iOS® 开发者

专业人士笔记

800+ 页

专业提示和技巧

GoalKicker.com
免费编程书籍

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

iOS® Developer

Notes for Professionals

800+ pages

of professional hints and tricks

GoalKicker.com
Free Programming Books

Disclaimer
This is an unofficial free book created for educational purposes and is not affiliated with official iOS® Developer group(s) or company(s).
All trademarks and registered trademarks are the property of their respective owners

目录

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

Contents

About 1
Chapter 1: Getting started with iOS Development 2
Section 1.1: Creating a default Single View Application 2
Section 1.2: Hello World 6
Section 1.3: Xcode Interface 11
Section 1.4: Create your first program in Swift 3 17
Chapter 2: UILabel 22
Section 2.1: Create a UILabel 22
Section 2.2: Number of Lines 24
Section 2.3: Set Font 25
Section 2.4: Text Color 26
Section 2.5: Background Color 27
Section 2.6: Size to fit 27
Section 2.7: Text alignment 30
Section 2.8: Calculate Content Bounds (for i.e. dynamic cell heights) 30
Section 2.9: Label Attributed Text 32
Section 2.10: Clickable Label 38
Section 2.11: Variable height using constraints 39
Section 2.12: LineBreakMode 39
Section 2.13: Add shadows to text 41
Section 2.14: Changing Text in an Existing Label 41
Section 2.15: Auto-size label to fit text 42
Section 2.16: Get UILabel's size strictly based on its text and font 43
Section 2.17: Highlighted and Highlighted Text Color 44
Section 2.18: Justify Text 44
Section 2.19: Dynamic label frame from unknown text length 45
Chapter 3: UILabel text underlining 47
Section 3.1: Underlining a text in a UILabel using Objective C 47
Section 3.2: Underlining a text in UILabel using Swift 47
Chapter 4: attributedText in UILabel 48
Section 4.1: HTML text in UILabel 48
Section 4.2: Set different property to text in single UILabel 48
Chapter 5: UIButton 50
Section 5.1: Creating a UIButton 50
Section 5.2: Attaching a Method to a Button 50
Section 5.3: Setting Font 51
Section 5.4: Set Image 51
Section 5.5: Get UIButton's size strictly based on its text and font 51
Section 5.6: Disabling a UIButton 52
Section 5.7: Set title 52
Section 5.8: Set title color 52
Section 5.9: Horizontally aligning contents 53
Section 5.10: Getting the title label 53
Section 5.11: Adding an action to an UIButton via Code (programmatically) 54
Chapter 6: UIDatePicker 55
Section 6.1: Create a Date Picker 55
Section 6.2: Setting Minimum-Maximum Date 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
第7章：UILocalNotification 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
第8章：UIImage 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
第9章：将NSAttributedString转换为UIImage 70
第9.1节：NSAttributedString到UIImage的转换 70
第10章：UIImagePickerController 71
第10.1节：UIImagePickerController的通用用法 71
第11章：UIImageView 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
第12章：调整UIImage大小 83
第12.1节：按尺寸和质量调整任意图像大小 83
第13章：将UIImage裁剪成圆形 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
第14章：UITableView 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
第15章：UITableViewController 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
Chapter 7: UILocalNotification 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
Chapter 8: UIImage 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
Chapter 9: Convert NSAttributedString to UIImage 70
Section 9.1: NSAttributedString to UIImage Conversion 70
Chapter 10: UIImagePickerController 71
Section 10.1: Generic usage of UIImagePickerController 71
Chapter 11: UIImageView 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
Chapter 12: Resizing UIImage 83
Section 12.1: Resize any image by size & quality 83
Chapter 13: Cut a UIImage into a circle 84
Section 13.1: Cut a image into a circle - Objective C 84
Section 13.2: SWIFT 3 Example 85
Chapter 14: UITableView 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
Chapter 15: UITableViewController 108
Section 15.1: TableView with dynamic properties with tableviewCellStyle basic 108

第15.2节：自定义单元格的表视图 109
第16章：UIRefreshControl 表视图 111
第16.1节：在表视图上设置refreshControl： 111
第16.2节：Objective-C 示例 111
第17章：UITableViewCell 113
第17.1节：UITableViewCell 的Xib文件 113
第18章：UITableViewCell 选择的自定义方法 114
第18.1节：行的单选与双选的区别 114
第19章：UITableViewCell的自定义选择方法 115
第19.1节：行的单选与双选的区别 115
第20章：UIView 116
第20.1节：使视图圆角化 116
第20.2节：使用IBInspectable和IBDesignable 118
第20.3节：拍摄快照 121
第20.4节：创建UIView 121
第20.5节：使视图抖动 121
第20.6节：利用内在内容尺寸 122
第20.7节：通过编程管理UIView插入和从另一个UIView中删除 124
第20.8节：使用自动布局创建UIView 125
第20.9节：UIView的动画 127
第20.10节：UIView的尺寸和框架属性扩展 127
第21章：UIView的快照 129
第21.1节：获取快照 129
第21.2节：包含其他标记和文本的子视图快照 129
第22章：UIAlertController 131
第22.1节：使用UIAlertController的警告视图 131
第22.2节：使用UIAlertController的操作表 132
第22.3节：在UIAlertController中添加文本字段，类似提示框 135
第22.4节：高亮显示操作按钮 135
第22.5节：显示和处理警告 136
第23章：UIColor 141
第23.1节：创建UIColor 141
第23.2节：从十六进制数字或字符串创建UIColor 142
第23.3节：带Alpha组件的颜色 144
第23.4节：未公开的方法 145
第23.5节：从图像模式创建UIColor 146
第23.6节：给定UIColor的较浅和较深色调 147
第23.7节：使用户定义属性应用CGColor数据类型 148
第24章：UITextView 149
第24.1节：设置富文本 149
第24.2节：更改字体 149
第24.3节：自动检测链接、地址、日期等 149
第24.4节：更改文本 150
第24.5节：更改文本对齐方式 150
第24.6节：UITextViewDelegate 方法 150
第24.7节：更改文本颜色 151
第24.8节：移除多余的填充以适应精确测量的文本 151
第24.9节：获取和设置光标位置 151

Section 15.2: TableView with Custom Cell 109
Chapter 16: UIRefreshControl TableView 111
Section 16.1: Set up refreshControl on tableView: 111
Section 16.2: Objective-C Example 111
Chapter 17: UITableViewCell 113
Section 17.1: Xib file of UITableViewCell 113
Chapter 18: Custom methods of selection of UITableViewCells 114
Section 18.1: Distinction between single and double selection on row 114
Chapter 19: Custom methods of selection of UITableViewCells 115
Section 19.1: Distinction between single and double selection on row 115
Chapter 20: UIView 116
Section 20.1: Make the view rounded 116
Section 20.2: Using IBInspectable and IBDesignable 118
Section 20.3: Taking a snapshot 121
Section 20.4: Create a UIView 121
Section 20.5: Shake a View 121
Section 20.6: Utilizing Intrinsic Content Size 122
Section 20.7: Programmatically manage UIView insertion and deletion into and from another UIView 124
Section 20.8: Create UIView using Autolayout 125
Section 20.9: Animating a UIView 127
Section 20.10: UIView extension for size and frame attributes 127
Chapter 21: Snapshot of UIView 129
Section 21.1: Getting the Snapshot 129
Section 21.2: Snapshot with subview with other markup and text 129
Chapter 22: UIAlertController 131
Section 22.1: AlertViews with UIAlertController 131
Section 22.2: Action Sheets with UIAlertController 132
Section 22.3: Adding Text Field in UIAlertController like a prompt Box 135
Section 22.4: Highlighting an action button 135
Section 22.5: Displaying and handling alerts 136
Chapter 23: UIColor 141
Section 23.1: Creating a UIColor 141
Section 23.2: Creating a UIColor from hexadecimal number or string 142
Section 23.3: Color with Alpha component 144
Section 23.4: Undocumented Methods 145
Section 23.5: UIColor from an image pattern 146
Section 23.6: Lighter and Darker Shade of a given UIColor 147
Section 23.7: Make user defined attributes apply the CGColor datatype 148
Chapter 24: UITextView 149
Section 24.1: Set attributed text 149
Section 24.2: Change font 149
Section 24.3: Auto Detect Links, Addresses, Dates, and more 149
Section 24.4: Change text 150
Section 24.5: Change text alignment 150
Section 24.6: UITextViewDelegate methods 150
Section 24.7: Change text color 151
Section 24.8: Remove extra paddings to fit to a precisely measured text 151
Section 24.9: Getting and Setting the Cursor Postition 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
第25章：UITextField代理 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
第26章：UINavigationController 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
第27章：UIGestureRecognizer 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
第28章：UIBarButtonItem 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
第29章：UIScrollView 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
第30章：UIStackView 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
第31章：动态更新UIStackView 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
第32章：带有StackView子视图的UIScrollView 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
第33章：UIScrollView自动布局 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
第34章：UITextField 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
Chapter 25: UITextField Delegate 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
Chapter 26: UINavigationController 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 NavigationController 157
Chapter 27: UIGestureRecognizer 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
Chapter 28: UIBarButtonItem 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
Chapter 29: UIScrollView 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
Chapter 30: UIStackView 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
Chapter 31: Dynamically updating a UIStackView 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
Chapter 32: UIScrollView with StackView child 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
Chapter 33: UIScrollView AutoLayout 189
Section 33.1: ScrollableController 189
Section 33.2: UIScrollView dynamic content size through Storyboard 193
Chapter 34: UITextField 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
第34.4节：输入辅助视图（工具栏） 196
第34.5节：UITextView成为第一响应者时滚动视图移动 197
第34.6节：键盘类型 199
第34.7节：更改占位符颜色和字体 199
第34.8节：用UIPickerView替换键盘 200
第34.9节：创建UITextField 203
第34.10节：获取和设置光标位置 204
第34.11节：收起键盘 205
第34.12节：初始化文本字段 208
第34.13节：自动大写 208
第34.14节：设置对齐方式 209
第35章：自定义UITextField 210
第35.1节：用于过滤输入文本的自定义UITextField 210
第35.2节：禁止所有操作（如复制、粘贴等）的自定义UITextField 210
第36章：UIViewController 212
第36.1节：子类化 212
第36.2节：访问容器视图控制器 214
第36.3节：以编程方式设置视图 214
第36.4节：从Storyboard实例化 214
第36.5节：创建一个实例 215
第36.6节：添加/移除子视图控制器 216
第37章：UISwitch 217
第37.1节：设置开启/关闭状态的图片 217
第37.2节：设置开启/关闭 217
第37.3节：设置背景颜色 217
第37.4节：设置色调颜色 218
第38章：UICollectionView 219
第38.1节：创建UICollectionView 219
第38.2节：UICollectionView - 数据源 219
第38.3节：Collection View的基本Swift示例 220
第38.4节：使用数据源和流布局管理多个Collection View 225
第38.5节：UICollectionViewDelegate设置及项目选择 227
第38.6节：以编程方式创建集合视图 228
第38.7节：Swift - UICollectionViewDelegateFlowLayout 229
第38.8节：执行批量更新 229
第39章：UISearchController 231
第39.1节：导航栏标题中的搜索栏 231
第39.2节：表视图头部的搜索栏 234
第39.3节：实现 237
第39.4节：Objective-C中的UISerachController 237
第40章：UITabBarController 239
第40.1节：创建实例 239
第40.2节：带有标签栏的导航控制器 239
第40.3节：标签栏颜色自定义 240
第40.4节：更改标签栏项目标题和图标 240
第40.5节：通过代码创建标签栏控制器，无需Storyboard 241
第40.6节：带自定义颜色选择的UITabBarController 242
第41章：UIWebView 245

Section 34.3: Hide blinking caret 196
Section 34.4: Input accessory view (toolbar) 196
Section 34.5: Moving scroll when UITextView becomes first responder 197
Section 34.6: KeyboardType 199
Section 34.7: Change placeholder color and font 199
Section 34.8: Replace keyboard with UIPickerView 200
Section 34.9: Create a UITextField 203
Section 34.10: Getting and Setting the Cursor Position 204
Section 34.11: Dismiss Keyboard 205
Section 34.12: Initialize text field 208
Section 34.13: Autocapitalization 208
Section 34.14: Set Alignment 209
Chapter 35: Custom UITextField 210
Section 35.1: Custom UITextField for Filtering Input Text 210
Section 35.2: Custom UITextField to Disallow All Actions like Copy, Paste, etc 210
Chapter 36: UIViewController 212
Section 36.1: Subclassing 212
Section 36.2: Access the container view controller 214
Section 36.3: Set the view programmatically 214
Section 36.4: Instantiate from a Storyboard 214
Section 36.5: Create an instance 215
Section 36.6: Adding/removing a child view controller 216
Chapter 37: UISwitch 217
Section 37.1: Set Image for On/Off state 217
Section 37.2: Set On / Off 217
Section 37.3: Set Background Color 217
Section 37.4: Set Tint Color 218
Chapter 38: UICollectionView 219
Section 38.1: Create a UICollectionView 219
Section 38.2: UICollectionView - Datasource 219
Section 38.3: Basic Swift example of a Collection View 220
Section 38.4: Manage Multiple Collection view with DataSource and Flowlayout 225
Section 38.5: UICollectionViewDelegate setup and item selection 227
Section 38.6: Create a Collection View Programmatically 228
Section 38.7: Swift - UICollectionViewDelegateFlowLayout 229
Section 38.8: Performing batch updates 229
Chapter 39: UISearchController 231
Section 39.1: Search Bar in Navigation Bar Title 231
Section 39.2: Search Bar in Table View Header 234
Section 39.3: Implementation 237
Section 39.4: UISerachController in Objective-C 237
Chapter 40: UITabBarController 239
Section 40.1: Create an instance 239
Section 40.2: Navigation Controller with TabBar 239
Section 40.3: Tab Bar color customizing 240
Section 40.4: Changing Tab Bar Item Title and Icon 240
Section 40.5: Create Tab Bar controller programmatically without Storyboard 241
Section 40.6: UITabBarController with custom color selection 242
Chapter 41: UIWebView 245

第41.1节：创建UIWebView实例 . 245
第41.2节：确定内容大小 . 245
第41.3节：加载HTML字符串 . 245
第41.4节：发起URL请求 . 246
第41.5节：加载JavaScript . 246
第41.6节：停止加载网页内容 . 247
第41.7节：重新加载当前网页内容 . 247
第41.8节：加载.pdf、.txt、.doc等文档文件 . 247
第41.9节：在webView中加载本地HTML文件 . 247
第41.10节：使UIWebView内的链接可点击 . 248
第42章：UIActivityViewController . 250
第42.1节：初始化活动视图控制器 . 250
第43章：UIControl - 使用块进行事件处理 . 251
第43.1节：介绍 . 251
第44章：UISplitViewController . 254
第44.1节：使用Objective C中的代理实现主视图和详细视图交互 . 254
第45章：UISlider . 264
第45.1节：UISlider . 264
第45.2节：SWIFT示例 . 264
第45.3节：添加自定义滑块图像 . 264
第46章：UIStoryboard . 266
第46.1节：以编程方式获取UIStoryboard实例 . 266
第46.2节：打开另一个故事板 . 266
第47章：UIPageViewController . 267
第47.1节：通过代码创建水平分页的UIPageViewController . 267
第47.2节：创建水平分页视图控制器（无限页）的简单方法 . 268
第48章：UISplitViewController . 272
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 . 272
第49章：UIFont . 276
第49.1节：声明和初始化UIFont . 276
第49.2节：更改标签的字体 . 276
第50章：UIDevice . 277
第50.1节：获取iOS设备型号名称 . 277
第50.2节：识别设备和操作系统 . 278
第50.3节：获取设备方向 . 279
第50.4节：获取设备电池状态 . 280
第50.5节：使用接近传感器 . 281
第50.6节：获取电池状态和电池电量 . 281
第51章：使UIView的选定角变圆角 . 282
第51.1节：使用Objective C代码使UIView的选定角变圆角 . 282
第52章：来自XIB文件的自定义UIView . 283
第52.1节：布线元件 . 283
第52.2节：如何使用XIB制作自定义可重用UIView . 297
第53章：UIBezierPath . 298
第53.1节：设计和绘制贝塞尔曲线路径 . 298
第53.2节：如何为UIBezierPath绘制的矩形应用圆角半径 . 303
第53.3节：如何将阴影应用于UIBezierPath . 305
第53.4节：如何使用UIBezierPath创建简单形状 . 307

Section 41.1: Create a UIWebView instance . 245
Section 41.2: Determining content size . 245
Section 41.3: Load HTML string . 245
Section 41.4: Making a URL request . 246
Section 41.5: Load JavaScript . 246
Section 41.6: Stop Loading Web Content . 247
Section 41.7: Reload Current Web Content . 247
Section 41.8: Load Document files like .pdf, .txt, .doc etc . 247
Section 41.9: Load local HTML file in webView . 247
Section 41.10: Make links That inside UIWebview clickable . 248
Chapter 42: UIActivityViewController . 250
Section 42.1: Initializing the Activity View Controller . 250
Chapter 43: UIControl - Event Handling with Blocks . 251
Section 43.1: Introduction . 251
Chapter 44: UISplitViewController . 254
Section 44.1: Master and Detail View interaction using Delegates in Objective C . 254
Chapter 45: UISlider . 264
Section 45.1: UISlider . 264
Section 45.2: SWIFT Example . 264
Section 45.3: Adding a custom thumb image . 264
Chapter 46: UIStoryboard . 266
Section 46.1: Getting an instance of UIStoryboard programmatically . 266
Section 46.2: Open another storyboard . 266
Chapter 47: UIPageViewController . 267
Section 47.1: Create a horizontal paging UIPageViewController programmatically . 267
Section 47.2: A simple way to create horizontal page view controllers (infinite pages) . 268
Chapter 48: UISplitViewController . 272
Section 48.1: Interacting Between Master and Detail View using Delegates in Objective C . 272
Chapter 49: UIFont . 276
Section 49.1: Declaring and initializing UIFont . 276
Section 49.2: Changing the font of a label . 276
Chapter 50: UIDevice . 277
Section 50.1: Get iOS device model name . 277
Section 50.2: Identifying the Device and Operating . 278
Section 50.3: Getting the Device Orientation . 279
Section 50.4: Getting the Device Battery State . 280
Section 50.5: Using the Proximity Sensor . 281
Section 50.6: Getting Battery Status and Battery Level . 281
Chapter 51: Make selective UIView corners rounded . 282
Section 51.1: Objective C code to make selected corner of a UiView rounded . 282
Chapter 52: Custom UIViews from XIB files . 283
Section 52.1: Wiring elements . 283
Section 52.2: How to make custom reusable UIView using XIB . 297
Chapter 53: UIBezierPath . 298
Section 53.1: Designing and drawing a Bezier Path . 298
Section 53.2: How to apply corner radius to rectangles drawn by UIBezierPath . 303
Section 53.3: How to apply shadows to UIBezierPath . 305
Section 53.4: How to create a simple shapes using UIBezierPath . 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
第54章：UIPickerView 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
第55章：UI反馈生成器 315
第55.1节：触发冲击触觉反馈 315
第56章：UI外观 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
第57章：使用UICollectionView的UIKit动力学 319
第57.1节：使用UIDynamicAnimator创建自定义拖拽行为 319
第58章：UIPheonix——简单、灵活、动态且高度可扩展的UI框架 329
第58.1节：示例UI组件 329
第58.2节：示例用法 331
第59章：UIKit动力学 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
第60章：用户界面测试 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
第61章：更改状态栏颜色 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
第62章：UISegmentedControl 377
第62.1节：通过代码创建UISegmentedControl 377
第63章：视图控制器之间传递数据 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
第64章：管理键盘 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
Chapter 54: UIPickerView 313
Section 54.1: Basic example 313
Section 54.2: Changing pickerView Background Color and text color 314
Chapter 55: UIFeedbackGenerator 315
Section 55.1: Trigger Impact Haptic 315
Chapter 56: UIAppearance 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
Chapter 57: UIKit Dynamics with UICollectionView 319
Section 57.1: Creating a Custom Dragging Behavior with UIDynamicAnimator 319
Chapter 58: UIPheonix - easy, flexible, dynamic & highly scalable UI framework 329
Section 58.1: Example UI Components 329
Section 58.2: Example Usage 331
Chapter 59: UIKit Dynamics 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
Chapter 60: UI Testing 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
Chapter 61: Change Status Bar Color 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
Chapter 62: UISegmentedControl 377
Section 62.1: Creating UISegmentedControl via code 377
Chapter 63: Passing Data between View Controllers 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
Chapter 64: Managing the Keyboard 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

第65章：检查网络连接 ... 396
第65.1节：创建Reachability监听器 ... 396
第65.2节：添加网络变化观察者 ... 396
第65.3节：网络不可用时的警报 ... 396
第65.4节：连接变为WIFI或蜂窝网络时的警报 ... 396
第65.5节：验证是否已连接到网络 ... 397
第66章：无障碍功能 ... 399
第66.1节：使视图可访问 ... 399
第66.2节：无障碍框架 ... 399
第66.3节：布局更改 ... 399
第66.4节：无障碍容器 ... 399
第66.5节：隐藏元素 ... 400
第66.6节：屏幕切换 ... 400
第66.7节：公告 ... 400
第66.8节：元素排序 ... 400
第66.9节：模态视图 ... 401
第67章：自动布局 ... 402
第67.1节：均匀分布视图 ... 402
第67.2节：中心约束 ... 403
第67.3节：通过编程设置约束 ... 406
第67.4节：UILabel及其父视图大小根据UILabel中的文本调整 ... 407
第67.5节：UILabel的固有尺寸 ... 412
第67.6节：视觉格式语言基础：代码中的约束！ ... 422
第67.7节：解决UILabel优先级冲突 ... 424
第67.8节：如何使用自动布局进行动画 ... 426
第67.9节：NSLayoutConstraint：代码中的约束！ ... 427
第67.10节：比例布局 ... 428
第67.11节：自动布局与非自动布局的混合使用 ... 429
第67.12节：如何使用自动布局 ... 429
第68章：MKMapView ... 431
第68.1节：更改地图类型 ... 431
第68.2节：模拟自定义位置 ... 435
第68.3节：设置地图缩放/区域 ... 435
第68.4节：使用MKLocalSearch实现本地搜索 ... 436
第68.5节：OpenStreetMap瓦片覆盖 ... 436
第68.6节：滚动到坐标和缩放级别 ... 438
第68.7节：使用注释 ... 440
第68.8节：添加MKMapView ... 440
第68.9节：显示用户位置和用户跟踪示例 ... 441
第68.10节：在地图上添加图钉/点注释 ... 444
第68.11节：调整地图视图的可见区域以显示所有注释 ... 445
第69章：NSArray ... 446
第69.1节：将数组转换为json字符串 ... 446
第70章：NSAttributedString ... 447
第70.1节：创建具有自定义字距（字母间距）的字符串 ... 447
第70.2节：更改单词或字符串的颜色 ... 447
第70.3节：移除所有属性 ... 448
第70.4节：在Swift中追加带属性的字符串和加粗文本 ... 448
第70.5节：创建带删除线的字符串 ... 448

Chapter 65: Checking for Network Connectivity ... 396
Section 65.1: Creating a Reachability listener ... 396
Section 65.2: Add observer to network changes ... 396
Section 65.3: Alert when network becomes unavailable ... 396
Section 65.4: Alert when connection becomes a WIFI or cellular network ... 396
Section 65.5: Verify if is connected to network ... 397
Chapter 66: Accessibility ... 399
Section 66.1: Make a View Accessible ... 399
Section 66.2: Accessibility Frame ... 399
Section 66.3: Layout Change ... 399
Section 66.4: Accessibility Container ... 399
Section 66.5: Hiding Elements ... 400
Section 66.6: Screen Change ... 400
Section 66.7: Announcement ... 400
Section 66.8: Ordering Elements ... 400
Section 66.9: Modal View ... 401
Chapter 67: Auto Layout ... 402
Section 67.1: Space Views Evenly ... 402
Section 67.2: Center Constraints ... 403
Section 67.3: Setting constraints programmatically ... 406
Section 67.4: UILabel & Parentview size According to Text in UILabel ... 407
Section 67.5: UILabel Intrinsic Size ... 412
Section 67.6: Visual Format Language Basics: Constraints in Code! ... 422
Section 67.7: Resolve UILabel Priority Conflict ... 424
Section 67.8: How to animate with Auto Layout ... 426
Section 67.9: NSLayoutConstraint: Contraints in code! ... 427
Section 67.10: Proportional Layout ... 428
Section 67.11: Mixed usage of Auto Layout with non-Auto Layout ... 429
Section 67.12: How to use Auto Layout ... 429
Chapter 68: MKMapView ... 431
Section 68.1: Change map-type ... 431
Section 68.2: Simulate a custom location ... 435
Section 68.3: Set Zoom/Region for Map ... 435
Section 68.4: Local search implementation using MKLocalSearch ... 436
Section 68.5: OpenStreetMap Tile-Overlay ... 436
Section 68.6: Scroll to coordinate and zoom-level ... 438
Section 68.7: Working With Annotation ... 440
Section 68.8: Add MKMapView ... 440
Section 68.9: Show UserLocation and UserTracking example ... 441
Section 68.10: Adding Pin/Point Annotation on map ... 444
Section 68.11: Adjust the map view's visible rect in order to display all annotations ... 445
Chapter 69: NSArray ... 446
Section 69.1: Convert Array into json string ... 446
Chapter 70: NSAttributedString ... 447
Section 70.1: Creating a string that has custom kerning (letter spacing) ... 447
Section 70.2: Change the color of a word or string ... 447
Section 70.3: Removing all attributes ... 448
Section 70.4: Appending Attributed Strings and bold text in Swift ... 448
Section 70.5: Create a string with strikethrough text ... 448

第71章：HTML与NSAttributed字符串的相互转换 ... 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
第72章：NSTimer ... 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
第73章：NSDate ... 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
第74章：NSNotificationCenter ... 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
第75章：NSURLSession ... 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
第76章：NSUserDefaults ... 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
第77章：NSHTTPCookieStorage ... 477
第77.1节：从NSUserDefault存储和读取Cookie ... 477
第78章：NSURLConnection ... 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
第79章：NSURL ... 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
第80章：NSData ... 482

Chapter 71: Convert HTML to NSAttributed string and vice verse ... 450
Section 71.1: Objective C code to convert HTML string to NSAttributedString and Vice Versa ... 450
Chapter 72: NSTimer ... 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
Chapter 73: NSDate ... 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
Chapter 74: NSNotificationCenter ... 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
Chapter 75: NSURLSession ... 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
Chapter 76: NSUserDefaults ... 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 NSUserDefaults ... 475
Section 76.6: Getting Default Values ... 475
Chapter 77: NSHTTPCookieStorage ... 477
Section 77.1: Store and read the cookies from NSUserDefault ... 477
Chapter 78: NSURLConnection ... 479
Section 78.1: Delegate methods ... 479
Section 78.2: Synchronous Request ... 479
Section 78.3: Asynchronous request ... 480
Chapter 79: NSURL ... 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
Chapter 80: NSData ... 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
第81章：NSInvocation 484
第81.1节：NSInvocation Objective-C 484
第82章：NSUserActivity 486
第82.1节：创建NSUserActivity 486
第83章：NSPredicate 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
第84章：NSBundle 491
第84.1节：通过路径获取Bundle 491
第84.2节：获取主Bundle 491
第85章：CAAnimation 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
第86章：并发 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
第87章：CAGradientLayer 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
第88章：Safari服务 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
第89章：CALayer 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
第90章：iOS - 使用Robbie Hanson框架实现XMPP 516
第90.1节：iOS XMPP Robbie Hanson示例与Openfire 516
第91章：Swift与Objective-C的互操作性 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
Chapter 81: NSInvocation 484
Section 81.1: NSInvocation Objective-C 484
Chapter 82: NSUserActivity 486
Section 82.1: Creating a NSUserActivity 486
Chapter 83: NSPredicate 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
Chapter 84: NSBundle 491
Section 84.1: Getting Bundle by Path 491
Section 84.2: Getting the Main Bundle 491
Chapter 85: CAAnimation 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
Chapter 86: Concurrency 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
Chapter 87: CAGradientLayer 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
Chapter 88: Safari Services 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
Chapter 89: CALayer 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
Chapter 90: iOS - Implementation of XMPP with Robbie Hanson framework 516
Section 90.1: iOS XMPP Robbie Hanson Example with Openfire 516
Chapter 91: Swift and Objective-C interoperability 519
Section 91.1: Using Objective-C Classes in Swift 519

第91.2节：在Objective-C中使用Swift类 521
第92章：自定义字体 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
第93章：AVSpeechSynthesizer 529
第93.1节：创建基本的文本转语音 529
第94章：本地化 530
第94.1节：iOS中的本地化 530
第95章：Alamofire 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
第96章：iBeacon 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
第97章：CLLocation 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
第98章：检查iOS版本 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
第99章：通用链接 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
第100章：iOS中的PDF创建 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
第101章：应用内购买 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
第102章：CGContext参考 552
第102.1节：绘制线条 552
第102.2节：绘制文本 552
第103章：核心定位（Core Location） 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
Chapter 92: Custom fonts 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
Chapter 93: AVSpeechSynthesizer 529
Section 93.1: Creating a basic text to speech 529
Chapter 94: Localization 530
Section 94.1: Localization in iOS 530
Chapter 95: Alamofire 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
Chapter 96: iBeacon 533
Section 96.1: iBeacon Basic Operation 533
Section 96.2: Ranging iBeacons 533
Section 96.3: Scanning specific Beacons 534
Chapter 97: CLLocation 535
Section 97.1: Distance Filter using 535
Section 97.2: Get User Location Using CLLocationManager 535
Chapter 98: Checking iOS version 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
Chapter 99: Universal Links 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
Chapter 100: PDF Creation in iOS 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
Chapter 101: In-App Purchase 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
Chapter 102: CGContext Reference 552
Section 102.1: Draw line 552
Section 102.2: Draw Text 552
Chapter 103: Core Location 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
第104章：FacebookSDK ... 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
第105章：AFNetworking ... 564
第105.1节：在自定义线程上调度完成块 ... 564
第106章：CTCallCenter ... 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
第107章：推送通知 ... 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
第108章：丰富推送通知扩展 - iOS 10 ... 578
第108.1节：通知内容扩展 ... 578
第108.2节：实现 ... 578
第109章：丰富通知 ... 580
第109.1节：创建一个简单的UNNotificationContentExtension ... 580
第110章：键值编码-键值观察 ... 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
第111章：初始化惯用法 ... 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
第112章：Storyboard ... 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
第113章：后台模式和事件 ... 591
第113.1节：后台播放音频 ... 591
第114章：Fastlane ... 593
第114.1节：fastlane工具 ... 593
第115章：CAShapeLayer ... 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
Chapter 104: FacebookSDK ... 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
Chapter 105: AFNetworking ... 564
Section 105.1: Dispatching completion block on a custom thread ... 564
Chapter 106: CTCallCenter ... 565
Section 106.1: CallKit - ios 10 ... 565
Section 106.2: Intercepting calls from your app even from the background ... 565
Chapter 107: Push Notifications ... 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
Chapter 108: Extension for rich Push Notification - iOS 10. ... 578
Section 108.1: Notification Content Extension ... 578
Section 108.2: Implementation ... 578
Chapter 109: Rich Notifications ... 580
Section 109.1: Creating a simple UNNotificationContentExtension ... 580
Chapter 110: Key Value Coding-Key Value Observation ... 585
Section 110.1: Observing a property of a NSObject subclass ... 585
Section 110.2: Use of context for KVO Observation ... 586
Chapter 111: Initialization idioms ... 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
Chapter 112: Storyboard ... 590
Section 112.1: Initialize ... 590
Section 112.2: Fetch Initial ViewController ... 590
Section 112.3: Fetch ViewController ... 590
Chapter 113: Background Modes and Events ... 591
Section 113.1: Play Audio in Background ... 591
Chapter 114: Fastlane ... 593
Section 114.1: fastlane tools ... 593
Chapter 115: CAShapeLayer ... 594
Section 115.1: Draw Rectangle ... 594
Section 115.2: Draw Circle ... 594
Section 115.3: CAShapeLayer Animation ... 595

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

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

第130.2节：使用CloudKit仪表板 642
第130.3节：将数据保存到CloudKit 643
第131章：GameplayKit 644
第131.1节：生成随机数 644
第131.2节：GKEntity和GKComponent 645
第132章：从命令行构建和归档Xcode 648
第132.1节：构建与归档 648
第133章：XCTest框架 - 单元测试 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
第134章：AVPlayer和AVPlayerViewController 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
第135章：iOS中的深度链接 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
第136章：核心图形 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
第137章：界面切换（Segues） 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
第138章：EventKit 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
第139章：SiriKit 668
第139.1节：向应用添加Siri扩展 668
第140章：联系人框架 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
第141章：iOS 10语音识别API 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音 673
第142章：StoreKit 675
第142.1节：从App Store获取本地化的产品信息 675
第143章：代码签名 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
Chapter 131: GameplayKit 644
Section 131.1: Generating random numbers 644
Section 131.2: GKEntity and GKComponent 645
Chapter 132: Xcode Build & Archive From Command Line 648
Section 132.1: Build & Archive 648
Chapter 133: XCTest framework - Unit Testing 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
Chapter 134: AVPlayer and AVPlayerViewController 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
Chapter 135: Deep Linking in iOS 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
Chapter 136: Core Graphics 660
Section 136.1: Creating a Core Graphics Context 660
Section 136.2: Presenting the Drawn Canvas to User 660
Chapter 137: Segues 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 a triggering a Segue 663
Section 137.4: Deciding if an invoked Segue should be performed 663
Section 137.5: Trigger Segue Programmatically 663
Chapter 138: EventKit 665
Section 138.1: Accessing different types of calendars 665
Section 138.2: Requesting Permission 665
Section 138.3: Adding an event 666
Chapter 139: SiriKit 668
Section 139.1: Adding Siri Extension to App 668
Chapter 140: Contacts Framework 670
Section 140.1: Adding a Contact 670
Section 140.2: Authorizing Contact Access 670
Section 140.3: Accessing Contacts 671
Chapter 141: iOS 10 Speech Recognition API 673
Section 141.1: Speech to text: Recognize speech from a bundle contained audio recording 673
Chapter 142: StoreKit 675
Section 142.1: Get localized product information from the App Store 675
Chapter 143: Code signing 676
Section 143.1: Provisioning Profiles 676

第144章：使用Application Loader创建上传到App Store的.ipa文件 677
第144.1节：创建.ipa文件以通过Application Loader上传应用到App Store 677
第145章：尺寸类别与自适应 681
第145.1节：通过Storyboard实现尺寸类别与自适应 681
第146章：MKDistanceFormatter 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
第147章：3D触控 689
第147.1节：使用Swift的3D Touch 689
第147.2节：3D Touch Objective-C示例 690
第148章：GameCenter排行榜 692
第148.1节：GameCenter排行榜 692
第149章：钥匙串 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
第150章：使用宏处理多环境 701
第150.1节：使用多个目标和宏处理多环境 701
第151章：设置视图背景 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
第152章：Block 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
第153章：自动布局中的内容拥抱/内容压缩 720
第153.1节：定义：内在内容大小 720
第154章：iOS谷歌地点API 721
第154.1节：从当前位置获取附近地点 721
第155章：导航栏 723
第155.1节：SWIFT示例 723
第155.2节：自定义默认导航栏外观 723
第156章：应用程序范围操作 724
第156.1节：获取最顶层的UIViewController 724
第156.2节：拦截系统事件 724
第157章：CoreImage滤镜 725
第157.1节：核心图像滤镜示例 725
第158章：使用CoreImage/OpenCV进行人脸检测 731
第158.1节：人脸和特征检测 731
第159章：MPMediaPickerDelegate 734
第159.1节：使用MPMediaPickerControllerDelegate加载音乐并用AVAudioPlayer播放 734

Chapter 144: Create .ipa File to upload on appstore with Applicationloader 677
Section 144.1: create .ipa file to upload app to appstore with Application Loader 677
Chapter 145: Size Classes and Adaptivity 681
Section 145.1: Size Classes and Adaptivity through Storyboard 681
Chapter 146: MKDistanceFormatter 687
Section 146.1: String from distance 687
Section 146.2: Distance units 687
Section 146.3: Unit style 687
Chapter 147: 3D Touch 689
Section 147.1: 3D Touch with Swift 689
Section 147.2: 3 D Touch Objective-C Example 690
Chapter 148: GameCenter Leaderboards 692
Section 148.1: GameCenter Leaderboards 692
Chapter 149: Keychain 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
Chapter 150: Handle Multiple Environment using Macro 701
Section 150.1: Handle multiple environment using multiple target and macro 701
Chapter 151: Set View Background 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
Chapter 152: Block 718
Section 152.1: Custom completion block for Custom Methods 718
Section 152.2: UIView Animations 718
Section 152.3: Modify captured variable 718
Chapter 153: Content Hugging/Content Compression in Autolayout 720
Section 153.1: Definition: Intrinsic Content Size 720
Chapter 154: iOS Google Places API 721
Section 154.1: Getting Nearby Places from Current Location 721
Chapter 155: Navigation Bar 723
Section 155.1: SWIFT Example 723
Section 155.2: Customize default navigation bar appearance 723
Chapter 156: App wide operations 724
Section 156.1: Get the top most UIViewController 724
Section 156.2: Intercept System Events 724
Chapter 157: CoreImage Filters 725
Section 157.1: Core Image Filter Example 725
Chapter 158: Face Detection Using CoreImage/OpenCV 731
Section 158.1: Face and Feature Detection 731
Chapter 159: MPMediaPickerDelegate 734
Section 159.1: Load music with MPMediaPickerControllerDelegate and play it with AVAudioPlayer 734

第160章：图表（Coreplot） 736
第160.1节：使用CorePlot制作图表 736
第161章：Swift中的FCM消息传递 738
第161.1节：在Swift中初始化FCM 738
第162章：在iOS中创建自定义框架 740
第162.1节：用Swift创建框架 740
第163章：自定义键盘 741
第163.1节：自定义键盘示例 741
第164章：AirDrop 748
第164.1节：AirDrop 748
第165章：SLComposeViewController 749
第165.1节：用于Twitter、Facebook、新浪微博和腾讯微博的SLComposeViewController 749
第166章：iOS中的AirPrint教程 751
第166.1节：AirPrint打印横幅文本 751
第167章：Carthage iOS设置 753
第167.1节：Carthage安装Mac 753
第168章：Healthkit 754
第168.1节：HealthKit 754
第169章：iOS中的Core SpotLight 757
第169.1节：Core-Spotlight 757
第170章：Core Motion 759
第170.1节：访问气压计以获取相对高度 759
第171章：二维码扫描器 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
第172章：iOS的plist 763
第172.1节：示例： 763
第172.2节：从Plist保存和编辑/删除数据 765
第173章：WCSessionDelegate 767
第173.1节：手表套件控制器（WKInterfaceController） 767
第174章：应用代理（AppDelegate） 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
第175章：应用提交流程 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
第176章：文件句柄 774
第176.1节：从文档目录分块读取文件 774
第177章：基本文本文件输入输出 776
第177.1节：从文档文件夹读取和写入 776
第178章：iOS 语音合成（TTS） 778
第178.1节：文本转语音 778
第179章：MPVolumeView 779

Chapter 160: Graph (Coreplot) 736
Section 160.1: Making graphs with CorePlot 736
Chapter 161: FCM Messaging in Swift 738
Section 161.1: Initialize FCM in Swift 738
Chapter 162: Create a Custom framework in iOS 740
Section 162.1: Create Framework in Swift 740
Chapter 163: Custom Keyboard 741
Section 163.1: Custom KeyBoard Example 741
Chapter 164: AirDrop 748
Section 164.1: AirDrop 748
Chapter 165: SLComposeViewController 749
Section 165.1: SLComposeViewController for Twitter, facebook, SinaWelbo and TencentWelbo 749
Chapter 166: AirPrint tutorial in iOS 751
Section 166.1: AirPrint printing Banner Text 751
Chapter 167: Carthage iOS Setup 753
Section 167.1: Carthage Installation Mac 753
Chapter 168: Healthkit 754
Section 168.1: HealthKit 754
Chapter 169: Core SpotLight in iOS 757
Section 169.1: Core-Spotlight 757
Chapter 170: Core Motion 759
Section 170.1: Accessing Barometer to get relative altitude 759
Chapter 171: QR Code Scanner 760
Section 171.1: Scanning QR code with AVFoudation framework 760
Section 171.2: UIViewController scanning for QR and displaying video input 761
Chapter 172: plist iOS 763
Section 172.1: Example: 763
Section 172.2: Save and edit/delete data from Plist 765
Chapter 173: WCSessionDelegate 767
Section 173.1: Watch kit controller (WKInterfaceController) 767
Chapter 174: AppDelegate 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
Chapter 175: App Submission Process 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
Chapter 176: FileHandle 774
Section 176.1: Read file from document directory in chunks 774
Chapter 177: Basic text file I/O 776
Section 177.1: Read and write from Documents folder 776
Chapter 178: iOS TTS 778
Section 178.1: Text To Speech 778
Chapter 179: MPVolumeView 779

第179.1节：添加 MPVolumeView 779
第180章：Objective-C 关联对象 780
第180.1节：基本关联对象示例 780
第181章：在视图控制器之间传递数据（带消息框概念） 781
第181.1节：简单示例用法 781
第182章：MVVM 782
第182.1节：无响应式编程的MVVM 782
第183章：缓存在线图片 785
第183.1节：AlamofireImage 785
第184章：队列中的链式块（使用MKBlockQueue） 786
第184.1节：示例代码 786
第185章：模拟器 788
第185.1节：启动模拟器 788
第185.2节：3D / 力触控模拟 788
第185.3节：更改设备型号 789
第185.4节：模拟器导航 790
第186章：后台模式 791
第186.1节：开启后台模式功能 791
第186.2节：后台获取 791
第186.3节：测试后台获取 792
第186.4节：后台音频 794
第187章：OpenGL 795
第187.1节：示例项目 795
第188章：MVP架构 796
第188.1节：Dog.swift 797
第188.2节：DoggyService.swift 797
第188.3节：DoggyPresenter.swift 797
第188.4节：DoggyView.swift 798
第188.5节：DoggyListViewController.swift 798
第189章：使用CoreBluetooth配置信标 800
第189.1节：显示所有蓝牙低功耗（BLE）名称 800
第189.2节：连接并读取主值 801
第189.3节：写入主值 802
第190章：核心数据 804
第190.1节：核心数据操作 804
第191章：带仪器的配置文件 805
第191.1节：时间分析器 805
第192章：应用评分/审核请求 814
第192.1节：评分/审核iOS应用 814
第193章：MyLayout布局 815
第193.1节：使用MyLayout的简单演示 815
第194章：模拟器构建 817
第194.1节：在模拟器上手动安装构建 817
第195章：使用GPX文件模拟iOS位置 818
第195.1节：您的.gpx文件：MPS_HQ.gpx 818
第195.2节：设置此位置： 818
第196章：SqlCipher集成 820

Section 179.1: Adding a MPVolumeView 779
Chapter 180: Objective-C Associated Objects 780
Section 180.1: Basic Associated Object Example 780
Chapter 181: Passing Data between View Controllers (with MessageBox-Concept) 781
Section 181.1: Simple Example Usage 781
Chapter 182: MVVM 782
Section 182.1: MVVM Without Reactive Programming 782
Chapter 183: Cache online images 785
Section 183.1: AlamofireImage 785
Chapter 184: Chain Blocks in a Queue (with MKBlockQueue) 786
Section 184.1: Example Code 786
Chapter 185: Simulator 788
Section 185.1: Launching Simulator 788
Section 185.2: 3D / Force Touch simulation 788
Section 185.3: Change Device Model 789
Section 185.4: Navigating Simulator 790
Chapter 186: Background Modes 791
Section 186.1: Turning on the Background Modes capability 791
Section 186.2: Background Fetch 791
Section 186.3: Testing background fetch 792
Section 186.4: Background Audio 794
Chapter 187: OpenGL 795
Section 187.1: Example Project 795
Chapter 188: MVP Architecture 796
Section 188.1: Dog.swift 797
Section 188.2: DoggyService.swift 797
Section 188.3: DoggyPresenter.swift 797
Section 188.4: DoggyView.swift 798
Section 188.5: DoggyListViewController.swift 798
Chapter 189: Configure Beacons with CoreBluetooth 800
Section 189.1: Showing names of all Bluetooth Low Energy (BLE) 800
Section 189.2: Connect and read major value 801
Section 189.3: Write major value 802
Chapter 190: Core Data 804
Section 190.1: Operations on core data 804
Chapter 191: Profile with Instruments 805
Section 191.1: Time Profiler 805
Chapter 192: Application rating/review request 814
Section 192.1: Rate/Review iOS Application 814
Chapter 193: MyLayout 815
Section 193.1: A Simple Demo to use MyLayout 815
Chapter 194: Simulator Builds 817
Section 194.1: Installing the build manually on simulator 817
Chapter 195: Simulating Location Using GPX files iOS 818
Section 195.1: Your .gpx file: MPS_HQ.gpx 818
Section 195.2: To set this location: 818
Chapter 196: SqlCipher integration 820

第196.1节：代码集成： 820
第197章：安全性 821
第197.1节：保护iTunes备份中的数据 821
第197.2节：使用SSL的传输安全 822
第198章：应用传输安全（ATS） 824
第198.1节：加载所有HTTP内容 824
第198.2节：选择性加载HTTP内容 824
第198.3节：端点要求SSL 825
第199章：选择最佳iOS架构模式的指南 826
第199.1节：MVP模式 826
第199.2节：MVVM模式 827
第199.3节：VIPER模式 828
第199.4节：MVC模式 829
第200章：多播代理 830
第200.1节：适用于任何控件的多播代理 830
第201章：使用图像资源 834
第201.1节：使用图像资源的启动图像 834
第201.2节：使用图像资源的应用图标 837
第202章：Objective-C中的运行时 842
第202.1节：使用关联对象 842
第203章：模态展示样式 844
第203.1节：使用界面构建器探索模态展示样式 844
第204章：CydiaSubstrate调整 853
第204.1节：使用Theos创建新调整 853
第205章：从图像创建视频 855
第205.1节：从UIImages创建视频 855
第206章：可编码（Codable） 857
第206.1节：Swift 4中使用Codable与JSONEncoder和JSONDecoder 857
第207章：异步加载图像 859
第207.1节：最简单的方法 859
第207.2节：下载后检查单元格是否仍然可见 859
第208章：添加Swift桥接头文件 860
第208.1节：如何手动创建Swift桥接头文件 860
第208.2节：Xcode自动创建 860
第209章：创建应用程序ID 862
第209.1节：创建应用内购买产品 862
第209.2节：创建沙盒用户 864
第210章：Swift：在AppDelegate中更改rootViewController以呈现主界面或登录/引导流程 865
第210.1节：选项1：替换根视图控制器（良好） 865
第210.2节：选项2：以模态方式呈现替代流程（更佳） 865
鸣谢 867
你可能也喜欢 874

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

关于

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

About

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

第1章：iOS开发入门

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

1. 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。）

2. 打开Xcode。将会打开以下窗口：

Chapter 1: Getting started with iOS Development

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

1. 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.)

