运行。
block	接收到通知时要执行的块。该块由通知中心复制，并（复印件）一直保留，直到移除观察者注册。

iOS 通知是一种简单且强大的方式，用于以松耦合的方式发送数据。也就是说，通知的发送者不必关心谁（如果有的话）接收通知，它只是将通知发布到应用的其他部分，可能会被许多对象接收，也可能因应用状态不同而无人接收。

来源：[- HACKING with Swift](#)

## 第 74.1 节：移除观察者

### Swift 2.3

```
//移除单次通知的观察者  
NSNotificationCenter.defaultCenter().removeObserver(self, name: "TestNotification", object: nil)  
  
//移除所有通知的观察者  
NotificationCenter.defaultCenter().removeObserver(self)
```

### Swift 3

```
//移除单次通知的观察者  
NotificationCenter.default.removeObserver(self, name: NSNotification.Name(rawValue:  
"TestNotification"), object: nil)  
  
//移除所有通知的观察者  
NotificationCenter.default.removeObserver(self)
```

### Objective-C

```
//移除单次通知的观察者  
[[NSNotificationCenter defaultCenter] removeObserver:self name:@"TestNotification" object:nil];  
  
//移除所有通知的观察者  
[[NSNotificationCenter defaultCenter] removeObserver:self];
```

## 第74.2节：添加观察者

### 命名规范

通知由全局NSString对象标识，其名称构成如下：

关联类名 + Did | Will + 名称的唯一部分 + Notification

例如：

- NSApplicationDidBecomeActiveNotification
- NSWindowDidMiniaturizeNotification

# Chapter 74: NSNotificationCenter

Parameter	Details
name	The name of the notification for which to register the observer; that is, only notifications with this name are used to add the block to the operation queue. If you pass nil, the notification center doesn't use a notification's name to decide whether to add the block to the operation queue.
obj	The object whose notifications the observer wants to receive; that is, only notifications sent by this sender are delivered to the observer. If you pass nil, the notification center doesn't use a notification's sender to decide whether to deliver it to the observer.
queue	The operation queue to which block should be added. If you pass nil, the block is run synchronously on the posting thread.
block	The block to be executed when the notification is received. The block is copied by the notification center and (the copy) held until the observer registration is removed.

iOS 通知是一种简单且强大的方式，用于以松耦合的方式发送数据。也就是说，通知的发送者不必关心谁（如果有的话）接收通知，它只是将通知发布到应用的其他部分，可能会被许多对象接收，也可能因应用状态不同而无人接收。

Source : [- HACKING with Swift](#)

## Section 74.1: Removing Observers

### Swift 2.3

```
//Remove observer for single notification  
NSNotificationCenter.defaultCenter().removeObserver(self, name: "TestNotification", object: nil)  
  
//Remove observer for all notifications  
NotificationCenter.defaultCenter().removeObserver(self)
```

### Swift 3

```
//Remove observer for single notification  
NotificationCenter.default.removeObserver(self, name: NSNotification.Name(rawValue:  
"TestNotification"), object: nil)  
  
//Remove observer for all notifications  
NotificationCenter.default.removeObserver(self)
```

### Objective-C

```
//Remove observer for single notification  
[[NSNotificationCenter defaultCenter] removeObserver:self name:@"TestNotification" object:nil];  
  
//Remove observer for all notifications  
[[NSNotificationCenter defaultCenter] removeObserver:self];
```

## Section 74.2: Adding an Observer

### Naming Convention

Notifications are identified by global NSString objects whose names are composed in this way:

Name of associated class + Did | Will + UniquePartOfName + Notification

For example:

- NSApplicationDidBecomeActiveNotification
- NSWindowDidMiniaturizeNotification

- NSTextViewDidChangeSelectionNotification
- NSColorPanelColorDidChangeNotification

### Swift 2.3

```
NSNotificationCenter.defaultCenter().addObserver(self,
                                             selector: #selector(self.testNotification(_:)),
                                             name: "TestNotification",
                                             object: nil)
```

### Swift 3

```
NotificationCenter.default.addObserver(self,
                                      selector: #selector(self.testNotification(_:)),
                                      name: NSNotification.Name(rawValue: "TestNotification"),
                                      object: nil)
```

### Objective-C

```
[[NSNotificationCenter defaultCenter] addObserver:self
                                         selector:@selector(testNotification:)
                                         name:@"TestNotification"
                                         object:nil];
```

附注：同样值得注意的是，添加观察者的次数必须与移除观察者的次数完全相同。一个新手常犯的错误是在viewWillAppear:中添加观察者  
UIViewController，但在viewDidUnload:中移除观察者，会导致推送次数不均，从而导致观察者泄漏以及通知选择器被多余地调用。

## 第74.3节：发布带有数据的通知

### Swift

```
let userInfo: [String: AnyObject] = ["someKey": myObject]
NSNotificationCenter.defaultCenter().postNotificationName("TestNotification", object: self,
                                                       userInfo: userInfo)
```

### Objective-C

```
NSDictionary *userInfo = [NSDictionary dictionaryWithObject:myObject forKey:@"someKey"];
[[NSNotificationCenter defaultCenter] postNotificationName: @"TestNotification" object:nil
                                               userInfo:userInfo];
```