2. 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。
- 检出现有项目： 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:

这是一个普通的应用程序开发模板。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—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.

uktcSchedule 〉 uktcShedule 〉 Classes 〉 Slide-OutView 〉 LeftViewController.swift 〉 No Selection

```swift
//
//  LeftViewController.swift
//  uktcShedule
//
//  Created by Deyan Marinov on 10/9/15.
//  Copyright © 2015 Deyan Marinov. All rights reserved.
//

import UIKit

enum LeftMenu: Int {
    case Main = 0
    case Swift
    case Java
}

protocol LeftMenuProtocol : class {
    func changeViewController(menu: LeftMenu)
}

class LeftViewController : UIViewController, LeftMenuProtocol {

    @IBOutlet weak var tableView: UITableView!
    var menus = ["Main", "Swift", "Java"]
    var mainViewController: UIViewController!
    var swiftViewController: UIViewController!
    var javaViewController: UIViewController!
    var goViewController: UIViewController!
    var nonMenuViewController: UIViewController!
    var imageHeaderView: ImageHeaderView!

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        self.tableView.separatorColor = UIColor(red: 224/255, green: 224/255, blue: 224/255, alpha: 1.0)

        let storyboard = UIStoryboard(name: "Main", bundle: nil)
        let swiftViewController = storyboard.instantiateViewControllerWithIdentifier("SwiftViewController") as!
            SwiftViewController
        self.swiftViewController = UINavigationController(rootViewController: swiftViewController)

        let javaViewController = storyboard.instantiateViewControllerWithIdentifier("JavaViewController") as!
            JavaViewController
        self.javaViewController = UINavigationController(rootViewController: javaViewController)
```

- 界面构建器。 图形化创建和编辑用户界面文件。

uktcSchedule 〉 uktcShedule 〉 Classes 〉 Slide-OutView 〉 LeftViewController.swift 〉 No Selection

```swift
//
//  LeftViewController.swift
//  uktcShedule
//
//  Created by Deyan Marinov on 10/9/15.
//  Copyright © 2015 Deyan Marinov. All rights reserved.
//

import UIKit

enum LeftMenu: Int {
    case Main = 0
    case Swift
    case Java
}

protocol LeftMenuProtocol : class {
    func changeViewController(menu: LeftMenu)
}

class LeftViewController : UIViewController, LeftMenuProtocol {

    @IBOutlet weak var tableView: UITableView!
    var menus = ["Main", "Swift", "Java"]
    var mainViewController: UIViewController!
    var swiftViewController: UIViewController!
    var javaViewController: UIViewController!
    var goViewController: UIViewController!
    var nonMenuViewController: UIViewController!
    var imageHeaderView: ImageHeaderView!

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        self.tableView.separatorColor = UIColor(red: 224/255, green: 224/255, blue: 224/255, alpha: 1.0)

        let storyboard = UIStoryboard(name: "Main", bundle: nil)
        let swiftViewController = storyboard.instantiateViewControllerWithIdentifier("SwiftViewController") as!
            SwiftViewController
        self.swiftViewController = UINavigationController(rootViewController: swiftViewController)

        let javaViewController = storyboard.instantiateViewControllerWithIdentifier("JavaViewController") as!
            JavaViewController
        self.javaViewController = UINavigationController(rootViewController: javaViewController)
```

- **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可以快速访问以下资源：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UIViewControlle
r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.

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。

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

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 `UILabel`s, 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：

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)
```


Like as below:

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 = attributesString;
```

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 = attributesString;
```

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 objectAtIndexPath: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;
}
```

Swfit 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 objectAtIndexPath: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;
}
```

Swfit 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

Vie...cene
Vie...troller
View
View
View
User Name
Change to Attributed
Label
Text
Plain
Chathuranga Silva
Color
Font
Avenir Book 17.0
Alignment
Lines
1
Behavior
Enabled
Highlighted
Baseline
Align Baselines
Line Breaks
Truncate Tail
Autoshrink
Fixed Font Size
Tighten Letter Spacing
Highlighted
Default
Shadow
Default
Shadow Offset
0
-1
Horizontal
Vertical
View
Mode
Left
Semantic
Unspecified
Chathuranga Silva
Changed To Attributed text
Text
Attributed
Avenir Book 17.0
Chathuranga Silva
Lines
1
Behavior
Enabled
Highlighted
Baseline
Align Baselines
Line Breaks
Truncate Tail
Autoshrink
Fixed Font Size
Tighten Letter Spacing
Highlighted
Default
Shadow
Default
Shadow Offset
0
-1
Horizontal
Vertical
View
Mode
Left
Semantic
Unspecified
Tag
0
Interaction
User Interaction Enabled
Chathuranga Silva
iOS Developer .....qdqdqdqdqdqdqdqd

步骤 2

Vie...cene
Vie...troller
View
View
View
User Name
Change to Attributed
Label
Text
Plain
Chathuranga Silva
Color
Font
Avenir Book 17.0
Alignment
Lines
1
Behavior
Enabled
Highlighted
Baseline
Align Baselines
Line Breaks
Truncate Tail
Autoshrink
Fixed Font Size
Tighten Letter Spacing
Highlighted
Default
Shadow
Default
Shadow Offset
0
-1
Horizontal
Vertical
View
Mode
Left
Semantic
Unspecified
Chathuranga Silva
Changed To Attributed text
Text
Attributed
Avenir Book 17.0
Chathuranga Silva
Lines
1
Behavior
Enabled
Highlighted
Baseline
Align Baselines
Line Breaks
Truncate Tail
Autoshrink
Fixed Font Size
Tighten Letter Spacing
Highlighted
Default
Shadow
Default
Shadow Offset
0
-1
Horizontal
Vertical
View
Mode
Left
Semantic
Unspecified
Tag
0
Interaction
User Interaction Enabled
Chathuranga Silva
iOS Developer .....qdqdqdqdqdqdqdqd

Step 2

点击标签文本并右键点击

步骤 3

然后点击字体 -> 显示字体

Click the label text and Right click

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 shaddow/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。

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()
```

If you don't See the preview click the show image in settings

Finally change shaddow and offset according to your preferences.

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：

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.GreaterThanOrEqual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: .GreaterThanOrEqual,
    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
```

第2.12节：换行模式

使用代码


```
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:

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 `.GreaterThanOrEqual` on the `.Height` attribute

Version ≥ iOS 6

Swift

```
label.numberOfLines = 0

let heightConstraint = NSLayoutConstraint(
    item: label,
    attribute: .Height,
    relatedBy: .GreaterThanOrEqual,
    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
```

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:NSLineBreakByTruncatingTail];
```

- `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:NSLineBreakByTruncatingTail];
```

- `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 asign 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:NSUnicodeStringEncoding] 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NS AttributedString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 添加属性: NSForegroundColorAttributeName
                  值: [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:NSUnicodeStringEncoding] 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 preform 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!

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 系统
    case 详细披露
    case 信息亮
    case 信息暗
    case 添加联系人
    static var 圆角矩形: 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:
.touchUpInside)
```

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:
.touchUpInside)
```

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：

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:

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(x: 0, y: 0, width: 320,
height: 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 日期和时间
    case 倒计时器
}
```

- 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(x: 0, y: 0, width: 320,
height: 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^⌘H（control-command-H）返回主屏幕，然后按⌘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 ^⌘H (control-command-H) to go home and then type ⌘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 `fireDate` 和 `timeZone` 属性。

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 {
    返回
}

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 102. 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 = NSString.localizedUserNotificationString(forKey: "Elon said:", arguments:
nil)
content.body = NSString.localizedUserNotificationString(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 = NSString.localizedUserNotificationString(forKey: "Elon said:", arguments:
nil)
    content.body = NSString.localizedUserNotificationString(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(withIdentifiers: ["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！"
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.
UNTimeIntervalNotificationTrigger *trigger = [UNTimeIntervalNotificationTrigger
                                              triggerWithTimeInterval:5.f
                                              repeats:NO];
UNNotificationRequest *request = [UNNotificationRequest requestWithIdentifier:@"FiveSecond"
                                                                  内容:content
触发器:trigger];
/// 3. 安排本地通知
UNUserNotificationCenter *center = [UNUserNotificationCenter currentNotificationCenter];
[center addNotificationRequest:request withCompletionHandler:^(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(withIdentifiers: ["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.
UNTimeIntervalNotificationTrigger *trigger = [UNTimeIntervalNotificationTrigger
                                              triggerWithTimeInterval:5.f
                                              repeats:NO];
UNNotificationRequest *request = [UNNotificationRequest requestWithIdentifier:@"FiveSecond"
                                                                    content:content
                                                                    trigger:trigger];
/// 3. schedule localNotification
UNUserNotificationCenter *center = [UNUserNotificationCenter currentNotificationCenter];
[center addNotificationRequest:request withCompletionHandler:^(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, withCompletionHandler: 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, withCompletionHandler: 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 imageFromAttributedString:str];
```

函数 imageFromAttributedString 定义如下：

```
- (UIImage *)imageFromAttributedString:(NSAttributedString *)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 imageFromAttributedString:str];
```

The function `imageFromAttributedString` is as defined below:

```
- (UIImage *)imageFromAttributedString:(NSAttributedString *)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;

}

@end
```

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;

}

@end
```

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UIImage

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
```

结果如下：

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: UIImage 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 property 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。

居中

图像在视图中居中，但图像的长度和宽度未被拉伸。

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。

底部

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.

Center

The image is centered in the view, but the length and width of the image are not stretched.

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.

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
```

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
```

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
```

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UIImage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 `UIImages` 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UIImage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 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UITableViewCells，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。

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
```


example of what a custom cell may look like.

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中将其出列：

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 = UITableViewCellSeparatorStyleNone;
```

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 = UITableViewCellSeparatorStyleNone;
```

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
}
```

> cellForRowAtIndexPath: 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, cellForRowAtIndexPath indexPath: NSIndexPath) ->
UITableViewCell {
    let cell = tableView.dequeueReusableCellWithIdentifier("MyCustomCell", forIndexPath: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。

Objective-C

```
- (NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section {
```



```
    // 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
}
```

> `cellForRowAtIndexPath:` 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, cellForRowAtIndexPath indexPath: NSIndexPath) ->
UITableViewCell {
    let cell = tableView.dequeueReusableCellWithIdentifier("MyCustomCell", forIndexPath: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 {
```

```
    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UITableVie w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 删除行，索引路径为 @[indexPath]
使用行动画 UITableViewRowAnimationAutomatic];
            break;
        默认:
            // 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, 提交编辑样式 editingStyle:
UITableViewCellEditingStyle, 对于行索引路径 indexPath: NSIndexPath) {
    switch editingStyle {
        case .Insert:
            self.向数据模型中插入新数据()
tableView.插入行，索引路径为([indexPath]), 使用行动画:.Automatic)
        case .Delete:
            self.从数据模型中移除数据，索引为indexPath.row)
            tableView.删除行，索引路径为([indexPath]), 使用行动画:.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 editingStyle:
UITableViewCellEditingStyle, forRowAtIndexPath indexPath: NSIndexPath) {
    switch editingStyle {
        case .Insert:
            self.insertNewDataIntoDataModel()
            tableView.insertRowsAtIndexPaths([indexPath], withRowAnimation:.Automatic)
        case .Delete:
            self.removeDataFromDataModelAtIndex(indexPath.row)
            tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation:.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, heightForRowAtIndexPath indexPath: NSIndexPath) -> 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, heightForRowAtIndexPath indexPath: NSIndexPath) -> 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 whats 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:cellForRowAtIndexPath，要求你为在tableView:numberOfRowsInSection中指定的每一行创建并返回一个单元格。所以，如果你说需要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:cellForRowAtIndexPath`, 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dataSource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 orer 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`, this is called whenever a row is tapped, which allows you to do something in response. For our example, we'll just print a confirmation statement to the Xcode log.

```
// 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

```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 {

        // 如有需要，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

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

@interface ViewController: UIViewController <UITableViewDelegate, UITableViewDataSource> {
    IBOutlet UITableView *myTableView;
    NSArray *myDataArray;
}
```



```swift
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

```objc
#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 vartableView: 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:@"%@ %d",[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:@"%@ %d",[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:[NSIndexSet 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:[NSIndexSet 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 UITableViewContr oller，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 tablevie wCellStyle 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 constrains, 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UITableViewCell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"


    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
}
```



```
}


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"


    return cell
}

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 TableView

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cellForRowAtIndexPath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 `UITableView` 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 `cellForRowAtIndexPath` method

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath
{
    //Creted 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 = 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 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 = 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.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` `UIView`s. 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，提供了一些文本以使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
```


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
```

```
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
```



```
                                    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];

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-cordinate 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:

```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：

```objc
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

```swift
let alertController = UIAlertController(title: "演示", message: "带有两个按钮的演示",
preferredStyle: UIAlertControllerStyle.actionSheet)
```

Objective-C

```objc
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


```swift
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:

```objc
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

```swift
let alertController = UIAlertController(title: "Demo", message: "A demo with two buttons",
preferredStyle: UIAlertControllerStyle.actionSheet)
```

Objective-C

```objc
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: "Lauching 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UIAlertActionStyle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 `UIAlertActionStyle` 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 *pattenColor = [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

- 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 *pattenColor = [UIColor colorWithPatternImage:[UIImage imageNamed:@"myImage.png"]];
```

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;

        默认:
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")!)
```

Carrier 4:48 AM

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;
    如果 ([c getRed:&r green:&g blue:&b alpha:&a])
```


Carrier 4:48 AM

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])
```

```
        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]

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。

```
        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]`

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.

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 Postition

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>";
NSAttributedString *attributedString = [[NSMutableAttributedString alloc]
                                        initWithData: [htmlString
dataUsingEncoding:NSUnicodeStringEncoding]
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>";
NSAttributedString *attributedString = [[NSMutableAttributedString alloc]
                                        initWithData: [htmlString
dataUsingEncoding:NSUnicodeStringEncoding]
                                        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:"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvxyz").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:"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvxyz").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 NavigationController

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。
- 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。
- 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。____

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。____


```
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。

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。

That will give you the default Refresh icon.

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.

注意事项

- 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;
```

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];
```


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;
```

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</kbd 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UIView，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 `UIViews` 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

- YouTube视频教程：UIScrollView - 如何保持视图在屏幕上

第29.2节：创建UIScrollView

创建一个以CGRect为框架的UIScrollView实例。

Swift

```
let scrollview = UIScrollView.init(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

- YouTube video tutorial: UIScrollView - How to keep your views on screen

Section 29.2: Create a UIScrollView

Create an instance of `UIScrollView` with a `CGRect` as frame.

Swift

```
let scrollview = UIScrollView.init(frame: CGRect(x: 0, y: 0, width: 320, height: 400))
```

Objective-C

```
UIScrollView *scrollview = [[UIScrollView alloc] initWithFrame:CGRectMake(0, 0, 320, 400)];
```

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

- UIScrollView教程

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的大小。当可滚动区域，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
```


- UIScrollView Tutorial

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 therefor 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

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 5: Add both Stackview in one Stackview

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 Constrain 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 = .水平
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:CGRectZero];
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: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。
```



```
UIStackViewDistributionFillProportionally, UIStackViewDistributionEqualSpacing,
UIStackViewDistributionEqualCentering

UILabel *label = [[UILabel alloc] initWithFrame:CGRectZero];
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:CGRectZero];
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 with/heigh alerts on interface builder. As this answer explained, it is necessary to:

1. 在UIScrollView中添加一个UIView（contentScrollView）；
2. 在该contentScrollView中，将上下左右边距设置为0
3. 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.contentScrollView.scrollToShowView(view: textView)
}
```


1. Add in the UIScrollView a UIView (the contentScrollView);
2. In this contentScrollView, set top, bottom, left and right margins to 0
3. 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.contentScrollView.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 在 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 在 约束集合 {
            self.scrollView.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
            约束, options: NSLayoutFormatOptions(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: NSLayoutFormatOptions(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 在 约束集合 {
            self.contentView.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
            约束, options: NSLayoutFormatOptions(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: NSLayoutFormatOptions(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: NSLayoutFormatOptions(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: NSLayoutFormatOptions(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`

Carrier
5:08 PM

Carrier
5:08 PM

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.

View Controller Scene
View Controller
Top Layout Guide
Bottom Layout Guide
View
Scroll View
View
Sample text.....
Sample text.....
Text View
Image View
Label
Constraints
Constraints
trailing = View.trailing
bottom = View.bottom
View.top = top
View.leading = leading
Constraints
trailing = Scroll View.trailing
Scroll View.top = Top Layout Guid...
bottom = Scroll View.bottom
Scroll View.leading = leading
View.height = height + 25 @250
View.width = width
Navigation Item
First Responder
Exit
Navigation Controller Scene
Sample text.....
Sample text.....
Lorem ipsum dolor sit er elit lamet, consectetaur cillium adipisicing pecu, 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
Image View
Scroll View
Indicator Insets
Top
Bottom
Left
Right
View
Show
Frame Rectangle
X
Y
Width
Height
Arrange
Position View
Layout Margins
Default
Preserve Superview Mar...
Follow Readable Width
Constraints
All
This Size Class
Sibling & Ancestor Constraints
Trailing Space to: Superview
Edit
Leading Space to: Superview
Edit
Bottom Space to: Superview
Edit
Top Space to: Top Layout...
Edit
Label
Label - A variably sized amount of static text.

View Controller Scene
View Controller
Top Layout Guide
Bottom Layout Guide
View
Scroll View
View
Sample text.....
Sample text.....
Text View
Image View
Label
Constraints
Constraints
trailing = View.trailing
bottom = View.bottom
View.top = top
View.leading = leading
Constraints
trailing = Scroll View.trailing
Scroll View.top = Top Layout Guid...
bottom = Scroll View.bottom
Scroll View.leading = leading
View.height = height + 25 @250
View.width = width
Navigation Item
First Responder
Exit
Navigation Controller Scene
Sample text.....
Sample text.....
Lorem ipsum dolor sit er elit lamet, consectetaur cillium adipisicing pecu, 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
Image View
Scroll View
Indicator Insets
Top
Bottom
Left
Right
View
Show
Frame Rectangle
X
Y
Width
Height
Arrange
Position View
Layout Margins
Default
Preserve Superview Mar...
Follow Readable Width
Constraints
All
This Size Class
Sibling & Ancestor Constraints
Trailing Space to: Superview
Edit
Leading Space to: Superview
Edit
Bottom Space to: Superview
Edit
Top Space to: Top Layout...
Edit
Label
Label - A variably sized amount of static text.

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 <

```
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，假设它是我们视图的子视图，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 compaitiblity 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`UITextFields`属性上单独设置以下类型：`keyboardType`

```
typedef NS_ENUM(NSInteger, UIKeyboardType) {
    UIKeyboardTypeDefault,                // 当前输入法的默认类型。
UIKeyboardTypeASCIICapable,           // 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 = UIKeyboardTypeASCIICapable, // 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 apropriately
    _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 `UITextFields` property: `keyboardType`

```
typedef NS_ENUM(NSInteger, UIKeyboardType) {
    UIKeyboardTypeDefault,                // Default type for the current input method.
    UIKeyboardTypeASCIICapable,           // 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 = UIKeyboardTypeASCIICapable, // 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()
    }
}

/**
存储当前正在编辑的 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. Dismsses 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:

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：

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

//
// Created by Ali on 7/21/16.
// Copyright © 2016 mag. All rights reserved.
//
import UIKit
class ViewController: UIViewController {
Insert Outlet, Action, or Outlet Collection
super.viewDidLoad()
// Do any additional setup after loading the view, typically from a nib.
}
override func didReceiveMemoryWarning() {
super. didReceiveMemoryWarning ()
// Dispose of any resources that can be recreated.
}
}
me:
Submit
Connection Outlet
Object View Controller
Name textField
Type UITextField
Storage Strong
Cancel
Connect

//
// Created by Ali on 7/21/16.
// Copyright © 2016 mag. All rights reserved.
//
import UIKit
class ViewController: UIViewController {
Insert Outlet, Action, or Outlet Collection
super.viewDidLoad()
// Do any additional setup after loading the view, typically from a nib.
}
override func didReceiveMemoryWarning() {
super. didReceiveMemoryWarning ()
// Dispose of any resources that can be recreated.
}
}
me:
Submit
Connection Outlet
Object View Controller
Name textField
Type UITextField
Storage Strong
Cancel
Connect

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 vieDidLoad 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 = .无
```

Objective-C

```
textField.autocapitalizationType = UITextAutocapitalizationTypeNone;
```

所有选项：

- `.无` \ `UITextAutocapitalizationTypeNone` ：不自动大写任何内容
- `.单词` \ `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）

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`。

- `.Sentences` \ `UITextAutocapitalizationTypeSentences` : Autocapitalize the first word in a sentence
- `.AllCharacters` \ `UITextAutocapitalizationTypeAllCharacters` : Autocapitalize every letter (i.e. caps lock)

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 out 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")

//设置关闭状态图片
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 蓝色];
[mySwitch setTintColor: [UIColor 蓝色]];

//开启状态
mySwitch.onTintColor = [UIColor 青色];
[mySwitch setOnTintColor: [UIColor 青色]];
```

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 UICollectionView DelegateFlowLayout 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.

进一步学习

- 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(withReuseIdentifier:
"ProductCategoryCell", for: indexPath) as! ProductCategoryCell
                cell.viewBackground.layer.borderWidth = 0.5
                 //Do some thing as per use
                return cell
            }

            let cell = collectionView.dequeueReusableCell(withReuseIdentifier: 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(withReuseIdentifier:
"ProductCategoryCell", for: indexPath) as! ProductCategoryCell
                  cell.viewBackground.layer.borderWidth = 0.5
                  //Do some thing as per use
                  return cell
            }

            let cell = collectionView.dequeueReusableCell(withReuseIdentifier: 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
        }

    }
```

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 UICollectionDataSource *>
```

Swift

在你的MyViewController.swift中添加以下内容

```
class MyViewController : UICollectionViewDelegate {
```

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 UICollectionDataSource *>
```

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
```

```
UICollectionView, didSelectItemAtIndexPath indexPath:
NSIndexPath)
{
}
```

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: CGRect(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: CGRect(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.

```
collecitonView.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]

collecitonView.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.obscuresBackgroundDuringPresentation	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. *(read-only)*
`UISearchController.searchResultsUpdater`	The object responsible for updating the contents of the search results controller.
`UISearchController.isActive`	The presented state of the search interface.
`UISearchController.obscuresBackgroundDuringPresentation`	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

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。
            返回 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: searchText)
            // 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]
```


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 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。
                返回 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: searchText)
                // 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 ?
```


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
    searchController.dimsBackgroundDuringPresentation = 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: UTableViewController, UISearchResultsUpdating {
    let searchController = UISearchController(searchResultsController: nil)
}
```

Add the search bar:

```
override func viewDidLoad() {
    super.viewDidLoad()

    searchController.searchResultsUpdater = self
    searchController.hidesNavigationBarDuringPresentation = false
    searchController.dimsBackgroundDuringPresentation = 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
}
- (void)searchBarSearchButtonClicked:(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 secondTabNavigationCntroller : UINavigationController!
    var thirdTabNavigationCntroller : UINavigationController!
    var fourthTabNavigationCntroller : UINavigationController!
    var fifthTabNavigationCntroller : 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 secondTabNavigationCntroller : UINavigationController!
    var thirdTabNavigationCntroller : UINavigationController!
    var fourthTabNavigationCntroller : UINavigationController!
    var fifthTabNavigationCntroller : 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())
secondTabNavigationCntroller = UINavigationController.init(rootViewController:
SecondViewController())
thirdTabNavigationController = UINavigationController.init(rootViewController:
ThirdViewController())
fourthTabNavigationCntroller = UINavigationController.init(rootViewController:
FourthViewController())
fifthTabNavigationController = UINavigationController.init(rootViewController:
FifthViewController()

tabBarController.viewControllers = [firstTabNavigationController,
secondTabNavigationCntroller, thirdTabNavigationController, fourthTabNavigationCntroller,
fifthTabNavigationController]


        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
        secondTabNavigationCntroller.tabBarItem = item2
        thirdTabNavigationController.tabBarItem = item3
        fourthTabNavigationCntroller.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.navigationCntroller?.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()

        firstTabNavigationCntroller = UINavigationCntroller.init(rootViewCntroller:
FirstViewCntroller())
        secondTabNavigationCntroller = UINavigationCntroller.init(rootViewCntroller:
SecondViewCntroller())
        thirdTabNavigationCntroller = UINavigationCntroller.init(rootViewCntroller:
ThirdViewCntroller())
        fourthTabNavigationCntroller = UINavigationCntroller.init(rootViewCntroller:
FourthViewCntroller())
        fifthTabNavigationCntroller = UINavigationCntroller.init(rootViewCntroller:
FifthViewCntroller())

        tabBarCntroller.viewCntrollers = [firstTabNavigationCntroller,
secondTabNavigationCntroller, thirdTabNavigationCntroller, fourthTabNavigationCntroller,
fifthTabNavigationCntroller]


        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)

        firstTabNavigationCntroller.tabBarItem = item1
        secondTabNavigationCntroller.tabBarItem = item2
        thirdTabNavigationCntroller.tabBarItem = item3
        fourthTabNavigationCntroller.tabBarItem = item4
        fifthTabNavigationCntroller.tabBarItem = item5

        UITabBar.appearance().tintColor = UIColor(red: 0/255.0, green: 146/255.0, blue: 248/255.0,
alpha: 1.0)

        self.window?.rootViewCntroller = tabBarCntroller

        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 TabbarCntroller: UITabBarCntroller {

    override func viewDidLoad() {
        super.viewDidLoad()

        self.navigationCntroller?.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 viewWillAppear(_ 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 viewWillAppear(_ 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*

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

In 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	描述
activityItems	包含执行活动的对象数组。该数组不能为空，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	Description
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

@objc
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

@objc
protocol ButtonEvent {
    @objc optional func 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，DetailVie wController 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 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 `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UIViewControllers并将它们添加到一个数组中。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 `UIViewControllers` 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 添加子视图: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`

iPad Pro - iPad Pro / iOS 9.3 (13E230)
Carrier
8:26 PM
100%
My VC at index 0
My VC at index 1
My VC at index 2
My VC at index 3
My VC at index 4

iPad Pro - iPad Pro / iOS 9.3 (13E230)
Carrier
8:26 PM
100%
My VC at index 0
My VC at index 1
My VC at index 2
My VC at index 3
My VC at index 4

iPad Pro - iPad Pro / iOS 9.3 (13E230)
Carrier
8:26 PM
100%
My VC at index 0
My VC at index 1
My VC at index 2
My VC at index 3
My VC at index 4

iPad Pro - iPad Pro / iOS 9.3 (13E230)
Carrier
8:26 PM
100%
My VC at index 0
My VC at index 1
My VC at index 2
My VC at index 3
My VC at index 4

关于横向布局

on Landscape Orientiation

iPad Pro - iPad Pro / iOS 9.3 (13E230)
Carrier
8:26 PM
100%
My VC at index 0
My VC at index 1
My VC at index 2
My VC at index 3
My VC at index 4

iPad Pro - iPad Pro / iOS 9.3 (13E230)
Carrier
8:26 PM
100%
My VC at index 0
My VC at index 1
My VC at index 2
My VC at index 3
My VC at index 4

第45章：UISlider

第45.1节：UISlider

Objective-C

在ViewController.h中或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];
[self.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, 宽度: 320, 高度: 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];
[self.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的实例

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UIPageViewController

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 = [(你的视图控制器基类 *)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 it's 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 Controler 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 Controler 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(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(arc4random_uniform(360))/360, saturation: 0.5,
```

```
brightness: 0.8, alpha: 1)
        }
        let storyboard = UIStoryboard(name: "Main", bundle: nil)
        let controller = storyboard.instantiateViewController(withIdentifier: "View
Controller")
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: "View
Controller")
        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 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`UIViewControllers`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 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 `UIViewControllers` 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];
}
[cell.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
名称	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:返回 标识符
        }
    }
}

if UIDevice.current.modelName == "iPhone 7" {
    // 是 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);
```



```
    // is 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
}
```

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(@"%.f",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];
}
-(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(@"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(@"%.f",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:(NSNotificationCenter *)notification
{
    if ([[UIDevice currentDevice] proximityState] == YES)
    {
NSLog(@"设备靠近用户。");
    }

    else
    {
NSLog(@"设备未靠近用户。");
    }
}
```

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 50.5: 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:(NSNotificationCenter *)notification
{
    if ([[UIDevice currentDevice] proximityState] == YES)
    {
        NSLog(@"Device is close to user.");
    }

    else
    {
        NSLog(@"Device is not closer to user.");
    }
}
```

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。以下是我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。

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。

Click on the Size Inspector and resize the view.
For this example we'll be using width 321 and height 256.

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。

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：

Add Auto-Layout constraints by clicking on "Resolve Auto Layout Issues" (bottom-right) and selecting "Add Missing Constraints" under "All Views".

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".

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”。

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".

在下一个窗口中，选择你的目标并点击“创建”。

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：

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 duplicate your custom Pokemon view.
Give it a different size, say 150x150.
Choose another Pokemon image, observe:

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:

iPhone 5 - iPhone 5 / iOS 9.3 (13E230)
Carrier
3:54 PM
Game Center
Extras
Watch
CustomUIV...
Safari

iPhone 5 - iPhone 5 / iOS 9.3 (13E230)
Carrier
3:54 PM
Game Center
Extras
Watch
CustomUIV...
Safari

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。

loadNibNamed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loadNibNamed。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 loadNibNamed 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 loadNibNamed 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 shap 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 = CGPoint(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 = CGRect(x: 100, y: 100, 宽度: 50, 高度: 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 = CGPoint(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 = CGRect(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` 比绘制到图层要慢，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（有趣且配有良好的视觉示例）
- 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) 通过圆角处理： UIRectCornerTopLeft 圆角半径： 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)
通过圆角处理： UIRectCornerTopRight 圆角半径： 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)
通过圆角处理： UIRectCornerBottomLeft 圆角半径： 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 cxtRef = 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(cxtRef, 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(cxtRef, 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 cxtRef = 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(cxtRef, 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(cxtRef, 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 = CGPoint(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 cxtRef = 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 = CGPoint(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 cxtRef = 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(cxtRef, rectPath.CGPath);

        // 3.
        [[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];
CGContextDrawPath(cxtRef, 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(cxtRef, rectPath.CGPath);

        // 3.
        [[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];
        CGContextDrawPath(cxtRef, 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.btnFolderPress),
forControlEvents: UIControlEvents.TouchUpInside)
    }

    @objc private func btnFolderPress() {
        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,", "Secound row,","Third row,","Fourth row"]

    override func viewDidLoad() {
        super.viewDidLoad()
        self.btnFolder.addTarget(self, action: #selector(CreateListVC.btnFolderPress),
forControlEvents: UIControlEvents.TouchUpInside)
    }

    @objc private func btnFolderPress() {
        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 `UIFeedbackGenerator` 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UICollectionView Layout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，我们有一个允许多选项选择的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UIAttachment Behavior，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 for 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 selectedIndexPaths: [IndexPath], from point: CGPoint)
    {
indexPathsForDraggingElements = selectedIndexPaths
        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] = selectedIndexPaths.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 selectedIndexPaths: [IndexPath], from point: CGPoint)
    {
        indexPathsForDraggingElements = selectedIndexPaths
        animator = UIDynamicAnimator(collectionViewLayout: self)
        animator?.delegate = self

        // Get all of the draggable attributes but change zIndex so above other cells
        let draggableAttributes: [UICollectionViewLayoutAttributes] = selectedIndexPaths.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(attributes.indexPath) ?? 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(attributes.indexPath) ?? 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:attributes.indexPath])
        {
            [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:attributes.indexPath])
        {
            [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(withReuseIdentifier: "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
    {
        return 1000
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) ->
UICollectionViewCell
    {
        let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "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.translatesAutoresizingMaskIntoConstraints = 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.translatesAutoresizingMaskIntoConstraints = 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 == UIGestureRecognizerStateEnded ||  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 == UIGestureRecognizerStateEnded ||  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

iOS - Collection View

iOS - Table View

macOS - Collection View

macOS - Table View

tvOS - Collection View

tvOS - Table View

iOS - Collection View

iOS - Table View

macOS - Collection View

macOS - Table View

tvOS - Collection View

tvOS - Table View

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
```


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
```

```
        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 添加子视图:_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
```

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 were 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 添加子视图:_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 () <UIViewControllerTransitioningDelegate>
@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:(UIViewControl
ler *)presented presentingController:(UIViewController *)presenting
sourceController:(UIViewController *)source
{
    return [[DropOutAnimator alloc]initWithDuration: 1.5 appearing:YES];
}

- (id<UIViewControllerAnimatedTransitioning>)animationControllerForDismissedController:(UIViewControl
ler *)dismissed
{
    return [[DropOutAnimator alloc] initWithDuration:4.0 appearing:NO];
}

@end
```

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 () <UIViewControllerTransitioningDelegate>
@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:(UIViewControl
ler *)presented presentingController:(UIViewController *)presenting
sourceController:(UIViewController *)source
{
    return [[DropOutAnimator alloc]initWithDuration: 1.5 appearing:YES];
}

-
(id<UIViewControllerAnimatedTransitioning>)animationControllerForDismissedController:(UIViewControl
ler *)dismissed
{
    return [[DropOutAnimator alloc] initWithDuration:4.0 appearing:NO];
}

@end
```

Here we create the view controller that is presented. Also because `ModalViewController` is it's 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: 时间间隔
    let isAppearing: 布尔值

    var transitionContext: UIViewControllerContextTransitioning?
    var hasElapsedTimeExceededDuration = false
    var finishTime: 时间间隔 = 0.0
    var collisionBehavior: UICollisionBehavior?
    var attachmentBehavior: UIAttachmentBehavior?
    var animator: UIDynamicAnimator?

    init(duration: 时间间隔 = 1.0,  isAppearing: 布尔值)
    {
        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 hasElapsedTimeExceededDuration = 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 refrence 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.hasElapsedTimeExceededDuration = 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 befor 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.hasElapsedTimeExceededDuration = 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 composit 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 bouncying 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 refrence 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 befor 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 bouncying 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。

```
            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.

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，来源：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

-
(UIViewControl
ler *)presented presentingController:(UIViewController *)presenting
sourceController:(UIViewController *)source
{
    return [[EmptyAnimator alloc] init];
}

-
(UIViewControl
ler *)dismissed
{
    return [[EmptyAnimator alloc] init];
}
{
id <
>)
:
(id<UIViewControlle
rAnimatedTransitioning>)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:(UIViewControl
ler *)presented presentingController:(UIViewController *)presenting
sourceController:(UIViewController *)source
{
    return [[EmptyAnimator alloc] init];
}

-
(id<UIViewControllerAnimatedTransitioning>)animationControllerForDismissedController:(UIViewControl
ler *)dismissed
{
    return [[EmptyAnimator alloc] init];
}

-
(id<UIViewControllerInteractiveTransitioning>)interactionControllerForPresentation:(id<UIViewContro
llerAnimatedTransitioning>)animator
{
    return self.presentingAnimator;
}
```

```
{
 id <                                                    >)__                                                    :(id<UIViewControllerAni
matedTransitioning>)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<UIViewControlle
rAnimatedTransitioning>)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 refrence 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
-
(void)setupViewForTransitionWithContext:(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 == UIGestureRecognizerStateEnded || 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 == UIGestureRecognizerStateEnded || 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 it's 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:
.touchUpInside)
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:
.touchUpInside)
        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 propogate
        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 heigh: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 refrence 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(宽度: 30.0, 高度: 30.0)
    let centerPoint = CGPoint(x: self.animationView.bounds.midX - (squareSize.width/2), y:
self.animationView.bounds.midY - (squareSize.height/2))
    let frame = CGRect(起点: centerPoint, 大小: 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.

Setps:

- 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

1. 在info.plist中将View controller-based status bar appearance设置为YES
2. 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

1. In info.plist set `View controller-based status bar appearance` to `YES`
2. 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 (not 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;

}
-(void)userDidEnterInformation:(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 执行Segue，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 overrided 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` is the name of the second view controller and `isDetailingEnabled` is a public variable in that view controller.

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 it's 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 == "YourCustomIdentifer"
    {
        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 == "YourCustomIdentifer"
    {
        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
secondViewController.callback = { text in self.label.text = text }
        }
    }

    // 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 *xibFileName = @"Keyboard"; // 不包含 xib 扩展名
    UIView *view = [[[NSBundle mainBundle] loadNibNamed:xibFileName 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 *xibFileName = @"Keyboard"; // xib extention not included
    UIView *view = [[[NSBundle mainBundle] loadNibNamed:xibFileName 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，我发现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: NSTimeInterval, 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: 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)

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 viewWillAppear(animated: Bool) {
        super.viewWillAppear(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: NSTimeInterval, 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: 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)

        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 viewWillAppear(animated: Bool) {
        super.viewWillAppear(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，完成：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：

1. 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)isConnntectedToNetwork
{
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)isConnntectedToNetwork
{
    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 a 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 `shouldGroupAccessibiltyChildren`:

```
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 deserve 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:|[leftSpac
e][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:|[leftSpac
e][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” ⌘ + ⌥ + = （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 ⌘ + ⌥ + = (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 veritcally 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 it's 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。

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 4: Most tricky give a bottom to UILabel from UIView .

Step 5: (Optional) Set constrain to UIButton

1. Leading 2) Bottom 3) Trailing 4) Fixed Height (From mainview)

CircleDemo
CircleDemo
Main.storyboard
Main.stor...rd (Base)
View Controller Scene
View Controller
View
Add Text Increase size
View Controller Scene
View Controller
Top Layout Guide
Bottom Layout...
View
View
Lbl1
Lbl2
Constraints
Add Text Incr...
Constraints
First Responder
Exit
Storyboard Entry Point
I am Label
I am Label
Add Text Increase size
View
Show
Frame Rectangle
20
X
280
Width
Arrange
Position View
Layout Margins
Default
Preserve Supervie
Follow Readable W
Constraints
The selected views have no constraints
time, explicit left, top, width, and
constraints will be generated for th
Content Hugging Priority
Horizontal
250
Vertical
250
Content Compression Resistance Pr
Horizontal
750
Vertical
750
Intrinsic Size
Default (System Defi
Button
Button - Intercepts touch ev
sends an action message to
object when it's tapped.
uibutt

输出：

CircleDemo
CircleDemo
Main.storyboard
Main.stor...rd (Base)
View Controller Scene
View Controller
View
Add Text Increase size
View Controller Scene
View Controller
Top Layout Guide
Bottom Layout...
View
View
Lbl1
Lbl2
Constraints
Add Text Incr...
Constraints
First Responder
Exit
Storyboard Entry Point
I am Label
I am Label
Add Text Increase size
View
Show
Frame Rectangle
20
X
280
Width
Arrange
Position View
Layout Margins
Default
Preserve Supervie
Follow Readable W
Constraints
The selected views have no constraints
time, explicit left, top, width, and
constraints will be generated for th
Content Hugging Priority
Horizontal
250
Vertical
250
Content Compression Resistance Pr
Horizontal
750
Vertical
750
Intrinsic Size
Default (System Defi
Button
Button - Intercepts touch ev
sends an action message to
object when it's tapped.
uibutt

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 in 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 out 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 contraints
    // 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：

1. 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)
```

2. 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];
}]
```

3. 更改约束的优先级。这比添加和移除约束的CPU消耗更低。
4. **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。
5. **使用不会干扰预期动画的约束。**
6. 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。
7. 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];
```

8. 重写 layoutSubviews。调用 [super layoutSubviews] 并微调约束。
9. 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:

1. **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)
```

2. **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];
}]
```

3. **Change the priority of the constraints**. This is less CPU intensive than adding and removing constraints.
4. **Remove all constraints and use autosizing masks**. For the later, you have to set `view.translatesAutoresizingMaskIntoConstraints = YES`.
5. **Use constraints that don't interfere with the intended animation**.
6. **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.
7. **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];
```

8. **Override layoutSubviews**. Call `[super layoutSubviews]` and fine tune the constraints.
9. **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: Contraints 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 contraints
    // 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 urlTeplate = "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*.

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 urlTeplate = "http://tile.openstreetmap.org/{z}/{x}/{y}.png"
```

```
let overlay = MKTileOverlay(urlTemplate: urlTeplate)
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: urlTeplate)
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：

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)
```



```
}
```

This will return the correct `MKOverlayRenderer` to your `mapView`. If you run your project now, you should see a map like this:

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 == -180.00000000{
            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. :-)

     - author: 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 == -180.00000000{
            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];

}
```

第68.8节：添加MKMapView

Swift


```
        }
    }
}
```

(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];

}
```

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
```

CUBACEL
11:35 AM
100%
NORTH AMERICA
Toronto
New York
os Angeles
North Atlanti Ocean
Mexico City
Bogotá
Quito
SOUTH AMERICA
Lima
La Paz
Brasília
Pacific ean
Asunción
Rio de Jar
Santiago
Buenos Aires
Legal

CUBACEL
11:35 AM
100%
NORTH AMERICA
Toronto
New York
os Angeles
North Atlanti Ocean
Mexico City
Bogotá
Quito
SOUTH AMERICA
Lima
La Paz
Brasília
Pacific ean
Asunción
Rio de Jar
Santiago
Buenos Aires
Legal

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 dequed pin if available
        MKAnnotationView *pinView = [mapView
dequeueReusableAnnotationViewWithIdentifier:@"PinAnnotationView"];

        if (!pinView)
        {
            // If not dequed, 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..<string.length))
```

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:
[NSFontAttributeName: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..<string.length))
```

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:
[NSFontAttributeName: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NSAttributed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:NSUnicodeStringEncoding]
                                        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 NSAttributed string and vice verse

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:NSUnicodeStringEncoding]
                                        options: @{ NSDocumentTypeDocumentAttribute:
NSHTMLTextDocumentType }
                                        documentAttributes: nil
                                        error: nil ];
```

NSAttributedString to HTML Conversion:

```
//Dictionary to hold all the attributes of NSAttributed String
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 beforing 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 false, fire the timer only once. If true, fire the timer every interval 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 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	示例1	示例2
y	至少包含1位数字的年份。	公元175年 → “175”	公元2016年 → “2016”
yy	恰好包含2位数字的年份。	公元5年 → “05”	公元2016年 → “16”
yyy	至少包含3位数字的一年。	公元5年 → “005”	公元2016年 → “2016”
yyyy	至少包含4位数字的年份。	公元5年 → “0005”	公元2016年 → “2016”
M	至少包含1位数字的月份。	7月 → “7”	“十一月” → “11”
MM	至少包含2位数字的月份。	7月 → “07”	“十一月” → “11”
MMM	三字母月份缩写。	七月 → “Jul”	“十一月” → “Nov”
MMMM	月份全称。	七月 → “July”	“十一月” → “November”
MMMMM	单字母月份缩写（1月、6月、7月均为‘J’）。	七月 → “J”	“十一月” → “N”
d	至少包含一位数字的日期。	8 → “8”	29 → “29”
dd	至少包含两位数字的日期。	8 → “08”	29 → “29”
“E”, “EE”, 或”EEE”	日期名称的三个字母缩写。	星期一 → “Mon”	星期四 → “Thu”
EEEE	完整的星期名称。	星期一 → “星期一”	星期四 → “星期四”
EEEEE	星期名称的1个字母缩写。（星期四和星期二都用'T'表示）	星期一 → “M”	星期四 → “T”
EEEEEE	星期名称的2个字母缩写。	星期一 → “Mo”	星期四 → “Th”
a	一天中的时段（上午/下午）。	晚上10点 → “PM”	上午2点 → “上午”
h	基于1-12的小时，至少包含1位数字。	晚上10点 → “10”	凌晨2点 → “2”
hh	基于1-12的小时，至少包含2位数字。	晚上10点 → “10”	凌晨2点 → “02”
H	基于0-23的小时，至少包含1位数字。	晚上10点 → “14”	凌晨2点 → “2”
HH	基于0-23的小时，至少包含2位数字。	晚上10点 → “14”	凌晨2点 → “02”
m	至少包含1位数字的分钟。	7 → “7”	29 → “29”
毫米	至少包含2位数字的分钟。	7 → “07”	29 → “29”
秒	至少包含1位数字的秒。	7 → “7”	29 → “29”
秒	至少包含2位数字的秒。	7 → “07”	29 → “29”

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”
EEEEE	1 letter day abbreviation of day name.(Thu and Tue will be 'T')	Monday → “M”	Thursday → “T”
EEEEEE	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 swiftly-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 swiftly-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 then 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(@"%llu", 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(@"%llu",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:[NSLo
cale 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`dateWithTimerIntervalSinceNow(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().dateWithTimerIntervalSinceNow(totalSecondsInWeek)

//使用正值表示从今天起的未来日期
let lastWeek = NSDate().dateWithTimerIntervalSinceNow(-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)getUTCOffSetIntervalWithCurrentTimeZone:(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 `dateWithTimerIntervalSinceNow(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().dateWithTimerIntervalSinceNow(totalSecondsInWeek)

//Using positive value for future date from today
let lastWeek = NSDate().dateWithTimerIntervalSinceNow(-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)getUTCOffSetIntervalWithCurrentTimeZone:(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（复制件）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 notifications are a simple and powerful way to send data in a loosely coupled way. That is, the sender of a notification doesn't have to care about who (if anyone) receives the notification, it just posts it out there to the rest of the app and it could be picked up by lots of things or nothing depending on your app's state.

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",
                                                 对象：nil)
```

Swift 3

```
NSNotificationCenter.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 = NSNotificationCenter.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
NSNotificationCenter.defaultCenter().removeObserver(observer)
```

第74.5节：发送通知

Swift

```
NSNotificationCenter.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

```
NSNotificationCenter.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 = NSNotificationCenter.defaultCenter().addObserverForName("nameOfTheNotification",
object: nil, queue: nil) { (notification) in
    // Do operations with the notification in this block
}

// Remove observer
NSNotificationCenter.defaultCenter().removeObserver(observer)
```

Section 74.5: Posting a Notification

Swift

```
NSNotificationCenter.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 = NSURLSession(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）/URLSessionDidFinish EventsForBackgroundURLSession（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 = NSURLSession(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 NSURLSession 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 cacheing
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 NSURLSession
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 NSURLSession'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
completionHandler(NSURLSessionResponseAllow);
}

- (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
    completionHandler(NSURLSessionResponseAllow);
}

- (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(NSURLSessionAuthChallengeUseCredential,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(NSURLSessionAuthChallengeUseCredential,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 = @"--------------------------147378098314664998827466414 49";

// 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(NSURLSessionAuthChallengeUseCredential,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(NSURLSessionAuthChallengeUseCredential,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 = @"--------------------------147378098314664998827466414 49";

// 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]dataUsingEncod
ing: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_n
ame\"\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 ap
pendData:[[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
* appendDataString = [NSString stringWithFormat:@"Content-Disposition: form-data;name=\"userfile\"; filenam
e=\"%@.jpg\"\r", picFileName];
[body appendData:[appendDataString dataUsingEncoding:NSUTF8StringEncoding]];[body appe
ndData:[@"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]dataUsingEncodi
ng: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 * appendDataString = [NSString stringWithFormat:@"Content-Disposition: form-data;
name=\"userfile\"; filename=\"%@.jpg\"\r\n", picFileName];
[body appendData:[appendDataString 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 cacheing
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.

第76章：NSUserDefaults

第76.1节：设置值

要在NSUserDefaults中设置值，可以使用以下函数：

Swift 3之前版本

```
setBool(_:forKey:)
setFloat(_:forKey:)
setInteger(_:forKey:)
setObject(_:forKey:)
setDouble(_:forKey:)
setURL(_:forKey:)
```

Swift 3

在Swift 3中，函数名称改为set，后面不再跟类型。

```
set(_:forKey:)
```

Objective-C

```
-(void)setBool:(BOOL)value forKey:(nonnull NSString *)defaultName;
-(void)setFloat:(float)value forKey:(nonnull NSString *)defaultName;
-(void)setInteger:(NSInteger)value forKey:(nonnull NSString *)defaultName;
-(void)setObject:(nullable id)value forKey:(nonnull NSString *)defaultName;
-(void)setDouble:(double)value forKey:(nonnull NSString *)defaultName;
-(void)setURL:(nullable NSURL *)value forKey:(nonnull NSString *)defaultName;
```

示例用法如下：

Swift 3之前版本

```
NSUserDefaults.standardUserDefaults.setObject("Netherlands", forKey: "HomeCountry")
```

Swift 3

```
UserDefaults.standard.set("Netherlands", forKey: "HomeCountry")
```

Objective-C

```
[[NSUserDefaults standardUserDefaults] setObject:@"Netherlands" forKey:@"HomeCountry"];
```

自定义对象

要将自定义对象保存到 `NSUserDefaults`，您需要让您的 CustomClass 遵循 `NSCoding` 协议。您需要实现以下方法：Swift

```
public func encodeWithCoder(aCoder: NSCoder) {
    aCoder.encodeObject(name, forKey:"name")
    aCoder.encodeObject(unitId, forKey: "unitId")
}

required public init(coder aDecoder: NSCoder) {
    super.init()
name = aDecoder.decodeObjectForKey("name") as? String
    unitId = aDecoder.decodeIntegerForKey("unitId") as? NSInteger
}
```


Chapter 76: NSUserDefaults

Section 76.1: Setting values

To set a value in `NSUserDefaults`, you can use the following functions:

Swift < 3

```
setBool(_:forKey:)
setFloat(_:forKey:)
setInteger(_:forKey:)
setObject(_:forKey:)
setDouble(_:forKey:)
setURL(_:forKey:)
```

Swift 3

In Swift 3 the names of function is changed to `set` insted of `set` folloed by the type.

```
set(_:forKey:)
```

Objective-C

```
-(void)setBool:(BOOL)value forKey:(nonnull NSString *)defaultName;
-(void)setFloat:(float)value forKey:(nonnull NSString *)defaultName;
-(void)setInteger:(NSInteger)value forKey:(nonnull NSString *)defaultName;
-(void)setObject:(nullable id)value forKey:(nonnull NSString *)defaultName;
-(void)setDouble:(double)value forKey:(nonnull NSString *)defaultName;
-(void)setURL:(nullable NSURL *)value forKey:(nonnull NSString *)defaultName;
```

Example usage would be:

Swift < 3

```
NSUserDefaults.standardUserDefaults.setObject("Netherlands", forKey: "HomeCountry")
```

Swift 3

```
UserDefaults.standard.set("Netherlands", forKey: "HomeCountry")
```

Objective-C

```
[[NSUserDefaults standardUserDefaults] setObject:@"Netherlands" forKey:@"HomeCountry"];
```

Custom objects

To save custom objects into the `NSUserDefaults` you need to make your CustomClass confirm to protocol of `NSCoding`. You need to implement the following methods: **Swift**

```
public func encodeWithCoder(aCoder: NSCoder) {
    aCoder.encodeObject(name, forKey:"name")
    aCoder.encodeObject(unitId, forKey: "unitId")
}

required public init(coder aDecoder: NSCoder) {
    super.init()
    name = aDecoder.decodeObjectForKey("name") as? String
    unitId = aDecoder.decodeIntegerForKey("unitId") as? NSInteger
}
```

Objective-C

```
- (id)initWithCoder:(NSCoder *)coder {
    self = [super init];
    if (self) {
name = [coder decodeObjectForKey:@"name"];
        unitId = [coder decodeIntegerForKey:@"unitId"];
    }
    return self;
}

- (void)encodeWithCoder:(NSCoder*)coder {
    [coder encodeObject:name forKey:@"name"];
    [coder encodeInteger:unitId forKey:@"unitId"];
}
```

第76.2节：Swift 3中UserDefaults的使用

每个应用程序都需要在UserDefaults中存储用户会话或用户相关的详细信息。因此，我们将整个逻辑封装在一个类中，以更好地管理UserDefaults。

Swift 3

```
import Foundation

public struct Session {

fileprivate static let defaults = UserDefaults.standard

    enum userValues: String {
        case auth_token
        case email
        case fname
        case mobile
        case title
        case userId
        case userType
        case OTP
        case isApproved
    }


    //MARK: - 获取用户详细信息
    static func getUserSessionDetails()->[String:AnyObject]? {
        let dictionary = defaults.object(forKey: "LoginSession") as? [String:AnyObject]
        return  dictionary
    }

    //MARK: - 保存设备令牌
    static func saveDeviceToken(_ token:String){
       guard (gettingDeviceToken() ?? "").isEmpty else {
            return
        }
defaults.removeObject(forKey: "deviceToken")
        defaults.set(token, forKey: "deviceToken")
        defaults.synchronize()
    }

    //MARK: - 获取令牌
    static func gettingDeviceToken()->String?{
        let token = defaults.object(forKey: "deviceToken") as? String
```


Objective-C

```
- (id)initWithCoder:(NSCoder *)coder {
    self = [super init];
    if (self) {
        name = [coder decodeObjectForKey:@"name"];
        unitId = [coder decodeIntegerForKey:@"unitId"];
    }
    return self;
}

- (void)encodeWithCoder:(NSCoder*)coder {
    [coder encodeObject:name forKey:@"name"];
    [coder encodeInteger:unitId forKey:@"unitId"];
}
```

Section 76.2: UserDefaults uses in Swift 3

Every application needed to store User Session or User related details inside application in UserDefaults.So we made whole logic inside a Class for managing UserDefaults better way.

Swift 3

```
import Foundation

public struct Session {

    fileprivate static let defaults = UserDefaults.standard

    enum userValues: String {
        case auth_token
        case email
        case fname
        case mobile
        case title
        case userId
        case userType
        case OTP
        case isApproved
    }


    //MARK: - Getting here User Details
    static func getUserSessionDetails()->[String:AnyObject]? {
        let dictionary = defaults.object(forKey: "LoginSession") as? [String:AnyObject]
        return  dictionary
    }

    //MARK: - Saving Device Token
    static func saveDeviceToken(_ token:String){
        guard (gettingDeviceToken() ?? "").isEmpty else {
            return
        }
        defaults.removeObject(forKey: "deviceToken")
        defaults.set(token, forKey: "deviceToken")
        defaults.synchronize()
    }

    //MARK: - Getting Token here
    static func gettingDeviceToken()->String?{
        let token = defaults.object(forKey: "deviceToken") as? String
```

```
        if token == nil{
            return ""
        }else{ return token}
    }

    //MARK: - 设置用户详情
    static func setUserSessionDetails(_ dic :[String : AnyObject]){
        defaults.removeObject(forKey: "LoginSession")
        defaults.set(dic, forKey: "LoginSession")
        defaults.synchronize()
    }

    //MARK: - 移除所有默认值
    static func userSessionLogout(){
        //设置活动
defaults.removeObject(forKey: "LoginSession")
        defaults.synchronize()
    }

    //MARK: - 在此处从会话获取值
    static func getUserValues(value: userValues) -> String? {
        let dic = getUserSessionDetails() ?? [:]
guard let value = dic[value.rawValue] else{
            return ""
        }
        return value as? String
    }

}
```

UserDefaults 类的使用

```
//保存用户详情
Session.setUserSessionDetails(json ?? [:])

//获取用户详情
let userId = Session.getUserValues(value: .userId) ?? ""
```

第76.3节：使用管理器保存和读取数据

虽然你可以在任何地方使用NSUserDefaults方法，但有时定义一个管理器来为你保存和读取NSUserDefaults会更好，然后使用该管理器来读取或写入数据。

假设我们想将用户的分数保存到NSUserDefaults。我们可以创建一个如下的类，它至少有两个方法：setHighScore和高分。无论何时想访问高分，都创建该类的一个实例。

Swift

```
public class ScoreManager: NSObject {

    let highScoreDefaultKey = "HighScoreDefaultKey"

    var highScore = {
        set {
            // 此方法包含保存最高分的实现
            // 你可以使用 NSUserDefaults 或其他数据存储方式，如 CoreData 或
            // SQLite 等。

            NSUserDefaults.standardUserDefaults().setInteger(newValue, forKey: highScoreDefaultKey)
```



```
        if token == nil{
            return ""
        }else{ return token}
    }

    //MARK: - Setting here User Details
    static func setUserSessionDetails(_ dic :[String : AnyObject]){
        defaults.removeObject(forKey: "LoginSession")
        defaults.set(dic, forKey: "LoginSession")
        defaults.synchronize()
    }

    //MARK: Removing here all Default Values
    static func userSessionLogout(){
        //Set Activity
        defaults.removeObject(forKey: "LoginSession")
        defaults.synchronize()
    }

    //MARK: - Get value from session here
    static func getUserValues(value: userValues) -> String? {
        let dic = getUserSessionDetails() ?? [:]
        guard let value = dic[value.rawValue] else{
            return ""
        }
        return value as? String
    }

}
```

Use of UserDefaults Class

```
//Saving user Details
Session.setUserSessionDetails(json ?? [:])

//Retriving user Details
let userId = Session.getUserValues(value: .userId) ?? ""
```

Section 76.3: Use Managers to Save and Read Data

While you can use the `NSUserDefaults` methods anywhere, it can sometimes be better to define a manager that saves and reads from `NSUserDefaults` for you and then use that manager for reading or writing your data.

Suppose that we want to save a user's score into `NSUserDefaults`. We can create a class like the one below that has at two methods: `setHighScore` and `highScore`. Anywhere you want to access the high scores, create an instance of this class.

Swift

```
public class ScoreManager: NSObject {

    let highScoreDefaultKey = "HighScoreDefaultKey"

    var highScore = {
        set {
            // This method includes your implementation for saving the high score
            // You can use NSUserDefaults or any other data store like CoreData or
            // SQLite etc.

            NSUserDefaults.standardUserDefaults().setInteger(newValue, forKey: highScoreDefaultKey)
```

```swift
            NSUserDefaults.standardUserDefaults().synchronize()
        }
        get {
        // 此方法包含读取最高分的实现

            let score = NSUserDefaults.standardUserDefaults().objectForKey(highScoreDefaultKey)

            if (score != nil) {
                return score.integerValue;
            } else {
                //没有可用的高分，所以返回 -1
                return -1;
            }
        }
    }
}
```

Objective-C

```objc
#import "ScoreManager.h"

#define HIGHSCRORE_KEY @"highScore"

@implementation ScoreManager

- (void)setHighScore:(NSUInteger) highScore {
    // 此方法包含保存高分的实现
    // 你可以使用 NSUserDefaults 或其他数据存储方式，如 CoreData 或
    // SQLite 等。

    [[NSUserDefaults standardUserDefaults] setInteger:highScore forKey:HIGHSCRORE_KEY];
    [[NSUserDefaults standardUserDefaults] synchronize];
}

- (NSInteger)highScore
{

    // 此方法包含读取最高分的实现

NSNumber *highScore = [[NSUserDefaults standardUserDefaults] objectForKey:HIGHSCRORE_KEY];
    if (highScore) {
        return highScore.integerValue;
    }else
    {
        //没有可用的高分，因此返回-1

        return -1;
    }

}

@end
```

优点如下：

1. 你的读写过程的实现只在一个地方，可以随时更改（例如从NSUserDefaults切换到Core Data），而不必担心更改所有使用高分的地方。

2. 只需调用一个方法即可访问或写入分数。


```swift
            NSUserDefaults.standardUserDefaults().synchronize()
        }
        get {
        //This method includes your implementation for reading the high score

            let score = NSUserDefaults.standardUserDefaults().objectForKey(highScoreDefaultKey)

            if (score != nil) {
                return score.integerValue;
            } else {
                //No high score available, so return -1
                return -1;
            }
        }
    }
}
```

Objective-C

```objc
#import "ScoreManager.h"

#define HIGHSCRORE_KEY @"highScore"

@implementation ScoreManager

- (void)setHighScore:(NSUInteger) highScore {
    // This method includes your implementation for saving the high score
    // You can use NSUserDefaults or any other data store like CoreData or
    // SQLite etc.

    [[NSUserDefaults standardUserDefaults] setInteger:highScore forKey:HIGHSCRORE_KEY];
    [[NSUserDefaults standardUserDefaults] synchronize];
}

- (NSInteger)highScore
{

    //This method includes your implementation for reading the high score

    NSNumber *highScore = [[NSUserDefaults standardUserDefaults] objectForKey:HIGHSCRORE_KEY];
    if (highScore) {
        return highScore.integerValue;
    }else
    {
        //No high score available, so return -1

        return -1;
    }

}

@end
```

The advantages are that:

1. The implementation of your read and write process is only in one place and you can change it (for example switch from `NSUserDefaults` to Core Data) whenever you want and not worry about changing all places that you are working with the high score.

2. Simply call only one method when you want to access to score or write it.

3. 当你发现错误或类似问题时，调试也很简单。

> **注意**
>
> 如果你担心同步问题，最好使用一个单例类来管理同步。

第76.4节：保存数值

NSUserDefaults 由系统定期写入磁盘，但有时你希望立即保存更改，例如当应用程序进入后台状态时。这可以通过调用 synchronize 来完成。

Swift

```
NSUserDefaults.standardUserDefaults().synchronize()
```

Objective-C

```
[[NSUserDefaults standardUserDefaults] synchronize];
```

第76.5节：清除NSUserDefaults

Swift

```
let bundleIdentifier = NSBundle.mainBundle().bundleIdentifier()

NSUserDefaults.standardUserDefaults().removePersistentDomainForName(bundleIdentifier)
```

Objective-C

```
NSString *bundleIdentifier = [[NSBundle mainBundle] bundleIdentifier];

[[NSUserDefaults standardUserDefaults] removePersistentDomainForName: bundleIdentifier];
```

第76.6节：获取默认值

要在NSUserDefaults中获取值，可以使用以下函数：

Swift

```
arrayForKey(_:)
boolForKey(_:)
dataForKey(_:)
dictionaryForKey(_:)
floatForKey(_:)
integerForKey(_:)
objectForKey(_:)
stringArrayForKey(_:)
stringForKey(_:)
doubleForKey(_:)
URLForKey(_:)
```

Objective-C

```
-(nullable NSArray *)arrayForKey:(nonnull NSString *)defaultName;
-(BOOL)boolForKey:(nonnull NSString *)defaultName;
-(nullable NSData *)dataForKey:(nonnull NSString *)defaultName;
-(nullable NSDictionary<NSString *, id> *)dictionaryForKey:(nonnull NSString *)defaultName;
```


3. Simply debug it when you see a bug or something like this.

> **Note**
>
> If you are worried about synchronization, it is better to use a singleton class that manages the synchronization.

Section 76.4: Saving Values

`NSUserDefaults` are written to disk periodically by the system, but there are times when you want your changes saved immediately, such as when the app transitions into background state. This is done by calling `synchronize`.

Swift

```
NSUserDefaults.standardUserDefaults().synchronize()
```

Objective-C

```
[[NSUserDefaults standardUserDefaults] synchronize];
```

Section 76.5: Clearing NSUserDefaults

Swift

```
let bundleIdentifier = NSBundle.mainBundle().bundleIdentifier()

NSUserDefaults.standardUserDefaults().removePersistentDomainForName(bundleIdentifier)
```

Objective-C

```
NSString *bundleIdentifier = [[NSBundle mainBundle] bundleIdentifier];

[[NSUserDefaults standardUserDefaults] removePersistentDomainForName: bundleIdentifier];
```

Section 76.6: Getting Default Values

To get a value in NSUserDefaults you can use the following functions:

Swift

```
arrayForKey(_:)
boolForKey(_:)
dataForKey(_:)
dictionaryForKey(_:)
floatForKey(_:)
integerForKey(_:)
objectForKey(_:)
stringArrayForKey(_:)
stringForKey(_:)
doubleForKey(_:)
URLForKey(_:)
```

Objective-C

```
-(nullable NSArray *)arrayForKey:(nonnull NSString *)defaultName;
-(BOOL)boolForKey:(nonnull NSString *)defaultName;
-(nullable NSData *)dataForKey:(nonnull NSString *)defaultName;
-(nullable NSDictionary<NSString *, id> *)dictionaryForKey:(nonnull NSString *)defaultName;
```

```
-(float)floatForKey:(nonnull NSString *)defaultName;
-(NSInteger)integerForKey:(nonnull NSString *)defaultName;
-(nullable id)objectForKey:(nonnull NSString *)key;
-(nullable NSArray<NSString *> *)stringArrayForKey:(nonnull NSString *)defaultName;
-(nullable NSString *)stringForKey:(nonnull NSString *)defaultName;
-(double)doubleForKey:(nonnull NSString *)defaultName;
-(nullable NSURL *)URLForKey:(nonnull NSString *)defaultName;
```

示例用法如下：

Swift

```
let homeCountry = NSUserDefaults.standardUserDefaults().stringForKey("HomeCountry")
```

Objective-C

```
NSString *homeCountry = [[NSUserDefaults standardUserDefaults] stringForKey:@"HomeCountry"];
```



```
-(float)floatForKey:(nonnull NSString *)defaultName;
-(NSInteger)integerForKey:(nonnull NSString *)defaultName;
-(nullable id)objectForKey:(nonnull NSString *)key;
-(nullable NSArray<NSString *> *)stringArrayForKey:(nonnull NSString *)defaultName;
-(nullable NSString *)stringForKey:(nonnull NSString *)defaultName;
-(double)doubleForKey:(nonnull NSString *)defaultName;
-(nullable NSURL *)URLForKey:(nonnull NSString *)defaultName;
```

Example usage would be:

Swift

```
let homeCountry = NSUserDefaults.standardUserDefaults().stringForKey("HomeCountry")
```

Objective-C

```
NSString *homeCountry = [[NSUserDefaults standardUserDefaults] stringForKey:@"HomeCountry"];
```

第77章：NSHTTPCookieStorage

第77.1节：从NSUserDefault存储和读取Cookie

```
import Foundation


class CookiesSingleton {

static let instance : CookiesSingleton = CookiesSingleton()
static var enableDebug = true

func loadCookies() {
    if let cookiesDetails = NSUserDefaults.standardUserDefaults().objectForKey("customeWebsite")  {
        for (keys,_) in cookiesDetails as! NSDictionary{
                if let  cookieDict = NSUserDefaults.standardUserDefaults().objectForKey(keys as!
字符串){
                    if let cookie = NSHTTPCookie(properties:cookieDict as! [String:AnyObject]) {
                        NSHTTPCookieStorage.sharedHTTPCookieStorage().setCookie(cookie)
                        if(CookiesSingleton.enableDebug){
                            print("Each Cookies",cookieDict)
                        }
                    }
                }
        }
    }
}

func  removeCookies(){
    NSURLCache.sharedURLCache().removeAllCachedResponses()
    NSURLCache.sharedURLCache().diskCapacity = 0
    NSURLCache.sharedURLCache().memoryCapacity = 0

    let storage : NSHTTPCookieStorage = NSHTTPCookieStorage.sharedHTTPCookieStorage()
    for cookie in storage.cookies! {
storage.deleteCookie(cookie as NSHTTPCookie)
    }

    NSUserDefaults.standardUserDefaults().setValue("", forKey: "customeWebsite")
    NSUserDefaults.standardUserDefaults().synchronize()

    if(CookiesSingleton.enableDebug){
        print("Cookies Removed")
    }
}

func saveCookies() {

    let cookieArray = NSMutableArray()
    let savedC = NSHTTPCookieStorage.sharedHTTPCookieStorage().cookies

    let allCookiesDic:NSMutableDictionary = NSMutableDictionary()

    for c : NSHTTPCookie in savedC! {

        let cookieProps = NSMutableDictionary()
        cookieArray.addObject(c.name)
        cookieProps.setValue(c.name, forKey: NSHTTPCookieName)
        cookieProps.setValue(c.value, forKey: NSHTTPCookieValue)
        cookieProps.setValue(c.domain, forKey: NSHTTPCookieDomain)
```


Chapter 77: NSHTTPCookieStorage

Section 77.1: Store and read the cookies from NSUserDefault

```
import Foundation


class CookiesSingleton {

static let instance : CookiesSingleton = CookiesSingleton()
static var enableDebug = true

func loadCookies() {
    if let cookiesDetails = NSUserDefaults.standardUserDefaults().objectForKey("customeWebsite")  {
        for (keys,_) in cookiesDetails as! NSDictionary{
                if let  cookieDict = NSUserDefaults.standardUserDefaults().objectForKey(keys as!
String){
                    if let cookie = NSHTTPCookie(properties:cookieDict as! [String:AnyObject]) {
                        NSHTTPCookieStorage.sharedHTTPCookieStorage().setCookie(cookie)
                        if(CookiesSingleton.enableDebug){
                            print("Each Cookies",cookieDict)
                        }
                    }
                }
        }
    }
}

func  removeCookies(){
    NSURLCache.sharedURLCache().removeAllCachedResponses()
    NSURLCache.sharedURLCache().diskCapacity = 0
    NSURLCache.sharedURLCache().memoryCapacity = 0

    let storage : NSHTTPCookieStorage = NSHTTPCookieStorage.sharedHTTPCookieStorage()
    for cookie in storage.cookies! {
        storage.deleteCookie(cookie as NSHTTPCookie)
    }

    NSUserDefaults.standardUserDefaults().setValue("", forKey: "customeWebsite")
    NSUserDefaults.standardUserDefaults().synchronize()

    if(CookiesSingleton.enableDebug){
        print("Cookies Removed")
    }
}

func saveCookies() {

    let cookieArray = NSMutableArray()
    let savedC = NSHTTPCookieStorage.sharedHTTPCookieStorage().cookies

    let allCookiesDic:NSMutableDictionary = NSMutableDictionary()

    for c : NSHTTPCookie in savedC! {

        let cookieProps = NSMutableDictionary()
        cookieArray.addObject(c.name)
        cookieProps.setValue(c.name, forKey: NSHTTPCookieName)
        cookieProps.setValue(c.value, forKey: NSHTTPCookieValue)
        cookieProps.setValue(c.domain, forKey: NSHTTPCookieDomain)
```

```
cookieProps.setValue(c.path, forKey: NSHTTPCookiePath)
        cookieProps.setValue(c.version, forKey: NSHTTPCookieVersion)
        cookieProps.setValue(NSDate().dateByAddingTimeInterval(2629743), forKey:
NSHTTPCookieExpires)

allCookiesDic.setValue(cookieProps, forKey: c.name)

    }
    NSUserDefaults.standardUserDefaults().setValue(allCookiesDic, forKey: "customeWebsite")
    NSUserDefaults.standardUserDefaults().synchronize()

    if(CookiesSingleton.enableDebug){
        print("Cookies Saved")
    }
}

}
```



```
        cookieProps.setValue(c.path, forKey: NSHTTPCookiePath)
        cookieProps.setValue(c.version, forKey: NSHTTPCookieVersion)
        cookieProps.setValue(NSDate().dateByAddingTimeInterval(2629743), forKey:
NSHTTPCookieExpires)

        allCookiesDic.setValue(cookieProps, forKey: c.name)

    }
    NSUserDefaults.standardUserDefaults().setValue(allCookiesDic, forKey: "customeWebsite")
    NSUserDefaults.standardUserDefaults().synchronize()

    if(CookiesSingleton.enableDebug){
        print("Cookies Saved")
    }
}

}
```

第78章：NSURLConnection

第78.1节：代理方法

//符合NSURLConnectionDelegate协议。

```
@interface ViewController : UIViewController<NSURLConnectionDelegate>
{
    NSMutableData *_responseData;
}
```

// NSURLConnection 协议方法的实现。

```
#pragma mark NSURLConnection 代理方法

- (void)connection:(NSURLConnection *)connection didReceiveResponse:(NSURLResponse *)response {
    // 已收到响应，这里初始化你创建的实例变量
    // 以便在 didReceiveData 方法中追加数据
    // 此外，每次发生重定向时都会调用此方法，因此重新初始化
    // 也起到清空的作用
_responseData = [[NSMutableData alloc] init];
}

- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data {
    // 将新数据追加到你声明的实例变量中
    [_responseData appendData:data];
}

- (NSCachedURLResponse *)connection:(NSURLConnection *)connection
                willCacheResponse:(NSCachedURLResponse*)cachedResponse {
    // 返回 nil 表示不需要为此连接存储缓存响应
    return nil;
}

- (void)connectionDidFinishLoading:(NSURLConnection *)connection {
    // 请求已完成，数据已接收
    // 你现在可以解析实例变量中的内容了

}

- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError *)error {
    // 请求因某种原因失败了！
    // 检查 error 变量
}
```

第78.2节：同步请求

```
NSURLRequest * urlRequest = [NSURLRequest requestWithURL:[NSURL
URLWithString:@"http://google.com"]];
NSURLResponse * response = nil;
NSError * error = nil;
NSData * data = [NSURLConnection sendSynchronousRequest:urlRequest
                                returningResponse:&response
                                            error:&error];

if (error == nil)
{
    // 在这里解析数据
```


Chapter 78: NSURLConnection

Section 78.1: Delegate methods

//conform the NSURLConnectionDelegate protocol.

```
@interface ViewController : UIViewController<NSURLConnectionDelegate>
{
    NSMutableData *_responseData;
}
```

//Implementation of the NSURLConnection protocol methods.

```
#pragma mark NSURLConnection Delegate Methods

- (void)connection:(NSURLConnection *)connection didReceiveResponse:(NSURLResponse *)response {
    // A response has been received, this is where we initialize the instance var you created
    // so that we can append data to it in the didReceiveData method
    // Furthermore, this method is called each time there is a redirect so reinitializing it
    // also serves to clear it
    _responseData = [[NSMutableData alloc] init];
}

- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data {
    // Append the new data to the instance variable you declared
    [_responseData appendData:data];
}

- (NSCachedURLResponse *)connection:(NSURLConnection *)connection
                willCacheResponse:(NSCachedURLResponse*)cachedResponse {
    // Return nil to indicate not necessary to store a cached response for this connection
    return nil;
}

- (void)connectionDidFinishLoading:(NSURLConnection *)connection {
    // The request is complete and data has been received
    // You can parse the stuff in your instance variable now

}

- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError *)error {
    // The request has failed for some reason!
    // Check the error var
}
```

Section 78.2: Synchronous Request

```
NSURLRequest * urlRequest = [NSURLRequest requestWithURL:[NSURL
URLWithString:@"http://google.com"]];
NSURLResponse * response = nil;
NSError * error = nil;
NSData * data = [NSURLConnection sendSynchronousRequest:urlRequest
                                returningResponse:&response
                                            error:&error];

if (error == nil)
{
    // Parse data here
```

```
}
```

第78.3节：异步请求

```
// 创建请求实例。
NSURLRequest *request = [NSURLRequest requestWithURL:[NSURL URLWithString:@"http://google.com"]];

// 创建URL连接并发送请求
NSURLConnection *conn = [[NSURLConnection alloc] initWithRequest:request delegate:self];
```



```
}
```

Section 78.3: Asynchronous request

```
// Create the request instance.
NSURLRequest *request = [NSURLRequest requestWithURL:[NSURL URLWithString:@"http://google.com"]];

// Create url connection and fire request
NSURLConnection *conn = [[NSURLConnection alloc] initWithRequest:request delegate:self];
```

第79章：NSURL

第79.1节：如何从NSURL字符串获取最后的字符串组件

```
NSURL *url = [NSURL URLWithString:@"http://www.example.com/images/apple-tree.jpg"];
NSString *fileName = [url lastPathComponent];
// fileName = "apple-tree.jpg"
```

第79.2节：如何从Swift中的URL（NSURL）获取最后的字符串组件

Swift 2.3

```
let url = NSURL(string: "http://google.com/lastPath")
let lastPath = url?.lastPathComponent
```

Swift 3.0

```
let url = URL(string: "http://google.com/lastPath")
let lastPath = url?.lastPathComponent
```


Chapter 79: NSURL

Section 79.1: How to get last string component from NSURL String

```
NSURL *url = [NSURL URLWithString:@"http://www.example.com/images/apple-tree.jpg"];
NSString *fileName = [url lastPathComponent];
// fileName = "apple-tree.jpg"
```

Section 79.2: How to get last string component from URL (NSURL) in Swift

Swift 2.3

```
let url = NSURL(string: "http://google.com/lastPath")
let lastPath = url?.lastPathComponent
```

Swift 3.0

```
let url = URL(string: "http://google.com/lastPath")
let lastPath = url?.lastPathComponent
```

第80章：NSData

第80.1节：将NSData转换为十六进制字符串

NSData 可以表示为十六进制字符串，类似于其 description 方法输出的内容。

Swift

```
extension NSData {

    func hexString() -> String {
        return UnsafeBufferPointer&lt;UInt8&gt;(start: UnsafePointer&lt;UInt8&gt;(bytes), count: length)
          .reduce("") { $0 + String(format: "%02x", $1) }
    }

}
```

Objective-C

```
@implementation NSData (HexRepresentation)

- (NSString *)hexString {
const unsigned char *bytes = (const unsigned char *)self.bytes;
    NSMutableString *hex = [NSMutableString new];
    for (NSInteger i = 0; i < self.length; i++) {
        [hex appendFormat:@"%02x", bytes[i]];
    }
    return [hex copy];
}

@end
```

第80.2节：创建NSData对象

使用文件

Swift

```
let data = NSData(contentsOfFile: filePath) //假设filePath是有效路径
```

Objective-C

```
NSData *data = [NSData dataWithContentsOfFile:filePath]; //假设filePath是有效路径
```

使用字符串对象

Swift

```
let data = (string as NSString).dataUsingEncoding(NSUTF8StringEncoding) //假设 string 是一个
String 对象
```

Objective-C

```
NSData *data = [string dataUsingEncoding:NSUTF8StringEncoding]; //假设 string 是一个 String
对象
```

第 80.3 节：将 NSData 转换为其他类型

转换为字符串

Swift

```
let string = String(NSString(data: data, encoding: NSUTF8StringEncoding)) //假设 data 是一个
```


Chapter 80: NSData

Section 80.1: Converting NSData to HEX string

`NSData` can be represented as hexadecimal string, similar to what it outputs in its `description` method.

Swift

```
extension NSData {

    func hexString() -> String {
        return UnsafeBufferPointer<UInt8>(start: UnsafePointer<UInt8>(bytes), count: length)
          .reduce("") { $0 + String(format: "%02x", $1) }
    }

}
```

Objective-C

```
@implementation NSData (HexRepresentation)

- (NSString *)hexString {
    const unsigned char *bytes = (const unsigned char *)self.bytes;
    NSMutableString *hex = [NSMutableString new];
    for (NSInteger i = 0; i < self.length; i++) {
        [hex appendFormat:@"%02x", bytes[i]];
    }
    return [hex copy];
}

@end
```

Section 80.2: Creating NSData objects

Using a file

Swift

```
let data = NSData(contentsOfFile: filePath) //assuming filePath is a valid path
```

Objective-C

```
NSData *data = [NSData dataWithContentsOfFile:filePath]; //assuming filePath is a valid path
```

Using a String object

Swift

```
let data = (string as NSString).dataUsingEncoding(NSUTF8StringEncoding) //assuming string is a
String object
```

Objective-C

```
NSData *data = [string dataUsingEncoding:NSUTF8StringEncoding]; //assuming string is a String
object
```

Section 80.3: Converting NSData to other types

To String

Swift

```
let string = String(NSString(data: data, encoding: NSUTF8StringEncoding)) //assuming data is a
```

```
有效的 NSData 对象
```

Objective-C

```
NSString *string = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding]; //假设 data 是一个有效的 NSData 对象
[string release];
```

转换为数组

Swift

```
let array = data.bytes as! NSMutableArray //假设 data 是一个有效的 NSData 对象
```

Objective-C

```
NSMutableArray *array = (NSMutableArray *)[data bytes]; //假设 data 是一个有效的 NSData 对象
```

转换为字节数组

Swift

```
let bytesArray = data.bytes as! UInt8 //假设 data 是一个有效的 NSData 对象
```

Objective-C

```
UInt8 *bytesArray = (UInt8 *)data.bytes; //假设 data 是一个有效的 NSData 对象
```



```
valid NSData object
```

Objective-C

```
NSString *string = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding]; //assuming data is a valid NSData object
[string release];
```

To Array

Swift

```
let array = data.bytes as! NSMutableArray //assuming data is a valid NSData object
```

Objective-C

```
NSMutableArray *array = (NSMutableArray *)[data bytes]; //assuming data is a valid NSData object
```

To Bytes Array

Swift

```
let bytesArray = data.bytes as! UInt8 //assuming data is a valid NSData object
```

Objective-C

```
UInt8 *bytesArray = (UInt8 *)data.bytes; //assuming data is a valid NSData object
```

第81章：NSInvocation

第81.1节：NSInvocation Objective-C

参考这篇 原文 由 e.James

根据 苹果的 NSInvocation 类参考文档：

> NSInvocation 是一个 Objective-C 消息的静态表现，也就是说，它是一个被转化为对象的动作。

更详细一点：

消息的概念是Objective-C哲学的核心。每当你调用某个对象的方法或访问其变量时，你就是在向它发送一条消息。 NSInvocation 在你想在不同时间点向对象发送消息，或多次发送同一条消息时非常有用。 NSInvocation 允许你描述你将要发送的消息，然后稍后调用它（实际上发送给目标对象）。

例如，假设你想向数组添加一个字符串。你通常会发送addObject:消息，方式如下：

```
[myArray addObject:myString];
```

现在，假设你想使用NSInvocation在其他时间点发送这条消息：

首先，你需要为NSMutableArray的addObject:选择器准备一个NSInvocation对象：

```
NSMethodSignature * mySignature = [NSMutableArray
    instanceMethodSignatureForSelector:@selector(addObject:)];
NSInvocation * myInvocation = [NSInvocation
    invocationWithMethodSignature:mySignature];
```

接下来，你需要指定要向哪个对象发送消息：

```
[myInvocation setTarget:myArray];
```

指定你希望发送给该对象的消息：

```
[myInvocation setSelector:@selector(addObject:)];
```

并填写该方法的所有参数：

```
[myInvocation setArgument:&myString atIndex:2];
```

请注意，对象参数必须以指针方式传递。感谢Ryan McCuaig指出这一点，
更多详情请参见Apple的文档。

此时，`myInvocation` 是一个完整的对象，描述了可以发送的消息。要实际发送该消息，你需要调用：

```
[myInvocation invoke];
```

这最后一步将导致消息被发送，实质上执行了[myArray addObject:myString];。

Chapter 81: NSInvocation

Section 81.1: NSInvocation Objective-C

Refer to this original Post by e.James

According to Apple's NSInvocation class reference:

> An `NSInvocation` is an Objective-C message rendered static, that is, it is an action turned into an object.

And, in a *little* more detail:

The concept of messages is central to the objective-c philosophy. Any time you call a method, or access a variable of some object, you are sending it a message. `NSInvocation` comes in handy when you want to send a message to an object at a different point in time, or send the same message several times. `NSInvocation` allows you to *describe* the message you are going to send, and then *invoke* it (actually send it to the target object) later on.

For example, let's say you want to add a string to an array. You would normally send the `addObject:` message as follows:

```
[myArray addObject:myString];
```

Now, let's say you want to use `NSInvocation` to send this message at some other point in time:

First, you would prepare an `NSInvocation` object for use with `NSMutableArray`'s `addObject:` selector:

```
NSMethodSignature * mySignature = [NSMutableArray
    instanceMethodSignatureForSelector:@selector(addObject:)];
NSInvocation * myInvocation = [NSInvocation
    invocationWithMethodSignature:mySignature];
```

Next, you would specify which object to send the message to:

```
[myInvocation setTarget:myArray];
```

Specify the message you wish to send to that object:

```
[myInvocation setSelector:@selector(addObject:)];
```

And fill in any arguments for that method:

```
[myInvocation setArgument:&myString atIndex:2];
```

Note that object arguments must be passed by pointer. Thank you to Ryan McCuaig for pointing that out, and please see Apple's documentation for more details.

At this point, `myInvocation` is a complete object, describing a message that can be sent. To actually send the message, you would call:

```
[myInvocation invoke];
```

This final step will cause the message to be sent, essentially executing `[myArray addObject:myString];`.

可以把它想象成发送电子邮件。你打开一封新邮件（NSInvocation对象），填写你想发送的人的地址（对象），为收件人输入消息（指定selector和参数），然后点击“发送”（调用invoke）。

更多信息请参见使用NSInvocation。

NSUndoManager使用NSInvocation对象来实现撤销命令。基本上，你所做的是创建一个NSInvocation对象，表示：“嘿，如果你想撤销我刚才做的操作，就向该对象发送这条消息，并带上这些参数”。你将NSInvocation对象交给NSUndoManager，它会将该对象添加到可撤销操作的数组中。如果用户调用“撤销”，NSUndoManager只需查找数组中最近的操作，并调用存储的NSInvocation对象来执行必要的操作。

有关详细信息，请参阅注册撤销操作。

Think of it like sending an email. You open up a new email (`NSInvocation` object), fill in the address of the person (object) who you want to send it to, type in a message for the recipient (specify a `selector` and arguments), and then click "send" (call `invoke`).

See Using NSInvocation for more information.

`NSUndoManager` uses `NSInvocation` objects so that it can *reverse* commands. Essentially, what you are doing is creating an `NSInvocation` object to say: "Hey, if you want to undo what I just did, send this message to that object, with these arguments". You give the `NSInvocation` object to the `NSUndoManager`, and it adds that object to an array of undoable actions. If the user calls "Undo", `NSUndoManager` simply looks up the most recent action in the array, and invokes the stored `NSInvocation` object to perform the necessary action.