## 第74.4节：为名称添加和移除观察者

```
// 添加观察者
let observer = NotificationCenter.defaultCenter().addObserverForName("nameOfTheNotification",
object: nil, queue: nil) { (notification) in
    // 在此代码块中对通知进行操作
}

// 移除观察者
NotificationCenter.defaultCenter().removeObserver(observer)
```

## 第74.5节：发送通知

### Swift

```
NotificationCenter.defaultCenter().postNotificationName("TestNotification", object: self)
```

### Objective-C

```
[[NSNotificationCenter defaultCenter] postNotificationName:@"TestNotification" object:nil];
```

- NSTextViewDidChangeSelectionNotification
- NSColorPanelColorDidChangeNotification

### Swift 2.3

```
NSNotificationCenter.defaultCenter().addObserver(self,
                                             selector: #selector(self.testNotification(_:)),
                                             name: "TestNotification",
                                             object: nil)
```

### Swift 3

```
NotificationCenter.default.addObserver(self,
                                      selector: #selector(self.testNotification(_:)),
                                      name: NSNotification.Name(rawValue: "TestNotification"),
                                      object: nil)
```

### Objective-C

```
[[NSNotificationCenter defaultCenter] addObserver:self
                                         selector:@selector(testNotification:)
                                         name:@"TestNotification"
                                         object:nil];
```

PS: It is also worth noting that the number of times an observer has been added has to be exactly the number of times the observer is removed. A rookie mistake is to add the observer in the viewWillAppear: of a UIViewController, but removing the observer in viewDidUnload:, will cause an uneven number of pushes and thus leaking the observer and the notification selector getting called in a superfluous manner.

## Section 74.3: Posting a Notification with Data

### Swift

```
let userInfo: [String: AnyObject] = ["someKey": myObject]
NSNotificationCenter.defaultCenter().postNotificationName("TestNotification", object: self,
                                                       userInfo: userInfo)
```

### Objective-C

```
NSDictionary *userInfo = [NSDictionary dictionaryWithObject:myObject forKey:@"someKey"];
[[NSNotificationCenter defaultCenter] postNotificationName: @"TestNotification" object:nil
                                               userInfo:userInfo];
```

## Section 74.4: Add and remove observer for name

```
// Add observer
let observer = NotificationCenter.defaultCenter().addObserverForName("nameOfTheNotification",
object: nil, queue: nil) { (notification) in
    // Do operations with the notification in this block
}

// Remove observer
NotificationCenter.defaultCenter().removeObserver(observer)
```

## Section 74.5: Posting a Notification

### Swift

```
NotificationCenter.defaultCenter().postNotificationName("TestNotification", object: self)
```

### Objective-C

```
[[NSNotificationCenter defaultCenter] postNotificationName:@"TestNotification" object:nil];
```

## 第74.6节：观察通知

### Swift

```
func testNotification(notification: NSNotification) {  
    let userInfo = notification.userInfo  
    let myObject: MyObject = userInfo["someKey"]  
}
```

### Objective-C

```
- (void)testNotification:(NSNotification *)notification {  
    NSDictionary *userInfo = notification.userInfo;  
    MyObject *myObject = [userInfo objectForKey:@"someKey"];  
}
```

## 第74.7节：使用Block添加/移除观察者

可以使用Block代替通过选择器添加观察者：

```
id testObserver = [[NSNotificationCenter defaultCenter] addObserverForName:@"TestNotification"  
                           object:nil  
                           queue:nil  
                           usingBlock:^(NSNotification*  
notification) {  
    NSDictionary *userInfo = notification.userInfo;  
    MyObject *myObject = [userInfo objectForKey:@"someKey"];  
});
```

然后可以使用以下方法移除观察者：

```
[[NSNotificationCenter defaultCenter] removeObserver:testObserver  
                                         name:@"TestNotification"  
                                         object:nil];
```

## Section 74.6: Observing a Notification

### Swift

```
func testNotification(notification: NSNotification) {  
    let userInfo = notification.userInfo  
    let myObject: MyObject = userInfo["someKey"]  
}
```

### Objective-C

```
- (void)testNotification:(NSNotification *)notification {  
    NSDictionary *userInfo = notification.userInfo;  
    MyObject *myObject = [userInfo objectForKey:@"someKey"];  
}
```

## Section 74.7: Adding/Removing an Observer with a Block

Instead of adding an observer with a selector, a block can be used:

```
id testObserver = [[NSNotificationCenter defaultCenter] addObserverForName:@"TestNotification"  
                           object:nil  
                           queue:nil  
                           usingBlock:^(NSNotification*  
notification) {  
    NSDictionary *userInfo = notification.userInfo;  
    MyObject *myObject = [userInfo objectForKey:@"someKey"];  
});
```

The observer can then be removed with:

```
[[NSNotificationCenter defaultCenter] removeObserver:testObserver  
                                         name:@"TestNotification"  
                                         object:nil];
```