See Registering Undo Operations for more details.

第82章：NSUserActivity

NSUserActivity对象可用于协调应用程序中的重要事件与系统的交互。它是不同运行iOS和macOS设备之间Handoff功能的基础。此外，它还可用于改进公共索引，并增强或创建应用程序的Spotlight搜索结果。从iOS 10开始，它还可用于协调您的应用程序与Siri之间通过SiriKit的交互。

第82.1节：创建NSUserActivity

要创建一个NSUserActivity对象，您的应用必须在其Info.plist文件中声明支持的活动类型。
支持的活动由您的应用定义，且应当唯一。活动使用反向域名风格的命名方案定义（例如“com.companyName.productName.activityName”）。以下是Info.plist中的一个示例条目：

键	值
NSUserActivityTypes	[数组]
- 项目0	com.companyName.productName.activityName01
- 项目1	com.companyName.productName.activityName02

一旦定义了所有支持的活动类型，您就可以开始在应用程序的

代码中访问和使用它们。

要创建一个NSUserActivity对象，您必须执行以下操作

```
// 初始化活动对象并从应用程序的
plist中指定的类型之一设置类型
NSUserActivity *currentActivity = [[NSUserActivity alloc]
initWithActivityType:@"com.companyName.productName.activityName01"];

// 设置活动的标题。
// 该标题可能会显示给用户，因此请确保其已本地化且易于理解
currentActivity.title = @"当前活动";

// 配置其他属性，如将包含在活动中的userInfo
currentActivity.userInfo = @{@"informationKey" : @"value"};

// 配置活动，以便系统知道可以对其执行的操作
// 重要的是，您只对应用程序支持的任务设置YES
// 在此示例中，我们仅启用该活动以用于Handoff
[currentActivity setEligibleForHandoff:YES];
[currentActivity setEligibleForSearch:NO]; // 默认为NO
[currentActivity setEligibleForPublicIndexing:NO]; // 默认为NO

// 将此活动设置为当前用户活动
// 在同一时间内，设备上只能有一个活动处于当前状态。调用此方法会使任何其他当前活动失效。

[currentActivity becomeCurrent];
```

之后，上述活动应该可以用于接力（Handoff）（尽管还需要更多工作来正确处理“接力”）。

Chapter 82: NSUserActivity

An NSUserActivity object can be used to coordinate significant events in an app with the system. It is the basis for Handoff between different devices running iOS and macOS. Additionally, it may also be used to improve public-indexing and augment or create Spotlight Search results for an app. As of iOS 10, it may also be used to coordinate interactions between your app and Siri using SiriKit.

Section 82.1: Creating a NSUserActivity

To create a `NSUserActivity` object, your app must declare the types of activities it supports in its `Info.plist` file. Supported activities are defined by your application and should be unique. An activity is defined using a reverse-domain style naming scheme (i.e. "com.companyName.productName.activityName"). Here is what an entry in your Info.plist might look like:

Key	Value
NSUserActivityTypes	[Array]
- item0	com.companyName.productName.activityName01
- item1	com.companyName.productName.activityName02

Once you have defined all supported activity types, you may begin to access and use them in your application's code.

To create a `NSUserActivity` object you must do the following

```
// Initialize the activity object and set its type from one of the ones specified in your app's
plist
NSUserActivity *currentActivity = [[NSUserActivity alloc]
initWithActivityType:@"com.companyName.productName.activityName01"];

// Set the title of the activity.
// This title may be displayed to the user, so make sure it is localized and human-readable
currentActivity.title = @"Current Activity";

// Configure additional properties like userInfo which will be included in the activity
currentActivity.userInfo = @{@"informationKey" : @"value"};

// Configure the activity so the system knows what may be done with it
// It is important that you only set YES to tasks that your application supports
// In this example, we will only enable the activity for use with Handoff
[currentActivity setEligibleForHandoff:YES];
[currentActivity setEligibleForSearch:NO]; // Defaults to NO
[currentActivity setEligibleForPublicIndexing:NO]; // Defaults to NO

// Set this activity as the current user activity
// Only one activity may be current at a time on a device. Calling this method invalidates any
other current activities.
[currentActivity becomeCurrent];
```

After this, the activity above should be available for Handoff (although more work is required to properly handle the "Handoff").

第83章：NSPredicate

第83.1节：使用NSPredicate进行表单验证

```
NSString *emailRegex = @"[A-Z0-9a-z]([A-Z0-9a-z._-]{0,64})+[A-Z0-9a-z]+@[A-Z0-9a-z]+([A-Za-z0-9.-
]{0,64})+([A-Z0-9a-z])+\\.[A-Za-z]{2,4}";    NSString *firstNameRegex = @"[0-9A-Za-z\"'-]{2,32}$";
NSString *firstNameRegex = @"[ 0-9A-Za-z]{2,32}$";
NSString *lastNameRegex = @"[0-9A-Za-z\"'-]{2,32}$";
NSString *mobileNumberRegEx = @"^[0-9]{10}$";
NSString *zipcodeRegEx = @"^[0-9]{5}$";
NSString *SSNRegEx = @"^\\d{3}-?\\d{2}-?\\d{4}$";
NSString *addressRegEx = @"^[ A-Za-z0-9]{2,32}$";
NSString *cityRegEx = @"^[ A-Za-z0-9]{2,25}$";
NSString *PINRegEx = @"^[0-9]{4}$";
NSString *driversLiscRegEx = @"^[0-9a-zA-Z]{5,20}$";

-(BOOL)validateEmail {
                        //当电子邮件地址以“.”、“-”、“_”开头时，电子邮件地址字段应显示错误。
    NSPredicate *emailPredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", emailRegex];

    return ([emailPredicate evaluateWithObject:self.text] && self.text.length <= 64 && ([self.text
rangeOfString:@".."].location == NSNotFound));
}

- (BOOL)validateFirstName {
    NSPredicate *firstNamePredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@",
firstNameRegex];
    return [firstNamePredicate evaluateWithObject:self.text];
}

- (BOOL)validateLastName {
    NSPredicate *lastNamePredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@",
lastNameRegex];
    return [lastNamePredicate evaluateWithObject:self.text];
}

- (BOOL)validateAlphaNumericMin2Max32 {
    NSPredicate *firstNamePredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@",
firstNameRegex];
    return [firstNamePredicate evaluateWithObject:self.text];
}

- (BOOL)validateMobileNumber {
    NSString *strippedMobileNumber =  [[[[self.text stringByReplacingOccurrencesOfString:@"("
withString:@""]
stringByReplacingOccurrencesOfString:@")" withString:@""]
                                       stringByReplacingOccurrencesOfString:@"-" withString:@""]
                                       stringByReplacingOccurrencesOfString:@" " withString:@""];

    NSPredicate *mobileNumberPredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@",
mobileNumberRegEx];

    return [mobileNumberPredicate evaluateWithObject:strippedMobileNumber];
}

- (BOOL)validateZipcode {
    NSPredicate *zipcodePredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@",
zipcodeRegEx];

    return [zipcodePredicate evaluateWithObject:self.text];
```


Chapter 83: NSPredicate

Section 83.1: Form validation using NSPredicate

```
NSString *emailRegex = @"[A-Z0-9a-z]([A-Z0-9a-z._-]{0,64})+[A-Z0-9a-z]+@[A-Z0-9a-z]+([A-Za-z0-9.-
]{0,64})+([A-Z0-9a-z])+\\.[A-Za-z]{2,4}";    NSString *firstNameRegex = @"[0-9A-Za-z\"'-]{2,32}$";
NSString *firstNameRegex = @"[ 0-9A-Za-z]{2,32}$";
NSString *lastNameRegex = @"[0-9A-Za-z\"'-]{2,32}$";
NSString *mobileNumberRegEx = @"^[0-9]{10}$";
NSString *zipcodeRegEx = @"^[0-9]{5}$";
NSString *SSNRegEx = @"^\\d{3}-?\\d{2}-?\\d{4}$";
NSString *addressRegEx = @"^[ A-Za-z0-9]{2,32}$";
NSString *cityRegEx = @"^[ A-Za-z0-9]{2,25}$";
NSString *PINRegEx = @"^[0-9]{4}$";
NSString *driversLiscRegEx = @"^[0-9a-zA-Z]{5,20}$";

-(BOOL)validateEmail {
    //Email address field should give an error when the email address begins with ".","-","_" .
    NSPredicate *emailPredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", emailRegex];

    return ([emailPredicate evaluateWithObject:self.text] && self.text.length <= 64 && ([self.text
rangeOfString:@".."].location == NSNotFound));
}

- (BOOL)validateFirstName {
    NSPredicate *firstNamePredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@",
firstNameRegex];
    return [firstNamePredicate evaluateWithObject:self.text];
}

- (BOOL)validateLastName {
    NSPredicate *lastNamePredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@",
lastNameRegex];
    return [lastNamePredicate evaluateWithObject:self.text];
}

- (BOOL)validateAlphaNumericMin2Max32 {
    NSPredicate *firstNamePredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@",
firstNameRegex];
    return [firstNamePredicate evaluateWithObject:self.text];
}

- (BOOL)validateMobileNumber {
    NSString *strippedMobileNumber =  [[[[self.text stringByReplacingOccurrencesOfString:@"("
withString:@""]
                                         stringByReplacingOccurrencesOfString:@")" withString:@""]
                                         stringByReplacingOccurrencesOfString:@"-" withString:@""]
                                         stringByReplacingOccurrencesOfString:@" " withString:@""];

    NSPredicate *mobileNumberPredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@",
mobileNumberRegEx];

    return [mobileNumberPredicate evaluateWithObject:strippedMobileNumber];
}

- (BOOL)validateZipcode {
    NSPredicate *zipcodePredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@",
zipcodeRegEx];

    return [zipcodePredicate evaluateWithObject:self.text];
```

```
}

- (BOOL)validateSSN {
NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", SSNRegEx];

return [predicate evaluateWithObject:self.text];
}

- (BOOL)validateAddress {
          NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", addressRegEx];

    return [predicate evaluateWithObject:self.text];
}

- (BOOL)validateCity {
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", cityRegEx];
    return [predicate evaluateWithObject:self.text];
}

- (BOOL)validatePIN {
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", PINRegEx];
    return [predicate evaluateWithObject:self.text];
}
   - (BOOL)validateDriversLiscNumber {
    if([self.text length] > 20) {
        return NO;
    }
    NSPredicate *driversLiscPredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@",
driversLiscRegEx];

    return [driversLiscPredicate evaluateWithObject:self.text];
}
```

第83.2节：使用

predicateWithBlock 创建 NSPredicate

Objective-C

```
NSPredicate *predicate = [NSPredicate predicateWithBlock:^BOOL(id item,
                                                              NSDictionary *bindings) {
    return [item isKindOfClass:[UILabel class]];
}];
```

Swift

```
let predicate = NSPredicate { (item, bindings) -> Bool in
    return item.isKindOfClass(UILabel.self)
}
```

在此示例中，谓词将匹配属于 UILabel 类的项目。

第83.3节：使用

predicateWithFormat 创建 NSPredicate

Objective-C

```
NSPredicate *predicate = [NSPredicate predicateWithFormat: @"self[SIZE] = %d", 5)];
```

Swift


```
}

- (BOOL)validateSSN {
NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", SSNRegEx];

return [predicate evaluateWithObject:self.text];
}

- (BOOL)validateAddress {
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", addressRegEx];

    return [predicate evaluateWithObject:self.text];
}

- (BOOL)validateCity {
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", cityRegEx];
    return [predicate evaluateWithObject:self.text];
}

- (BOOL)validatePIN {
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", PINRegEx];
    return [predicate evaluateWithObject:self.text];
}
   - (BOOL)validateDriversLiscNumber {
    if([self.text length] > 20) {
        return NO;
    }
    NSPredicate *driversLiscPredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@",
driversLiscRegEx];

    return [driversLiscPredicate evaluateWithObject:self.text];
}
```

Section 83.2: Creating an NSPredicate Using predicateWithBlock

Objective-C

```
NSPredicate *predicate = [NSPredicate predicateWithBlock:^BOOL(id item,
                                                              NSDictionary *bindings) {
    return [item isKindOfClass:[UILabel class]];
}];
```

Swift

```
let predicate = NSPredicate { (item, bindings) -> Bool in
    return item.isKindOfClass(UILabel.self)
}
```

In this example, the predicate will match items that are of the class `UILabel`.

Section 83.3: Creating an NSPredicate Using predicateWithFormat

Objective-C

```
NSPredicate *predicate = [NSPredicate predicateWithFormat: @"self[SIZE] = %d", 5)];
```

Swift

```
let predicate = NSPredicate(format: "self[SIZE] >= %d", 5)
```

在此示例中，谓词将匹配长度至少为5的数组项目。

第83.4节：使用替换变量创建 NSPredicate

NSPredicate 可以使用替换变量以允许动态绑定值。

Objective-C

```
NSPredicate *template = [NSPredicate predicateWithFormat: @"self BEGINSWITH $letter"];
NSDictionary *variables = @{ @"letter": @"r" };
NSPredicate *beginsWithR = [template predicateWithSubstitutionVariables: variables];
```

Swift

```
let template = NSPredicate(format: "self BEGINSWITH $letter")
let variables = ["letter": "r"]
let beginsWithR = template.predicateWithSubstitutionVariables(variables)
```

模板谓词不会被`predicateWithSubstitutionVariables`修改。相反，会创建一个副本，该副本接收替换变量。

第83.5节：带有`AND`、`OR`和`NOT`条件的NSPredicate

使用`NSCompoundPredicate`类可以使条件谓词更简洁且更安全，该类为给定的谓词提供基本的布尔运算符。

Objective-C

AND - 条件

```
  NSPredicate *predicate = [NSPredicate predicateWithFormat:@"samplePredicate"];
  NSPredicate *anotherPredicate = [NSPredicate predicateWithFormat:@"anotherPredicate"];
  NSPredicate *combinedPredicate = [NSCompoundPredicate andPredicateWithSubpredicates:
@[predicate,anotherPredicate]];
```

OR - 条件

```
 NSPredicate *predicate = [NSPredicate predicateWithFormat:@"samplePredicate"];
 NSPredicate *anotherPredicate = [NSPredicate predicateWithFormat:@"anotherPredicate"];
 NSPredicate *combinedPredicate = [NSCompoundPredicate orPredicateWithSubpredicates:
@[predicate,anotherPredicate]];
```

非 - 条件

```
 NSPredicate *predicate = [NSPredicate predicateWithFormat:@"samplePredicate"];
 NSPredicate *anotherPredicate = [NSPredicate predicateWithFormate:@"anotherPredicate"];
 NSPredicate *combinedPredicate = [NSCompoundPredicate notPredicateWithSubpredicate:
@[predicate,anotherPredicate]];
```

第83.6节：使用NSPredicate过滤数组

Objective-C

```
NSArray *heroes = @[@"tracer", @"bastion", @"reaper", @"junkrat", @"roadhog"];

NSPredicate *template = [NSPredicate predicateWithFormat:@"self BEGINSWITH $letter"];
```



```
let predicate = NSPredicate(format: "self[SIZE] >= %d", 5)
```

In this example, the predicate will match items that are arrays with length of at least 5.

Section 83.4: Creating an NSPredicate with Substitution Variables

An `NSPredicate` can use substitution variables to allow values to be bound on the fly.

Objective-C

```
NSPredicate *template = [NSPredicate predicateWithFormat: @"self BEGINSWITH $letter"];
NSDictionary *variables = @{ @"letter": @"r" };
NSPredicate *beginsWithR = [template predicateWithSubstitutionVariables: variables];
```

Swift

```
let template = NSPredicate(format: "self BEGINSWITH $letter")
let variables = ["letter": "r"]
let beginsWithR = template.predicateWithSubstitutionVariables(variables)
```

The template predicate is not modified by `predicateWithSubstitutionVariables`. Instead, a copy is created, and that copy receives the substitution variables.

Section 83.5: NSPredicate with `AND`, `OR` and `NOT` condition

Conditional predicate will be cleaner and safer by using the `NSCompoundPredicate` class which provides basic boolean operators for the given predicates.

Objective-c

AND - Condition

```
  NSPredicate *predicate = [NSPredicate predicateWithFormat:@"samplePredicate"];
  NSPredicate *anotherPredicate = [NSPredicate predicateWithFormat:@"anotherPredicate"];
  NSPredicate *combinedPredicate = [NSCompoundPredicate andPredicateWithSubpredicates:
@[predicate,anotherPredicate]];
```

OR - Condition

```
 NSPredicate *predicate = [NSPredicate predicateWithFormat:@"samplePredicate"];
 NSPredicate *anotherPredicate = [NSPredicate predicateWithFormat:@"anotherPredicate"];
 NSPredicate *combinedPredicate = [NSCompoundPredicate orPredicateWithSubpredicates:
@[predicate,anotherPredicate]];
```

NOT - Condition

```
 NSPredicate *predicate = [NSPredicate predicateWithFormat:@"samplePredicate"];
 NSPredicate *anotherPredicate = [NSPredicate predicateWithFormat:@"anotherPredicate"];
 NSPredicate *combinedPredicate = [NSCompoundPredicate notPredicateWithSubpredicate:
@[predicate,anotherPredicate]];
```

Section 83.6: Using NSPredicate to Filter an Array

Objective-C

```
NSArray *heroes = @[@"tracer", @"bastion", @"reaper", @"junkrat", @"roadhog"];

NSPredicate *template = [NSPredicate predicateWithFormat:@"self BEGINSWITH $letter"];
```

```
NSDictionary *beginsWithRVariables = @{ @"letter": @"r"};
NSPredicate *beginsWithR = [template predicateWithSubstitutionVariables: beginsWithRVariables];

NSArray *beginsWithRHeroes = [heroes filteredArrayUsingPredicate: beginsWithR];
// ["reaper", "roadhog"]

NSDictionary *beginsWithTVariables = @{ @"letter": @"t"};
NSPredicate *beginsWithT = [template predicateWithSubstitutionVarables: beginsWithTVariables];

NSArray *beginsWithTHeroes = [heroes filteredArrayUsingPredicate: beginsWithT];
// ["tracer"]
```

Swift

```
let heroes = ["tracer", "bastion", "reaper", "junkrat", "roadhog"]

let template = NSPredicate(format: "self BEGINSWITH $letter")

let beginsWithRVariables = ["letter": "r"]
let beginsWithR = template.predicateWithSubstitutionVariables(beginsWithRVariables)

let beginsWithRHeroes = heroes.filter { beginsWithR.evaluateWithObject($0) }
// ["reaper", "roadhog"]

let beginsWithTVariables = ["letter": "t"]
let beginsWithT = template.predicateWithSubstitutionVariables(beginsWithTVariables)

let beginsWithTHeroes = heroes.filter { beginsWithT.evaluateWithObject($0) }
// ["tracer"]
```



```
NSDictionary *beginsWithRVariables = @{ @"letter": @"r"};
NSPredicate *beginsWithR = [template predicateWithSubstitutionVariables: beginsWithRVariables];

NSArray *beginsWithRHeroes = [heroes filteredArrayUsingPredicate: beginsWithR];
// ["reaper", "roadhog"]

NSDictionary *beginsWithTVariables = @{ @"letter": @"t"};
NSPredicate *beginsWithT = [template predicateWithSubstitutionVarables: beginsWithTVariables];

NSArray *beginsWithTHeroes = [heroes filteredArrayUsingPredicate: beginsWithT];
// ["tracer"]
```

Swift

```
let heroes = ["tracer", "bastion", "reaper", "junkrat", "roadhog"]

let template = NSPredicate(format: "self BEGINSWITH $letter")

let beginsWithRVariables = ["letter": "r"]
let beginsWithR = template.predicateWithSubstitutionVariables(beginsWithRVariables)

let beginsWithRHeroes = heroes.filter { beginsWithR.evaluateWithObject($0) }
// ["reaper", "roadhog"]

let beginsWithTVariables = ["letter": "t"]
let beginsWithT = template.predicateWithSubstitutionVariables(beginsWithTVariables)

let beginsWithTHeroes = heroes.filter { beginsWithT.evaluateWithObject($0) }
// ["tracer"]
```

第84章：NSBundle

第84.1节：通过路径获取Bundle

1. 使用路径定位Cocoa Bundle

> 要使用Cocoa获取特定路径下的bundle，调用NSBundle的bundleWithPath:类方法

```
NSBundle *myBundle;
// 获取一个可加载bundle的引用
myBundle = [NSBundle bundleWithPath:@"/Library/MyBundle.bundle";
```

2. 使用路径定位Cocoa Foundation Bundle

> 要使用 Core Foundation 在特定路径获取捆绑包，调用 CFBundleCreate 函数，且必须使用 CFURLRef 类型。

```
CFURLRef bundleURL;
CFBundleRef myBundle;
// 从捆绑包路径的 CFString 表示创建一个 CFURLRef。
bundleURL = CFURLCreateWithFileSystemPath(kCFAllocatorDefault,
CFSTR("/Library/MyBundle.bundle"), kCFURLPOSIXPathStyle, true);
// 使用 URLRef 创建捆绑包实例。
myBundle = CFBundleCreate(kCFAllocatorDefault, bundeURL);
// 现在可以释放 URL 了。
CFRelease(bundleURL);
// 使用捆绑包 ...
// 使用完成后释放捆绑包。
CFRelease(myBundle);
```

第 84.2 节：获取主捆绑包

1. 使用 Cocoa 获取主包的引用。

> 要获取Cocoa应用程序中的主包，调用NSBundle类的mainBundle类方法。

```
NSBundle *mainBundle;
// 获取应用程序的主包；
mainBundle = [NSBundle mainBundle];
```

2. 使用Core Foundation获取主包的引用。

> 使用CFBundleGetMainBundle函数来检索基于C的应用程序的主包。

```
CFBundleRef mainBundle;
// 获取应用程序的主包
mainBundle = CFBundleGetMainBundle();
```


Chapter 84: NSBundle

Section 84.1: Getting Bundle by Path

1. Locating a Cocoa bundle using its path

> To obtain the bundle at a specific path using Cocoa, call the ***bundleWithPath:*** class method of the **NSBundle**

```
NSBundle *myBundle;
// obtain a reference to a loadable bundle
myBundle = [NSBundle bundleWithPath:@"/Library/MyBundle.bundle";
```

2. Locating a Cocoa Foundation bundle using its Path

> To obtain the bundle at a specific path using Core Foundation, call the ***CFBundleCreate*** function and must use **CFURLRef** type.

```
CFURLRef bundleURL;
CFBundleRef myBundle;
// Make a CFURLRef from the CFString representation of the bundle's path.
bundleURL = CFURLCreateWithFileSystemPath(kCFAllocatorDefault,
CFSTR("/Library/MyBundle.bundle"), kCFURLPOSIXPathStyle, true);
// Make a bundle instance using the URLRef.
myBundle = CFBundleCreate(kCFAllocatorDefault, bundeURL);
// You can release the URL now.
CFRelease(bundleURL);
// Use the bundle ...
// Release the bundle when done.
CFRelease(myBundle);
```

Section 84.2: Getting the Main Bundle

1. Getting a reference to the main bundle using Cocoa.

> To get the main bundle in Cocoa application, call the ***mainBundle*** class method of the **NSBundle** class.

```
NSBundle *mainBundle;
// Get the main bundle for the app;
mainBundle = [NSBundle mainBundle];
```

2. Getting a reference to the main bundle using Core Foundation.

> Use the ***CFBundleGetMainBundle*** function to retrieve the main bundle for your C-based application.

```
CFBundleRef mainBundle;
// Get the main bundle for the app
mainBundle = CFBundleGetMainBundle();
```

第85章：CAAnimation

第85.1节：将视图从一个位置动画到另一个位置

Objective-C

```
CABasicAnimation *animation = [CABasicAnimation animationWithKeyPath:@"position.x"];
animation.fromValue = @0;
animation.toValue = @320;
animation.持续时间 = 1;

[_label.图层 添加动画:animation forKey:@"basic"];
```

Swift

```
let animation = CABasicAnimation(keyPath: "position.x")
animation.起始值 = NSNumber(value: 0.0)
animation.结束值 = NSNumber(value: 320.0)

_label.图层.添加动画(animation, forKey: "basic")
```

视图将在水平方向上从0移动到320。如果你想让视图垂直移动，只需将keyPath替换为：

```
"position.y"
```

第85.2节：动画视图 - 投掷

OBJECTIVE-C

```
CATransition* transition = [CATransition animation];
transition.开始进度 = 0;
transition.结束进度 = 1.0;
transition.类型 = @"flip";
transition.子类型 = @"fromLeft";
transition.持续时间 = 0.8;
transition.repeatCount = 5;
[_label.layer 添加动画:transition forKey:@"transition"];
```

SWIFT

```
var transition = CATransition()
transition.startProgress = 0
transition.endProgress = 1.0
transition.type = "flip"
transition.subtype = "fromLeft"
transition.duration = 0.8
transition.repeatCount = 5
label.layer.addAnimation(transition, forKey: "transition")
```

第85.3节：推入视图动画

Objective C

```
CATransition *animation = [CATransition animation];
[animation setSubtype:kCATransitionFromRight];//kCATransitionFromLeft
[animation setDuration:0.5];
[animation setType:kCATransitionPush];
[animation setTimingFunction:[CAMediaTimingFunction
functionWithName:kCAMediaTimingFunctionEaseInEaseOut]];
[[yourView layer] addAnimation:animation forKey:@"SwitchToView1"];
```


Chapter 85: CAAnimation

Section 85.1: Animate a view from one position to another

Objective-C

```
CABasicAnimation *animation = [CABasicAnimation animationWithKeyPath:@"position.x"];
animation.fromValue = @0;
animation.toValue = @320;
animation.duration = 1;

[_label.layer addAnimation:animation forKey:@"basic"];
```

Swift

```
let animation = CABasicAnimation(keyPath: "position.x")
animation.fromValue = NSNumber(value: 0.0)
animation.toValue = NSNumber(value: 320.0)

_label.layer.addAnimation(animation, forKey: "basic")
```

The view will move from 0 to 320 horizontally. if you want to Move view to Vertically just replace keypath like this:

```
"position.y"
```

Section 85.2: Animate View - Toss

OBJECTIVE-C

```
CATransition* transition = [CATransition animation];
transition.startProgress = 0;
transition.endProgress = 1.0;
transition.type = @"flip";
transition.subtype = @"fromLeft";
transition.duration = 0.8;
transition.repeatCount = 5;
[_label.layer addAnimation:transition forKey:@"transition"];
```

SWIFT

```
var transition = CATransition()
transition.startProgress = 0
transition.endProgress = 1.0
transition.type = "flip"
transition.subtype = "fromLeft"
transition.duration = 0.8
transition.repeatCount = 5
label.layer.addAnimation(transition, forKey: "transition")
```

Section 85.3: Push View Animation

Objective C

```
CATransition *animation = [CATransition animation];
[animation setSubtype:kCATransitionFromRight];//kCATransitionFromLeft
[animation setDuration:0.5];
[animation setType:kCATransitionPush];
[animation setTimingFunction:[CAMediaTimingFunction
functionWithName:kCAMediaTimingFunctionEaseInEaseOut]];
[[yourView layer] addAnimation:animation forKey:@"SwitchToView1"];
```

Swift

```
let animation = CATransition()
animation.subtype = kCATransitionFromRight//kCATransitionFromLeft
animation.duration = 0.5
animation.type = kCATransitionPush
animation.timingFunction = CAMediaTimingFunction(name: kCAMediaTimingFunctionEaseInEaseOut)
yourView.layer.addAnimation(animation, forKey: "SwitchToView1")
```

第85.4节：旋转视图

```
 CGRect boundingRect = CGRectMake(-150, -150, 300, 300);

CAKeyframeAnimation *orbit = [CAKeyframeAnimation animation];
orbit.keyPath = @"position";
orbit.path = CFAutorelease(CGPathCreateWithEllipseInRect(boundingRect, NULL));
orbit.duration = 4;
orbit.additive = YES;
orbit.repeatCount = HUGE_VALF;
orbit.calculationMode = kCAAnimationPaced;
orbit.rotationMode = kCAAnimationRotateAuto;

[_label.layer addAnimation:orbit forKey:@"orbit"];
```

第85.5节：摇动视图

Objective-C

```
CAKeyframeAnimation *animation = [CAKeyframeAnimation animationWithKeyPath:@"position.x"];
animation.values = @[ @0, @10, @-10, @10, @0 ];
animation.keyTimes = @[ @0, @(1 / 6.0), @(3 / 6.0), @(5 / 6.0), @1 ];
animation.duration = 0.4;
animation.additive = YES;
[_label.layer addAnimation:animation forKey:@"shake"];
```

Swift 3

```
let animation = CAKeyframeAnimation(keyPath: "position.x")
animation.values = [ 0, 10, -10, 10, 0 ]
        animation.keyTimes = [ 0, NSNumber(value: (1 / 6.0)), NSNumber(value: (3 / 6.0)), NSNumber(value:
(5 / 6.0)), 1 ]
animation.duration = 0.4
animation.isAdditive = true
label.layer.add(animation, forKey: "shake")
```


Swift

```
let animation = CATransition()
animation.subtype = kCATransitionFromRight//kCATransitionFromLeft
animation.duration = 0.5
animation.type = kCATransitionPush
animation.timingFunction = CAMediaTimingFunction(name: kCAMediaTimingFunctionEaseInEaseOut)
yourView.layer.addAnimation(animation, forKey: "SwitchToView1")
```

Section 85.4: Revolve View

```
 CGRect boundingRect = CGRectMake(-150, -150, 300, 300);

CAKeyframeAnimation *orbit = [CAKeyframeAnimation animation];
orbit.keyPath = @"position";
orbit.path = CFAutorelease(CGPathCreateWithEllipseInRect(boundingRect, NULL));
orbit.duration = 4;
orbit.additive = YES;
orbit.repeatCount = HUGE_VALF;
orbit.calculationMode = kCAAnimationPaced;
orbit.rotationMode = kCAAnimationRotateAuto;

[_label.layer addAnimation:orbit forKey:@"orbit"];
```

Section 85.5: Shake View

Objective-C

```
CAKeyframeAnimation *animation = [CAKeyframeAnimation animationWithKeyPath:@"position.x"];
animation.values = @[ @0, @10, @-10, @10, @0 ];
animation.keyTimes = @[ @0, @(1 / 6.0), @(3 / 6.0), @(5 / 6.0), @1 ];
animation.duration = 0.4;
animation.additive = YES;
[_label.layer addAnimation:animation forKey:@"shake"];
```

Swift 3

```
let animation = CAKeyframeAnimation(keyPath: "position.x")
animation.values = [ 0, 10, -10, 10, 0 ]
animation.keyTimes = [ 0, NSNumber(value: (1 / 6.0)), NSNumber(value: (3 / 6.0)), NSNumber(value:
(5 / 6.0)), 1 ]
animation.duration = 0.4
animation.isAdditive = true
label.layer.add(animation, forKey: "shake")
```

第86章：并发

队列 queue	调度块中的代码将在哪个队列中运行。队列（queue）类似于（但不完全等同于）线程；不同队列中的代码可以并行运行。使用dispatch_get_main_queue可以获取主线程的队列。要创建一个新队列（这也会创建一个新线程），使用dispatch_queue_create("QUEUE_NAME", DISPATCH_QUEUE_CONCURRENT)。第一个参数是队列的名称，如果在块仍在运行时暂停调试器，该名称会显示在调试器中。第二个参数除非你想在多个dispatch_async或dispatch_sync调用中使用同一个队列，否则无关紧要。它描述了当另一个块被放入同一队列时会发生什么；DISPATCH_QUEUE_CONCURRENT会使两个块同时运行，而DISPATCH_QUEUE_SERIAL会让第二个块等待第一个块完成后再运行。
块 block	此块中的代码将在队列queue中运行；将你想在独立队列上运行的代码放在这里。一个有用的提示：如果你在Xcode中编写此代码，并且块参数周围有蓝色轮廓，双击该参数，Xcode会自动生成一个空块（这适用于任何函数或方法中的所有块参数）

相关主题：Grand Central Dispatch（GCD）

第86.1节：调度组 - 等待其他线程完成

```
dispatch_group_t preapreWaitingGroup = dispatch_group_create();

dispatch_group_enter(preapreWaitingGroup);
[self doAsynchronousTaskWithComplete:^(id someResults, NSError *error) {
    // 通知该任务已完成。
dispatch_group_leave(preapreWaitingGroup);
}]

dispatch_group_enter(preapreWaitingGroup);
[self 执行其他异步任务并完成:^(id someResults, NSError *error) {
    dispatch_group_leave(preapreWaitingGroup);
}]

dispatch_group_notify(preapreWaitingGroup, dispatch_get_main_queue(), ^{
    // 当上述所有线程完成并调用 dispatch_group_leave 后，此代码块将被执行
    NSLog(@"准备完成。我准备好了");
});
```

更新 1. Swift 3 版本。

```
let prepareGroup = DispatchGroup()
prepareGroup.enter()
doAsynchronousTaskWithComplete() { (someResults, error) in
    // 通知该任务已完成。
prepareGroup.leave()
}

prepareGroup.enter()
doOtherAsynchronousTaskWithComplete() { (someResults, error) in
    // 通知该任务已完成。
prepareGroup.leave()
}

prepareGroup.notify(queue: DispatchQueue.main) {
    // 当上述所有线程完成后，此代码块将被执行并调用 dispatch_group_leave
    print("准备完成。我准备好了")
}
```


Chapter 86: Concurrency

queue	**The queue that the code in the dispatch block will run in. A *queue* is like (but not exactly the same as) a thread; code in different queues can run in parallel. Use `dispatch_get_main_queue` to get the queue for the main thread To create a new queue, which in turn creates a new thread, use `dispatch_queue_create("QUEUE_NAME", DISPATCH_QUEUE_CONCURRENT)`. The first parameter is the name of the queue, which is displayed in the debugger if you pause while the block is still running. The second parameter doesn't matter unless you want to use the same queue for multiple `dispatch_async` or `dispatch_sync` calls. It describes what happens when another block is put on the same queue; `DISPATCH_QUEUE_CONCURRENT` will cause both blocks to run at the same time, while `DISPATCH_QUEUE_SERIAL` will make the second block wait for the first block to finish**
block	Code in this block will run in the queue queue; put code you want to run on the separate queue here. A helpful tip: If you're writing this in Xcode and the block argument has the blue outline around it, double click on the argument and Xcode will automatically make an empty block (this applies to all block arguments in any function or method)

Related topic: Grand Central Dispatch

Section 86.1: Dispatch group - waiting for other threads completed

```
dispatch_group_t preapreWaitingGroup = dispatch_group_create();

dispatch_group_enter(preapreWaitingGroup);
[self doAsynchronousTaskWithComplete:^(id someResults, NSError *error) {
    // Notify that this task has been completed.
    dispatch_group_leave(preapreWaitingGroup);
}]

dispatch_group_enter(preapreWaitingGroup);
[self doOtherAsynchronousTaskWithComplete:^(id someResults, NSError *error) {
    dispatch_group_leave(preapreWaitingGroup);
}]

dispatch_group_notify(preapreWaitingGroup, dispatch_get_main_queue(), ^{
    // This block will be executed once all above threads completed and call dispatch_group_leave
    NSLog(@"Prepare completed. I'm readyyyy");
});
```

Update 1. Swift 3 version.

```
let prepareGroup = DispatchGroup()
prepareGroup.enter()
doAsynchronousTaskWithComplete() { (someResults, error) in
    // Notify that this task has been completed.
    prepareGroup.leave()
}

prepareGroup.enter()
doOtherAsynchronousTaskWithComplete() { (someResults, error) in
    // Notify that this task has been completed.
    prepareGroup.leave()
}

prepareGroup.notify(queue: DispatchQueue.main) {
    // This block will be executed once all above threads completed and call dispatch_group_leave
    print("Prepare completed. I'm readyyyy")
}
```

第86.2节：在主线程上执行

在异步执行任务时，通常需要确保某段代码在主线程上运行。例如，你可能想异步调用REST API，但将结果放入屏幕上的UILabel中。在更新UILabel之前，必须确保代码在主线程上运行：

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
    //执行耗时任务
    //...

    //现在在更新UI之前，确保回到主线程
    dispatch_async(dispatch_get_main_queue(), ^{
       label.text = //....
    });
}
```

每当你更新屏幕上的视图时，务必确保在主线程上执行，否则可能会出现
未定义的行为。

第86.3节：并发运行代码——在运行其他代码时运行代码

假设你想在执行某个操作时（在本例中是记录“Foo”），同时做另一件事（记录“Bar”）。通常，如果不使用并发，其中一个操作会被完全执行，另一个操作只有在前一个完全结束后才会运行。但使用并发，你可以让两个操作同时运行：

```
dispatch_async(dispatch_queue_create("Foo", DISPATCH_QUEUE_CONCURRENT), ^{
    for (int i = 0; i < 100; i++) {
NSLog(@"Foo");
        usleep(100000);
    }
});

for (int i = 0; i < 100; i++) {
    NSLog(@"Bar");
usleep(50000);
}
```

这将打印“Foo”100次，每次打印后暂停100毫秒，但所有操作都将在一个单独的线程上进行。虽然
正在打印Foo，“Bar”也会以50毫秒的间隔同时打印。理想情况下，你应该看到“Foo”和“Bar”混合在一起的输出

Section 86.2: Executing on the main thread

When performing tasks asynchronously there typically becomes a need to ensure a piece of code is run on the main thread. For example you may want to hit a REST API asynchronously, but put the result in a UILabel on the screen. Before updating the UILabel you must ensure that your code is run on the main thread:

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
    //Perform expensive tasks
    //...

    //Now before updating the UI, ensure we are back on the main thread
    dispatch_async(dispatch_get_main_queue(), ^{
        label.text = //....
    });
}
```

Whenever you update views on the screen, always ensure you are doing so on the main thread, otherwise undefined behavior could occur.

Section 86.3: Running code concurrently -- Running code while running other code

Say you want to perform in action (in this case, logging "Foo"), while doing something else (logging "Bar"). Normally, if you don't use concurrency, one of these actions is going to be fully executed, and the other run will run only after it's completely finished. But with concurrency, you can make both actions run at the same time:

```
dispatch_async(dispatch_queue_create("Foo", DISPATCH_QUEUE_CONCURRENT), ^{
    for (int i = 0; i < 100; i++) {
        NSLog(@"Foo");
        usleep(100000);
    }
});

for (int i = 0; i < 100; i++) {
    NSLog(@"Bar");
    usleep(50000);
}
```

This will log "Foo" 100 times, pausing for 100ms each time it logs, but it will do all this on a separate thread. While Foo is being logged, "Bar" will also be logged in 50ms intervals, at the same time. You should ideally see an output with "Foo"s and "Bars" mixed together

第87章：CAGradientLayer

参数	详细信息
颜色	一个包含CGColorRef对象的数组，定义每个渐变停靠点的颜色。可动画。
位置	一个可选的包含NSNumber对象的数组，定义每个渐变停靠点的位置。可动画。
终点	在图层坐标空间中绘制渐变的终点。可动画。
起点	在图层坐标空间中绘制渐变的起点。可动画。
类型	图层绘制的渐变样式。默认为kCAGradientLayerAxial。

第87.1节：创建CAGradientLayer

```
// 用于承载CAGradientLayer的视图。
let view: UIView = UIView(frame: CGRect(x: 0, y: 0, width: 320, height: 320))

// 初始化渐变图层。
let gradientLayer: CAGradientLayer = CAGradientLayer()

// 设置渐变图层的尺寸。
gradientLayer.frame = view.bounds

// 渐变顶部的颜色。
let topColor: CGColor = UIColor.red.cgColor

// 渐变底部的颜色。
let bottomColor: CGColor = UIColor.yellow.cgColor

// 设置颜色数组。
gradientLayer.colors = [topColor, bottomColor]

// 设置颜色位置。
gradientLayer.locations = [0.0, 1.0]

// 将渐变图层插入视图的图层层级中。
view.layer.insertSublayer(gradientLayer, at: 0)
```

结果：

Chapter 87: CAGradientLayer

Parameter	Details
color	An array of `CGColorRef` objects defining the color of each gradient stop. Animatable.
locations	An optional array of `NSNumber` objects defining the location of each gradient stop. Animatable.
endPoint	The end point of the gradient when drawn in the layer's coordinate space. Animatable.
startPoint	The start point of the gradient when drawn in the layer's coordinate space. Animatable.
type	Style of gradient drawn by the layer. Defaults to `kCAGradientLayerAxial`.

Section 87.1: Creating a CAGradientLayer

```
// View to hold the CAGradientLayer.
let view: UIView = UIView(frame: CGRect(x: 0, y: 0, width: 320, height: 320))

// Initialize gradient layer.
let gradientLayer: CAGradientLayer = CAGradientLayer()

// Set frame of gradient layer.
gradientLayer.frame = view.bounds

// Color at the top of the gradient.
let topColor: CGColor = UIColor.red.cgColor

// Color at the bottom of the gradient.
let bottomColor: CGColor = UIColor.yellow.cgColor

// Set colors.
gradientLayer.colors = [topColor, bottomColor]

// Set locations of the colors.
gradientLayer.locations = [0.0, 1.0]

// Insert gradient layer into view's layer heirarchy.
view.layer.insertSublayer(gradientLayer, at: 0)
```

Result :

第87.2节：在CAGradientLayer中实现颜色变化动画

```
// 获取渐变的当前颜色。
let oldColors = self.gradientLayer.colors

// 定义渐变的新颜色。
let newColors = [UIColor.red.cgColor, UIColor.yellow.cgColor]

// 设置渐变的新颜色。
self.gradientLayer.colors = newColors

// 初始化用于更改渐变颜色的新动画。
let animation: CABasicAnimation = CABasicAnimation(keyPath: "colors")

// 设置当前颜色值。
animation.fromValue = oldColors

// 设置新颜色值。
animation.toValue = newColors

// 设置动画持续时间。
animation.duration = 0.3

// 设置动画完成后移除。
animation.isRemovedOnCompletion = true

// 设置接收器在动画完成后保持可见的最终状态。
animation.fillMode = kCAFillModeForwards

// 设置线性节奏，使动画在其持续时间内均匀发生。
animation.timingFunction = CAMediaTimingFunction(name: kCAMediaTimingFunctionLinear)

// 设置动画的代理。
animation.delegate = self

// 添加动画。
self.gradientLayer.addAnimation(animation, forKey: "animateGradientColorChange")
```

结果：

Section 87.2: Animating a color change in CAGradientLayer

```
// Get the current colors of the gradient.
let oldColors = self.gradientLayer.colors

// Define the new colors for the gradient.
let newColors = [UIColor.red.cgColor, UIColor.yellow.cgColor]

// Set the new colors of the gradient.
self.gradientLayer.colors = newColors

// Initialize new animation for changing the colors of the gradient.
let animation: CABasicAnimation = CABasicAnimation(keyPath: "colors")

// Set current color value.
animation.fromValue = oldColors

// Set new color value.
animation.toValue = newColors

// Set duration of animation.
animation.duration = 0.3

// Set animation to remove once its completed.
animation.isRemovedOnCompletion = true

// Set receiver to remain visible in its final state when the animation is completed.
animation.fillMode = kCAFillModeForwards

// Set linear pacing, which causes an animation to occur evenly over its duration.
animation.timingFunction = CAMediaTimingFunction(name: kCAMediaTimingFunctionLinear)

// Set delegate of animation.
animation.delegate = self

// Add the animation.
self.gradientLayer.addAnimation(animation, forKey: "animateGradientColorChange")
```

Result :

第87.3节：创建水平CAGradientLayer

```
// 用于承载CAGradientLayer的视图。
let view: UIView = UIView(frame: CGRect(x: 0, y: 0, width: 320, height: 320))

// 初始化渐变图层。
```

Section 87.3: Creating a horizontal CAGradientLayer

```
// View to hold the CAGradientLayer.
let view: UIView = UIView(frame: CGRect(x: 0, y: 0, width: 320, height: 320))

// Initialize gradient layer.
```

```
let gradientLayer: CAGradientLayer = CAGradientLayer()

// 设置渐变图层的尺寸。
gradientLayer.frame = view.bounds

// 渐变顶部的颜色。
let topColor: CGColor = UIColor.redColor().CGColor

// 渐变顶部的颜色。
let bottomColor: CGColor = UIColor.yellowColor().CGColor

// 设置颜色。
gradientLayer.colors = [topColor, bottomColor]

// 设置颜色数组。
gradientLayer.startPoint = CGPoint(x: 0.0, y: 0.5)

// 设置起点。
gradientLayer.endPoint = CGPoint(x: 1.0, y: 0.5)

// 设置终点。
view.layer.insertSublayer(gradientLayer, atIndex: 0)
```

结果：

第87.4节：创建一个带有

多种颜色的水平CAGradientLayer

```
// 用于承载CAGradientLayer的视图。
let view: UIView = UIView(frame: CGRect(x: 0, y: 0, width: 320, height: 320))

// 初始化渐变图层。
let gradientLayer: CAGradientLayer = CAGradientLayer()

// 设置渐变图层的尺寸。
gradientLayer.frame = view.bounds

// 渐变顶部的颜色。
let topColor: CGColor = UIColor.greenColor().CGColor

// 渐变中间的颜色。
```



```
let gradientLayer: CAGradientLayer = CAGradientLayer()

// Set frame of gradient layer.
gradientLayer.frame = view.bounds

// Color at the top of the gradient.
let topColor: CGColor = UIColor.redColor().CGColor

// Color at the bottom of the gradient.
let bottomColor: CGColor = UIColor.yellowColor().CGColor

// Set colors.
gradientLayer.colors = [topColor, bottomColor]

// Set start point.
gradientLayer.startPoint = CGPoint(x: 0.0, y: 0.5)

// Set end point.
gradientLayer.endPoint = CGPoint(x: 1.0, y: 0.5)

// Insert gradient layer into view's layer heirarchy.
view.layer.insertSublayer(gradientLayer, atIndex: 0)
```

Result :

Section 87.4: Creating a horizontal CAGradientLayer with multiple colors

```
// View to hold the CAGradientLayer.
let view: UIView = UIView(frame: CGRect(x: 0, y: 0, width: 320, height: 320))

// Initialize gradient layer.
let gradientLayer: CAGradientLayer = CAGradientLayer()

// Set frame of gradient layer.
gradientLayer.frame = view.bounds

// Color at the top of the gradient.
let topColor: CGColor = UIColor.greenColor().CGColor

// Color at the middle of the gradient.
```

```
let middleColor: CGColor = UIColor.blueColor().CGColor

// 渐变底部的颜色。
let bottomColor: CGColor = UIColor.blackColor().CGColor

// 设置颜色。
gradientLayer.colors = [topColor, middleColor, bottomColor]

// 设置起点。
gradientLayer.startPoint = CGPoint(x: 0.0, y: 0.5)

// 设置起点。
gradientLayer.endPoint = CGPoint(x: 1.0, y: 0.5)

// 设置终点。
view.layer.insertSublayer(gradientLayer, atIndex: 0)
```

结果：

第87.5节：创建具有多种颜色的CGGradientLayer

```
// 用于承载CAGradientLayer的视图。
let view: UIView = UIView(frame: CGRect(x: 0, y: 0, width: 320, height: 320))

// 初始化渐变图层。
let gradientLayer: CAGradientLayer = CAGradientLayer()

// 设置渐变图层的尺寸。
gradientLayer.frame = view.bounds

// 渐变顶部的颜色。
let topColor: CGColor = UIColor.blue.cgColor

// 渐变的顶部颜色。
let middleColor: CGColor = UIColor.yellow.cgColor

// 渐变的中间颜色。
let bottomColor: CGColor = UIColor.green.cgColor

// 设置颜色。
gradientLayer.colors = [topColor, middleColor, bottomColor]
```



```
let middleColor: CGColor = UIColor.blueColor().CGColor

// Color at the bottom of the gradient.
let bottomColor: CGColor = UIColor.blackColor().CGColor

// Set colors.
gradientLayer.colors = [topColor, middleColor, bottomColor]

// Set start point.
gradientLayer.startPoint = CGPoint(x: 0.0, y: 0.5)

// Set end point.
gradientLayer.endPoint = CGPoint(x: 1.0, y: 0.5)

// Insert gradient layer into view's layer heirarchy.
view.layer.insertSublayer(gradientLayer, atIndex: 0)
```

Result :

Section 87.5: Creating a CGGradientLayer with multiple colors

```
// View to hold the CAGradientLayer.
let view: UIView = UIView(frame: CGRect(x: 0, y: 0, width: 320, height: 320))

// Initialize gradient layer.
let gradientLayer: CAGradientLayer = CAGradientLayer()

// Set frame of gradient layer.
gradientLayer.frame = view.bounds

// Color at the top of the gradient.
let topColor: CGColor = UIColor.blue.cgColor

// Color at the middle of the gradient.
let middleColor: CGColor = UIColor.yellow.cgColor

// Color at the bottom of the gradient.
let bottomColor: CGColor = UIColor.green.cgColor

// Set colors.
gradientLayer.colors = [topColor, middleColor, bottomColor]
```

```
// 设置颜色的位置。
gradientLayer.locations = [0.0, 0.5, 1.0]

// 将渐变图层插入视图的图层层级中。
view.layer.insertSublayer(gradientLayer, at: 0)
```

结果：

```
// Set locations of the colors.
gradientLayer.locations = [0.0, 0.5, 1.0]

// Insert gradient layer into view's layer heirarchy.
view.layer.insertSublayer(gradientLayer, at: 0)
```

Result :

第88章：Safari服务

第88.1节：使用SafariViewController打开URL

别忘了先导入必要的框架。

```
import SafariServices
//Objective-C
@import SafariServices;
```

实例化一个SafariViewController对象。

```
let safariVC = SFSafariViewController(URL: URL(string: "your_url")!)
//Objective-C
@import SafariServices;
NSURL *URL = [NSURL URLWithString:[NSString stringWithFormat:@"http://www.google.com"]];
SFSafariViewController *sfvc = [[SFSafariViewController alloc] initWithURL:URL];
```

你也可以选择在加载完成后让SafariViewController进入阅读模式（如果可能）。

```
let safariVC = SFSafariViewController(URL: URL(string: "your_url")!, entersReaderIfAvailable: true)
//Objective-C
NSURL *URL = [NSURL URLWithString:[NSString stringWithFormat:@"http://www.google.com"]];
SFSafariViewController *sfvc = [[SFSafariViewController alloc] initWithURL:URL
entersReaderIfAvailable:YES];
```

展示视图控制器。

```
present(safariVC, animated: true, completion: nil)
//Objective-C
[self presentViewController:sfvc animated:YES completion:nil];
```

第88.2节：实现SFSafariViewControllerDelegate

你应该实现SFSafariViewControllerDelegate，这样当用户点击SafariViewController上的完成按钮时，你的类会收到通知，并且你可以关闭它。

首先声明你的类实现该协议。

```
class MyClass: SFSafariViewControllerDelegate {

}
```

实现代理方法以接收关闭通知。

```
func safariViewControllerDidFinish(controller: SFSafariViewController) {
    // 完成时关闭SafariViewController
controller.dismissViewControllerAnimated(true, completion: nil)
}
```

别忘了将你的类设置为SafariViewController的代理。

```
let safariVC = SFSafariViewController(URL: yourURL)
safariVC.delegate = self
```


Chapter 88: Safari Services

Section 88.1: Open a URL with SafariViewController

Don't forget to import the necessary framework first.

```
import SafariServices
//Objective-C
@import SafariServices;
```

Instantiate a `SafariViewController` instance.

```
let safariVC = SFSafariViewController(URL: URL(string: "your_url")!)
//Objective-C
@import SafariServices;
NSURL *URL = [NSURL URLWithString:[NSString stringWithFormat:@"http://www.google.com"]];
SFSafariViewController *sfvc = [[SFSafariViewController alloc] initWithURL:URL];
```

Optionally you can also tell SafariViewController to enter reading mode if possible once it's done loading.

```
let safariVC = SFSafariViewController(URL: URL(string: "your_url")!, entersReaderIfAvailable: true)
//Objective-C
NSURL *URL = [NSURL URLWithString:[NSString stringWithFormat:@"http://www.google.com"]];
SFSafariViewController *sfvc = [[SFSafariViewController alloc] initWithURL:URL
entersReaderIfAvailable:YES];
```

Present the view controller.

```
present(safariVC, animated: true, completion: nil)
//Objective-C
[self presentViewController:sfvc animated:YES completion:nil];
```

Section 88.2: Implement SFSafariViewControllerDelegate

You should implement `SFSafariViewControllerDelegate` so that your class is notified when the user hits the Done button on the SafariViewController and you can dismiss it as well.

First declare your class to implement the protocol.

```
class MyClass: SFSafariViewControllerDelegate {

}
```

Implement the delegate method to be notified on dismissal.

```
func safariViewControllerDidFinish(controller: SFSafariViewController) {
    // Dismiss the SafariViewController when done
    controller.dismissViewControllerAnimated(true, completion: nil)
}
```

Don't forget to set your class as the SafariViewController's delegate.

```
let safariVC = SFSafariViewController(URL: yourURL)
safariVC.delegate = self
```

您可以实现的其他代理方法有：

```
// 当初始 URL 加载完成时调用。
safariViewController(_ controller: SFSafariViewController, didCompleteInitialLoad
didLoadSuccessfully: Bool) { }

// 当用户点击操作按钮时调用。
safariViewController(_ controller: SFSafariViewController, activityItemsFor URL: URL, title:
String?) -> [UIActivity] { }
```

第 88.3 节：向 Safari 阅读列表添加项目

您可以通过调用 SSReadingList 单例的 addItem 方法，将项目添加到用户的 Safari 阅读列表中。

```
let readingList = SSReadingList.default()
readingList?.addItem(with: yourURL, title: "optional title", previewText: "optional preview text")
```

默认的阅读列表可以是nil，如果不允许访问阅读列表。

此外，你可以通过调用supportsURL来检查阅读列表是否支持某个URL。

```
SSReadingList.default().supportsURL(URL(string: "https://example.com")!)
```

这将返回true或false，表示Safari阅读列表是否支持给定的URL。例如，可以用来判断是否显示添加URL到阅读列表的按钮。

Additional delegate methods you can implement are:

```
// Called when the initial URL load is complete.
safariViewController(_ controller: SFSafariViewController, didCompleteInitialLoad
didLoadSuccessfully: Bool) { }

// Called when the user taps an Action button.
safariViewController(_ controller: SFSafariViewController, activityItemsFor URL: URL, title:
String?) -> [UIActivity] { }
```

Section 88.3: Add Items to Safari Reading List

You can add items to a user's Reading List in Safari by calling the `addItem` method on the `SSReadingList` singleton.

```
let readingList = SSReadingList.default()
readingList?.addItem(with: yourURL, title: "optional title", previewText: "optional preview text")
```

The default Reading List can be `nil` if access to the Reading List is not permitted.

Additionally you can check if the Reading List supports a URL by calling `supportsURL`.

```
SSReadingList.default().supportsURL(URL(string: "https://example.com")!)
```

This will return either `true` or `false` indicating if the given URL is supported by Safari Reading List. Use this for example to determine whether to show a button to add a URL to the Reading List.

第89章：CALayer

第89.1节：向CALayer添加变换（平移、旋转、缩放）

基础知识

你可以对图层进行多种不同的变换，但基本的有

- 平移（移动）
- 缩放
- 旋转

要对`CALayer`进行变换，您需要将图层的`transform`属性设置为`CATransform3D`类型。例如，要平移一个图层，可以这样写：

```
myLayer.transform = CATransform3DMakeTranslation(20, 30, 0)
```

单词`Make`用于创建初始变换的名称中：CATransform3D**Make**Translation。后续应用的变换则省略了`Make`。例如，下面是先旋转再平移的操作：

```
let rotation = CATransform3DMakeRotation(CGFloat(30.0 * M_PI / 180.0), 20, 20, 0)
myLayer.transform = CATransform3DTranslate(rotation, 20, 30, 0)
```

现在我们已经了解了如何创建变换的基础，接下来看看每种变换的示例。不过，首先我会展示如何搭建项目，以防你也想试一试。

设置

在以下示例中，我创建了一个单视图应用，并在故事板中添加了一个背景为浅蓝色的UIView。我用以下代码将视图连接到视图控制器：

```
import UIKit

class ViewController: UIViewController {

    var myLayer = CATextLayer()
    @IBOutlet weak var myView: UIView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // 设置子图层
        addSubLayer()
```


Chapter 89: CALayer

Section 89.1: Adding Transforms to a CALayer (translate, rotate, scale)

Basics

There are a number of different transforms you can do on a layer, but the basic ones are

- translate (move)
- scale
- rotate

To do transforms on a `CALayer`, you set the layer's `transform` property to a `CATransform3D` type. For example, to translate a layer, you would do something like this:

```
myLayer.transform = CATransform3DMakeTranslation(20, 30, 0)
```

The word `Make` is used in the name for creating the initial transform: CATransform3D**Make**Translation. Subsequent transforms that are applied omit the `Make`. See, for example, this rotation followed by a translation:

```
let rotation = CATransform3DMakeRotation(CGFloat(30.0 * M_PI / 180.0), 20, 20, 0)
myLayer.transform = CATransform3DTranslate(rotation, 20, 30, 0)
```

Now that we have the basis of how to make a transform, let's look at some examples of how to do each one. First, though, I'll show how I set up the project in case you want to play around with it, too.

Setup

For the following examples I set up a Single View Application and added a `UIView` with a light blue background to the storyboard. I hooked up the view to the view controller with the following code:

```
import UIKit

class ViewController: UIViewController {

    var myLayer = CATextLayer()
    @IBOutlet weak var myView: UIView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // setup the sublayer
        addSubLayer()
```

```
        // 执行变换
        transformExample()
    }

    func addSubLayer() {
myLayer.frame = CGRect(x: 0, y: 0, width: 100, height: 40)
        myLayer.backgroundColor = UIColor.blueColor().CGColor
myLayer.string = "Hello"
        myView.layer.addSublayer(myLayer)
    }

    //******** 用下面的示例替换此函数 ********

    func transformExample() {

        // 在这里添加变换代码 ...


    }

}
```

有许多不同种类的CALayer，但我选择使用CATextLayer，这样变换在视觉上会更清晰。

翻译

平移变换会移动图层。基本语法是

```
CATransform3DMakeTranslation(tx: CGFloat, ty: CGFloat, tz: CGFloat)
```

其中 tx 是 x 坐标的变化，ty 是 y 坐标的变化，tz 是 z 坐标的变化。

示例

在 iOS 中，坐标系的原点位于左上角，因此如果我们想将图层向右移动 90 点
并向下移动 50 点，可以这样做：

```
myLayer.transform = CATransform3DMakeTranslation(90, 50, 0)
```

注意事项

- 请记住，你可以将此代码粘贴到上面项目代码中的 transformExample() 方法中。
- 由于这里我们只处理二维，tz 设置为 0。
- 上图中的红线从原始位置的中心延伸到新位置的中心。这是因为变换是相对于锚点进行的，而锚点默认位于图层的中心。


```
        // do the transform
        transformExample()
    }

    func addSubLayer() {
        myLayer.frame = CGRect(x: 0, y: 0, width: 100, height: 40)
        myLayer.backgroundColor = UIColor.blueColor().CGColor
        myLayer.string = "Hello"
        myView.layer.addSublayer(myLayer)
    }

    //******** Replace this function with the examples below ********

    func transformExample() {

        // add transform code here ...


    }

}
```

There are many different kinds of `CALayer`, but I chose to use `CATextLayer` so that the transforms will be more clear visually.

Translate

The translation transform moves the layer. The basic syntax is

```
CATransform3DMakeTranslation(tx: CGFloat, ty: CGFloat, tz: CGFloat)
```

where `tx` is the change in the x coordinates, `ty` is the change in y, and `tz` is the change in z.

Example

In iOS the origin of the coordinate system is in the top left, so if we wanted to move the layer 90 points to the right and 50 points down, we would do the following:

```
myLayer.transform = CATransform3DMakeTranslation(90, 50, 0)
```

Notes

- Remember that you can paste this into the `transformExample()` method in the project code above.
- Since we are just going to deal with two dimensions here, `tz` is set to `0`.
- The red line in the image above goes from the center of the original location to the center of the new location. That's because transforms are done in relation to the anchor point and the anchor point by default is in the center of the layer.

缩放

缩放变换会拉伸或压缩图层。基本语法是

```
CATransform3DMakeScale(sx: CGFloat, sy: CGFloat, sz: CGFloat)
```

其中 sx、sy 和 sz 分别是对 x、y 和 z 坐标进行缩放（乘法）的数值。

示例

如果我们想将宽度缩小一半，高度放大三倍，可以这样写

```
myLayer.transform = CATransform3DMakeScale(0.5, 3.0, 1.0)
```

注意事项

- 由于我们只在二维空间中操作，所以对 z 坐标乘以 1.0 以保持其不变。
- 上图中的红点表示锚点。注意缩放是相对于锚点进行的，也就是说，所有内容都是向锚点方向拉伸或远离锚点拉伸。

旋转

旋转变换围绕锚点（默认情况下为图层中心）旋转图层。基本语法是

```
CATransform3DMakeRotation(angle: CGFloat, x: CGFloat, y: CGFloat, z: CGFloat)
```

其中angle是图层应旋转的弧度角，x、y和z是旋转的轴。将某个轴设置为0则取消该轴的旋转。

示例

Scale

The scale transform stretches or squishes the layer. The basic syntax is

```
CATransform3DMakeScale(sx: CGFloat, sy: CGFloat, sz: CGFloat)
```

where `sx`, `sy`, and `sz` are the numbers by which to scale (multiply) the x, y, and z coordinates respectively.

Example

If we wanted to half the width and triple the height, we would do the following

```
myLayer.transform = CATransform3DMakeScale(0.5, 3.0, 1.0)
```

Notes

- Since we are only working in two dimensions, we just multiply the z coordinates by 1.0 to leave them unaffected.
- The red dot in the image above represents the anchor point. Notice how the scaling is done in relation to the anchor point. That is, everything is either stretched toward or away from the anchor point.

Rotate

The rotation transform rotates the layer around the anchor point (the center of the layer by default). The basic syntax is

```
CATransform3DMakeRotation(angle: CGFloat, x: CGFloat, y: CGFloat, z: CGFloat)
```

where `angle` is the angle in radians that the layer should be rotated and x, y, and z are the axes about which to rotate. Setting an axis to 0 cancels a rotation around that particular axis.

Example

如果我们想顺时针旋转图层30度，可以这样做：

```
let degrees = 30.0
let radians = CGFloat(degrees * M_PI / 180)
myLayer.transform = CATransform3DMakeRotation(radians, 0.0, 0.0, 1.0)
```

注意事项

- 由于我们在二维空间中工作，只想让xy平面绕z轴旋转。因此我们将 x和 y设置为0.0，将 z设置为1.0。
- 这使图层顺时针旋转。我们也可以通过将 z设置为-1.0来实现逆时针旋转。
- 红点显示了锚点的位置。旋转是围绕锚点进行的。

多重变换

为了组合多个变换，我们可以像这样使用连接（concatination）

```
CATransform3DConcat(a: CATransform3D, b: CATransform3D)
```

但是，我们将依次执行每个变换。第一个变换将在其名称中使用`Make`。后续的变换将不使用`Make`，但它们会将前一个变换作为参数。

示例

这次我们将之前的三个变换全部组合起来。

```
let degrees = 30.0
let radians = CGFloat(degrees * M_PI / 180)

// 平移
var transform = CATransform3DMakeTranslation(90, 50, 0)

// 旋转
transform = CATransform3DRotate(transform, radians, 0.0, 0.0, 1.0)
```

If we wanted to rotate a layer clockwise 30 degrees, we would do the following:

```
let degrees = 30.0
let radians = CGFloat(degrees * M_PI / 180)
myLayer.transform = CATransform3DMakeRotation(radians, 0.0, 0.0, 1.0)
```

Notes

- Since we are working in two dimentions, we only want the xy plane to be rotated around the z axis. Thus we set x and y to `0.0` and set z to `1.0`.
- This rotated the layer in a clockwise direction. We could have rotated counterclockwise by setting z to `-1.0`.
- The red dot shows where the anchor point is. The rotation is done around the anchor point.

Multiple transforms

In order to combine multiple transforms we could use concatination like this

```
CATransform3DConcat(a: CATransform3D, b: CATransform3D)
```

However, we will just do one after another. The first transform will use the `Make` in its name. The following transforms will not use `Make`, but they will take the previous transform as a parameter.

Example

This time we combine all three of the previous transforms.

```
let degrees = 30.0
let radians = CGFloat(degrees * M_PI / 180)

// translate
var transform = CATransform3DMakeTranslation(90, 50, 0)

// rotate
transform = CATransform3DRotate(transform, radians, 0.0, 0.0, 1.0)
```

```
// 缩放
transform = CATransform3DScale(transform, 0.5, 3.0, 1.0)

// 应用变换
myLayer.transform = transform
```

注意事项

- 变换的顺序很重要。
- 所有操作都是相对于锚点（红点）进行的。

关于锚点和位置的说明

我们在上面进行了所有变换而没有改变锚点。不过有时确实需要改变锚点，
比如你想绕中心点以外的某个点旋转。然而，这可能有点棘手。

锚点和位置都在同一个地方。锚点以图层的坐标系单位表示（默认是0.5，0.5），位置以父图层的坐标系表示。它们可以这样设置

```
myLayer.anchorPoint = CGPoint(x: 0.0, y: 1.0)
myLayer.position = CGPoint(x: 50, y: 50)
```

如果只设置锚点而不改变位置，帧会发生变化，使得位置处于正确的位置。更准确地说，帧是基于新的锚点和旧的位置重新计算的。这通常会产生意想不到的结果。以下两篇文章对此有很好的讨论。

- 关于anchorPoint
- 平移 旋转 平移？

另见

- 边框、圆角和阴影在CALayer上的应用
- 使用带有贝塞尔路径的边框为图层

此示例最初来自这个Stack Overflow示例。

第89.2节：阴影

您可以在每个图层上使用5个属性来配置阴影：

- shadowOffset - 此属性用于将阴影向左/右或向上/下移动

```
self.layer.shadowOffset = CGSizeMake(-1, -1); // 向左上移动1像素

self.layer.shadowOffset = CGSizeMake(1, 1); // 向右下移动1像素
```

- shadowColor - 此属性设置阴影的颜色

```
self.layer.shadowColor = [UIColor blackColor].CGColor;
```

- shadowOpacity - 阴影的不透明度，范围从0到1

```
self.layer.shadowOpacity = 0.2;
```

- shadowRadius - 模糊半径（相当于Sketch或Photoshop中的模糊属性）


```
// scale
transform = CATransform3DScale(transform, 0.5, 3.0, 1.0)

// apply the transforms
myLayer.transform = transform
```

Notes

- The order that the transforms are done in matters.
- Everything was done in relation to the anchor point (red dot).

A Note about Anchor Point and Position

We did all our transforms above without changing the anchor point. Sometimes it is necessary to change it, though, like if you want to rotate around some other point besides the center. However, this can be a little tricky.

The anchor point and position are both at the same place. The anchor point is expressed as a unit of the layer's coordinate system (default is `0.5, 0.5`) and the position is expressed in the superlayer's coordinate system. They can be set like this

```
myLayer.anchorPoint = CGPoint(x: 0.0, y: 1.0)
myLayer.position = CGPoint(x: 50, y: 50)
```

If you only set the anchor point without changing the position, then the frame changes so that the position will be in the right spot. Or more precisely, the frame is recalculated based on the new anchor point and old position. This usually gives unexpected results. The following two articles have an excellent discussion of this.

- About the anchorPoint
- Translate rotate translate?

See also

- Border, rounded corners, and shadow on a `CALayer`
- Using a border with a Bezier path for a layer

This example originally comes from this Stack Overflow example.

Section 89.2: Shadows

You can use 5 properties on each layer to configure your shadows:

- `shadowOffset` - this property moves your shadow left/right or up/down

```
self.layer.shadowOffset = CGSizeMake(-1, -1); // 1px left and up

self.layer.shadowOffset = CGSizeMake(1, 1); // 1px down and right
```

- `shadowColor` - this sets the color of your shadow

```
self.layer.shadowColor = [UIColor blackColor].CGColor;
```

- `shadowOpacity` - this is the opacity of the shadow, from `0` to `1`

```
self.layer.shadowOpacity = 0.2;
```

- `shadowRadius` - this is the blur radius (equivalent of the blur property in Sketch or Photoshop)

```
self.layer.shadowRadius = 6;
```

- shadowPath - 这是一个对性能非常重要的属性，当未设置时，iOS 会基于视图的 alpha 通道来绘制阴影，对于带有 alpha 的复杂 PNG 来说，这可能会导致性能开销较大。该属性允许你强制指定阴影的形状，从而提升性能。

Objective-C

```
self.layer.shadowPath = [UIBezierPath bezierPathWithOvalInRect:CGRectMake(0,0,100,100)]; // 这会生成一个圆形阴影
```

Swift 3

```
self.layer.shadowPath = UIBezierPath(ovalIn: CGRect(x: 0, y: 0, width: 100, height: 100)).cgPath
```

第89.3节：带自定义图像的发射器视图

例如，我们将创建一个包含发射器图层并能动画粒子的视图。

```
import QuartzCore

class ConfettiView: UIView {
    // 主发射器图层
    var emitter: CAEmitterLayer!

    // 发射颜色数组
    var colors: [UIColor]!

    // 出现强度
    var intensity: Float!

    private var active :Bool!

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
setup()
    }

    override init(frame: CGRect) {
        super.init(frame: frame)
       setup()
    }

    func setup() {
        // 初始化
colors = [UIColor.redColor(),
                  UIColor.greenColor(),
                  UIColor.blueColor()
                  ]
intensity = 0.2

        active = false
    }

    func startConfetti() {
        emitter = CAEmitterLayer()

发射器.发射器位置 = CGPoint(x: frame.尺寸.宽度 / 2.0, y: -20)
       发射器.发射器形状 = kCAEmitterLayerLine
```



```
self.layer.shadowRadius = 6;
```

- shadowPath - this is an important property for performance, when unset iOS bases the shadow on the alpha channel of the view, which can be performance intensive with a complex PNG with alpha. This property lets you force a shape for your shadow and be more performant because of it.

Objective-C

```
self.layer.shadowPath = [UIBezierPath bezierPathWithOvalInRect:CGRectMake(0,0,100,100)]; //this does a circular shadow
```

Swift 3

```
self.layer.shadowPath = UIBezierPath(ovalIn: CGRect(x: 0, y: 0, width: 100, height: 100)).cgPath
```

Section 89.3: Emitter View with custom image

For example we will create view that contains emitter layer and animates particles.

```
import QuartzCore

class ConfettiView: UIView {
    // main emitter layer
    var emitter: CAEmitterLayer!

    // array of color to emit
    var colors: [UIColor]!

    // intensity of appearance
    var intensity: Float!

    private var active :Bool!

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        setup()
    }

    override init(frame: CGRect) {
        super.init(frame: frame)
        setup()
    }

    func setup() {
        // initialization
        colors = [UIColor.redColor(),
                  UIColor.greenColor(),
                  UIColor.blueColor()
                  ]
        intensity = 0.2

        active = false
    }

    func startConfetti() {
        emitter = CAEmitterLayer()

        emitter.emitterPosition = CGPoint(x: frame.size.width / 2.0, y: -20)
        emitter.emitterShape = kCAEmitterLayerLine
```

```
发射器.发射器大小 = CGSize(宽度: frame.尺寸.宽度, 高度: 1)

        var 细胞 = [CAEmitterCell]()
        for 颜色 in 颜色数组 {
细胞.添加(confettiWithColor(颜色))
        }

发射器.发射器细胞 = 细胞
        图层.添加子图层(发射器)
        激活 = true
    }

    func 停止彩带() {
        发射器?.出生率 = 0
        激活 = false
    }

    func 彩带颜色(颜色: UIColor) -> CAEmitterCell {
        let 彩带 = CAEmitterCell()

彩带.出生率 = 10.0 * 强度
        彩带.生命周期 = 180.0 * 强度
        彩带.生命周期范围 = 0
彩带.颜色 = 颜色.CGColor
        彩带.速度 = CGFloat(350.0 * 强度)
        彩带.速度范围 = CGFloat(40.0 * 强度)
        彩带.发射经度 = CGFloat(M_PI)
confetti.emissionRange = CGFloat(M_PI_4)
        confetti.spin = CGFloat(3.5 * 强度)
        confetti.spinRange = CGFloat(4.0 * 强度)

        // 警告：图层可以将此属性设置为 CGImageRef，以显示图像作为其
内容。
confetti.contents = UIImage(named: "confetti")?.CGImage
        return confetti
    }

    internal func isActive() -> Bool {
        return self.active
    }
}
```

您需要添加“confetti”图像或使用confetti.contentsRect定义矩形

第89.4节：圆角

```
layer.masksToBounds = true;
layer.cornerRadius = 8;
```

第89.5节：使用 CAEmitterLayer 创建粒子

CAEmitterLayer类为 Core Animation 提供了粒子发射系统。粒子由CAEmitterCell的实例定义。

粒子绘制在图层的背景色和边框之上。

```
        var emitter = CAEmitterLayer()

发射器。emitterPosition = CGPoint(x: frame.size.width / 2.0, y: -20)
```



```
        emitter.emitterSize = CGSize(width: frame.size.width, height: 1)

        var cells = [CAEmitterCell]()
        for color in colors {
            cells.append(confettiWithColor(color))
        }

        emitter.emitterCells = cells
        layer.addSublayer(emitter)
        active = true
    }

    func stopConfetti() {
        emitter?.birthRate = 0
        active = false
    }

    func confettiWithColor(color: UIColor) -> CAEmitterCell {
        let confetti = CAEmitterCell()

        confetti.birthRate = 10.0 * intensity
        confetti.lifetime = 180.0 * intensity
        confetti.lifetimeRange = 0
        confetti.color = color.CGColor
        confetti.velocity = CGFloat(350.0 * intensity)
        confetti.velocityRange = CGFloat(40.0 * intensity)
        confetti.emissionLongitude = CGFloat(M_PI)
        confetti.emissionRange = CGFloat(M_PI_4)
        confetti.spin = CGFloat(3.5 * intensity)
        confetti.spinRange = CGFloat(4.0 * intensity)

        // WARNING: A layer can set this property to a CGImageRef to display the image as its
contents.
        confetti.contents = UIImage(named: "confetti")?.CGImage
        return confetti
    }

    internal func isActive() -> Bool {
        return self.active
    }
}
```

You need to add "confetti" image or define rect with **confetti.contentsRect**

Section 89.4: Rounded corners

```
layer.masksToBounds = true;
layer.cornerRadius = 8;
```

Section 89.5: Creating particles with CAEmitterLayer

The **CAEmitterLayer** class provides a particle emitter system for Core Animation. The particles are defined by instances of **CAEmitterCell**.

The particles are drawn above the layer's background color and border.

```
        var emitter = CAEmitterLayer()

        emitter.emitterPosition = CGPoint(x: frame.size.width / 2.0, y: -20)
```

```
发射器。emitterShape = kCAEmitterLayerLine
        发射器。emitterSize = CGSize(width: frame.size.width, height: 1)

        发射器。emitterCells = cells
图层。addSublayer(emitter)
```

第89.6节：如何向CALayer添加UIImage

你可以通过使用视图的layer的contents属性，简单地向视图添加图像：

```
myView.layer.contents = UIImage(named: "star")?.CGImage
```

- 注意，UIImage需要转换为CGImage。

如果你希望将图像添加到它自己的图层中，可以这样做：

```
let myLayer = CALayer()
let myImage = UIImage(named: "star")?.CGImage
myLayer.frame = myView.bounds
myLayer.contents = myImage
myView.layer.addSublayer(myLayer)
```

修改外观

上述代码生成了如下视图。浅蓝色是UIView，深蓝色的星星是UIImage。

如你所见，它看起来有像素化现象。这是因为UIImage比UIView小，所以它被缩放以填充视图，这是默认行为，除非你另有指定。

下面的示例展示了图层的contentsGravity属性的不同变化。代码如下：

```
myView.layer.contents = UIImage(named: "star")?.CGImage
myView.layer.contentsGravity = kCAGravityTop
myView.layer.geometryFlipped = true
```

在iOS中，如果你使用顶部或底部对齐，可能需要将geometryFlipped属性设置为true，否则它的表现会与你预期相反。（只有重力方向垂直翻转，内容渲染不会翻转。如果内容被翻转导致问题，请参见this Stack Overflow answer。）

下面每个contentsGravity设置都有两个UIView示例，一个视图比UIImage大，另一个比UIImage小。这样你可以看到缩放和重力的效果。

kCAGravityResize

这是默认值。


```
        emitter.emitterShape = kCAEmitterLayerLine
        emitter.emitterSize = CGSize(width: frame.size.width, height: 1)

        emitter.emitterCells = cells
        layer.addSublayer(emitter)
```

Section 89.6: How to add a UIImage to a CALayer

You can add an image to a view's `layer` simply by using its `contents` property:

```
myView.layer.contents = UIImage(named: "star")?.CGImage
```

- Note that the `UIImage` needs to be converted to a `CGImage`.

If you wish to add the image in its own layer, you can do it like this:

```
let myLayer = CALayer()
let myImage = UIImage(named: "star")?.CGImage
myLayer.frame = myView.bounds
myLayer.contents = myImage
myView.layer.addSublayer(myLayer)
```

Modifying the appearance

The above code produces a view like this. The light blue is the `UIView` and the dark blue star is the `UIImage`.

As you can see, though, it looks pixelated. This is because the `UIImage` is smaller than the `UIView` so it is being scaled to fill the view, which is the default it you don't specify anything else.

The examples below show variations on the layer's `contentsGravity` property. The code looks like this:

```
myView.layer.contents = UIImage(named: "star")?.CGImage
myView.layer.contentsGravity = kCAGravityTop
myView.layer.geometryFlipped = true
```

In iOS, you may want to set the `geometryFlipped` property to `true` if you are doing anything with top or bottom gravity, otherwise it will be the opposite of what you expect. (Only the gravity is flipped vertically, not the content rendering. If you are having trouble with the content being flipped, see this Stack Overflow answer.)

There are two `UIView` examples below for every `contentsGravity` setting, one view is larger than the `UIImage` and the other is smaller. This way you can see the effects of the scaling and gravity.

kCAGravityResize

This is the default.

kCAGravityResizeAspect

kCAGravityResizeAspectFill

kCAGravityCenter

kCAGravityTop

kCAGravityResizeAspect

kCAGravityResizeAspectFill

kCAGravityCenter

kCAGravityTop

kCAGravityBottom

kCAGravityLeft

kCAGravityRight

kCAGravityTopLeft

kCAGravityBottom

kCAGravityLeft

kCAGravityRight

kCAGravityTopLeft

kCAGravityTopRight

kCAGravityBottomLeft

kCAGravityBottomRight

相关

- 视图的内容模式属性
- 在drawRect中使用CGContextDrawImage绘制UIImage
- CALayer教程：入门

注意事项

- 此示例最初来自这个Stack Overflow答案。

第89.7节：禁用动画

CALayer属性动画默认启用。当不需要时，可以按如下方式禁用。

Swift

```
CATransaction.begin()
CATransaction.setDisableActions(true)

// 更改你不想动画的图层属性
```


kCAGravityTopRight

kCAGravityBottomLeft

kCAGravityBottomRight

Related

- Content mode property of a view
- Drawing a `UIImage` in `drawRect` with `CGContextDrawImage`
- CALayer Tutorial: Getting Started

Notes

- This example comes originally from this Stack Overflow answer.

Section 89.7: Disable Animations

`CALayer` property animations are enabled by default. When this is undesirable, they can be disabled as follows.

Swift

```
CATransaction.begin()
CATransaction.setDisableActions(true)

// change layer properties that you don't want to animate
```

```
CATransaction.commit()
```

Objective-C

```
[CATransaction begin];
[CATransaction setDisableActions:YES];

// 更改你不想动画的图层属性

[CATransaction commit];
```



```
CATransaction.commit()
```

Objective-C

```
[CATransaction begin];
[CATransaction setDisableActions:YES];

// change layer properties that you don't want to animate

[CATransaction commit];
```

第90章：iOS - 使用Robbie Hanson框架实现XMPP

第90.1节：iOS XMPP Robbie Hanson示例与Openfire

SRXMPPDemo
在此下载示例及所有类 - https://github.com/SahebRoy92/SRXMPPDemo

这是一个使用Objective C实现的XMPP演示，包含了各种简单和复杂的功能。所有XMPP的功能均通过"in band"的xmpp函数完成。该项目包含的部分功能有——

SRXMPP - 一个几乎包含一对一聊天应用所需所有功能的包装单例类。

- 一对一聊天
- 聊天（文本消息）的核心数据实现，因此可以保存之前的消息和离线消息。
- 通过Robbie Hanson自有框架提供的XML和核心数据实现vCard（用户个人资料信息，包括自己和他人）。
- 好友状态的可用性（在线/离线/正在输入）

操作步骤

如果你想将此项目作为参考，可以执行以下操作——

1. 在一台在线服务器上安装Openfire - 租用服务器，安装Openfire。

2. 想在自己的电脑上无忧尝试 - 你需要下载、安装并设置3个东西才能开始

a. Java -

- 下载并安装适用于Mac的Java。

b. XAMPP -

- 安装XAMPP相对简单。
- 安装后，只需启动XAMPP并启动数据库(SQL)和Apache服务器。

Chapter 90: iOS - Implementation of XMPP with Robbie Hanson framework

Section 90.1: iOS XMPP Robbie Hanson Example with Openfire

SRXMPPDemo
Download the example and all the classes here - https://github.com/SahebRoy92/SRXMPPDemo

A demo on XMPP in Objective C, with various simple and complex features implemented in it. All the features of XMPP is done by **"in band"** xmpp functions. Few features this project contains are --

SRXMPP - A wrapper Singleton class that almost has all features needed for one-to-one chat application.

- one to one chat
- Core data implementation of chat (text message) thus having saving of previous messages, offline messages.
- implementation of vCard(profile information of user, own and others too) from XML and Core Data provided by Robbie Hanson's own framework.
- availability of friends status (online/offline/typing)

Steps to follow

You want to use this project as a reference then you can do the following--

1. Installed Openfire in a live server - Rent a server, install openfire.

2. Want to try it out without a hassle in your own computer - You need to download, install and setup 3 things to start

a. Java -

- Download and install Java for Mac.

b. XAMPP -

- Install XAMPP is relatively easy.
- After installation just start the XAMPP and start **Database(SQL)** and **Apache Server**.

- 然后打开浏览器，粘贴此URL[http://localhost/phpmyadmin/]
- 。从左侧面板创建一个新的数据库。
- **给数据库命名，名称可以随意，但请记住这个名字，假设我们命名为ChatDB**

c. Openfire -

- 安装Openfire并运行应用程序，然后"启动Openfire"

- Then open browser and paste this URL [http://localhost/phpmyadmin/]
- . Create a new DB from the left hand side panel.
- **Name the DB anything but remember this name, suppose we name it ChatDB**

c. Openfire -

- Install Openfire and run the application and "Start Openfire"

- 打开浏览器并粘贴此网址 - http://localhost:9090/setup/index.jsp
- 进行正常安装
  - 选择语言 >
  - 服务器设置，保持原样，直接继续 >
  - 数据库设置，保持为“标准数据库连接”如所选 >
  - 数据库设置 - 标准连接”。现在记住你设置的数据库名称是**ChatDB**。
  - 选择数据库驱动预设为*“*MySQL*”。JDBC驱动类保持不变。现在在数据库URL中你可以看到括号中提到的主机名和数据库名。只需将主机名改为“localhost”，数据库名改为“ChatDB”，或者你之前在设置XAMPP时设置的其他数据库名。用户名和密码保持为空。填写如图所示的详细信息

.

  - 接下来通过设置用户名和密码并确认完成安装。就这样，你完成了Openfire的设置。

现在到了你需要在代码中修改一个小细节的部分。

#**重要** 我们需要进入类**SRXMPP.m**，找到NSString外部变量**SRXMPP_Hostname**（在顶部）并将其值覆盖为

- 安装了 OpenFire 的服务器的 IP，**或者**
- 如果你是在本地安装的，请将该值覆盖为**"localhost"**。

就是这样，你已经准备好使用这个示例项目，开始编码并将其打造为你自己的更好项目了。

这个入门包将帮助你更好地理解 XMPP 结构，并掌握 XMPP 协议。

你可以在本网站找到其他 XMPP 协议 - https://xmpp.org/rfcs/rfc3920.html

开发工作仍在进行中，部分内容我希望以后能加入

1. 群聊
2. 图片发送支持

简而言之，这个示例项目连同单例几乎包含了一对一聊天应用所需的所有功能。

- Open Browser and Paste this URL - http://localhost:9090/setup/index.jsp
- Do normal setup
  - Select Language >
  - Server settings, leave as it is, just do continue >
  - Database Settings, leave as it is as "Standard Database Connection as selected >
  - Database Settings - Standard Connection". Now remember the name of the DB you set was **ChatDB**.
  - Select Database Driver Presets as *"*MySQL*". Leave JDBC Driver Class as it is. Now in the Database URL you can see, brackets mentioning hostname and Database Name. Just change Hostname to **"localhost"**, and database name to **"ChatDB"**, or any other name of DB you have set earlier, while seting up XAMPP. Leave the Username and password as blank.Fill up details like the image here

.

  - Next complete setup by giving a username and password and reconfirming it. Thats it your done Setting up Openfire.

Now the part comes when you have to change a tiny detail in the code.

#**Important** We need to go to the class - **SRXMPP.m**, locate the NSString extern **SRXMPP_Hostname** (in the top) and overwrite the value of it to the

- IP of the server where OpenFire is installed , **OR**
- if you have installed it locally, overwrite the value to - **"localhost"**.

Thats it, you are ready to use this example project and start coding and making it into a better project of your own.

This starter pack will help you in understanding XMPP structure better as well as getting a grasp into XMPP protocols.

You can find other XMPP protocols here in this site - https://xmpp.org/rfcs/rfc3920.html

Development is still left and parts where I hope to include them later on

1. Group Chat
2. Image sending support

In short this example project along with the singleton has almost all features that are needed for a One-to-One chat application to have.

第91章：Swift 与 Objective-C 互操作性

第91.1节：在 Swift 中使用 Objective-C 类

> 如果你有一个现有的类想要使用，执行**步骤2**然后跳到**步骤5**。（在某些情况下，我不得不在较旧的 ObjC 文件中显式添加`#import <Foundation/Foundation.h`）

步骤1：添加Objective-C实现文件 -- .m

向你的类中添加一个`.m`文件，并命名为`CustomObject.m`

步骤2：添加桥接头文件

添加你的.m文件时，系统很可能会弹出如下提示：

点击YES！

如果没有看到提示，或者不小心删除了桥接头文件，请在项目中添加一个新的`.h`文件，命名为`<#YourProjectName#>-Bridging-Header.h`

在某些情况下，尤其是使用ObjC框架时，你不会显式添加Objective-C类，Xcode无法找到链接器。此时，创建上述命名的`.h`文件，确保在目标的项目设置中链接其路径，如下所示：

Chapter 91: Swift and Objective-C interoperability

Section 91.1: Using Objective-C Classes in Swift

> If you have an existing class that you'd like to use, perform **Step 2** and then skip to **Step 5**. (For some cases, I had to add an explicit `#import <Foundation/Foundation.h` to an older ObjC File)

Step 1: Add Objective-C Implementation -- .m

Add a `.m` file to your class, and name it `CustomObject.m`

Step 2: Add Bridging Header

When adding your `.m` file, you'll likely be hit with a prompt that looks like this:

Click **YES** !