# 第75章：NSURLSession

## 第75.1节：Objective-C 创建会话和数据任务

```
NSURL *url = [NSURL URLWithString:@"http://www.example.com/"];
NSURLSessionConfiguration *configuration = [NSURLSessionConfiguration defaultSessionConfiguration];

// 在此配置会话。

NSURLSession *session = [NSURLSession sessionWithConfiguration:configuration];

[[session dataTaskWithURL:url
completionHandler:^(NSData *data, NSURLResponse *response, NSError *error)
{
    // response对象包含元数据 (HTTP头, 状态码)

    // data对象包含响应体

    // error对象包含任何客户端错误 (例如连接
    // 失败), 在某些情况下, 也可能报告服务器端错误。
    // 但通常, 你应通过
    // 检查response对象中的HTTP状态码来检测服务器端错误。
} resume];
```

## 第75.2节：设置后台配置

创建后台会话

```
// Swift :
let mySessionID = "com.example.bgSession"
let bgSessionConfig =
NSURLSessionConfiguration.backgroundSessionConfigurationWithIdentifier(mySessionID)

let session = URLSession(configuration: bgSessionConfig)

// 在此添加任务

// Objective-C :
NSString *mySessionID = @"com.example.bgSession";
NSURLSessionConfiguration *configuration =
[NSURLSessionConfiguration backgroundSessionConfigurationWithIdentifier: mySessionID];
NSURLSession *session = [NSURLSession sessionWithConfiguration:configuration
delegate:self]
```

此外，在 iOS 中，您必须设置支持后台应用重新启动的处理。当您的应用程序的  
当调用 `application:handleEventsForBackgroundURLSession:completionHandler:` 方法 (Objective-C) 或 applicat  
ion(`_:handleEventsForBackgroundURLSession:completionHandler:`) 方法 (Swift) 时，表示您的应用已在后台重新启动以  
处理会话上的活动。

在该方法中，您应使用提供的标识符创建一个新的会话，并配置一个代理来处理事件，就像您通常在前台那样操作。  
此外，您应将提供的完成处理程序存储在一个字典中，使用会话作为键。

当代理的 `URLSessionDidFinishEventsForBackgroundURLSession:` (Obj-C) / `URLSessionDidFinishEventsForBackgroundURLSession:` (Swift)  
方法被调用，通知您没有更多事件需要处理时，您的应用应查找该会话的完成处理  
程序，并将该会话移除

# Chapter 75: NSURLSession

## Section 75.1: Objective-C Create a Session And Data Task

```
NSURL *url = [NSURL URLWithString:@"http://www.example.com/"];
NSURLSessionConfiguration *configuration = [NSURLSessionConfiguration defaultSessionConfiguration];

// Configure the session here.

NSURLSession *session = [NSURLSession sessionWithConfiguration:configuration];

[[session dataTaskWithURL:url
completionHandler:^(NSData *data, NSURLResponse *response, NSError *error)
{
    // The response object contains the metadata (HTTP headers, status code)

    // The data object contains the response body

    // The error object contains any client-side errors (e.g. connection
    // failures) and, in some cases, may report server-side errors.
    // In general, however, you should detect server-side errors by
    // checking the HTTP status code in the response object.
} resume];
```

## Section 75.2: Setting up background configuration

To create a background session

```
// Swift :
let mySessionID = "com.example.bgSession"
let bgSessionConfig =
NSURLSessionConfiguration.backgroundSessionConfigurationWithIdentifier(mySessionID)

let session = URLSession(configuration: bgSessionConfig)

// add tasks here

// Objective-C :
NSString *mySessionID = @"com.example.bgSession";
NSURLSessionConfiguration *configuration =
[NSURLSessionConfiguration backgroundSessionConfigurationWithIdentifier: mySessionID];
NSURLSession *session = [NSURLSession sessionWithConfiguration:configuration
delegate:self]
```

Additionally, in iOS, you must set up support for handling background app relaunch. When your app's  
application:handleEventsForBackgroundURLSession:completionHandler: method (Objective-C) or  
application(\_:handleEventsForBackgroundURLSession:completionHandler:) method (Swift) gets called, it  
means your app has been relaunched in the background to handle activity on a session.

In that method, you should create a new session with the provided identifier and configure it with a delegate to  
handle events just like you normally would in the foreground. Additionally, you should store the provided  
completion handler in a dictionary, using the session as the key.

When the delegate's `URLSessionDidFinishEventsForBackgroundURLSession:` (Obj-C) /  
`URLSessionDidFinishEventsForBackgroundURLSession:` (Swift) method gets called to tell you that there are no  
more events to handle, your app should look up the completion handler for that session, remove the session from

字典，并调用完成处理程序，从而告诉操作系统您不再有任何与会话相关的未完成处理。（如果您在收到该代理调用时仍在进行某些操作，请等待完成。）一旦您调用该方法，后台会话会立即获得已作废。

如果您的应用随后收到一个application:application:didFinishLaunchingWithOptions:调用（很可能表示用户在您处理后台事件时将应用切换到前台），则可以安全地使用相同的标识符创建一个后台会话，因为具有该标识符的旧会话已不存在。

如果你对细节感兴趣，简单来说，当你创建一个后台会话时，你实际上在做两件事：

- 在外部守护进程（nsurlsessiond）中创建会话以处理下载
- 在您的应用程序内创建一个会话，通过 NSXPC 与该外部守护进程通信

通常，在应用程序的单次启动中创建两个具有相同会话ID的会话是危险的，因为它们都试图与后台守护进程中的同一会话通信。这就是官方文档中建议绝不要创建多个具有相同标识符的会话的原因。然而，如果第一个会话是作为handleEventsForBackgroundURLSession调用的一部分创建的临时会话，那么现在已失效的应用内会话与后台守护进程中的会话之间的关联将不复存在。