If you did not see the prompt, or accidentally deleted your bridging header, add a new `.h` file to your project and name it `<#YourProjectName#>-Bridging-Header.h`

In some situations, particularly when working with ObjC frameworks, you don't add an Objective-C class explicitly and Xcode can't find the linker. In this case, create your `.h` file named as mentioned above, then make sure you link its path in your target's project settings like so:

注意

最佳实践是使用$(SRCROOT)宏来链接项目，这样如果你移动项目，或者与他人通过远程仓库协作，项目仍然能正常工作。 $(SRCROOT)可以理解为包含你的 .xcodeproj文件的目录。它可能看起来像这样：

$(SRCROOT)/Folder/Folder/<#YourProjectName#>-Bridging-Header.h

步骤3：添加Objective-C头文件 -- .h

添加另一个`.h`文件并命名为`CustomObject.h`

步骤4：构建你的Objective-C类

在CustomObject.h中

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

@interface CustomObject : NSObject

@property (strong, nonatomic) id someProperty;

- (void) someMethod;

@end
```

在CustomObject.m中

Note

It's best practice to link your project using the `$(SRCROOT)` macro so that if you move your project, or work on it with others using a remote repo, it will still work. `$(SRCROOT)` can be thought of as the directory that contains your .xcodeproj file. It might look like this:

`$(SRCROOT)/Folder/Folder/<#YourProjectName#>-Bridging-Header.h`

Step 3: Add Objective-C Header -- .h

Add another `.h` file and name it `CustomObject.h`

Step 4: Build your Objective-C Class

In `CustomObject.h`

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

@interface CustomObject : NSObject

@property (strong, nonatomic) id someProperty;

- (void) someMethod;

@end
```

In `CustomObject.m`

```
#import "CustomObject.h"

@implementation CustomObject

- (void) someMethod {
    NSLog(@"SomeMethod Ran");
}

@end
```

步骤5：将类添加到Bridging-Header

在YourProject-Bridging-Header.h中：

```
#import "CustomObject.h"
```

步骤6：使用你的对象

在SomeSwiftFile.swift中：

```
var instanceOfCustomObject: CustomObject = CustomObject()
instanceOfCustomObject.someProperty = "Hello World"
println(instanceOfCustomObject.someProperty)
instanceOfCustomObject.someMethod()
```

无需显式导入，这就是桥接头文件的作用。

第91.2节：在Objective-C中使用Swift类

步骤1：创建新的Swift类

向你的项目中添加一个`.swift`文件，命名为`MySwiftObject.swift`

在`MySwiftObject.swift`中：

```
import Foundation

class MySwiftObject : NSObject {

    var someProperty: AnyObject = "Some Initializer Val"

    init() {}

    func someFunction(someArg:AnyObject) -> String {
        var returnVal = "你发送给我的 \(someArg)"
        return returnVal
    }

}
```

步骤 2：将 Swift 文件导入到 ObjC 类中

在 `SomeRandomClass.m` 中：

```
#import "<#YourProjectName#>-Swift.h"
```

文件：`<#YourProjectName#>-Swift.h` 应该已经在你的项目中自动创建，即使你看不到它。


```
#import "CustomObject.h"

@implementation CustomObject

- (void) someMethod {
    NSLog(@"SomeMethod Ran");
}

@end
```

Step 5: Add Class to Bridging-Header

In `YourProject-Bridging-Header.h`:

```
#import "CustomObject.h"
```

Step 6: Use your Object

In `SomeSwiftFile.swift`:

```
var instanceOfCustomObject: CustomObject = CustomObject()
instanceOfCustomObject.someProperty = "Hello World"
println(instanceOfCustomObject.someProperty)
instanceOfCustomObject.someMethod()
```

No need to import explicitly, that's what the bridging header is for.

Section 91.2: Using Swift Classes in Objective-C

Step 1: Create New Swift Class

Add a `.swift` file to your project, and name it `MySwiftObject.swift`

In `MySwiftObject.swift`:

```
import Foundation

class MySwiftObject : NSObject {

    var someProperty: AnyObject = "Some Initializer Val"

    init() {}

    func someFunction(someArg:AnyObject) -> String {
        var returnVal = "You sent me \(someArg)"
        return returnVal
    }

}
```

Step 2: Import Swift Files to ObjC Class

In `SomeRandomClass.m`:

```
#import "<#YourProjectName#>-Swift.h"
```

The file:`<#YourProjectName#>-Swift.h` should already be created automatically in your project, even if you can not

看不到它。

步骤 3：使用你的类

```
MySwiftObject * myOb = [MySwiftObject new];
NSLog(@"MyOb.someProperty: %@", myOb.someProperty);
myOb.someProperty = @"Hello World";
NSLog(@"MyOb.someProperty: %@", myOb.someProperty);
NSString * retString = [myOb someFunction:@"Arg"];
NSLog(@"RetString: %@", retString);
```

注意：

1. 代码补全的表现没有达到我期望的准确度。在我的系统上，使用“cmd + r”快速构建似乎有助于 Swift 找到一些 Objc 代码，反之亦然。似乎有助于 Swift 找到一些 Objc 代码，反之亦然。

2.如果你向旧项目中添加了.swift文件并出现错误：dyld: Library not loaded: @rpath/libswift_stdlib_core.dylib，尝试完全重新启动 Xcode。

3. 虽然最初可以通过使用@objc前缀在 Objective-C 中使用纯 Swift 类，但在 Swift 2.0 之后，这已不再可能。请参阅编辑历史以获取原始解释。如果未来 Swift 版本重新启用此功能，答案将相应更新。

see it.

Step 3: Use your class

```
MySwiftObject * myOb = [MySwiftObject new];
NSLog(@"MyOb.someProperty: %@", myOb.someProperty);
myOb.someProperty = @"Hello World";
NSLog(@"MyOb.someProperty: %@", myOb.someProperty);
NSString * retString = [myOb someFunction:@"Arg"];
NSLog(@"RetString: %@", retString);
```

Note:

1. CodeCompletion wasn't behaving as accurately as I'd like it to. On my system, running a quick build w/ "cmd + r" seemed to help Swift find some of the Objc code and vice versa.

2. If you add `.swift` file to an older project and get error: `dyld: Library not loaded: @rpath/libswift_stdlib_core.dylib`, try completely restarting Xcode.

3. While it was originally possible to use pure Swift classes in Objective-C by using the `@objc` prefix, after Swift 2.0, this is no longer possible. See edit history for original explanation. If this functionality is reenabled in future Swift versions, the answer will be updated accordingly.

第92章：自定义字体

第92.1节：嵌入自定义字体

> **自定义字体支持**
> 想要使用自定义字体的应用程序现在可以将这些字体包含在其应用程序包中，并通过在Info.plist文件中包含UIAppFonts键来向系统注册这些字体。该键的值是一个字符串数组，用于标识应用程序包中的字体文件。当系统检测到该键时，会加载指定的字体并使其可供应用程序使用。

字体设置在Info.plist后，您可以像使用其他字体一样在IB中或通过编程方式使用自定义字体。

1. 将字体拖放到Xcode的Supporting Files文件夹中。别忘了在“Add to targets”部分勾选您的应用程序。从此刻起，您可以在IB中使用该字体，并从字体面板中选择它。

2. 要使该字体在设备上可用，打开Info.plist并添加“Fonts provided by application”键（UIAppFonts）。将字体名称作为Item 0键的值添加。注意：字体名称可能与您的字体文件名不同。

3. 使用以下代码片段获取自定义添加的字体名称

[Swift 3]

```
for family in UIFont.familyNames {
            print("\(family)")

            for name in UIFont.fontNames(forFamilyName: family) {
                print("   \(name)")
            }
        }
```

[Objective - C]

```
for (NSString *familyName in [UIFont familyNames]){
        NSLog(@"Family name: %@", familyName);
        for (NSString *fontName in [UIFont fontNamesForFamilyName:familyName]) {
            NSLog(@"--Font name: %@", fontName);
        }
    }
```

第92.2节：在

Storyboard中将自定义字体应用于控件

以下示例展示了如何将自定义字体应用于导航栏，并包含了针对Xcode中一些奇怪行为的修复。也可以将自定义字体应用于任何其他UI控件，如UILabel，

Chapter 92: Custom fonts

Section 92.1: Embedding custom fonts

> **Custom Font Support**
> Applications that want to use custom fonts can now include those fonts in their application bundle and register those fonts with the system by including the UIAppFonts key in their Info.plist file. The value of this key is an array of strings identifying the font files in the application's bundle. When the system sees the key, it loads the specified fonts and makes them available to the application.

Once the fonts have been set in the `Info.plist`, you can use your custom fonts as any other font in IB or programmatically.

1. Drag and drop your font to Xcode Supporting Files folder. Don't forget to mark your app at "Add to targets" section. From this moment you can use this font in IB and choose it from font pallet.

2. To make this font available on the device, open `Info.plist` and add `Fonts provided by application key` (UIAppFonts). Add font name as the value to the Item 0 key. Note: Font name can vary from your font file name.

3. Get the custom added font name using below snippet

[*Swift 3*]

```
for family in UIFont.familyNames {
            print("\(family)")

            for name in UIFont.fontNames(forFamilyName: family) {
                print("   \(name)")
            }
        }
```

[*Objective - C*]

```
for (NSString *familyName in [UIFont familyNames]){
        NSLog(@"Family name: %@", familyName);
        for (NSString *fontName in [UIFont fontNamesForFamilyName:familyName]) {
            NSLog(@"--Font name: %@", fontName);
        }
    }
```

Section 92.2: Applying custom fonts to controls within a Storyboard

The following example shows how to apply custom fonts to a Navigation Bar and includes fixes for some quirky behaviors found in Xcode. One also may apply the custom fonts to **any other UIControls** such as **UILabels**,

UIButton等，方法是在将自定义字体添加到项目后，通过属性检查器进行设置。请注意文末附近的外部链接，包含可用示例和视频。

1. 在导航控制器中选择您的导航栏

2. 在属性检查器中更改标题字体

（在 Xcode 识别新字体之前，您可能需要切换导航栏的栏色）

注意事项（警告）

已验证此方法在 Xcode 7.1.1 及以上版本有效。（请参见下面的示例）

1. 您确实需要切换导航栏的栏色，字体才会生效（这似乎是 Xcode 的一个漏洞；您可以切换回默认，字体就会生效）
2. 如果选择系统字体 ~ 请确保字号不是 0.0（否则新字体将被忽略）

3. 当视图层级中只有一个导航栏时，这似乎能正常工作。看来同一堆栈中的次级导航栏会被忽略。（注意，如果显示主导航控制器的导航栏，所有其他自定义导航栏设置都会被忽略）

注意事项（二）

其中一些内容有重复，说明它们非常值得注意。

UIButtons, and more by using the attributes inspector after the custom font is added to the project. Please note the external links to working samples and videos near the bottom.

1. Select Your Navigation Bar within your Navigation Controller

2. Change the Title Font in the Attributes Inspector

(You will likely need to toggle the Bar Tint for the Navigation Bar before Xcode picks up the new font)

Notes (Caveats)

Verified that this does work on Xcode 7.1.1+. (**See the Samples below**)

1. You do need to toggle the nav bar tint before the font takes effect (seems like a bug in Xcode; you can switch it back to default and font will stick)
2. If you choose a system font ~ Be sure to make sure the size is not 0.0 (Otherwise the new font will be ignored)

3. Seems like this works with no problem when only one NavBar is in the view hierarchy. It appears that secondary NavBars in the same stack are ignored. (Note that if you show the master navigation controller's navBar all the other custom navBar settings are ignored).

Gotchas (deux)

Some of these are repeated which means they are very likely worth noting.

1. 有时故事板的 XML 会损坏。这需要您以
   源代码模式查看故事板结构（右键点击故事板文件 > 以...方式打开）
2. 在某些情况下，与用户定义的运行时属性关联的 navigationItem 标签被设置为 view 标签的子节点，而不是 view controller 标签的子节点。如果是这样，请将其从标签之间移除以确保正常运行。
3. 切换导航栏颜色以确保使用自定义字体。
4. 验证字体的大小参数，除非使用动态字体样式
5. 视图层级会覆盖设置。看起来每个堆栈只能使用一种字体。

结果

示例

- 展示多种字体的高级项目视频
- 简单源码下载
- 高级项目下载 ~ 展示多种导航栏字体和自定义字体解决方法
- 展示多种字体和自定义字体的视频

处理自定义字体

注意 ~ 可以在 Code With Chris 网站找到一份很好的检查清单，并且可以查看示例下载项目。

如果你有自己的字体并想在故事板中使用，那么在以下SO 问题中有一套不错的答案。其中一个答案列出了这些步骤。

1. 获取您的自定义字体文件（.ttf，.ttc）
2. 将字体文件导入到您的Xcode项目中
3. 在app-info.plist中，添加一个名为“Fonts provided by application”的键。它是数组类型，将所有字体文件名（包括文件扩展名）添加到数组中。
4. 在Storyboard中，选中导航栏，进入属性检查器，点击字体选择区域右侧的图标按钮。在弹出面板中，选择字体为自定义，并选择您嵌入字体的字体族名称。

自定义字体解决方法

所以Xcode看起来可以自然地处理UINavigationItem上的自定义字体，但该功能实际上没有正确更新（所选字体被忽略）。

1. Sometimes the storyboard xml gets corrupt. This requires you to review the structure in Storyboard as Source Code mode (right click the storyboard file > Open As ...)
2. In some cases the navigationItem tag associated with user defined runtime attribute was set as an xml child of the view tag instead of the view controller tag. If so remove it from between the tags for proper operation.
3. Toggle the NavBar Tint to ensure the custom font is used.
4. Verify the size parameter of the font unless using a dynamic font style
5. View hierarchy will override the settings. It appears that one font per stack is possible.

Result

Samples

- Video Showing Multiple Fonts In Advanced Project
- Simple Source Download
- Advanced Project Download ~ Shows Multiple NavBar Fonts & Custom Font Workaround
- Video Showing Multiple Fonts & Custom Fonts

Handling Custom Fonts

Note ~ A nice checklist can be found from the Code With Chris website and you can see the sample download project.

If you have your own font and want to use that in your storyboard, then there is a decent set of answers on the following SO Question. One answer identifies these steps.

1. Get you custom font file(.ttf,.ttc)
2. Import the font files to your Xcode project
3. In the app-info.plist,add a key named Fonts provided by application.It's an array type , add all your font file names to the array,note:including the file extension.
4. In the storyboard , on the NavigationBar go to the Attribute Inspector,click the right icon button of the Font select area.In the popup panel , choose Font to Custom, and choose the Family of you embeded font name.

Custom Font Workaround

So Xcode naturally looks like it can handle custom fonts on UINavigationItem but that feature is just not updating properly (The font selected is ignored).

> **解决方法如下：**
>
> 一种方法是结合Storyboard和添加一行代码来修复：首先向视图控制器添加一个UIView（UIButton、UILabel或其他UIView子类）（不是导航项.....Xcode目前不允许这样做）。添加控件后，您可以在Storyboard中修改字体，并将其作为Outlet引用添加到您的视图控制器。只需将该视图赋值给UINavigationItem.titleView。如果需要，也可以在代码中设置文本名称。已报告的Bug（23600285）。

```
@IBOutlet var customFontTitleView: UIButton!

//过了一段时间.....
self.navigationItem.titleView = customFontTitleView
```

注意 - 这个示例来源于我在SO（Stack Overflow）上发布的一个回答（here）。

第92.3节：使用Storyboard的自定义字体

Storyboard中UI组件的自定义字体可以通过用户定义运行时属性和Storyboard中的类别轻松实现。

其优点包括，

- 无需为UI元素定义outlet
- 无需通过代码为元素设置字体。

> **操作步骤**
>
> 1. 字体文件：将字体文件（.ttf）添加到应用程序包中，并在Info.plist中添加字体条目在应用程序提供的字体下，如本自定义字体文档所示。
> 2. 定义类别：添加一个类似UIKit+IBExtensions的文件，并为UI元素添加类别，如

> **To workaround this:**
>
> One way is to fix using the storyboard and adding a line of code: First add a UIView (UIButton, UILabel, or some other UIView subclass) to the View Controller (Not the Navigation Item...Xcode is not currently allowing one to do that). After you add the control you can modify the font in the storyboard and add a reference as an outlet to your View Controller. Just assign that view to the UINavigationItem.titleView. You could also set the text name in code if necessary. Reported Bug (23600285).

```
@IBOutlet var customFontTitleView: UIButton!

//Sometime later...
self.navigationItem.titleView = customFontTitleView
```

Note - This example is derived from an answer I posted on SO (here).

Section 92.3: Custom Fonts with Storyboard

Custom Fonts for UI components from storyboard can be easily achieved with User Defined Runtime Attributes in storyboard and Categories.

The advantages are like,

- No need to define outlets for the ui element
- No need to set font for elements programmatically.

> **Steps to follow**
>
> 1. **Font File:** Add the Font file (.ttf) to the application bundle and add the entry for the font in Info.plist under ***Font provided by application*** as in this documentation of custom fonts.
> 2. **Define Categories:** Add a file like ***UIKit+IBExtensions*** and add the categories for UI elements like

> UILabel、UIButton 等你想设置自定义字体的控件。所有类别都会有一个自定义属性，比如fontName。这个属性稍后将在故事板中用于设置自定义字体（如步骤4所示）。

UIKit+IBExtensions.h

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

// UILabel 的类别扩展
@interface UILabel (IBExtensions)

@property (nonatomic, copy) NSString *fontName;
@end

// UITextField 的类别扩展
@interface UITextField (IBExtensions)

@property (nonatomic, copy) NSString *fontName;
@end

// UIButton 的类别扩展
@interface UIButton (IBExtensions)

@property (nonatomic, copy) NSString *fontName;
@end
```

> 3. Getter 和 Setter： 为每个类别定义 fontName 属性的 getter 和 setter
> 已添加。

UIKit+IBExtensions.m

```
#import "UIKit+IBExtensions.h"

@implementation UILabel (IBExtensions)

- (NSString *)fontName {
    return self.font.fontName;
}

- (void)setFontName:(NSString *)fontName {
    self.font = [UIFont fontWithName:fontName size:self.font.pointSize];
}
@end

@implementation UITextField (IBExtensions)

- (NSString *)fontName {
    return self.font.fontName;
}

- (void)setFontName:(NSString *)fontName {
    self.font = [UIFont fontWithName:fontName size:self.font.pointSize];
}
@end

@implementation UIButton (IBExtensions)

- (NSString *)fontName {
    return self.titleLabel.font.fontName;
```


> UILabel, UIButton etc. for which you want to set custom font. All the categories will be having a custom property say **fontName**. This will be using from the storyboard later for setting custom font (as in step 4).

UIKit+IBExtensions.h

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

//Category extension for UILabel
@interface UILabel (IBExtensions)

@property (nonatomic, copy) NSString *fontName;
@end

// Category extension for UITextField
@interface UITextField (IBExtensions)

@property (nonatomic, copy) NSString *fontName;
@end

// Category extension for UIButton
@interface UIButton (IBExtensions)

@property (nonatomic, copy) NSString *fontName;
@end
```

> 3. **Getters and Setters:** Define getters and setters for the fontName property towards each category added.

UIKit+IBExtensions.m

```
#import "UIKit+IBExtensions.h"

@implementation UILabel (IBExtensions)

- (NSString *)fontName {
    return self.font.fontName;
}

- (void)setFontName:(NSString *)fontName {
    self.font = [UIFont fontWithName:fontName size:self.font.pointSize];
}
@end

@implementation UITextField (IBExtensions)

- (NSString *)fontName {
    return self.font.fontName;
}

- (void)setFontName:(NSString *)fontName {
    self.font = [UIFont fontWithName:fontName size:self.font.pointSize];
}
@end

@implementation UIButton (IBExtensions)

- (NSString *)fontName {
    return self.titleLabel.font.fontName;
```

```
}

- (void)setFontName:(NSString *)fontName{
    self.titleLabel.font = [UIFont fontWithName:fontName size:self.titleLabel.font.pointSize];
}
@end
```

> 4. 在Storyboard中设置字体： 在用户定义的运行时属性中添加一项，键路径为fontName，值为您的自定义字体名称，类型为字符串，如图所示。
> keyPath 和您的 Custom Font's Name 作为值，类型为字符串，如图所示。

这将在运行应用时设置您的自定义字体。

注意：

- Lato-Regular 是我使用的自定义字体。
- 在捆绑包中添加的 .ttf 文件中使用的相同名称应在故事板中使用，且不带扩展名。
- 字体大小将与 UI 元素属性检查器中定义的大小相同。


```
}

- (void)setFontName:(NSString *)fontName{
    self.titleLabel.font = [UIFont fontWithName:fontName size:self.titleLabel.font.pointSize];
}
@end
```

> 4. **Setting font in storyboard:** Add an entry in User Defined Runtime Attributes with ***fontName*** as keyPath and your ***Custom Font's Name*** as value with type as String as shown.

This will set your custom font while running the app.

Notes:

- Lato-Regular is the custom font I have used.
- Same name in the ***.ttf*** file added in bundle should be used without extension in storyboard.
- Font size will be same as it is defined in the UI element's attribute inspector.

第93章：AVSpeechSynthesizer

参数	详细信息
扬声器	AVSpeechSynthesizer 对象
语音	AVSpeechUtterance 对象

第93.1节：创建基本的文本转语音

使用AVSpeechSynthesizer的 speakUtterance: 方法将文本转换为语音。你需要传入一个AVSpeechUtterance对象给该方法，该对象包含你想要朗读的文本。

Objective C

```
AVSpeechSynthesizer *speaker = [[AVSpeechSynthesizer alloc] init];
AVSpeechUtterance *speech    = [AVSpeechUtterance speechUtteranceWithString:@"Hello World"];
[speaker speakUtterance:speech];
```

Swift

```
let speaker = AVSpeechSynthesizer()
let speech = AVSpeechUtterance(string: "Hello World")
speaker.speakUtterance(speech)
```


Chapter 93: AVSpeechSynthesizer

Parameter	Details
speaker	AVSpeechSynthesizer object
speech	AVSpeechUtterance object

Section 93.1: Creating a basic text to speech

Use the `speakUtterance:` method of `AVSpeechSynthesizer` to convert text to speech. You need to pass an `AVSpeechUtterance` object to this method, which contains the text that you want to be spoken.

Objective C

```
AVSpeechSynthesizer *speaker = [[AVSpeechSynthesizer alloc] init];
AVSpeechUtterance *speech    = [AVSpeechUtterance speechUtteranceWithString:@"Hello World"];
[speaker speakUtterance:speech];
```

Swift

```
let speaker = AVSpeechSynthesizer()
let speech = AVSpeechUtterance(string: "Hello World")
speaker.speakUtterance(speech)
```

第94章：本地化

本地化是iOS提供的功能，可以将你的应用翻译成多种语言。对于本地化，国际化是必要的。国际化是使iOS应用能够适应不同文化、语言和地区的过程。

第94.1节：iOS中的本地化

为每种语言创建一个单独的Localizable.strings文件。右侧内容会因语言不同而不同。可以将其视为键值对：

```
"str" = "str-language";
```

在 Objective-C 中访问字符串：

```
//尝试为本地化字符串提供描述，以便在需要时能够创建适当的文档

NSString *str = NSLocalizedString(@"string", @"字符串的描述");
```

在 Swift 中访问字符串：

```
let str = NSLocalizedString("string", comment: "语言");
```


Chapter 94: Localization

Localization is feature provided by iOS which translates your app into multiple language.For **Localisation**,**Internationalization** is necessary.**Internationalization** is process of making iOS app able to adapt different culture,language and regions.

Section 94.1: Localization in iOS

Create an individual `Localizable.strings` file for each language. The right side would be different for each language. Think of it as a key-value pair:

```
"str" = "str-language";
```

Access str in Objective-C:

```
//Try to provide description on the localized string to be able to create a proper documentation if
needed
NSString *str = NSLocalizedString(@"string", @"description of the string");
```

Access str in Swift:

```
let str = NSLocalizedString("string", comment: "language");
```

第95章：Alamofire

参数	详细信息
方法	.OPTIONS, .GET, .HEAD, .POST, .PUT, .PATCH, .DELETE, .TRACE, .CONNECT
URL字符串	URLStringConvertible
parameters	[String: AnyObject]?
encoding	ParameterEncoding
headers	[String: String]?

第95.1节：手动验证

```
Alamofire.request(.GET, "https://httpbin.org/get", parameters: ["foo": "bar"])
     .validate(statusCode: 200..<300)
.validate(contentType: ["application/json"])
     .response { response in
         print(response)
     }
```

第95.2节：自动验证

```
Alamofire.request("https://httpbin.org/get").validate().responseJSON { response in
switch response.result {
case .success:
    print("验证成功")
case .failure(let error):
    print(error)
 }
}
```

第95.3节：链式响应处理器

```
Alamofire.request(.GET, "https://httpbin.org/get")
     .validate()
.responseString { response in
         print("响应字符串: \(response.result.value)")
     }
.responseJSON { response in
         print("响应JSON: \(response.result.value)")
     }
```

第95.4节：发起请求

```
import Alamofire

Alamofire.request(.GET, "https://httpbin.org/get")
```

第95.5节：响应处理

```
Alamofire.request(.GET, "https://httpbin.org/get", 参数: ["foo": "bar"])
     .responseJSON { response in
         print(response.request)  // 原始URL请求
         print(response.response) // URL响应
         print(response.data)     // 服务器数据
         print(response.result)   // 响应序列化结果
```


Chapter 95: Alamofire

Parameter	Details
Method	.OPTIONS, .GET, .HEAD, .POST, .PUT, .PATCH, .DELETE, .TRACE, .CONNECT
URLString	URLStringConvertible
parameters	[String: AnyObject]?
encoding	ParameterEncoding
headers	[String: String]?

Section 95.1: Manual Validation

```
Alamofire.request(.GET, "https://httpbin.org/get", parameters: ["foo": "bar"])
     .validate(statusCode: 200..<300)
     .validate(contentType: ["application/json"])
     .response { response in
         print(response)
     }
```

Section 95.2: Automatic Validation

```
Alamofire.request("https://httpbin.org/get").validate().responseJSON { response in
switch response.result {
case .success:
    print("Validation Successful")
case .failure(let error):
    print(error)
 }
}
```

Section 95.3: Chained Response Handlers

```
Alamofire.request(.GET, "https://httpbin.org/get")
     .validate()
     .responseString { response in
         print("Response String: \(response.result.value)")
     }
     .responseJSON { response in
         print("Response JSON: \(response.result.value)")
     }
```

Section 95.4: Making a Request

```
import Alamofire

Alamofire.request(.GET, "https://httpbin.org/get")
```

Section 95.5: Response Handling

```
Alamofire.request(.GET, "https://httpbin.org/get", parameters: ["foo": "bar"])
     .responseJSON { response in
         print(response.request)  // original URL request
         print(response.response) // URL response
         print(response.data)     // server data
         print(response.result)   // result of response serialization
```

```
        if let JSON = response.result.value {
            print("JSON: \(JSON)")
        }
    }
```

第95.6节：响应处理器

```
Alamofire.request(.GET, "https://httpbin.org/get", 参数: ["foo": "bar"])
    .validate()
.response { request, response, data, error in
        print(request)
        print(response)
        print(data)
        print(error)
    }
```



```
        if let JSON = response.result.value {
            print("JSON: \(JSON)")
        }
    }
```

Section 95.6: Response Handler

```
Alamofire.request(.GET, "https://httpbin.org/get", parameters: ["foo": "bar"])
    .validate()
    .response { request, response, data, error in
        print(request)
        print(response)
        print(data)
        print(error)
    }
```

第96章：iBeacon

参数	详细信息
管理器	CLLocationManager 引用
区域	CLRegion 可以是圆形区域（地理围栏或信标区域）
信标	包含所有检测到信标的 CLBeacon 数组

第96.1节：iBeacon 基本操作

1. 设置监测信标

```
func initiateRegion(ref:BeaconHandler){
    let uuid: NSUUID = NSUUID(UUIDString: "<UUID>")
    let beacon = CLBeaconRegion(proximityUUID: uuid, identifier: "")
    locationManager?.requestAlwaysAuthorization()    //cllocation manager obj.
beacon?.notifyOnEntry = true
    beacon?.notifyOnExit = true
    beacon?.notifyEntryStateOnDisplay = true
    locationManager?.startMonitoringForRegion(beacon!)
    locationManager?.delegate = self;
    // 检查此设备是否支持信标监测
    if (!CLLocationManager.isMonitoringAvailableForClass(CLBeaconRegion)) {
        print("error")
    }
locationManager!.startRangingBeaconsInRegion(self.beacon!)
}
```

2.位置管理器进入和退出区域

```
func locationManager(manager: CLLocationManager, didEnterRegion region: CLRegion) {
    if(region.isKindOfClass(CLBeaconRegion)) {
locationManager!.startRangingBeaconsInRegion(self.beacon!)
    }
}

func locationManager(manager: CLLocationManager, didExitRegion region: CLRegion) {
    if(region.isKindOfClass(CLBeaconRegion)) {
locationManager!.stopRangingBeaconsInRegion(self.beacon!)
    }
}
```

3.位置管理器信标测距

```
func locationManager(manager: CLLocationManager, didRangeBeacons beacons: [CLBeacon], inRegion
region: CLBeaconRegion) {
    print(beacons.first.major)
}
```

第96.2节：测距iBeacon

首先，您必须请求定位服务的授权

```
let locationManager = CLLocationManager()
locationManager.delegate = self
locationManager.requestWhenInUseAuthorization()
// 或 locationManager.requestAlwaysAuthorization()
```


Chapter 96: iBeacon

Parameters	Details
manager	CLLocationManager reference
region	CLRegion could be circular region (geofence or beacon region)
beacons	Array of CLBeacon contains all ranged beacons

Section 96.1: iBeacon Basic Operation

1. Setup monitoring beacons

```
func initiateRegion(ref:BeaconHandler){
    let uuid: NSUUID = NSUUID(UUIDString: "<UUID>")
    let beacon = CLBeaconRegion(proximityUUID: uuid, identifier: "")
    locationManager?.requestAlwaysAuthorization()    //cllocation manager obj.
    beacon?.notifyOnEntry = true
    beacon?.notifyOnExit = true
    beacon?.notifyEntryStateOnDisplay = true
    locationManager?.startMonitoringForRegion(beacon!)
    locationManager?.delegate = self;
    // Check if beacon monitoring is available for this device
    if (!CLLocationManager.isMonitoringAvailableForClass(CLBeaconRegion)) {
        print("error")
    }
    locationManager!.startRangingBeaconsInRegion(self.beacon!)
}
```

2. Location manager enter and exit region

```
func locationManager(manager: CLLocationManager, didEnterRegion region: CLRegion) {
    if(region.isKindOfClass(CLBeaconRegion)) {
        locationManager!.startRangingBeaconsInRegion(self.beacon!)
    }
}

func locationManager(manager: CLLocationManager, didExitRegion region: CLRegion) {
    if(region.isKindOfClass(CLBeaconRegion)) {
        locationManager!.stopRangingBeaconsInRegion(self.beacon!)
    }
}
```

3. Location manager range beacon

```
func locationManager(manager: CLLocationManager, didRangeBeacons beacons: [CLBeacon], inRegion
region: CLBeaconRegion) {
    print(beacons.first.major)
}
```

Section 96.2: Ranging iBeacons

First, you have to request authorization of location services

```
let locationManager = CLLocationManager()
locationManager.delegate = self
locationManager.requestWhenInUseAuthorization()
// OR locationManager.requestAlwaysAuthorization()
```

然后，您可以在 didRangeBeacons 中获取所有iBeacon的信息

```
func locationManager(manager: CLLocationManager, didRangeBeacons beacons: [CLBeacon], inRegion
region: CLBeaconRegion) {
    for beacon in beacons {
        print(beacon.major)
        print(beacon.minor)
    }
}
```

第96.3节：扫描特定的Beacon

```
 beacon = CLBeaconRegion(proximityUUID: <#NSUUID#>, major: <#CLBeaconMajorValue#>, identifier:
<#String#>) // 监听所有具有指定UUID和主值的信标
 beacon =    CLBeaconRegion(proximityUUID: <##NSUUID#>, major: <##CLBeaconMajorValue#>, minor:
<##CLBeaconMinorValue#>, identifier: <##String#>) // 监听所有具有指定UUID和
主值及次值的信标
```


Then you can get all iBeacons' information inside `didRangeBeacons`

```
func locationManager(manager: CLLocationManager, didRangeBeacons beacons: [CLBeacon], inRegion
region: CLBeaconRegion) {
    for beacon in beacons {
        print(beacon.major)
        print(beacon.minor)
    }
}
```

Section 96.3: Scanning specific Beacons

```
 beacon = CLBeaconRegion(proximityUUID: <#NSUUID#>, major: <#CLBeaconMajorValue#>, identifier:
<#String#>) // listening to all beacons with given UUID and major value
 beacon =    CLBeaconRegion(proximityUUID: <##NSUUID#>, major: <##CLBeaconMajorValue#>, minor:
<##CLBeaconMinorValue#>, identifier: <##String#>) // listening to all beacons with given UUID and
major and minor value
```

第97章：CLLocation

第97.1节：使用距离过滤器

示例：

```
CLLocationManager *locationManager = [[CLLocationManager alloc] init];
   locationManager.delegate = self;
locationManager.desiredAccuracy = kCLLocationAccuracyBest;
   locationManager.distanceFilter = 5;
```

例如：在上述示例代码中，位置变化小于5米时不会发送回调，
而是被忽略。

第97.2节：使用CLLocationManager获取用户位置

1 - 在项目中包含 CoreLocation.framework；操作方法是点击：

```
根目录 -> 构建阶段 -> 连接二进制与库
```

点击 (+) 按钮，查找 CoreLocation.framework 并点击添加。

2- 修改 info.plist 文件以请求使用用户位置的权限，方法是以源代码形式打开它。在标签下添加以下任一键值对，以请求应用程序使用时访问用户位置的权限：

```
<key>NSLocationWhenInUseUsageDescription</key>
<string>请求权限时显示的消息</string>
```

3- 在将使用 CoreLocation 的 ViewController 中导入 CoreLocation。

```
import CoreLocation
```

4- 确保你的 ViewController 遵循 CLLocationManagerDelegate 协议

```
class ViewController: UIViewController,CLLocationManagerDelegate {}
```

完成以上步骤后，我们可以创建一个 CLLocationManager 对象作为实例变量，并在 ViewController 中使用它。

```
var manager:CLLocationManager!
```

我们这里不使用 'let'，因为我们将修改管理器以指定其代理、更新事件前的最小距离以及其精度

```
//初始化管理器
manager = CLLocationManager()

//指定代理
manager.delegate = self

//设置手机在触发更新事件前需要移动的最小距离（例如：100米）

manager.distanceFilter = 100

//根据您的使用情况设置精度为以下任意值
```


Chapter 97: CLLocation

Section 97.1: Distance Filter using

Example :

```
   CLLocationManager *locationManager = [[CLLocationManager alloc] init];
   locationManager.delegate = self;
   locationManager.desiredAccuracy = kCLLocationAccuracyBest;
   locationManager.distanceFilter = 5;
```

E.g. In the above example code above, location changes of less than 5 metres won't be sent to the callback, but instead be ignored.

Section 97.2: Get User Location Using CLLocationManager

1 - Include the CoreLocation.framework in your project; this is accomplished by clicking on:

```
root directory -> build phases -> Link Binary With Libraries
```

Click on the (+) button, look for CoreLocation.framework and click add.

2- Modify the info.plist file to ask for permission to use user location by opening it as source code. Add either of the following key:value pair under the tag to ask for usage of user's location while the application is in use:

```
<key>NSLocationWhenInUseUsageDescription</key>
<string>message to display when asking for permission</string>
```

3- import CoreLocation to the ViewController that will be using it.

```
import CoreLocation
```

4- Make sure your ViewController conforms to the CLLocationManagerDelagate protocol

```
class ViewController: UIViewController,CLLocationManagerDelegate {}
```

After these steps, we can create a CLLocationManager object as instance variable and use it in the ViewController.

```
var manager:CLLocationManager!
```

We do not use 'let' here because we will modify the manager to specify its delegate, minimum distance before update event, and its accuracy

```
//initialize the manager
manager = CLLocationManager()

//specify delegate
manager.delegate = self

//set the minimum distance the phone needs to move before an update event is triggered (for
example:  100 meters)
manager.distanceFilter = 100

//set Accuracy to any of the following depending on your use case
```

```
//let kCLLocationAccuracyBestForNavigation: CLLocationAccuracy
//let kCLLocationAccuracyBest: CLLocationAccuracy
//let kCLLocationAccuracyNearestTenMeters: CLLocationAccuracy
//let kCLLocationAccuracyHundredMeters: CLLocationAccuracy
//let kCLLocationAccuracyKilometer: CLLocationAccuracy
//let kCLLocationAccuracyThreeKilometers: CLLocationAccuracy

manager.desiredAccuracy = kCLLocationAccuracyBest

//请求用户授权
manager.requestWhenInUseAuthorization()

//开始收集位置信息
if #available(iOS 9.0, *) {

manager.requestLocation()

 } else {

manager.startUpdatingLocation()

   }
```

现在，为了获取位置更新，我们可以实现下面的函数，该函数会在达到 distanceFilter 时被调用。

```
func locationManager(manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {}
```

locations 参数是一个 CLLocation 对象数组，表示设备的实际位置。通过这些对象，可以访问以下属性：coordinate（坐标）、altitude（海拔）、floor（楼层）、horizontalAccuracy（水平精度）、verticalAccuracy（垂直精度）、timestamp（时间戳）、description（描述）、course（航向）、speed（速度），以及一个函数 distance(from:)，用于测量两个位置之间的距离。

注意：请求位置权限时，有两种不同类型的授权。

> “使用时”授权仅允许应用在使用中或前台时接收您的位置。
>
> “始终”授权则赋予应用后台权限，这可能会导致在应用关闭时电池寿命减少。
>
> 需要根据情况调整 Plist 文件。


```
//let kCLLocationAccuracyBestForNavigation: CLLocationAccuracy
//let kCLLocationAccuracyBest: CLLocationAccuracy
//let kCLLocationAccuracyNearestTenMeters: CLLocationAccuracy
//let kCLLocationAccuracyHundredMeters: CLLocationAccuracy
//let kCLLocationAccuracyKilometer: CLLocationAccuracy
//let kCLLocationAccuracyThreeKilometers: CLLocationAccuracy

manager.desiredAccuracy = kCLLocationAccuracyBest

//ask the user for permission
manager.requestWhenInUseAuthorization()

//Start collecting location information
if #available(iOS 9.0, *) {

   manager.requestLocation()

 } else {

   manager.startUpdatingLocation()

  }
```

Now to get access to the location updates, we can implement the function below which is called overtime the distanceFilter is reached.

```
func locationManager(manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {}
```

The locations parameter is an array of CLLocation objects that represent the actual location of the device.From these objects, one can get access to the following attributes: `coordinate`,`altitude`, `floor`, `horizontalAccuracy`, `verticalAccuracy`, `timestamp`, `description`, `course`, `speed`, and a function `distance(from:)` that measures the distance between two locations.

Note: While requesting permission for location, there are two different types of authorization.

> "When In Use" authorization only gives the app permission to receive your location when the app is in use or foreground.
>
> “Always" authorization, gives the app background permissions which may lead to decrease battery life in case your app is closed.
>
> Plist file should be adjusted as necessary.

第98章：检查iOS版本

第98.1节：iOS 8及更高版本

Swift 3：

```
let minimumVersion = OperatingSystemVersion(majorVersion: 8, minorVersion: 1, patchVersion: 2)
if ProcessInfo().isOperatingSystemAtLeast(minimumVersion) {
    //当前版本 >= (8.1.2)
} else {
    //当前版本 < (8.1.2)
}
```

第98.2节：Swift 2.0及更高版本

```
if #available(iOS 9, *) {
  // iOS 9
} else {
  // iOS 8或更早版本
}
```

第98.3节：版本比较

```
let minimumVersionString = "3.1.3"
let versionComparison = UIDevice.current.systemVersion.compare(minimumVersionString, options:
.numeric)
switch versionComparison {
    case .orderedSame, .orderedDescending:
        //当前版本 >= (3.1.3)
        break
    case .orderedAscending:
        //当前版本 < (3.1.3)
        fallthrough
    default:
        break;
}
```

Objective-C

```
NSString *version = @"3.1.3";
NSString *currentVersion = @"3.1.1";
NSComparisonResult result = [currentVersion compare:version options:NSNumericSearch];
switch(result){
  case: NSOrderedAscending:
        // 小于当前版本
  break;
  case: NSOrderedDescending:
  case: NSOrderedSame:
       // 等于或大于当前版本
  break;
}
```

第98.4节：设备iOS版本

这将返回当前系统版本。

Objective-C

Chapter 98: Checking iOS version

Section 98.1: iOS 8 and later

Swift 3:

```
let minimumVersion = OperatingSystemVersion(majorVersion: 8, minorVersion: 1, patchVersion: 2)
if ProcessInfo().isOperatingSystemAtLeast(minimumVersion) {
    //current version is >= (8.1.2)
} else {
    //current version is < (8.1.2)
}
```

Section 98.2: Swift 2.0 and later

```
if #available(iOS 9, *) {
  // iOS 9
} else {
  // iOS 8 or earlier
}
```

Section 98.3: Compare versions

```
let minimumVersionString = "3.1.3"
let versionComparison = UIDevice.current.systemVersion.compare(minimumVersionString, options:
.numeric)
switch versionComparison {
    case .orderedSame, .orderedDescending:
        //current version is >= (3.1.3)
        break
    case .orderedAscending:
        //current version is < (3.1.3)
        fallthrough
    default:
        break;
}
```

Objective-C

```
NSString *version = @"3.1.3";
NSString *currentVersion = @"3.1.1";
NSComparisonResult result = [currentVersion compare:version options:NSNumericSearch];
switch(result){
  case: NSOrderedAscending:
        //less than the current version
  break;
  case: NSOrderedDescending:
  case: NSOrderedSame:
       // equal or greater than the current version
  break;
}
```

Section 98.4: Device iOS Version

This will give current system version.

Objective-C

```
NSString *version = [[UIDevice currentDevice] systemVersion]
```

Swift

```
let version = UIDevice.currentDevice().systemVersion
```

Swift 3

```
let version = UIDevice.current.systemVersion
```



```
NSString *version = [[UIDevice currentDevice] systemVersion]
```

Swift

```
let version = UIDevice.currentDevice().systemVersion
```

Swift 3

```
let version = UIDevice.current.systemVersion
```

第99章：通用链接

第99.1节：设置iOS应用程序（启用通用链接）

应用程序端的设置需要两项内容：

1. 配置应用程序的权限，并通过开启项目中的关联域功能来启用通用链接能力。
2. 在你的AppDelegate中处理传入的链接。

1. 配置应用程序的权限，并启用通用链接。

配置应用程序权限的第一步是为您的应用程序标识符（App ID）启用该权限。请在 Apple 开发者会员中心进行操作。点击“证书、标识符与描述文件”，然后选择“标识符”。选择您的应用程序标识符（如有需要，请先创建），点击“编辑”，并启用“关联域名”权限。

ID: com.Universal-Links

Application Services:

Service	Development	Distribution
App Group	Disabled	Disabled
Associated Domains	Enabled	Enabled
Data Protection	Disabled	Disabled
Game Center	Enabled	Enabled
HealthKit	Disabled	Disabled
HomeKit	Disabled	Disabled
Wireless Accessory Configuration	Disabled	Disabled
iCloud	Disabled	Disabled
In-App Purchase	Enabled	Enabled
Inter-App Audio	Disabled	Disabled
Apple Pay	Disabled	Disabled
Wallet	Disabled	Disabled
Push Notifications	Disabled	Disabled
Personal VPN	Disabled	Disabled

Edit

Chapter 99: Universal Links

Section 99.1: Setup iOS Application (Enabling Universal Links)

The setup on the app side requires two things:

1. Configuring the app's entitlement, and enabling the universal links by turning on the Associated Domains capability in the project.
2. Handling Incoming Links in your `AppDelegate`.

1. Configuring the app's entitlement, and enabling universal links.

The first step in configuring your app's entitlements is to enable it for your App ID. Do this in the Apple Developer Member Center. Click on Certificates, Identifiers & Profiles and then Identifiers. Select your App ID (create it first if required), click Edit and enable the Associated Domains entitlement.