## 第75.3节：简单的GET请求

```
// 定义 URL
let url = NSURL(string: "https://urlToGet.com")

// 创建一个任务以从 URL 获取数据
let task = NSURLSession.sharedSession().dataTaskWithURL(url!)
{
    /*在此代码块内，我们可以访问 NSData *data、NSURLResponse *response 和 NSError
    *error，这些由 dataTaskWithURL() 函数返回*/
    (data, response, error) in

    if error == nil
    {
        // 可以在这里处理请求返回的数据
    }
    else
    {
        // 发生了错误
    }
}

// 发起请求
task.resume()
```

## 第 75.4 节：使用 NSURLSession 在 Objective-C 中发送带参数的 POST 请求

POST 请求体有两种常见的编码方式：URL 编码（application/x-www-form-urlencoded）和表单数据（multipart/form-data）。大部分代码相似，但构造请求体数据的方式不同。

### 使用URL编码发送请求

无论你是为你的小型应用程序搭建服务器，还是在团队中与一名完整的后端工程师合作，

the dictionary, and call the completion handler, thus telling the operating system that you no longer have any outstanding processing related to the session. (If you are still doing something for some reason when you get that delegate call, wait until done.) As soon as you call that method, the background session immediately gets invalidated.

If your application then receives an application:application:didFinishLaunchingWithOptions: call (likely indicating that the user foregrounded your app while you were busy processing background events), it is safe to create a background session with that same identifier, because the old session with that identifier no longer exists.

If you're curious about the details, at a high level, when you create a background session, you're doing two things:

- Creating a session in an external daemon (nsurlsessiond) to handle the downloads
- Creating a session within your app that talks to that external daemon via NSXPC

Normally, it is dangerous to create two sessions with the same session ID in a single launch of the app, because they both are trying to talk to the same session in the background daemon. This is why the official documentation says to never create multiple sessions with the same identifier. However, if the first session was a temporary session created as part of a handleEventsForBackgroundURLSession call, the association between the now-invalidated in-app session and the session in the background daemon no longer exists.

## Section 75.3: Simple GET request

```
// define url
let url = NSURL(string: "https://urlToGet.com")

//create a task to get data from a url
let task = NSURLSession.sharedSession().dataTaskWithURL(url!)
{
    /*inside this block, we have access to NSData *data, NSURLResponse *response, and NSError
    *error returned by the dataTaskWithURL() function*/
    (data, response, error) in

    if error == nil
    {
        // Data from the request can be manipulated here
    }
    else
    {
        // An error occurred
    }
}

//make the request
task.resume()
```

## Section 75.4: Sending a POST Request with arguments using NSURLSession in Objective-C

There are two common ways to encode a POST request body: URL encoding (application/x-www-form-urlencoded) and form data (multipart/form-data). Much of the code is similar, but the way you construct the body data is different.

### Sending a request using URL encoding

Be it you have a server for your small application or your working in a team with a full out back-end engineer, you'll

你总有一天会想通过你的iOS应用程序与该服务器通信。

在下面的代码中，我们将构造一串参数字符串，目标服务器脚本将使用这些参数来执行根据你的情况而变化的操作。例如，我们可能想发送字符串：

```
name=Brendon&password=abcde
```

当用户注册你的应用程序时发送给服务器，以便服务器可以将这些信息存储到数据库中。

让我们开始吧。你需要使用以下代码创建一个NSURLSession的POST请求。

```
// 创建配置，这是必要的，以便我们可以取消缓存等操作。
NSURLSessionConfiguration * defaultConfigObject = [NSURLSessionConfiguration
defaultSessionConfiguration];
// 禁用缓存
defaultConfigObject.requestCachePolicy = NSURLRequestReloadIgnoringLocalCacheData;
NSURLSession * defaultSession = [NSURLSession sessionWithConfiguration:defaultConfigObject
delegate:self delegateQueue:[NSOperationQueue mainQueue]];

NSString * scriptURL = [NSString stringWithFormat:@"https://server.io/api/script.php"];
// 将URL字符串转换为NSURLSession可用的URL
NSMutableURLRequest * urlRequest = [NSMutableURLRequest requestWithURL:[NSURL
URLWithString:scriptURL]];
NSString * postDataString = [NSString stringWithFormat:@"name=%@&password=%@", [self nameString],
[self URLencode:passwordString]];
[urlRequest setHTTPMethod:@"POST"];
[urlRequest setHTTPBody:[postDataString dataUsingEncoding:NSUTF8StringEncoding]];

NSURLSessionDataTask * dataTask = [defaultSession dataTaskWithRequest:urlRequest];
// 启动数据任务。
[dataTask resume];
```

以上代码只是创建并发送了POST请求到服务器。请记住，脚本URL和POST数据字符串会根据你的具体情况而变化。如果你正在阅读这段内容，你应该知道如何填写这些变量。

你还需要添加一个用于URL编码的小方法：

```
- (NSString *)URLencode:(NSString *)originalString encoding:(NSStringEncoding)encoding
{
    return (__bridge_transfer NSString *)CFURLCreateStringByAddingPercentEscapes(
        kCFAllocatorDefault,
        (__bridge CFStringRef)originalString,
        NULL,
        CFSTR(":/?#[!]@!$&'()*+,;="),
        CFStringConvertNSStringEncodingToEncoding(encoding));
}
```

因此，当服务器处理完这些数据后，它会向你的iOS应用发送一个返回值。所以我们需要处理这个返回值，但该如何处理呢？

我们使用事件驱动编程，并使用NSURLSession的代理方法。这意味着当服务器发送回响应时，这些方法将开始触发。以下5个方法是在整个请求过程中，每次请求时都会被触发的：

```
- (void)URLSession:(NSURLSession *)session dataTask:(NSURLSessionDataTask *)dataTask
didReceiveResponse:(NSURLResponse *)response
completionHandler:(void (^)(NSURLSessionResponseDisposition disposition))completionHandler;
```

want to talk to that server at one point with your iOS application.

In the following code we will be composing a string of arguments that the destination server script will use to do something that changes depending on your case. For example we may want to send the string:

```
name=Brendon&password=abcde
```

To the server when a user signs up to your application, so the server can store this information in a database.

Let's get started. You'll want to create a URLSession POST request with the following code.

```
// Create the configuration, which is necessary so we can cancel cacheing amongst other things.
NSURLSessionConfiguration * defaultConfigObject = [NSURLSessionConfiguration
defaultSessionConfiguration];
// Disables caching
defaultConfigObject.requestCachePolicy = NSURLRequestReloadIgnoringLocalCacheData;
NSURLSession * defaultSession = [NSURLSession sessionWithConfiguration:defaultConfigObject
delegate:self delegateQueue:[NSOperationQueue mainQueue]];

NSString * scriptURL = [NSString stringWithFormat:@"https://server.io/api/script.php"];
// Converts the URL string to a URL usable by URLSession
NSMutableURLRequest * urlRequest = [NSMutableURLRequest requestWithURL:[NSURL
URLWithString:scriptURL]];
NSString * postDataString = [NSString stringWithFormat:@"name=%@&password=%@", [self nameString],
[self URLencode:passwordString]];
[urlRequest setHTTPMethod:@"POST"];
[urlRequest setHTTPBody:[postDataString dataUsingEncoding:NSUTF8StringEncoding]];

NSURLSessionDataTask * dataTask = [defaultSession dataTaskWithRequest:urlRequest];
// Fire the data task.
[dataTask resume];
```

The above code just created and fired the POST request to the server. Remember that the script URL and the POST data string changes depending on your situation. If you're reading this, you'll know what to fill those variables with.

You'll also need to add a small method that does the URL encoding:

```
- (NSString *)URLencode:(NSString *)originalString encoding:(NSStringEncoding)encoding
{
    return (__bridge_transfer NSString *)CFURLCreateStringByAddingPercentEscapes(
        kCFAllocatorDefault,
        (__bridge CFStringRef)originalString,
        NULL,
        CFSTR(":/?#[!]@!$&'()*+,;="),
        CFStringConvertNSStringEncodingToEncoding(encoding));
}
```

So, when the server is finished processing this data it will send a return to your iOS app. So we need to process this return, but how?

We use event-driven programming and use URLSession's delegate methods. This means as the server sends back a response these methods will start triggering. The following 5 methods are the ones that'll be triggered throughout the ENTIRE request, each time one is made:

```
- (void)URLSession:(NSURLSession *)session dataTask:(NSURLSessionDataTask *)dataTask
didReceiveResponse:(NSURLResponse *)response
completionHandler:(void (^)(NSURLSessionResponseDisposition disposition))completionHandler;
```

```

- (void)URLSession:(NSURLSession *)session dataTask:(NSURLSessionDataTask *)dataTask
didReceiveData:(NSData *)data;

- (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task
didCompleteWithError:(NSError *)error;

- (void)URLSession:(NSURLSession *)session didReceiveChallenge:(NSURLAuthenticationChallenge *)
challenge completionHandler:(void (^)(NSURLSessionAuthChallengeDisposition, NSURLCredential *))completionHandler;

- (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task
didReceiveChallenge:(NSURLAuthenticationChallenge *)challenge completionHandler:(void
(^)(NSURLSessionAuthChallengeDisposition, NSURLCredential * _Nullable))completionHandler;