ID: com.Universal-Links

Application Services:

Service	Development	Distribution
App Group	Disabled	Disabled
Associated Domains	Enabled	Enabled
Data Protection	Disabled	Disabled
Game Center	Enabled	Enabled
HealthKit	Disabled	Disabled
HomeKit	Disabled	Disabled
Wireless Accessory Configuration	Disabled	Disabled
iCloud	Disabled	Disabled
In-App Purchase	Enabled	Enabled
Inter-App Audio	Disabled	Disabled
Apple Pay	Disabled	Disabled
Wallet	Disabled	Disabled
Push Notifications	Disabled	Disabled
Personal VPN	Disabled	Disabled

Edit

接下来，通过点击相应的 App ID 获取 App ID 的前缀和后缀。

App ID 的前缀和后缀应与 apple-app-site-association 文件中的一致。

接下来，在Xcode中，选择您的应用'的目标，点击功能（Capabilities），并将关联域（Associated Domains）切换为开启。为您的应用支持的每个域添加一条条目，前缀为app links:

例如**applinks:YourCustomDomainName.com**

对于示例应用，看起来像这样：

注意：确保您选择了相同的团队，并输入了与会员中心注册的 App ID 相同的 Bundle ID。还要确保 Xcode 包含了权限文件，方法是选中该文件，在文件检查器（File Inspector）中确认已勾选您的目标。

Next, get the App ID prefix and suffix by clicking on the respective App ID.

The App ID prefix and suffix should match the one in the apple-app-site-association file.

Next in Xcode, select your App's target, click Capabilities and toggle Associated Domains to On. Add an entry for each domain that your app supports, prefixed with **app links:**

For example **applinks:YourCustomDomainName.com**

Which looks like this for the sample app:

Note: Ensure you have selected the same team and entered the same Bundle ID as the registered App ID on the Member Center. Also ensure that the entitlements file is included by Xcode by selecting the file and in the File Inspector, ensure that your target is checked.

2. 在您的 AppDelegate 中处理传入链接

所有从 Safari 重定向到应用的通用链接都会通过应用的 AppDelegate 类中的以下方法。您需要解析此 URL 以确定应用中的正确操作。

```
[UIApplicationDelegate application: continueUserActivity: restorationHandler:]
```

Objective-C

```
-(BOOL)application:(UIApplication *)application continueUserActivity:(NSUserActivity *)userActivity
restorationHandler:(void (^)(NSArray * _Nullable))restorationHandler{
    ///检查活动是否来自网页重定向到应用。
    if ([userActivity.activityType isEqualToString: NSUserActivityTypeBrowsingWeb]) {
        ///从UserActivity对象获取URL。
        NSURL *url = userActivity.webpageURL;
        UIStoryboard *storyBoard = [UIStoryboard storyboardWithName:@"Main" bundle:nil];
        UINavigationController *navigationController = (UINavigationController
*)_window.rootViewController;
        if ([url.pathComponents containsObject:@"home"]) {
            [navigationController pushViewController:[storyBoard
instantiateViewControllerWithIdentifier:@"HomeScreenId"] animated:YES];
        }else if ([url.pathComponents containsObject:@"about"]){
            [navigationController pushViewController:[storyBoard
instantiateViewControllerWithIdentifier:@"AboutScreenId"] animated:YES];
        }
    }
    return YES;
}
```

Swift :

```
func application(application: UIApplication, continueUserActivity userActivity: NSUserActivity,
```

2. Handling Incoming Links in your AppDelegate

All redirects from Safari to the app for universal links go via the below method in the Application's AppDelegate class. You parse this URL to determine the right action in the app.

```
[UIApplicationDelegate application: continueUserActivity: restorationHandler:]
```

Objective-C

```
-(BOOL)application:(UIApplication *)application continueUserActivity:(NSUserActivity *)userActivity
restorationHandler:(void (^)(NSArray * _Nullable))restorationHandler{
    ///Checking whether the activity was from a web page redirect to the app.
    if ([userActivity.activityType isEqualToString: NSUserActivityTypeBrowsingWeb]) {
        ///Getting the URL from the UserActivty Object.
        NSURL *url = userActivity.webpageURL;
        UIStoryboard *storyBoard = [UIStoryboard storyboardWithName:@"Main" bundle:nil];
        UINavigationController *navigationController = (UINavigationController
*)_window.rootViewController;
        if ([url.pathComponents containsObject:@"home"]) {
            [navigationController pushViewController:[storyBoard
instantiateViewControllerWithIdentifier:@"HomeScreenId"] animated:YES];
        }else if ([url.pathComponents containsObject:@"about"]){
            [navigationController pushViewController:[storyBoard
instantiateViewControllerWithIdentifier:@"AboutScreenId"] animated:YES];
        }
    }
    return YES;
}
```

Swift :

```
func application(application: UIApplication, continueUserActivity userActivity: NSUserActivity,
```

```
restorationHandler: ([AnyObject]?) -> Void) -> Bool {
      if userActivity.activityType == NSUserActivityTypeBrowsingWeb {
          let url = userActivity.webpageURL!
          //处理 url
      }
      返回 true
  }
```

iOS 应用代码

应用代码可以在 master 分支 here 找到。

第 99.2 节：支持多个域名

应用中支持的每个域名都需要提供其自己的 apple-app-site-association 文件。如果每个域名提供的内容不同，那么文件内容也会相应更改以支持各自的路径。
否则，可以使用相同的文件，但该文件需要在每个支持的域名上都能访问。

第 99.3 节：为 App-Site-Association 文件签名

注意：如果您的服务器使用HTTPS提供内容，可以跳过此部分，直接进入应用程序设置指南。

如果您的应用程序针对的是iOS 9，且服务器使用HTTPS提供内容，则无需对文件进行签名。如果不是（例如在支持iOS 8的Handoff时），则必须使用受认可的证书颁发机构颁发的SSL证书对其进行签名。

注意：这不是Apple提供用于提交应用到App Store的证书。它应由第三方提供，建议使用与您的HTTPS服务器相同的证书（尽管这不是必需的）。

要签名文件，首先创建并保存一个简单的.txt版本。接着，在终端运行以下命令：

```
cat <unsigned_file>.txt | openssl smime -sign -inkey example.com.key -signer example.com.pem -
certfile intermediate.pem -noattr -nodetach -outform DER > apple-app-site-association
```

这将在当前目录输出签名后的文件。example.com.key、example.com.pem和 `intermediate.pem`是由您的证书颁发机构提供的文件。

注意：如果文件未签名，其`Content-Type`应为`application/json`。否则，应为 `application/pkcs7-mime`。

使用Apple应用搜索验证工具验证您的服务器
测试您的网页以支持iOS 9搜索API。输入URL，Applebot将爬取您的网页并展示如何优化以获得最佳效果 https://search.developer.apple.com/appsearch-validation-tool/

第99.4节：设置服务器

您需要有一台在线运行的服务器。为了安全地将您的 iOS 应用与服务器关联，苹果要求您提供一个配置文件，称为apple-app-site-association。该文件是一个JSON文件，描述了域名和支持的路由。

`apple-app-site-association` 文件需要通过HTTPS访问，且不能有任何重定向，路径为 *https://{domain}/apple-app-site-association*。


```
restorationHandler: ([AnyObject]?) -> Void) -> Bool {
      if userActivity.activityType == NSUserActivityTypeBrowsingWeb {
          let url = userActivity.webpageURL!
          //handle url
      }
      return true
  }
```

iOS Application Code

The app code can be found master branch here.

Section 99.2: Supporting Multiple Domains

Each domain supported in the app needs to make available its own apple-app-site-association file. If the content served by each domain is different, then the contents of the file will also change to support the respective paths. Otherwise, the same file can be used, but it needs to be accessible at every supported domain.

Section 99.3: Signing the App-Site-Association File

Note: *You can skip this part if your server uses HTTPS to serve content and jump to Application Setup guide.*

If your app targets iOS 9 and your server uses HTTPS to serve content, you don't need to sign the file. If not (e.g. when supporting Handoff on iOS 8), it has to be signed using a SSL certificate from a recognized certificate authority.

Note: This is not the certificate provided by Apple to submit your app to the App Store. It should be provided by a third-party, and it's recommended to use the same certificate you use for your HTTPS server (although it's not required).

To sign the file, first create and save a simple .txt version of it. Next, in the terminal, run the following command:

```
cat <unsigned_file>.txt | openssl smime -sign -inkey example.com.key -signer example.com.pem -
certfile intermediate.pem -noattr -nodetach -outform DER > apple-app-site-association
```

This will output the signed file in the current directory. The `example.com.key`, `example.com.pem`, and `intermediate.pem` are the files that would made available to you by your Certifying Authority.

Note: If the file is unsigned, it should have a `Content-Type` of `application/json`. Otherwise, it should be `application/pkcs7-mime`.

Validate your Server with Apple App search validation tool
Test your webpage for iOS 9 Search APIs. Enter a URL and Applebot will crawl your webpage and show how you can optimize for best results https://search.developer.apple.com/appsearch-validation-tool/

Section 99.4: Setup Server

You need to having a server running online. To securely associate your iOS app with a server, Apple requires that you make available a configuration file, called `apple-app-site-association`. This is a JSON file which describes the domain and supported routes.

The `apple-app-site-association` file needs to be accessible via HTTPS, without any redirects, at *https://{domain}/apple-app-site-association*.

该文件内容如下：

```
{
"applinks": {
    "apps": [ ],
    "details": [
        {
            "appID": "{app_prefix}.{app_identifier}",
            "paths": [ "/path/to/content", "/path/to/other/*", "NOT /path/to/exclude" ]
        },
        {
            "appID": "TeamID.BundleID2",
            "paths": [ "*" ]
        }
    ]
}
}
```

注意 - 不要在apple-app-site-association文件名后追加.json。

键如下：
apps：其值应为空数组，且必须存在。这是苹果的要求。
`details`：是一个字典数组，每个字典对应网站支持的一个iOS应用。每个字典包含关于该应用、团队和捆绑ID的信息。

定义路径有三种方式：
静态：整个支持的路径是硬编码的，用于识别特定链接，例如 /static/terms
通配符：可以使用 * 来匹配动态路径，例如 /books/* 可以匹配任何作者's 页面的路径。?在特定路径组件内，例如 books/1? 可用于匹配任何ID以1开头的书籍。
排除：在路径前加上 NOT 表示排除该路径的匹配。

路径在数组中出现的顺序很重要。索引越靠前优先级越高。一旦路径匹配，评估停止，其他路径被忽略。每个路径区分大小写。

#网站代码

网站代码可以在 gh-pages 分支的
https://github.com/vineetchoudhary/iOS-Universal-Links/tree/gh-pages 找到

The file looks like this:

```
{
"applinks": {
    "apps": [ ],
    "details": [
        {
            "appID": "{app_prefix}.{app_identifier}",
            "paths": [ "/path/to/content", "/path/to/other/*", "NOT /path/to/exclude" ]
        },
        {
            "appID": "TeamID.BundleID2",
            "paths": [ "*" ]
        }
    ]
}
}
```

NOTE - *Don't append `.json` to the `apple-app-site-association` filename.*

The keys are as follows:
`apps`: Should have an empty array as its value, and it must be present. This is how Apple wants it.
`details`: Is an array of dictionaries, one for each iOS app supported by the website. Each dictionary contains information about the app, the team and bundle IDs.

There are 3 ways to define paths:
`Static`: The entire supported path is hardcoded to identify a specific link, e.g. /static/terms
`Wildcards`: A * can be used to match dynamic paths, e.g. /books/* can matches the path to any author's page. ? inside specific path components, e.g. books/1? can be used to match any books whose ID starts with 1.
`Exclusions`: Prepending a path with NOT excludes that path from being matched.

The order in which the paths are mentioned in the array is important. Earlier indices have higher priority. Once a path matches, the evaluation stops, and other paths ignored. Each path is case-sensitive.

#Website Code

The website code can be found gh-pages branch on
https://github.com/vineetchoudhary/iOS-Universal-Links/tree/gh-pages

第100章：iOS中的PDF创建

第100.1节：创建PDF

```
UIGraphicsBeginPDFContextToFile(fileName, CGRectZero, nil);

UIGraphicsBeginPDFPageWithInfo(CGRectMake(0, 0, 612, 792), nil);

[self drawText];

UIGraphicsEndPDFContext();
```

fileName 是您将要追加或附加的文档文件

```
 NSString* temporaryFile = @"firstIOS.PDF";
    NSArray *arrayPaths =
NSSearchPathForDirectoriesInDomains(
                                    NSDocumentDirectory,
                                    NSUserDomainMask,
                                    YES);

    NSString *path = [arrayPaths objectAtIndex:0];

    NSString* fileName = [path stringByAppendingPathComponent:fileName];
```

其中 drawText 是

```
   (void)drawText
{
    NSString* textToDraw = @"Lorem Ipsum 是印刷和排版行业的虚拟文本。自1500年代以来，Lorem Ipsum一直是该
行业的标准虚拟文本，当时一位未知的印刷工人拿起一排字母并将其打乱，制作了一本字体样本书。";


    CFStringRef stringRef = (__bridge CFStringRef)textToDraw;

    CFAttributedStringRef currentText = CFAttributedStringCreate(NULL, stringRef, NULL);

    CTFramesetterRef framesetter = CTFramesetterCreateWithAttributedString(currentText);

    CGRect frameRect = CGRectMake(0, 0, 300, 100);

    CGMutablePathRef framePath = CGPathCreateMutable();

    CGPathAddRect(framePath, NULL, frameRect);

    CFRange currentRange = CFRangeMake(0, 0);

CTFrameRef frameRef = CTFramesetterCreateFrame(framesetter, currentRange, framePath, NULL);
    CGPathRelease(framePath);

    CGContextRef currentContext = UIGraphicsGetCurrentContext();


    CGContextSetTextMatrix(currentContext, CGAffineTransformIdentity);


CGContextTranslateCTM(currentContext, 0, 450);

    CGContextScaleCTM(currentContext, 2, -2);
```


Chapter 100: PDF Creation in iOS

Section 100.1: Create PDF

```
UIGraphicsBeginPDFContextToFile(fileName, CGRectZero, nil);

UIGraphicsBeginPDFPageWithInfo(CGRectMake(0, 0, 612, 792), nil);

[self drawText];

UIGraphicsEndPDFContext();
```

fileName is the document file where You are going to append or attach

```
 NSString* temporaryFile = @"firstIOS.PDF";
    NSArray *arrayPaths =
    NSSearchPathForDirectoriesInDomains(
                                        NSDocumentDirectory,
                                        NSUserDomainMask,
                                        YES);

    NSString *path = [arrayPaths objectAtIndex:0];

    NSString* fileName = [path stringByAppendingPathComponent:fileName];
```

Where **drawText** is

```
   (void)drawText
{
    NSString* textToDraw = @"Lorem Ipsum is simply dummy text of the printing and typesetting
industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an
unknown printer took a galley of type and scrambled it to make a type specimen book.";

    CFStringRef stringRef = (__bridge CFStringRef)textToDraw;

    CFAttributedStringRef currentText = CFAttributedStringCreate(NULL, stringRef, NULL);

    CTFramesetterRef framesetter = CTFramesetterCreateWithAttributedString(currentText);

    CGRect frameRect = CGRectMake(0, 0, 300, 100);

    CGMutablePathRef framePath = CGPathCreateMutable();

    CGPathAddRect(framePath, NULL, frameRect);

    CFRange currentRange = CFRangeMake(0, 0);

    CTFrameRef frameRef = CTFramesetterCreateFrame(framesetter, currentRange, framePath, NULL);
    CGPathRelease(framePath);

    CGContextRef currentContext = UIGraphicsGetCurrentContext();


    CGContextSetTextMatrix(currentContext, CGAffineTransformIdentity);


    CGContextTranslateCTM(currentContext, 0, 450);

    CGContextScaleCTM(currentContext, 2, -2);
```

```
CTFrameDraw(frameRef, currentContext);

    CFRelease(frameRef);

CFRelease(stringRef);

    CFRelease(framesetter);
}
```

第100.2节：显示PDF

```
  NSString* fileName = @"firstIOS.PDF";


NSArray *arrayPaths =
NSSearchPathForDirectoriesInDomains(
                                    NSDocumentDirectory,
                                    NSUserDomainMask,
                                    YES);

NSString *path = [arrayPaths objectAtIndex:0];

NSString* pdfFileName = [path stringByAppendingPathComponent:fileName];

UIWebView* webView = [[UIWebView alloc] initWithFrame:CGRectMake(0, 0, 320, 480)];

NSURL *url = [NSURL fileURLWithPath:pdfFileName];

NSURLRequest *request = [NSURLRequest requestWithURL:url];
```



```
    CTFrameDraw(frameRef, currentContext);

    CFRelease(frameRef);

    CFRelease(stringRef);

    CFRelease(framesetter);
}
```

Section 100.2: Show PDF

```
  NSString* fileName = @"firstIOS.PDF";


NSArray *arrayPaths =
NSSearchPathForDirectoriesInDomains(
                                    NSDocumentDirectory,
                                    NSUserDomainMask,
                                    YES);

NSString *path = [arrayPaths objectAtIndex:0];

NSString* pdfFileName = [path stringByAppendingPathComponent:fileName];

UIWebView* webView = [[UIWebView alloc] initWithFrame:CGRectMake(0, 0, 320, 480)];

NSURL *url = [NSURL fileURLWithPath:pdfFileName];

NSURLRequest *request = [NSURLRequest requestWithURL:url];
```

```
[webView setScalesPageToFit:YES];

[webView loadRequest:request];

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

第100.3节：多页PDF

```
UIGraphicsBeginPDFContextToFile(fileName, CGRectZero, nil);

    UIGraphicsBeginPDFPageWithInfo(CGRectMake(0, 0, 600, 792), nil);

    UIGraphicsBeginPDFPageWithInfo(CGRectMake(0, 0, 600, 792), nil);

    UIGraphicsBeginPDFPageWithInfo(CGRectMake(0, 0, 600, 792), nil);

   UIGraphicsEndPDFContext();
```

第100.4节：从UIWebview中加载的任何Microsoft文档创建PDF

```
#define kPaperSizeA4 CGSizeMake(595.2,841.8)
```

首先实现UIPrintPageRenderer协议

```
@interface UIPrintPageRenderer (PDF)

- (NSData*) printToPDF;

@end

@implementation UIPrintPageRenderer (PDF)

- (NSData*) printToPDF
{
    NSMutableData *pdfData = [NSMutableData data];
    UIGraphicsBeginPDFContextToData( pdfData, self.paperRect, nil );
    [self prepareForDrawingPages: NSMakeRange(0, self.numberOfPages)];
    CGRect bounds = UIGraphicsGetPDFContextBounds();
    for ( int i = 0 ; i < self.numberOfPages ; i++ )
    {
UIGraphicsBeginPDFPage();
        [self drawPageAtIndex: i inRect: bounds];
    }
UIGraphicsEndPDFContext();
    return pdfData;
}
@end
```

然后，在UIWebView文档加载完成后调用以下方法

```
-(void)createPDF:(UIWebView *)webView {

UIPrintPageRenderer *render = [[UIPrintPageRenderer alloc] init];
[render addPrintFormatter:webView.viewPrintFormatter startingAtPageAtIndex:0];

float padding = 10.0f;
CGRect paperRect = CGRectMake(0, 0, kPaperSizeA4.width, kPaperSizeA4.height);
```



```
[webView setScalesPageToFit:YES];

[webView loadRequest:request];

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

Section 100.3: Multiple page PDF

```
UIGraphicsBeginPDFContextToFile(fileName, CGRectZero, nil);

    UIGraphicsBeginPDFPageWithInfo(CGRectMake(0, 0, 600, 792), nil);

    UIGraphicsBeginPDFPageWithInfo(CGRectMake(0, 0, 600, 792), nil);

    UIGraphicsBeginPDFPageWithInfo(CGRectMake(0, 0, 600, 792), nil);

    UIGraphicsEndPDFContext();
```

Section 100.4: Create PDF from any Microsoft Document loaded in UIWebview

```
#define kPaperSizeA4 CGSizeMake(595.2,841.8)
```

First of all implement UIPrintPageRenderer protocol

```
@interface UIPrintPageRenderer (PDF)

- (NSData*) printToPDF;

@end

@implementation UIPrintPageRenderer (PDF)

- (NSData*) printToPDF
{
    NSMutableData *pdfData = [NSMutableData data];
    UIGraphicsBeginPDFContextToData( pdfData, self.paperRect, nil );
    [self prepareForDrawingPages: NSMakeRange(0, self.numberOfPages)];
    CGRect bounds = UIGraphicsGetPDFContextBounds();
    for ( int i = 0 ; i < self.numberOfPages ; i++ )
    {
        UIGraphicsBeginPDFPage();
        [self drawPageAtIndex: i inRect: bounds];
    }
    UIGraphicsEndPDFContext();
    return pdfData;
}
@end
```

Then, call below method after document finished loading in `UIWebView`

```
-(void)createPDF:(UIWebView *)webView {

UIPrintPageRenderer *render = [[UIPrintPageRenderer alloc] init];
[render addPrintFormatter:webView.viewPrintFormatter startingAtPageAtIndex:0];

float padding = 10.0f;
CGRect paperRect = CGRectMake(0, 0, kPaperSizeA4.width, kPaperSizeA4.height);
```

```
CGRect printableRect = CGRectMake(padding, padding, kPaperSizeA4.width-(padding * 2),
kPaperSizeA4.height-(padding * 2));

[render setValue:[NSValue valueWithCGRect:paperRect] forKey:@"paperRect"];
[render setValue:[NSValue valueWithCGRect:printableRect] forKey:@"printableRect"];

NSData *pdfData = [render printToPDF];

dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{

    if (pdfData) {
        [pdfData writeToFile:directoryPath atomically: YES];
    }
    else
    {
NSLog(@"PDF couldnot be created");
    }
});}
```



```
CGRect printableRect = CGRectMake(padding, padding, kPaperSizeA4.width-(padding * 2),
kPaperSizeA4.height-(padding * 2));

[render setValue:[NSValue valueWithCGRect:paperRect] forKey:@"paperRect"];
[render setValue:[NSValue valueWithCGRect:printableRect] forKey:@"printableRect"];

NSData *pdfData = [render printToPDF];

dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{

    if (pdfData) {
        [pdfData writeToFile:directoryPath atomically: YES];
    }
    else
    {
        NSLog(@"PDF couldnot be created");
    }
});}
```

第101章：应用内购买

第101.1节：Swift 2中的单一应用内购买

在iTunesConnect中创建应用内购买后：

在你想要购买的视图控制器中

```
import StoreKit
```

并添加相关的代理

```
class ViewController: UIViewController, SKProductsRequestDelegate, SKPaymentTransactionObserver {
```

声明一个包含来自iTunesConnect的产品ID的变量

```
var product_id: NSString?

重写函数 viewDidLoad() {


product_id = "YOUR_PRODUCT_ID"
    super.viewDidLoad()
SKPaymentQueue.defaultQueue().addTransactionObserver(self)

    //检查产品是否已购买
    if (NSUserDefaults.standardUserDefaults().boolForKey("purchased")){

        // 隐藏广告
adView.隐藏 = 真

    } else {
        打印("应该显示广告..." )

    }

}
```

将按钮连接到购买应用内购买功能的函数

```
@IBAction func 解锁操作(sender: AnyObject) {

    打印("即将获取产品..." )

// 可以进行支付
如果 (SKPaymentQueue.canMakePayments())
    {
        让 productID:NSSet = NSSet(对象: self.product_id!);
        让 productsRequest:SKProductsRequest = SKProductsRequest(productIdentifiers: productID as!
Set<NSString>);
productsRequest.代理 = self;
        productsRequest.开始();
        打印("正在获取产品");
    }否则{
        print("无法进行购买");
    }
```


Chapter 101: In-App Purchase

Section 101.1: Single IAP in Swift 2

After creating an IAP in iTunesConnect:

In the view controller that you want to buy in

```
import StoreKit
```

and add the relevant delegates

```
class ViewController: UIViewController, SKProductsRequestDelegate, SKPaymentTransactionObserver {
```

declare a variable with the product id from iTunesConnect

```
var product_id: NSString?

override func viewDidLoad() {


    product_id = "YOUR_PRODUCT_ID"
    super.viewDidLoad()
    SKPaymentQueue.defaultQueue().addTransactionObserver(self)

   //Check if product is purchased
   if (NSUserDefaults.standardUserDefaults().boolForKey("purchased")){

       // Hide ads
       adView.hidden = true

   } else {
       print("Should show ads...")

   }

}
```

wire a button to a function to purchase the IAP

```
@IBAction func unlockAction(sender: AnyObject) {

   print("About to fetch the product...")

// Can make payments
if (SKPaymentQueue.canMakePayments())
    {
        let productID:NSSet = NSSet(object: self.product_id!);
        let productsRequest:SKProductsRequest = SKProductsRequest(productIdentifiers: productID as!
Set<NSString>);
        productsRequest.delegate = self;
        productsRequest.start();
        println("Fetching Products");
    }else{
        print("Can't make purchases");
    }
```

```
}
```

以下是一些辅助方法

```
func buyProduct(product: SKProduct){
      println("向苹果发送支付请求");
      let payment = SKPayment(product: product)
SKPaymentQueue.defaultQueue().addPayment(payment);

 }
```

必须声明的代理方法

```
func productsRequest (request: SKProductsRequest, didReceiveResponse response: SKProductsResponse)
{

    let count : Int = response.products.count
    if (count>0) {
        var validProduct: SKProduct = response.products[0] as SKProduct
        if (validProduct.productIdentifier == self.product_id) {
            print(validProduct.localizedTitle)
            print(validProduct.localizedDescription)
            print(validProduct.price)
buyProduct(validProduct);
        } else {
            print(validProduct.productIdentifier)
        }
    } else {
        print("nothing")
    }
}


func request(request: SKRequest!, didFailWithError error: NSError!) {
     print("获取产品信息出错");
}

    func paymentQueue(_ queue: SKPaymentQueue,
updatedTransactions transactions: [SKPaymentTransaction])

{
    print("收到来自苹果的支付交易响应");

    for transaction:AnyObject in transactions {
        if let trans:SKPaymentTransaction = transaction as? SKPaymentTransaction{
            switch trans.transactionState {
            case .Purchased:
                print("产品已购买");
SKPaymentQueue.defaultQueue().finishTransaction(transaction as!
SKPaymentTransaction)
                // 处理购买
                NSUserDefaults.standardUserDefaults().setBool(true , forKey: "purchased")
                adView.hidden = true
                break;
            case .Failed:
                print("购买失败");
SKPaymentQueue.defaultQueue().finishTransaction(transaction as!
SKPaymentTransaction)
                break;
```



```
}
```

And here are some helper methods

```
func buyProduct(product: SKProduct){
    println("Sending the Payment Request to Apple");
    let payment = SKPayment(product: product)
    SKPaymentQueue.defaultQueue().addPayment(payment);

 }
```

the delegate methods that must be declared

```
func productsRequest (request: SKProductsRequest, didReceiveResponse response: SKProductsResponse)
{

    let count : Int = response.products.count
    if (count>0) {
        var validProduct: SKProduct = response.products[0] as SKProduct
        if (validProduct.productIdentifier == self.product_id) {
            print(validProduct.localizedTitle)
            print(validProduct.localizedDescription)
            print(validProduct.price)
            buyProduct(validProduct);
        } else {
            print(validProduct.productIdentifier)
        }
    } else {
        print("nothing")
    }
}


func request(request: SKRequest!, didFailWithError error: NSError!) {
    print("Error Fetching product information");
}

    func paymentQueue(_ queue: SKPaymentQueue,
updatedTransactions transactions: [SKPaymentTransaction])

{
    print("Received Payment Transaction Response from Apple");

    for transaction:AnyObject in transactions {
        if let trans:SKPaymentTransaction = transaction as? SKPaymentTransaction{
            switch trans.transactionState {
            case .Purchased:
                print("Product Purchased");
                SKPaymentQueue.defaultQueue().finishTransaction(transaction as!
SKPaymentTransaction)
                // Handle the purchase
                NSUserDefaults.standardUserDefaults().setBool(true , forKey: "purchased")
                adView.hidden = true
                break;
            case .Failed:
                print("Purchased Failed");
                SKPaymentQueue.defaultQueue().finishTransaction(transaction as!
SKPaymentTransaction)
                break;
```

```
            case .Restored:
                print("已购买");
SKPaymentQueue.defaultQueue().restoreCompletedTransactions()


                // 处理购买
                    NSUserDefaults.standardUserDefaults().setBool(true , forKey: "purchased")
                    adView.hidden = true
                    break;
            default:
                break;
            }
        }
    }

}
```

然后是恢复非消耗型应用内购买的代码

```
if (SKPaymentQueue.canMakePayments()) {
  SKPaymentQueue.defaultQueue().restoreCompletedTransactions()
}
```

第101.2节：购买/订阅应用内购买用户的最基本步骤

假设你知道productID：

首先

```
import StoreKit
```

然后在你的代码中

```
let productID: Set = ["premium"]
let request = SKProductsRequest(productIdentifiers: productID)
request.delegate = self
request.start()
```

以及在SKProductsRequestDelegate中：

```
func productsRequest(request: SKProductsRequest, didReceiveResponse response: SKProductsResponse) {
    if response.products.count > 0 {
        let product = response.products[0]
        let payment = SKPayment(product: product)
        SKPaymentQueue.defaultQueue().addPayment(payment)
    }
}
```

第101.3节：在iTunesConnect中设置

在iTunesConnect中，选择您想要添加内购项目（IAP）的应用程序。

点击功能，您将看到如下界面：


```
            case .Restored:
                print("Already Purchased");
               SKPaymentQueue.defaultQueue().restoreCompletedTransactions()


                // Handle the purchase
                    NSUserDefaults.standardUserDefaults().setBool(true , forKey: "purchased")
                    adView.hidden = true
                    break;
            default:
                break;
            }
        }
    }

}
```

And then the code to restore a non-consumable in app purchase

```
if (SKPaymentQueue.canMakePayments()) {
  SKPaymentQueue.defaultQueue().restoreCompletedTransactions()
}
```

Section 101.2: Most basic steps for purchasing/subscribing a user to an IAP

Assuming you know the `productID`:

First

```
import StoreKit
```

Then in your code

```
let productID: Set = ["premium"]
let request = SKProductsRequest(productIdentifiers: productID)
request.delegate = self
request.start()
```

and in the `SKProductsRequestDelegate`:

```
func productsRequest(request: SKProductsRequest, didReceiveResponse response: SKProductsResponse) {
    if response.products.count > 0 {
        let product = response.products[0]
        let payment = SKPayment(product: product)
        SKPaymentQueue.defaultQueue().addPayment(payment)
    }
}
```

Section 101.3: Set Up in iTunesConnect

In iTunesConnect, select the app which you want to add an IAP to.

Click on features and you will see this:

In-App Purchases (0) ⊕ View Shared Secret

Click + to add an In-App Purchase.

点击加号。然后您需要选择想要制作的内购项目类型。

接着，您需要填写内购项目的所有信息。

In-App Purchase Summary

Enter a reference name and a product ID for this In-App Purchase.

Reference Name:

Product ID:

Pricing and Availability

Enter the pricing and availability details for this In-App Purchase below.

Cleared for Sale Yes No

Price Tier Select

View Pricing Matrix

如果遇到任何问题，您可以参考内购项目设置指南。

In-App Purchases (0) ⊕ View Shared Secret

Click + to add an In-App Purchase.

Click the plus. You will then need to select which type of IAP you want to make.

Then you will need to fill out all of the information for your IAP.

In-App Purchase Summary

Enter a reference name and a product ID for this In-App Purchase.

Reference Name:

Product ID:

Pricing and Availability

Enter the pricing and availability details for this In-App Purchase below.

Cleared for Sale Yes No

Price Tier Select

View Pricing Matrix

If you have any trouble you can consult the IAP Set Up Guide.

第102章：CGContext参考

第102.1节：绘制线条

```
CGContextRef context = UIGraphicsGetCurrentContext();

CGContextSetLineWidth(context, 5.0);
CGColorSpaceRef colorspace = CGColorSpaceCreateDeviceRGB();
CGContextMoveToPoint(context, 200, 400);
CGContextAddLineToPoint(context, 100, 100);
CGContextStrokePath(context);
CGColorSpaceRelease(colorspace);
```

第102.2节：绘制文本

绘制需要在构建阶段添加Core Text框架

```
[NSString* textToDraw = @"Welcome to the world Of IOS";

    CFStringRef stringRef = (__bridge CFStringRef)textToDraw;

    CFAttributedStringRef currentText = CFAttributedStringCreate(NULL, stringRef, NULL);
    CTFramesetterRef framesetter = CTFramesetterCreateWithAttributedString(currentText);
    CGRect frameRect = CGRectMake(0, 0, 300, 100);
    CGMutablePathRef framePath = CGPathCreateMutable();
    CGPathAddRect(framePath, NULL, frameRect);

    CFRange currentRange = CFRangeMake(0, 0);
CTFrameRef frameRef = CTFramesetterCreateFrame(framesetter, currentRange, framePath, NULL);
    CGPathRelease(framePath);
    CGContextRef currentContext = UIGraphicsGetCurrentContext();

CGContextSetTextMatrix(currentContext, CGAffineTransformIdentity);
    CGContextTranslateCTM(currentContext, 200, 300);
CGContextScaleCTM(currentContext, 2, -2);
    CTFrameDraw(frameRef, currentContext);

    CFRelease(frameRef);
CFRelease(stringRef);
    CFRelease(framesetter);
```


Chapter 102: CGContext Reference

Section 102.1: Draw line

```
CGContextRef context = UIGraphicsGetCurrentContext();

CGContextSetLineWidth(context, 5.0);
CGColorSpaceRef colorspace = CGColorSpaceCreateDeviceRGB();
CGContextMoveToPoint(context, 200, 400);
CGContextAddLineToPoint(context, 100, 100);
CGContextStrokePath(context);
CGColorSpaceRelease(colorspace);
```

Section 102.2: Draw Text

Draw To requires **Core Text framework** to be added in the Build Phase

```
[NSString* textToDraw = @"Welcome to the world Of IOS";

    CFStringRef stringRef = (__bridge CFStringRef)textToDraw;

    CFAttributedStringRef currentText = CFAttributedStringCreate(NULL, stringRef, NULL);
    CTFramesetterRef framesetter = CTFramesetterCreateWithAttributedString(currentText);
    CGRect frameRect = CGRectMake(0, 0, 300, 100);
    CGMutablePathRef framePath = CGPathCreateMutable();
    CGPathAddRect(framePath, NULL, frameRect);

    CFRange currentRange = CFRangeMake(0, 0);
    CTFrameRef frameRef = CTFramesetterCreateFrame(framesetter, currentRange, framePath, NULL);
    CGPathRelease(framePath);
    CGContextRef currentContext = UIGraphicsGetCurrentContext();

    CGContextSetTextMatrix(currentContext, CGAffineTransformIdentity);
    CGContextTranslateCTM(currentContext, 200, 300);
    CGContextScaleCTM(currentContext, 2, -2);
    CTFrameDraw(frameRef, currentContext);

    CFRelease(frameRef);
    CFRelease(stringRef);
    CFRelease(framesetter);
```

Carrier
4:03 PM
Back
Welcome to the world Of IOS

Carrier
4:03 PM
Back
Welcome to the world Of IOS

第103章：核心定位

第103.1节：请求使用定位服务的权限

使用以下方法检查应用的授权状态：

```
//Swift
let status: CLAuthorizationStatus = CLLocationManager.authorizationStatus()

//Objective-C
CLAuthorizationStatus status = [CLLocationManager authorizationStatus];
```

将状态与以下常量进行比较：

```
//Swift
switch 状态 {
case .未确定:
    // 执行操作
case .始终授权:
    // 执行操作
case .使用时授权:
    // 执行操作
case .受限:
    // 执行操作
case .拒绝:
    // 执行操作
}

//Objective-C
switch (状态) {
    case kCLAuthorizationStatusNotDetermined:

        // 用户尚未选择是否允许您的应用使用定位服务。

        break;

    case kCLAuthorizationStatusAuthorizedAlways:

        //用户已允许您的应用始终使用定位服务，即使应用在后台。


        break;

    case kCLAuthorizationStatusAuthorizedWhenInUse:

        //用户仅允许您的应用在前台使用定位服务。

        break;

    case kCLAuthorizationStatusRestricted:

        //用户无法选择是否允许您的应用使用定位服务，这可能是由于家长控制等原因。


        break;

    case kCLAuthorizationStatusDenied:

        //用户已选择不允许您的应用使用定位服务。
```


Chapter 103: Core Location

Section 103.1: Request Permission to Use Location Services

Check the app's authorization status with:

```
//Swift
let status: CLAuthorizationStatus = CLLocationManager.authorizationStatus()

//Objective-C
CLAuthorizationStatus status = [CLLocationManager authorizationStatus];
```

Test the status against the follow constants:

```
//Swift
switch status {
case .NotDetermined:
    // Do stuff
case .AuthorizedAlways:
    // Do stuff
case .AuthorizedWhenInUse:
    // Do stuff
case .Restricted:
    // Do stuff
case .Denied:
    // Do stuff
}

//Objective-C
switch (status) {
    case kCLAuthorizationStatusNotDetermined:

        //The user hasn't yet chosen whether your app can use location services or not.

        break;

    case kCLAuthorizationStatusAuthorizedAlways:

        //The user has let your app use location services all the time, even if the app is in the
background.

        break;

    case kCLAuthorizationStatusAuthorizedWhenInUse:

        //The user has let your app use location services only when the app is in the foreground.

        break;

    case kCLAuthorizationStatusRestricted:

        //The user can't choose whether or not your app can use location services or not, this
could be due to parental controls for example.

        break;

    case kCLAuthorizationStatusDenied:

        //The user has chosen to not let your app use location services.
```

```
        break;

    default:
        break;
}
```

在应用使用时获取定位服务权限

最简单的方法是将定位管理器初始化为根视图控制器的属性，并将权限请求放在其viewDidLoad中。

这会弹出请求权限的警告控制器：

```
//Swift
let locationManager = CLLocationManager()
locationManager.requestWhenInUseAuthorization()

//Objective-C
CLLocationManager *locationManager = [[CLLocationManager alloc] init];
[locationManager requestWhenInUseAuthorization];
```

在你的Info.plist中添加NSLocationWhenInUseUsageDescription键。该值将用于警告控制器的message标签。


```
        break;

    default:
        break;
}
```

Getting Location Service Permission While App is in Use

Simplest method is to initialize the location manager as a property of your root view controller and place the permission request in its `viewDidLoad`.

This brings up the alert controller that asks for permission:

```
//Swift
let locationManager = CLLocationManager()
locationManager.requestWhenInUseAuthorization()

//Objective-C
CLLocationManager *locationManager = [[CLLocationManager alloc] init];
[locationManager requestWhenInUseAuthorization];
```

Add the **NSLocationWhenInUseUsageDescription** key to your *Info.plist*. The value will be used in the alert controller's `message` label.

获取始终允许使用定位服务的权限

如果想请求即使应用未激活时也能使用定位服务的权限，请使用以下调用：

```
//Swift
locationManager.requestAlwaysAuthorization()

//Objective-C
[locationManager requestAlwaysAuthorization];
```

然后在你的Info.plist中添加NSLocationAlwaysUsageDescription键。同样，该值将用于警告控制器的message标签。

Getting Location Service Permission Always

To ask for permission to use location services even when the app is not active, use the following call instead:

```
//Swift
locationManager.requestAlwaysAuthorization()

//Objective-C
[locationManager requestAlwaysAuthorization];
```

Then add the **NSLocationAlwaysUsageDescription** key to your *Info.plist*. Again, the value will be used in the alert controller's `message` label.

第103.2节：使用GPX文件添加自定义位置

要检查定位服务，我们需要真实设备，但出于测试目的，我们也可以使用模拟器并通过以下步骤添加我们自己的位置：

- 将新的GPX文件添加到您的项目中。
- 在GPX文件中添加航点，如下所示

```
<?xml version="1.0"?>
<gpx version="1.1" creator="Xcode">
<!--
提供一个或多个包含纬度/经度对的航点。如果只提供一个航点，Xcode将模拟该特定位置。如果提供多个航点，Xcode将模
拟访问每个航点的路线。

-->
<wpt lat="52.599878" lon="4.702029">
      <name>位置名称（例如佛罗里达）</name>
</wpt>
```

- 然后进入产品-->方案-->编辑方案，在运行（RUN）中将默认位置设置为您的GPX文件名。

第103.3节：链接CoreLocation框架

Section 103.2: Add own custom location using GPX file

To check for location services we need real device but for testing purpose we can also use simulator and add our own location by following below steps:

- add new GPX file into your project.
- in GPX file add waypoints like

```
<?xml version="1.0"?>
<gpx version="1.1" creator="Xcode">
<!--
      Provide one or more waypoints containing a latitude/longitude pair. If you provide one
      waypoint, Xcode will simulate that specific location. If you provide multiple waypoints,
      Xcode will simulate a route visitng each waypoint.
-->
<wpt lat="52.599878" lon="4.702029">
     <name>location name (eg. Florida)</name>
</wpt>
```

- then go to product-->Scheme-->Edit Scheme and into RUN set default location as your GPX file name.

Section 103.3: Link CoreLocation Framework

在使用 CoreLocation 功能的类中导入 CoreLocation 模块。

```
//Swift
import CoreLocation

//Objective-C
#import <CoreLocation/CoreLocation.h>
```

第 103.4 节：后台位置服务

要在应用程序处于后台时使用标准位置服务，首先需要在目标设置的功能（Capabilities）选项卡中开启后台模式（Background Modes）并选择位置更新（Location updates）。

或者，直接将其添加到 Info.plist 中。

```
<key>NSLocationAlwaysUsageDescription</key>
<string>我想在后台获取您的位置信息</string>
```

Import the CoreLocation module in your classes that use CoreLocation functionality.

```
//Swift
import CoreLocation

//Objective-C
#import <CoreLocation/CoreLocation.h>
```

Section 103.4: Location Services in the Background

To use standard location services while the application is in the background you need first turn on `Background Modes` in the Capabilities tab of the Target settings, and select `Location updates`.

Or, add it directly to the Info.plist.

```
<key>NSLocationAlwaysUsageDescription</key>
<string>I want to get your location Information in background</string>
```

```
<key>UIBackgroundModes</key>
<array>
    位置
```

然后你需要设置 CLLocationManager

Objective C

```
//位置管理器必须有强引用。
_locationManager = [[CLLocationManager alloc] init];
_locationManager.delegate = self;

//请求始终授权（iOS8及以上）
if ([_locationManager respondsToSelector:@selector(requestAlwaysAuthorization)]) {
    [_locationManager requestAlwaysAuthorization];
}

//允许后台定位更新（iOS9及以上）
if ([_locationManager respondsToSelector:@selector(allowsBackgroundLocationUpdates)]) {
    _locationManager.allowsBackgroundLocationUpdates = YES;
}

[_locationManager startUpdatingLocation];
```

Swift

```
self.locationManager.delegate = self

if #available (iOS 8.0,*) {
    self.locationManager.requestAlwaysAuthorization()
}

if #available (iOS 9.0,*) {
    self.locationManager.allowsBackgroundLocationUpdates = true
}

self.locationManager.startUpdatingLocation()
```



```
<key>UIBackgroundModes</key>
<array>
    <string>location</string>
</array>
```

Then you need to setup the CLLocationManager

Objective C

```
//The Location Manager must have a strong reference to it.
_locationManager = [[CLLocationManager alloc] init];
_locationManager.delegate = self;

//Request Always authorization (iOS8+)
if ([_locationManager respondsToSelector:@selector(requestAlwaysAuthorization)]) {
    [_locationManager requestAlwaysAuthorization];
}

//Allow location updates in the background (iOS9+)
if ([_locationManager respondsToSelector:@selector(allowsBackgroundLocationUpdates)]) {
    _locationManager.allowsBackgroundLocationUpdates = YES;
}

[_locationManager startUpdatingLocation];
```

Swift

```
self.locationManager.delegate = self

if #available (iOS 8.0,*) {
    self.locationManager.requestAlwaysAuthorization()
}

if #available (iOS 9.0,*) {
    self.locationManager.allowsBackgroundLocationUpdates = true
}

self.locationManager.startUpdatingLocation()
```

第104章：FacebookSDK

第104.1节：创建你自己的自定义“使用Facebook登录”按钮

有时我们想设计自己的“使用Facebook登录”按钮的用户界面，而不是使用FacebookSDK自带的原始按钮。

1. 在你的故事板中，拖拽UIButton并按你想要的方式设置它。
2. 按住Ctrl键，将按钮拖拽到视图控制器，作为IBAction连接。
3. **在IBAction方法内部，你需要模拟点击实际的Facebook按钮，方法如下：**

Swift:

```
 let loginButton = FBSDKLoginButton()
loginButton.delegate = self
 // 你的自定义权限数组
loginButton.readPermissions =
 [
                              "public_profile",
                              "email",
                              "user_about_me",
                              "user_photos"
 ]
 // 隐藏按钮
loginButton.hidden = true
 self.view.addSubview(loginButton)
 // 模拟点击实际的 Facebook SDK 按钮
loginButton.sendActionsForControlEvents(UIControlEvents.TouchUpInside)
```

Objective-C:

```
FBSDKLoginButton *FBButton = [FBSDKLoginButton new];

// 你的自定义权限数组
FBButton.readPermissions = @[@"public_profile",
                             @"email",
@"user_about_me",
                             @"user_photos"
                             ];
FBButton.loginBehavior = FBSDKLoginBehaviorNative;
[FBButton setDelegate:self];
[FBButton setHidden:true];
[loginButton addSubview:FBButton];

[FBButton sendActionsForControlEvents:UIControlEventTouchUpInside];
```

你完成了。

第104.2节：FacebookSDK集成

步骤1：安装SDK

你可以手动安装SDK，或者通过CocoaPods安装。强烈推荐后者。

将以下内容写入Podfile：

Chapter 104: FacebookSDK

Section 104.1: Creating your own custom "Sign In With Facebook" button

Sometimes we want to design our own UI for "Sign In With Facebook" button instead of the original button that comes with FacebookSDK.