```

下面你将看到上述方法在上下文中的使用。由于苹果的说明，这些方法的用途相当直观，但我还是对它们的用途做了注释：

```

// 响应处理代理
- (void)URLSession:(NSURLSession *)session dataTask:(NSURLSessionDataTask *)dataTask
didReceiveResponse:(NSURLResponse *)response
completionHandler:(void (^)(NSURLSessionResponseDisposition disposition))completionHandler{
    // 该处理器允许我们接收并解析来自服务器的响应
    completionHandlerNSURLSessionResponseAllow;
}

- (void)URLSession:(NSURLSession *)session dataTask:(NSURLSessionDataTask *)dataTask
didReceiveData:(NSData *)data{

    // 将接收到的 JSON 解析为 NSDictionary
    NSError * err = nil;
    NSDictionary * jsonDict = [NSJSONSerialization JSONObjectWithData:data
options:NSJSONReadingAllowFragments error:&err];

    if (!err){ // 如果没有发生错误，正常解析对象数组
        // 在这里解析 JSON 字典 'jsonDict'
    }否则{ // 发生错误，需要通知用户
        // 在此处理错误
    }
}

// 错误处理代理
- (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task
didCompleteWithError:(NSError *)error{
    if(error == nil){
        // 从API下载成功
        NSLog(@"数据网络请求已成功完成。");
    }否则{
        // 描述并记录阻止我们接收响应的错误
        NSLog(@"错误: %@", [error userInfo]);
    }

    // 处理网络错误，告知用户发生了什么。
}

// 当会话收到挑战（因为iOS 9应用传输安全阻止无效的SSL证书）时，我们使用以下方法告诉NSURLSession“放轻松，我可以信任我自己”。
// 除非您的服务器使用HTTP而非HTTPS，否则以下内容不是必需的

- (void)URLSession:(NSURLSession *)session didReceiveChallenge:(NSURLAuthenticationChallenge *)
challenge completionHandler:(void (^)(NSURLSessionAuthChallengeDisposition, NSURLCredential

```

```

- (void)URLSession:(NSURLSession *)session dataTask:(NSURLSessionDataTask *)dataTask
didReceiveData:(NSData *)data;

- (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task
didCompleteWithError:(NSError *)error;

- (void)URLSession:(NSURLSession *)session didReceiveChallenge:(NSURLAuthenticationChallenge *)
challenge completionHandler:(void (^)(NSURLSessionAuthChallengeDisposition, NSURLCredential *))completionHandler;

- (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task
didReceiveChallenge:(NSURLAuthenticationChallenge *)challenge completionHandler:(void
(^)(NSURLSessionAuthChallengeDisposition, NSURLCredential * _Nullable))completionHandler;

```

Below you'll see the above methods used in context. Each of their purposes are pretty self-explanatory thanks to Apple, but I've commented their uses anyway:

```

// Response handling delegates
- (void)URLSession:(NSURLSession *)session dataTask:(NSURLSessionDataTask *)dataTask
didReceiveResponse:(NSURLResponse *)response
completionHandler:(void (^)(NSURLSessionResponseDisposition disposition))completionHandler{
    // Handler allows us to receive and parse responses from the server
    completionHandlerNSURLSessionResponseAllow;
}

- (void)URLSession:(NSURLSession *)session dataTask:(NSURLSessionDataTask *)dataTask
didReceiveData:(NSData *)data{

    // Parse the JSON that came in into an NSDictionary
    NSError * err = nil;
    NSDictionary * jsonDict = [NSJSONSerialization JSONObjectWithData:data
options:NSJSONReadingAllowFragments error:&err];

    if (!err){ // if no error occurred, parse the array of objects as normal
        // Parse the JSON dictionary 'jsonDict' here
    }else{ // an error occurred so we need to let the user know
        // Handle your error here
    }
}

// Error handling delegate
- (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task
didCompleteWithError:(NSError *)error{
    if(error == nil){
        // Download from API was successful
        NSLog(@"Data Network Request Did Complete Successfully.");
    }else{
        // Describes and logs the error preventing us from receiving a response
        NSLog(@"Error: %@", [error userInfo]);
    }

    // Handle network error, letting the user know what happened.
}

// When the session receives a challenge (because of iOS 9 App Transport Security blocking non-
valid SSL certificates) we use the following methods to tell NSURLSession "Chill out, I can trust
me".
// The following is not necessary unless your server is using HTTP, not HTTPS

- (void)URLSession:(NSURLSession *)session didReceiveChallenge:(NSURLAuthenticationChallenge *)
challenge completionHandler:(void (^)(NSURLSessionAuthChallengeDisposition, NSURLCredential

```

```

*)completionHandler{
    if([challenge.protectionSpace.authenticationMethod
isEqualToString:NSURLAuthenticationMethodServerTrust]){
        if([challenge.protectionSpace.host isEqualToString:@"DomainNameOfServer.io"]){
            NSURLCredential * credential = [NSURLCredential
credentialForTrust:challenge.protectionSpace.serverTrust];
            completionHandler(NSURLConnectionAuthChallengeUseCredential,credential);
        }
    }
}

- (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task
didReceiveChallenge:(NSURLAuthenticationChallenge *)challenge completionHandler:(void
(^)(NSURLSessionAuthChallengeDisposition, NSURLCredential * _Nullable))completionHandler{
    if([challenge.protectionSpace.authenticationMethod
isEqualToString:NSURLAuthenticationMethodServerTrust]){
        if([challenge.protectionSpace.host isEqualToString:@"DomainNameOfServer.io"]){
            NSURLCredential * credential = [NSURLCredential
credentialForTrust:challenge.protectionSpace.serverTrust];
            completionHandler(NSURLConnectionAuthChallengeUseCredential,credential);
        }
    }
}

```

就是这样！这就是在 iOS 9 中发送、接收和解析 API 请求所需的全部代码！好吧.....代码确实有点多。但如果像上面那样正确实现，它将是万无一失的！务必按照上面建议的地方处理错误。

## 使用表单编码发送请求

URL 编码是一种广泛兼容的编码任意数据的方式。然而，对于上传二进制数据（如照片）来说，它效率相对较低，因为每个非 ASCII 字节都会变成三个字符的代码。它也不支持文件附件，因此你必须将文件名和文件数据作为单独的字段传递。

假设我们想以一种高效且在服务器端看起来像文件的方式上传一张照片。

实现这一点的一种方法是改用表单编码。为此，请按以下方式编辑创建 NSURLSession 的代码：

```

UIImage * imgToSend;

// UIImageJPEGRepresentation 的第二个参数表示压缩质量。0 表示最高压缩，1 表示最低压缩
// 使用 0.4 可能避免达到服务器上传限制，并减少服务器空间占用
NSData * imageData = UIImageJPEGRepresentation(imgToSend, 0.4f);

// 或者，如果照片保存在磁盘上，可以通过
// [NSData dataWithContentsOfURL:] 来获取

// 设置 POST 请求的主体内容

// 该边界作为一个表单字段与下一个字段之间的分隔符。
// 它不能出现在你打算上传的实际数据中。

NSString * boundary = @"-----14737809831466499882746641449";

// POST 方法的主体内容
NSMutableData * body = [NSMutableData data];

// 主体必须以两个连字符开头的边界开始，后跟回车换行符。

```

```

*)completionHandler{
    if([challenge.protectionSpace.authenticationMethod
isEqualToString:NSURLAuthenticationMethodServerTrust]){
        if([challenge.protectionSpace.host isEqualToString:@"DomainNameOfServer.io"]){
            NSURLCredential * credential = [NSURLCredential
credentialForTrust:challenge.protectionSpace.serverTrust];
            completionHandler(NSURLConnectionAuthChallengeUseCredential,credential);
        }
    }
}

- (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task
didReceiveChallenge:(NSURLAuthenticationChallenge *)challenge completionHandler:(void
(^)(NSURLSessionAuthChallengeDisposition, NSURLCredential * _Nullable))completionHandler{
    if([challenge.protectionSpace.authenticationMethod
isEqualToString:NSURLAuthenticationMethodServerTrust]){
        if([challenge.protectionSpace.host isEqualToString:@"DomainNameOfServer.io"]){
            NSURLCredential * credential = [NSURLCredential
credentialForTrust:challenge.protectionSpace.serverTrust];
            completionHandler(NSURLConnectionAuthChallengeUseCredential,credential);
        }
    }
}

```

So that's it! That's all the code you need to send, receive and parse a request for an API in iOS 9! Okay...it was kind of a lot of code. But if implemented right like above, it'll be fail-safe! Make sure to always handle errors where suggested above.

## Sending a request using form encoding

URL encoding is a broadly compatible way to encode arbitrary data. However, it is relatively inefficient for uploading binary data (such as photos) because every non-ASCII byte turns into a three-character code. It also does not support file attachments, so you would have to pass filenames and file data as separate fields.

Suppose we want to upload a photograph in a way that is efficient and actually looks like a file on the server side. One way to do that is to use form encoding instead. To do this, edit the code that creates the NSURLSession as follows:

```

UIImage * imgToSend;

// 2nd parameter of UIImageJPEGRepresentation represents compression quality. 0 being most
// compressed, 1 being the least
// Using 0.4 likely stops us hitting the servers upload limit and costs us less server space
NSData * imageData = UIImageJPEGRepresentation(imgToSend, 0.4f);

// Alternatively, if the photo is on disk, you can retrieve it with
// [NSData dataWithContentsOfURL:]

// Set up the body of the POST request.

// This boundary serves as a separator between one form field and the next.
// It must not appear anywhere within the actual data that you intend to
// upload.
NSString * boundary = @"-----14737809831466499882746641449";

// Body of the POST method
NSMutableData * body = [NSMutableData data];

// The body must start with the boundary preceded by two hyphens, followed
// by a carriage return and newline pair.

```

```

// 注意, 当我们实际将边界用作主体数据的一部分时, 会在边界前加上两个额外的连字符。
//
[body appendData:[[NSString stringWithFormat:@"\r--%@\r", boundary]dataUsingEncoding:NSUTF8StringEncoding]];

// 接下来是一系列针对第一个字段的头部, 然后是两个回车换行对 (CR-LF) 。
//
[body appendData:[[NSString stringWithFormat:@"Content-Disposition: form-data;name=\"%tag_name\"\r\r"] dataUsingEncoding:NSUTF8StringEncoding]];

// 接下来是该字段 (名为"tag_name") 的实际数据, 后面跟着一个回车换行对、一个边界和另一个回车换行对。
[body appendData:[strippedCompanyName dataUsingEncoding:NSUTF8StringEncoding]];[body appendData:[[NSString stringWithFormat:@"\r-%@\r", boundary] dataUsingEncoding:NSUTF8StringEncoding]];

// 将文件名和图像数据编码为"userfile"CGI参数。
// 这与前面的字段类似, 只是它作为实际的文件附件发送, 而不是一段数据块, 这意味着它既有文件名也有实际的文件内容。
//
// 重要: 文件名必须是纯ASCII码 (如果像这样编码, 文件名中不能包含引号) 。
//
NSString * picFileName = [NSString stringWithFormat:@"photoName"];NSString * appendString = [NSString stringWithFormat:@"Content-Disposition: form-data;name=\"userfile\"; filename=\"%@.jpg\"\r", picFileName];
[body appendData:[appendString dataUsingEncoding:NSUTF8StringEncoding]];[body appendData:[@"Content-Type: application/octet-stream\r\r" dataUsingEncoding:NSUTF8StringEncoding]];
[body appendData:[NSData dataWithData:imageData]];

// 用两个额外的连字符开头和两个额外的连字符结尾关闭请求体的最后一个
// 边界。
[body appendData:[[NSString stringWithFormat:@"\r--%--\r", boundary]dataUsingEncoding:NSUTF8StringEncoding]];

// 创建会话
// 我们可以使用代理来跟踪上传进度并禁用缓存
NSURLSessionConfiguration * defaultConfigObject = [NSURLSessionConfiguration defaultSessionConfiguration];
defaultConfigObject.requestCachePolicy = NSURLRequestReloadIgnoringLocalCacheData;
NSURLSession * defaultSession = [NSURLSession sessionWithConfiguration: defaultConfigObject delegate: self delegateQueue: [NSOperationQueue mainQueue]];

// 数据上传任务。
NSURL * url = [NSURL URLWithString:@"https://server.io/api/script.php"];
NSMutableURLRequest * request = [NSMutableURLRequest requestWithURL:url];
NSString * contentType = [NSString stringWithFormat:@"multipart/form-data; boundary=%@", boundary];
[request addValue:contentType forHTTPHeaderField:@"Content-Type"];
request.HTTPMethod = @"POST";
request.HTTPBody = body;
NSURLSessionDataTask * uploadTask = [defaultSession dataTaskWithRequest:request];
[uploadTask resume];

```

这将像之前一样创建并启动NSURLSession请求, 因此代理方法的行为将完全相同。请确保发送图像的脚本 (位于变量url中的URL) 正在接收图像并且能够正确解析它。

```

// Notice that we prepend two additional hyphens to the boundary when
// we actually use it as part of the body data.
//
[body appendData:[[NSString stringWithFormat:@"\r\n--%@\r\n", boundary] dataUsingEncoding:NSUTF8StringEncoding]];

// This is followed by a series of headers for the first field and then
// TWO CR-LF pairs.
[body appendData:[[NSString stringWithFormat:@"Content-Disposition: form-data; name=\"%tag_name\"\r\n\r\n"] dataUsingEncoding:NSUTF8StringEncoding]];

// Next is the actual data for that field (called "tag_name") followed by
// a CR-LF pair, a boundary, and another CR-LF pair.
[body appendData:[strippedCompanyName dataUsingEncoding:NSUTF8StringEncoding]];
[body appendData:[[NSString stringWithFormat:@"\r\n--%@\r\n", boundary] dataUsingEncoding:NSUTF8StringEncoding]];

// Encode the filename and image data as the "userfile" CGI parameter.
// This is similar to the previous field, except that it is being sent
// as an actual file attachment rather than a blob of data, which means
// it has both a filename and the actual file contents.
//
// IMPORTANT: The filename MUST be plain ASCII (and if encoded like this,
// must not include quotation marks in the filename).
//
NSString * picFileName = [NSString stringWithFormat:@"photoName"];
NSString * appendString = [NSString stringWithFormat:@"Content-Disposition: form-data; name=\"userfile\"; filename=\"%@.jpg\"\r\n", picFileName];
[body appendData:[appendString dataUsingEncoding:NSUTF8StringEncoding]];
[body appendData:[@"Content-Type: application/octet-stream\r\n\r\n" dataUsingEncoding:NSUTF8StringEncoding]];
[body appendData:[NSData dataWithData:imageData]];

// Close the request body with one last boundary with two
// additional hyphens prepended **and** two additional hyphens appended.
[body appendData:[[NSString stringWithFormat:@"\r\n--%--\r\n", boundary] dataUsingEncoding:NSUTF8StringEncoding]];

// Create the session
// We can use the delegate to track upload progress and disable caching
NSURLSessionConfiguration * defaultConfigObject = [NSURLSessionConfiguration defaultSessionConfiguration];
defaultConfigObject.requestCachePolicy = NSURLRequestReloadIgnoringLocalCacheData;
NSURLSession * defaultSession = [NSURLSession sessionWithConfiguration: defaultConfigObject delegate: self delegateQueue: [NSOperationQueue mainQueue]];

// Data uploading task.
NSURL * url = [NSURL URLWithString:@"https://server.io/api/script.php"];
NSMutableURLRequest * request = [NSMutableURLRequest requestWithURL:url];
NSString * contentType = [NSString stringWithFormat:@"multipart/form-data; boundary=%@", boundary];
[request addValue:contentType forHTTPHeaderField:@"Content-Type"];
request.HTTPMethod = @"POST";
request.HTTPBody = body;
NSURLSessionDataTask * uploadTask = [defaultSession dataTaskWithRequest:request];
[uploadTask resume];

```

This creates and fires the NSURLSession request just as before, and as a result the delegate methods will behave exactly the same way. Make sure that the script the image is being sent to (located at the url in the variable url) is expecting an image and can parse it correctly.