1. In your storyboard, drag your UIButton and set it however you want it to be.
2. Ctrl + drag your button to your view controller as IBAction.
3. **Inside** the IBAction method you will have simulate a tap on the actual Facebook button as follow:

Swift:

```
let loginButton = FBSDKLoginButton()
loginButton.delegate = self
// Your Custom Permissions Array
loginButton.readPermissions =
[
                          "public_profile",
                          "email",
                          "user_about_me",
                          "user_photos"
]
// Hiding the button
loginButton.hidden = true
self.view.addSubview(loginButton)
// Simulating a tap for the actual Facebook SDK button
loginButton.sendActionsForControlEvents(UIControlEvents.TouchUpInside)
```

Objective-C:

```
FBSDKLoginButton *FBButton = [FBSDKLoginButton new];

// Your Custom Permissions Array
FBButton.readPermissions = @[@"public_profile",
                             @"email",
                             @"user_about_me",
                             @"user_photos"
                             ];
FBButton.loginBehavior = FBSDKLoginBehaviorNative;
[FBButton setDelegate:self];
[FBButton setHidden:true];
[loginButton addSubview:FBButton];

[FBButton sendActionsForControlEvents:UIControlEventTouchUpInside];
```

You're done.

Section 104.2: FacebookSDK Integration

Step 1: Install the SDK

You can install the SDK manually or via `CocoaPods`. The latter option is highly recommended.

Put these lines in `Podfile`:

```
target 'MyApp' do
  use_frameworks!

pod 'FBSDKCoreKit'
  pod 'FBSDKLoginKit'
  pod 'FBSDKShareKit'
end
```

在终端运行 pod install ，然后打开 .xcworkspace 而不是 .xcodeproj 。

FBSDKLoginKit 和 FBSDKShareKit 是可选的。你可以选择是否需要它们。

步骤2：在Facebook上创建应用

前往 Quick Starts - Facebook for Developers 创建一个应用。

创建应用后，Facebook会要求你下载SDK。如果你已经通过

CocoaPods安装了SDK，可以跳过这一步。

步骤3：编辑 .plist

a. 为了让你的应用能够与Facebook"通信"，你需要在 .plist 文件中添加一些设置。Facebook会在Quick Starts页面提供定制的代码片段。

b. 编辑您的.plist文件作为源代码。

c. 将您自定义的代码片段粘贴到源代码中。注意！该代码片段必须是<dict>标签的直接子元素。您的源代码应类似于：

```
<plist version="1.0">
<dict>
    // ...
//一些默认设置
    // ...
    <key>CFBundleURLTypes</key>
    <array>
        <dict>
            <key>CFBundleURLSchemes</key>
            <array>
                <string>fb{FBAppId}</string>
            </array>


    <key>FacebookAppID</key>
    <string>{FBAppId}</string>
```



```
target 'MyApp' do
  use_frameworks!

  pod 'FBSDKCoreKit'
  pod 'FBSDKLoginKit'
  pod 'FBSDKShareKit'
end
```

Run `pod install` in the terminal and open `.xcworkspace` instead of `.xcodeproj` afterwards.

`FBSDKLoginKit` and `FBSDKShareKit` are optional. You may or may not need them.

Step 2: Create an app on Facebook

Go to Quick Starts - Facebook for Developers to create an app.

Facebook will ask you to download the SDK after creating the app. You can skip this part if you installed the SDK via CocoaPods already.

Step 3: Edit `.plist`

a. To make your app able to "communicate" with Facebook, you need to put some settings in your `.plist` file. Facebook will give you the customized snippet on the Quick Starts page.

b. Edit your `.plist` file as source code.

c. Paste your customized snippet in the source code. **Be careful!** The snippet must be exactly the child of the **`<dict>`** tag. Your source code should be something like:

```
<plist version="1.0">
<dict>
    // ...
    //some default settings
    // ...
    <key>CFBundleURLTypes</key>
    <array>
        <dict>
            <key>CFBundleURLSchemes</key>
            <array>
                <string>fb{FBAppId}</string>
            </array>
        </dict>
    </array>
    <key>FacebookAppID</key>
    <string>{FBAppId}</string>
```

```
    <key>FacebookDisplayName</key>
    <string>{FBAppName}</string>
    <key>LSApplicationQueriesSchemes</key>
    <array>
        fbapi
        fb-messenger-api
        fbauth2
        fbshareextension

<key>NSAppTransportSecurity</key>
<dict>
    <key>NSExceptionDomains</key>
    <dict>
        <key>facebook.com</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSExceptionRequiresForwardSecrecy</key>
            <false/>

        <key>fbcdn.net</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSExceptionRequiresForwardSecrecy</key>
            <false/>

        <key>akamaihd.net</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSExceptionRequiresForwardSecrecy</key>
            <false/>
        </dict>
    </dict>

</plist>
```

如果你将代码片段粘贴到错误的位置，会遇到问题。

步骤4：在快速入门页面告诉Facebook你的应用包标识符。

=> 如何获取包标识符

步骤5：编辑你的AppDelegate.swift

a.

```
import FBSDKCoreKit
```

b.

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject: AnyObject]?) -> Bool {
FBSDKApplicationDelegate.sharedInstance().application(application,
didFinishLaunchingWithOptions: launchOptions)
    返回 true
}

func application(application: UIApplication, openURL url: NSURL, sourceApplication: String?,
```



```
    <key>FacebookDisplayName</key>
    <string>{FBAppName}</string>
    <key>LSApplicationQueriesSchemes</key>
    <array>
        <string>fbapi</string>
        <string>fb-messenger-api</string>
        <string>fbauth2</string>
        <string>fbshareextension</string>
    </array>
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSExceptionDomains</key>
    <dict>
        <key>facebook.com</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSExceptionRequiresForwardSecrecy</key>
            <false/>
        </dict>
        <key>fbcdn.net</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSExceptionRequiresForwardSecrecy</key>
            <false/>
        </dict>
        <key>akamaihd.net</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSExceptionRequiresForwardSecrecy</key>
            <false/>
        </dict>
    </dict>
</dict>
</plist>
```

If you paste the snippet at a wrong place, you will run into problems.

Step 4: Tell Facebook your bundle identifier on the Quick Starts page.

=> How to get bundle identifier

Step 5: Edit your `AppDelegate.swift`

a.

```
import FBSDKCoreKit
```

b.

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject: AnyObject]?) -> Bool {
    FBSDKApplicationDelegate.sharedInstance().application(application,
didFinishLaunchingWithOptions: launchOptions)
    return true
}

func application(application: UIApplication, openURL url: NSURL, sourceApplication: String?,
```

```
annotation: AnyObject) -> Bool {
    return FBSDKApplicationDelegate.sharedInstance().application(application, openURL: url,
sourceApplication: sourceApplication, annotation: annotation)
}
```

第104.3节：获取Facebook用户数据

用户在您的应用中登录Facebook后，现在是时候获取您请求的数据了 `FBButton.readPermissions`.

Swift:

```
enum FacebookParametesField : String
{
    case FIELDS_KEY = "fields"
    case FIELDS_VALUE = "id, email, picture, first_name, last_name"
}


if FBSDKAccessToken.currentAccessToken() != nil
{
//  获取用户的Facebook数据
FBSDKGraphRequest(graphPath: "me",
parameters: [FacebookParametesField.FIELDS_KEY.rawValue :
FacebookParametesField.FIELDS_VALUE.rawValue])
.startWithCompletionHandler({ (graphConnection : FBSDKGraphRequestConnection!, result : AnyObject!,
error : NSError!) -> Void in

        if error == nil
        {
            print("Facebook Graph phaze")

            let email = result["email"]
            let facebookToken = FBSDKAccessToken.currentAccessToken().tokenString
            let userFacebookId = result["id"]
            let firstName = result["first_name"]
            let lastName = result["last_name"]

            if let result = result as? Dictionary<String, AnyObject>
            {
                if let picture = result["picture"] as? Dictionary<String,AnyObject>
                {
                    if let data = picture["data"] as? Dictionary <String,AnyObject>
                    {
                        if let url = data["url"] as? String
                        {
                            //  头像图片 URL
                            let profilePictureURL = url
                        }
                    }
                }
            }
        }
    })
}
```



```
annotation: AnyObject) -> Bool {
    return FBSDKApplicationDelegate.sharedInstance().application(application, openURL: url,
sourceApplication: sourceApplication, annotation: annotation)
}
```

Section 104.3: Fetching the facebook user data

After the user signed in to Facebook at your app, now it's time to fetch the data you requested at the `FBButton.readPermissions`.

Swift:

```
enum FacebookParametesField : String
{
    case FIELDS_KEY = "fields"
    case FIELDS_VALUE = "id, email, picture, first_name, last_name"
}


if FBSDKAccessToken.currentAccessToken() != nil
{
//  Getting user facebook data
FBSDKGraphRequest(graphPath: "me",
                  parameters: [FacebookParametesField.FIELDS_KEY.rawValue :
FacebookParametesField.FIELDS_VALUE.rawValue])
.startWithCompletionHandler({ (graphConnection : FBSDKGraphRequestConnection!, result : AnyObject!,
error : NSError!) -> Void in

        if error == nil
        {
            print("Facebook Graph phaze")

            let email = result["email"]
            let facebookToken = FBSDKAccessToken.currentAccessToken().tokenString
            let userFacebookId = result["id"]
            let firstName = result["first_name"]
            let lastName = result["last_name"]

            if let result = result as? Dictionary<String, AnyObject>
            {
                if let picture = result["picture"] as? Dictionary<String,AnyObject>
                {
                    if let data = picture["data"] as? Dictionary <String,AnyObject>
                    {
                        if let url = data["url"] as? String
                        {
                            //  Profile picture URL
                            let profilePictureURL = url
                        }
                    }
                }
            }
        }
    })
}
```

第105章：AFNetworking

第105.1节：在自定义线程上调度完成块

每当使用AFNetworking时，调用会被调度到AFNetworking提供的自定义线程。当调用返回到完成块时，它会在主线程上执行。

此示例设置了一个调度到完成块的自定义线程：

AFNetworking 2.xx：

```
// 使用您的名称和DISPATCH_QUEUE_SERIAL作为标志创建dispatch_queue_t
dispatch_queue_t myQueue = dispatch_queue_create("com.CompanyName.AppName.methodTest",
                    DISPATCH_QUEUE_SERIAL);

// 初始化AFNetworking的AFHTTPRequestOperation
operation = [[AFHTTPRequestOperation alloc] initWithRequest:request];

// 设置FMDB属性以在主线程之外运行
[operation setCompletionQueue:myQueue];
```

AFNetworking 3.xx：

```
AFHTTPSessionManager *manager = [[AFHTTPSessionManager alloc] init];
[self setCompletionQueue:myQueue];
```


Chapter 105: AFNetworking

Section 105.1: Dispatching completion block on a custom thread

Whenever AFNetworking is used the call is dispatched on a custom thread provided by AFNetworking. When the call returns to the completion block, it gets executed on the main thread.

This example sets a custom thread that dispatch to the completion block:

AFNetworking 2.xx:

```
// Create dispatch_queue_t with your name and DISPATCH_QUEUE_SERIAL as for the flag
dispatch_queue_t myQueue = dispatch_queue_create("com.CompanyName.AppName.methodTest",
                    DISPATCH_QUEUE_SERIAL);

// init AFHTTPRequestOperation of AFNetworking
operation = [[AFHTTPRequestOperation alloc] initWithRequest:request];

// Set the FMDB property to run off the main thread
[operation setCompletionQueue:myQueue];
```

AFNetworking 3.xx:

```
AFHTTPSessionManager *manager = [[AFHTTPSessionManager alloc] init];
[self setCompletionQueue:myQueue];
```

第106章：CTCallCenter

第106.1节：CallKit - iOS 10

```
//头文件

<CallKit/CXCallObserver.h>

CXCallObserver *callObserver = [[CXCallObserver alloc] init];

// 如果队列为nil，则回调将在主队列上执行

[callObserver setDelegate:self queue:nil];

// 不要忘记保存对callObserver的引用，以防止其被释放

self.callObserver = callObserver;

// 获取通话状态
- (void)callObserver:(CXCallObserver *)callObserver callChanged:(CXCall *)call {
    if (call.hasConnected) {
        // 执行必要的操作
    }
}
```

第106.2节：即使在后台也能拦截来自您应用的通话

来自苹果文档：

> 使用 CTCallCenter 类获取当前蜂窝电话的列表，并响应电话状态的变化，例如从拨号状态到连接状态。这些状态变化称为蜂窝电话事件。

CTCallCenter 的目的是让开发者有机会在通话期间暂停应用状态，以便为用户提供最佳体验。

Objective-C:

首先，我们将在想要处理拦截的类中定义一个新的类成员：

```
@property (atomic, strong) CTCallCenter *callCenter;
```

在类的初始化方法（构造函数）中，我们将为类成员分配新的内存：

```
[self setCallCenter:[CTCallCenter new]];
```

随后，我们将调用实际处理拦截的新方法：

```
- (void)registerPhoneCallListener
{
[[self callCenter] setCallEventHandler:^(CTCall * _Nonnull call) {
    NSLog(@"CallEventHandler called - interception in progress");
```


Chapter 106: CTCallCenter

Section 106.1: CallKit - ios 10

```
//Header File

<CallKit/CXCallObserver.h>

CXCallObserver *callObserver = [[CXCallObserver alloc] init];

// If queue is nil, then callbacks will be performed on main queue

[callObserver setDelegate:self queue:nil];

// Don't forget to store reference to callObserver, to prevent it from being released

self.callObserver = callObserver;

// get call status
- (void)callObserver:(CXCallObserver *)callObserver callChanged:(CXCall *)call {
    if (call.hasConnected) {
        // perform necessary actions
    }
}
```

Section 106.2: Intercepting calls from your app even from the background

From Apple documentation:

> Use the CTCallCenter class to obtain a list of current cellular calls, and to respond to state changes for calls such as from a dialing state to a connected state. Such state changes are known as cellular call events.

The purpose of CTCallCenter is to give the developer the opportunity to pause his app state during a call in order to give the user the best experience.

Objective-C:

First, we will define a new class member inside the class we want to handle the interceptions:

```
@property (atomic, strong) CTCallCenter *callCenter;
```

Inside our class init (constructor) we will allocate new memory for our class member:

```
[self setCallCenter:[CTCallCenter new]];
```

Afterwards, we will invoke our new method that actually handles the interceptions:

```
- (void)registerPhoneCallListener
{
[[self callCenter] setCallEventHandler:^(CTCall * _Nonnull call) {
    NSLog(@"CallEventHandler called - interception in progress");
```

```
    if ([call.callState isEqualToString: CTCallStateConnected])
    {
NSLog(@"已连接");
    }
    else if ([call.callState isEqualToString: CTCallStateDialing])
    {
NSLog(@"拨号中");
    }
    else if ([call.callState isEqualToString: CTCallStateDisconnected])
    {
NSLog(@"已断开");

    } else if ([call.callState isEqualToString: CTCallStateIncoming])
    {
NSLog(@"来电中");
    }
 }];
}
```

就是这样，如果用户使用你的应用并接到电话，你可以拦截此电话并处理你的应用
以保存状态。

值得一提的是，有4种通话状态你可以拦截：

```
CTCallStateDialing
CTCallStateIncoming
CTCallStateConnected
CTCallStateDisconnected
```

Swift:

在相关类中定义你的类成员并进行定义：

```
self.callCenter = CTCallCenter()
self.callCenter.callEventHandler = { call in
    // 处理你的拦截
    if call.callState == CTCallStateConnected
    {
    }
}
```

如果你的应用在后台，并且你需要在应用处于后台时拦截通话，会发生什么？

例如，如果你开发的是一个企业应用，你基本上只需在“功能”标签中添加两个功能（VOIP 和后台获取）：

你的项目目标 -> 功能 -> 后台模式 -> 勾选“语音 over IP”和“后台获取”


```
    if ([call.callState isEqualToString: CTCallStateConnected])
    {
        NSLog(@"Connected");
    }
    else if ([call.callState isEqualToString: CTCallStateDialing])
    {
        NSLog(@"Dialing");
    }
    else if ([call.callState isEqualToString: CTCallStateDisconnected])
    {
        NSLog(@"Disconnected");

    } else if ([call.callState isEqualToString: CTCallStateIncoming])
    {
        NSLog(@"Incomming");
    }
 }];
}
```

That's it, if the user will use your app and will receive a phone call you could intercept this call and handle your app for a save state.

It is worth mentioning that there are 4 call states you can intercept:

```
CTCallStateDialing
CTCallStateIncoming
CTCallStateConnected
CTCallStateDisconnected
```

Swift:

Define your class member at the relevant class and define it:

```
self.callCenter = CTCallCenter()
self.callCenter.callEventHandler = { call in
    //  Handle your interception
    if call.callState == CTCallStateConnected
    {
    }
}
```

What will happen if your app is in the background and you need to intercept calls while the app is in the background ?

For example, if you develop an **enterprise** app you can basically just add 2 capabilities (VOIP & Background fetch) in the Capabilities tab:

Your project target -> Capabilities -> Background Modes -> mark Voice over IP & Background fetch

第107章：推送通知

参数	描述
用户信息	一个包含远程通知信息的字典，可能包括应用图标的徽章数字、提醒声音、提醒消息、通知标识符和自定义数据。

第107.1节：注册设备以接收推送通知

要注册设备以接收推送通知，请将以下代码添加到您的AppDelegate文件中的 `didFinishLaunchingWithOptions`方法：

Swift

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject: AnyObject]?) -> Bool {
    // 应用启动后自定义的重写点。
    if UIDevice.currentDevice().systemVersion.compare(v, options: .NumericSearch) ==
NSOrderedAscending {
        // 如果运行的是iOS 8以下版本，则注册推送通知
        if application.respondsToSelector("registerUserNotificationSettings:") {
            let types:UIUserNotificationType = (.Alert | .Badge | .Sound)
            let settings:UIUserNotificationSettings = UIUserNotificationSettings(forTypes: types,
categories: nil)

application.registerUserNotificationSettings(settings)
            application.registerForRemoteNotifications()
        } 否则 {
            // 在 iOS 8 之前注册推送通知
          application.registerForRemoteNotificationTypes(.Alert | .Badge | .Sound)
        }
    } else {
        var center = UNUserNotificationCenter.currentNotificationCenter()
        center.delegate = self
center.requestAuthorizationWithOptions((UNAuthorizationOptionSound |
UNAuthorizationOptionAlert | UNAuthorizationOptionBadge)) {(granted: Bool, error: NSError) -> Void
in
            if !error {
                UIApplication.sharedApplication().registerForRemoteNotifications()
                // 需要此操作才能让应用处理推送通知
                print("推送注册成功。")
            } else {
                print("推送注册失败")
                print("错误: \(error.localizedFailureReason!) - \(error.localizedDescription)")
                print("建议: \(error.localizedRecoveryOptions) -
\(error.localizedRecoverySuggestion!)")
            })
    }

    return true
}
```

Objective-C

```
#define SYSTEM_VERSION_LESS_THAN(v) ([[[UIDevice currentDevice] systemVersion] compare:v
options:NSNumericSearch] == NSOrderedAscending)

if( SYSTEM_VERSION_LESS_THAN( @"10.0" ) )
{
        if ([application respondsToSelector:@selector(isRegisteredForRemoteNotifications)])
    {
```


Chapter 107: Push Notifications

Parameter	Description
userInfo	A dictionary that contains remote notification info, potentially including a badge number for the app icon, alert sound, alert message, a notification identifier, and custom data.

Section 107.1: Registering device for Push Notifications

To register your device for push notifications, add the following code to your AppDelegate file in `didFinishLaunchingWithOptions` method:

Swift

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject: AnyObject]?) -> Bool {
    // Override point for customization after application launch.
    if UIDevice.currentDevice().systemVersion.compare(v, options: .NumericSearch) ==
NSOrderedAscending {
        // Register for Push Notitications, if running iOS < 8
        if application.respondsToSelector("registerUserNotificationSettings:") {
            let types:UIUserNotificationType = (.Alert | .Badge | .Sound)
            let settings:UIUserNotificationSettings = UIUserNotificationSettings(forTypes: types,
categories: nil)

            application.registerUserNotificationSettings(settings)
            application.registerForRemoteNotifications()
        } else {
            // Register for Push Notifications before iOS 8
            application.registerForRemoteNotificationTypes(.Alert | .Badge | .Sound)
        }
    } else {
        var center = UNUserNotificationCenter.currentNotificationCenter()
        center.delegate = self
        center.requestAuthorizationWithOptions((UNAuthorizationOptionSound |
UNAuthorizationOptionAlert | UNAuthorizationOptionBadge)) {(granted: Bool, error: NSError) -> Void
in
            if !error {
                UIApplication.sharedApplication().registerForRemoteNotifications()
                // required to get the app to do anything at all about push notifications
                print("Push registration success.")
            } else {
                print("Push registration FAILED")
                print("ERROR: \(error.localizedFailureReason!) - \(error.localizedDescription)")
                print("SUGGESTIONS: \(error.localizedRecoveryOptions) -
\(error.localizedRecoverySuggestion!)")
            })
    }

    return true
}
```

Objective-C

```
#define SYSTEM_VERSION_LESS_THAN(v) ([[[UIDevice currentDevice] systemVersion] compare:v
options:NSNumericSearch] == NSOrderedAscending)

if( SYSTEM_VERSION_LESS_THAN( @"10.0" ) )
{
        if ([application respondsToSelector:@selector(isRegisteredForRemoteNotifications)])
    {
```

```
        // iOS 8 通知
        [application registerUserNotificationSettings:[UIUserNotificationSettings
settingsForTypes:(UIUserNotificationTypeSound | UIUserNotificationTypeAlert |
UIUserNotificationTypeBadge) categories:nil]];

        [application registerForRemoteNotifications];
    }
    else
    {
        // iOS 8 以下通知
        [application registerForRemoteNotificationTypes:
         (UIRemoteNotificationTypeBadge | UIRemoteNotificationTypeAlert |
UIRemoteNotificationTypeSound)];
    }

}
else
{
UNUserNotificationCenter *center = [UNUserNotificationCenter currentNotificationCenter];
    center.delegate = self;
    [center requestAuthorizationWithOptions:(UNAuthorizationOptionSound |
UNAuthorizationOptionAlert | UNAuthorizationOptionBadge) completionHandler:^(BOOL granted, NSError
* _Nullable error)
     {
         if( !error )
         {
             [[UIApplication sharedApplication] registerForRemoteNotifications];  // 需要此操作才能让应用处理推送通
知             NSLog( @"推送注册成功。" );

         }
         else
         {
NSLog( @"推送注册失败" );
             NSLog( @"错误: %@ - %@", error.localizedFailureReason, error.localizedDescription );
             NSLog( @"建议: %@ - %@", error.localizedRecoveryOptions,
error.localizedRecoverySuggestion );
         }
     }];
}


// 检查应用是否由推送通知启动
//-----------------------------------------------------
    // 处理推送通知
    if (launchOptions != nil)
    {
        // 这里应用将由推送通知启动
        // 远程通知
NSDictionary* dictionary1 = [launchOptions
objectForKey:UIApplicationLaunchOptionsRemoteNotificationKey];
        // 本地通知
NSDictionary* dictionary2 = [launchOptions
objectForKey:UIApplicationLaunchOptionsLocalNotificationKey];
        if (dictionary1 != nil)
        {
            //远程通知负载
NSLog(@"从推送通知启动: %@", dictionary1);
            //在这里处理你的推送通知
        }
        if (dictionary2 != nil)
        {
NSLog(@"从dictionary2dictionary2dictionary2通知启动: %@",
```



```
        // iOS 8 Notifications
        [application registerUserNotificationSettings:[UIUserNotificationSettings
settingsForTypes:(UIUserNotificationTypeSound | UIUserNotificationTypeAlert |
UIUserNotificationTypeBadge) categories:nil]];

        [application registerForRemoteNotifications];
    }
    else
    {
        // iOS < 8 Notifications
        [application registerForRemoteNotificationTypes:
         (UIRemoteNotificationTypeBadge | UIRemoteNotificationTypeAlert |
UIRemoteNotificationTypeSound)];
    }

}
else
{
    UNUserNotificationCenter *center = [UNUserNotificationCenter currentNotificationCenter];
    center.delegate = self;
    [center requestAuthorizationWithOptions:(UNAuthorizationOptionSound |
UNAuthorizationOptionAlert | UNAuthorizationOptionBadge) completionHandler:^(BOOL granted, NSError
* _Nullable error)
     {
         if( !error )
         {
             [[UIApplication sharedApplication] registerForRemoteNotifications];  // required to
get the app to do anything at all about push notifications
             NSLog( @"Push registration success." );
         }
         else
         {
             NSLog( @"Push registration FAILED" );
             NSLog( @"ERROR: %@ - %@", error.localizedFailureReason, error.localizedDescription );
             NSLog( @"SUGGESTIONS: %@ - %@", error.localizedRecoveryOptions,
error.localizedRecoverySuggestion );
         }
     }];
}


//to check if your App lunch from Push notification
//-----------------------------------------------------
    //Handel Push notification
    if (launchOptions != nil)
    {
        // Here app will open from pushnotification
        //RemoteNotification
        NSDictionary* dictionary1 = [launchOptions
objectForKey:UIApplicationLaunchOptionsRemoteNotificationKey];
        //LocalNotification
        NSDictionary* dictionary2 = [launchOptions
objectForKey:UIApplicationLaunchOptionsLocalNotificationKey];
        if (dictionary1 != nil)
        {
            //RemoteNotification Payload
            NSLog(@"Launched from push notification: %@", dictionary1);
            //here handle your push notification
        }
        if (dictionary2 != nil)
        {
            NSLog(@"Launched from dictionary2dictionary2dictionary2 notification: %@",
```

```
dictionary2);
            double delayInSeconds = 7;
dispatch_time_t popTime = dispatch_time(DISPATCH_TIME_NOW, (int64_t)(delayInSeconds *
NSEC_PER_SEC));
dispatch_after(popTime, dispatch_get_main_queue(), ^(void){
                // [self addMessageFromRemoteNotification:dictionary2 updateUI:NO];
            });
        }

    }
    else
        {}
    //-----------------------------------------------
```

上述代码将尝试与APNs服务器通信以获取设备令牌（前提是您的iOS配置文件中已启用APNs）。

一旦与APNs服务器建立了可靠连接，服务器将为您提供设备令牌。

在添加上述代码后，向AppDelegate类中添加以下方法：

Swift

```
func application(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken
deviceToken: NSData) {
    print("DEVICE TOKEN = \(deviceToken)")
}

func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError
error: NSError) {
    print(error)
}
```

Objective-C

```
- (void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
NSString * deviceTokenString = [[[[deviceToken description]
                    stringByReplacingOccurrencesOfString: @"<" withString: @""]
                   stringByReplacingOccurrencesOfString: @">" withString: @""]
                  stringByReplacingOccurrencesOfString: @" " withString: @""];

NSLog(@"生成的设备令牌字符串是 : %@",deviceTokenString);
}

- (void)application:(UIApplication*)application
didFailToRegisterForRemoteNotificationsWithError:(NSError*)error
{
NSLog(@"获取令牌失败，错误：%@", error.description);
}
```

上述方法根据注册成功或失败的场景被调用。

成功场景调用：

Swift

```
func application(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken
deviceToken: NSData) {
    print("DEVICE TOKEN = \(deviceToken)")
```



```
dictionary2);
            double delayInSeconds = 7;
            dispatch_time_t popTime = dispatch_time(DISPATCH_TIME_NOW, (int64_t)(delayInSeconds *
NSEC_PER_SEC));
            dispatch_after(popTime, dispatch_get_main_queue(), ^(void){
                // [self addMessageFromRemoteNotification:dictionary2 updateUI:NO];
            });
        }

    }
    else
        {}
    //-----------------------------------------------
```

The above code will try to communicate with APNs server to get device token (prerequiesites are you have APNs enabled in your iOS provisioning profile).

Once it establishes reliable connection with APNs server, the server provides you a device token.

After adding the code above, add these methods to the `AppDelegate` class:

Swift

```
func application(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken
deviceToken: NSData) {
    print("DEVICE TOKEN = \(deviceToken)")
}

func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError
error: NSError) {
    print(error)
}
```

Objective-C

```
- (void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
NSString * deviceTokenString = [[[[deviceToken description]
                    stringByReplacingOccurrencesOfString: @"<" withString: @""]
                   stringByReplacingOccurrencesOfString: @">" withString: @""]
                  stringByReplacingOccurrencesOfString: @" " withString: @""];

NSLog(@"The generated device token string is : %@",deviceTokenString);
}

- (void)application:(UIApplication*)application
didFailToRegisterForRemoteNotificationsWithError:(NSError*)error
{
    NSLog(@"Failed to get token, error: %@", error.description);
}
```

The above methods are called according to registration success or failure scenario.

Success scenario calls:

Swift

```
func application(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken
deviceToken: NSData) {
    print("DEVICE TOKEN = \(deviceToken)")
```

```
}
```

在 Swift3 中：

```
@objc(userNotificationCenter:willPresentNotification:withCompletionHandler:) @available(iOS 10.0,
*)
func userNotificationCenter(_ center: UNUserNotificationCenter, willPresent notification:
UNNotification, withCompletionHandler completionHandler: @escaping
(UNNotificationPresentationOptions) -> Void)
{
    //在前台显示通知。
   print("Userinfo2 \(notification.request.content.userInfo)")
}
```

Objective-C

```
- (void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
if(application.applicationState == UIApplicationStateInactive) {
    NSLog(@"非活动状态 - 用户在应用关闭或后台时点击了通知");

    //执行一些任务
    [self handelPushNotification:userInfo];
}
else if (application.applicationState == UIApplicationStateBackground) {
    NSLog(@"应用后台 - 应用在后台时收到通知");
    [self handelPushNotification:userInfo];
}
else {
NSLog(@"应用激活 - 应用打开时收到通知");
    //显示应用内横幅
    //执行任务
}
}
```

失败场景调用：

Swift

```
func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError
error: NSError) {
    print(error)
}
```

Objective-C

```
- (void)application:(UIApplication*)application
didFailToRegisterForRemoteNotificationsWithError:(NSError*)error
```

> **注意**
>
> 如果上述方法都没有被调用，说明您的设备无法与APNs服务器建立可靠连接，可能是由于网络访问问题。

第107.2节：注册应用ID以用于推送通知

> **所需物品**


```
}
```

In Swift3:

```
@objc(userNotificationCenter:willPresentNotification:withCompletionHandler:) @available(iOS 10.0,
*)
func userNotificationCenter(_ center: UNUserNotificationCenter, willPresent notification:
UNNotification, withCompletionHandler completionHandler: @escaping
(UNNotificationPresentationOptions) -> Void)
{
    //To show notifications in foreground.
   print("Userinfo2 \(notification.request.content.userInfo)")
}
```

Objective-C

```
- (void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
if(application.applicationState == UIApplicationStateInactive) {
    NSLog(@"Inactive - the user has tapped in the notification when app was closed or in
background");
    //do some tasks
    [self handelPushNotification:userInfo];
}
else if (application.applicationState == UIApplicationStateBackground) {
    NSLog(@"application Background - notification has arrived when app was in background");
    [self handelPushNotification:userInfo];
}
else {
    NSLog(@"application Active - notication has arrived while app was opened");
    //Show an in-app banner
    //do tasks
}
}
```

Failure scenario calls:

Swift

```
func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError
error: NSError) {
    print(error)
}
```

Objective-C

```
- (void)application:(UIApplication*)application
didFailToRegisterForRemoteNotificationsWithError:(NSError*)error
```

> **Note**
>
> If none of the above methods are getting called, your device is not able to create reliable connection with APNs server, which might be because of internet access problems.

Section 107.2: Registering App ID for use with Push Notifications

> **Things you need**

- 已付费的苹果开发者计划会员资格一个有效的应
- 用程序ID和标识符（如 com.example.MyApp），且之前未被使用过

- 访问developer.apple.com和会员中心
- 一台用于测试的iOS设备（因为推送通知在模拟器上无法使用）

在苹果开发者中心为应用程序ID启用APNs访问权限

1- 登录developer.apple.com会员中心（首页上的账户链接）

2- 进入“证书”

3- 从左侧面板选择“应用程序ID”

4- 点击右上角的“+”

5- 添加带有推送通知选项勾选的应用程序ID

6- 点击已创建的应用程序ID并选择编辑

7- 在推送通知面板点击配置

8- 在你的Mac上打开钥匙串访问应用

9- 在钥匙串访问菜单中，点击证书助理 -> 从证书颁发机构请求证书

10- 在第一个文本框中输入你的邮箱

11- 在第二个文本框中输入你的姓名

- A paid Apple Developer Program Membership
- A valid App ID and identifier for you app (like com.example.MyApp) which is not used before anywhere
- Access to developer.apple.com and Member Center
- An iOS Device to test (as Push Notifications don't work on Simulator)

Enabling the APNs access for App ID in Apple Developer Center

1- Log in to developer.apple.com Member Center (the Account link on the home page)

2- Go to "Certificates"

3- Select "App ID" from left panel

4- Click on "+" on top right

5- Add App ID with Push Notifications option checked

6- Click on created App ID and select Edit

7- Click Configure in Push Notifications panel

8- Open Keychain Access app in your Mac

9- From Keychain Access menu, click Certificate Assistant -> Request a Certificate from a Certificate Authority

10- Enter your mail in the first text field

11- Enter your name in the second text field

12- CA邮箱地址留空

13- 选择保存到磁盘，而不是通过邮件发送给CA

14- 点击继续并上传生成的文件

15- 下载Apple生成的文件并在钥匙串访问打开时打开它

在Xcode中启用APNs访问

1- 选择你的项目

2- 打开功能（Capabilities）标签

3- 找到推送通知并开启

4- 找到后台模式并开启，勾选远程通知

第107.3节：测试推送通知

即使在服务器端尚未准备好之前，测试推送通知的工作原理始终是一个好习惯，以确保你这边的设置正确。使用以下PHP脚本给自己发送推送通知是相当简单的。

1. 将脚本保存为文件（例如 send_push.php），并与您的证书（开发或生产环境）放在同一文件夹中
2. 编辑它以填写您的设备令牌和证书密码3. 选择用于建立连接的正确路径，dev_path 或 prod_path（这是脚本中“打开与 APNS 服务器的连接”发生的地方）
4. 在终端中切换到该文件夹并运行命令 'php send_push'

12- Leave CA Email Address empty

13- Select Saved to disk rather than Emailed to the CA

14- Click Continue and upload the generated file

15- Download the generated file by Apple and open it while Keychain Access is open

Enabling the APNs access in Xcode

1- Select your project

2- Open Capabilities tab

3- Find Push Notifications and turn it on

4-Find Background Modes and turn it on and check Remote Notifications

Section 107.3: Testing push notifications

It is always a good practice to test how push notifications work even before you have your server side ready for them, just to make sure that everything is set up correctly on your side. It is quite easy to send yourself a push notification using a following PHP script.

1. Save the script as a file (send_push.php for example) in the same folder as your certificate (development or production)
2. Edit it to put your device token, password from the certificate
3. Choose the correct path for opening a connection, dev_path or prod_path (this is where 'Open a connection to the APNS server' happens in the script)
4. cd to the folder in Terminal and run command 'php send_push'

5. 在您的设备上接收通知

```php
<?php

// 在此处填写您的设备令牌（无空格）：
$deviceToken = '20128697f872d7d39e48c4a61f50cb11d77789b39e6fc6b4cd7ec80582ed5229';
// 将您的最终pem证书名称放在这里。它应该与此脚本位于同一文件夹中
$cert_name = 'final_cert.pem';
// 在此处填写您的私钥密码短语：
$passphrase = '1234';

// 示例点
$alert = 'Hello world!';
$event = 'new_incoming_message';

// 您可以根据所使用的证书类型选择任一路径
$dev_path = 'ssl://gateway.sandbox.push.apple.com:2195';
$prod_path = 'ssl://gateway.push.apple.com:2195';

////////////////////////////////////////////////////////////////////////////////

$ctx = stream_context_create();
stream_context_set_option($ctx, 'ssl', 'local_cert', $cert_name);
stream_context_set_option($ctx, 'ssl', 'passphrase', $passphrase);

// 打开与 APNS 服务器的连接
$fp = stream_socket_client(
    $dev_path, $err,
    $errstr, 60, STREAM_CLIENT_CONNECT|STREAM_CLIENT_PERSISTENT, $ctx);

if (!$fp)
    exit("连接失败: $err $errstr" . PHP_EOL);

echo '已连接到 APNS' . PHP_EOL;

// 创建负载主体
// 应尽可能简短
// 如果通知未能送达，很可能是
// 生成的消息太长
$body['aps'] = array(
                        'alert' => $alert,
                        'sound' => 'default',
                        'event' => $event
                        );

// 将负载编码为 JSON
$payload = json_encode($body);

// 构建二进制通知
            $msg = chr(0) . pack('n', 32) . pack('H*', $deviceToken) . pack('n', strlen($payload)) . $payload;

// 发送到服务器
$result = fwrite($fp, $msg, strlen($msg));

if (!$result)
    echo '消息未发送' . PHP_EOL;
else
    echo '消息发送成功' . PHP_EOL;

// 关闭与服务器的连接
fclose($fp);
```


5. Receive the notification on your device

```php
<?php

// Put your device token here (without spaces):
$deviceToken = '20128697f872d7d39e48c4a61f50cb11d77789b39e6fc6b4cd7ec80582ed5229';
// Put your final pem cert name here. it is supposed to be in the same folder as this script
$cert_name = 'final_cert.pem';
// Put your private key's passphrase here:
$passphrase = '1234';

// sample point
$alert = 'Hello world!';
$event = 'new_incoming_message';

// You can choose either of the paths, depending on what kind of certificate you are using
$dev_path = 'ssl://gateway.sandbox.push.apple.com:2195';
$prod_path = 'ssl://gateway.push.apple.com:2195';

////////////////////////////////////////////////////////////////////////////////

$ctx = stream_context_create();
stream_context_set_option($ctx, 'ssl', 'local_cert', $cert_name);
stream_context_set_option($ctx, 'ssl', 'passphrase', $passphrase);

// Open a connection to the APNS server
$fp = stream_socket_client(
    $dev_path, $err,
    $errstr, 60, STREAM_CLIENT_CONNECT|STREAM_CLIENT_PERSISTENT, $ctx);

if (!$fp)
    exit("Failed to connect: $err $errstr" . PHP_EOL);

echo 'Connected to APNS' . PHP_EOL;

// Create the payload body
// it should be as short as possible
// if the notification doesn't get delivered that is most likely
// because the generated message is too long
$body['aps'] = array(
                        'alert' => $alert,
                        'sound' => 'default',
                        'event' => $event
                        );

// Encode the payload as JSON
$payload = json_encode($body);

// Build the binary notification
$msg = chr(0) . pack('n', 32) . pack('H*', $deviceToken) . pack('n', strlen($payload)) . $payload;

// Send it to the server
$result = fwrite($fp, $msg, strlen($msg));

if (!$result)
    echo 'Message not delivered' . PHP_EOL;
else
    echo 'Message successfully delivered' . PHP_EOL;

// Close the connection to the server
fclose($fp);
```

第107.4节：检查您的应用是否已注册推送通知

Swift

```
let isPushEnabled = UIApplication.sharedApplication().isRegisteredForRemoteNotifications()
```

第107.5节：从您的.cer文件生成.pem证书，以传递给服务器开发人员

1. 将aps.cer保存到一个文件夹中
2. 打开“钥匙串访问”，将该证书下的密钥导出为.p12文件（命名为key.p12）。操作方法是右键点击该密钥，选择导出。保存到与步骤1相同的文件夹。导出时会提示输入密码，随便设置一个并记住它。

3. 在终端中切换到该文件夹并执行以下命令：
4. 将.cer转换为.pem证书

```
openssl x509 -in aps.cer -inform der -out aps.pem
```

5. 将您的密钥转换为.pem格式。打开密钥时，输入步骤2中导出时设置的密码。然后，输入另一个密码以保护导出的文件。系统会提示您输入两次以确认。

```
openssl pkcs12 -nocerts -out key.pem -in key.p12
```

6. 将文件合并为一个最终文件

```
cat key.pem aps.pem > final_cert.pem
```

7. final_cert.pem 是最终结果。将其连同步骤5中的密码一起交给服务器开发人员，这样他们才能使用受保护的证书。

第107.6节：取消注册推送通知

要以编程方式取消注册远程通知，可以使用

Objective-C

```
[[UIApplication sharedApplication] unregisterForRemoteNotifications];
```

Swift

Section 107.4: Checking if your app is already registered for Push Notification

Swift

```
let isPushEnabled = UIApplication.sharedApplication().isRegisteredForRemoteNotifications()
```

Section 107.5: Generating a .pem certificate from your .cer file, to pass on to the server developer

1. Save aps.cer to a folder
2. Open "Keychain access" and export the key that is under that certificate to a .p12 file (call it key.p12). To do that right click on it and choose Export. Save it to the same folder as step 1. On export you will be prompted for a password. Make something up and memorize it.

3. cd to that folder in Terminal and execute the following commands:
4. Convert .cer to a .pem certificate

```
openssl x509 -in aps.cer -inform der -out aps.pem
```

5. Convert your key to .pem format. To open the key, enter the password you exported it with from the keychain, in step 2. Then, enter another password that will protect the exported file. You will be prompted to enter it twice for confirmation.

```
openssl pkcs12 -nocerts -out key.pem -in key.p12
```

6. Merge the files into one final file

```
cat key.pem aps.pem > final_cert.pem
```

7. The final_cert.pem is the final result. Pass it on to server developers with the password from step 5, so that they will be able to use the protected certificate.

Section 107.6: Unregistering From Push Notifications

To unregister from Remote Notifications programmatically you can use

Objective-C

```
[[UIApplication sharedApplication] unregisterForRemoteNotifications];
```

Swift

```
UIApplication.sharedApplication().unregisterForRemoteNotifications()
```

这类似于进入手机设置，手动关闭该应用的通知。

注意：在某些罕见情况下，您可能需要这样做（例如：当您的应用不再支持推送通知时）

如果您只是想允许用户暂时禁用通知，您应该实现一个方法，在服务器的数据库中移除设备令牌。否则，如果您仅在设备本地禁用通知，服务器仍会发送消息。

第107.7节：设置应用图标徽章数字

使用以下代码片段从应用程序内部设置徽章数字（假设 someNumber 已经声明）：

Objective-C

```
[UIApplication sharedApplication].applicationIconBadgeNumber = someNumber;
```

Swift

```
UIApplication.shared.applicationIconBadgeNumber = someNumber
```

要完全移除徽章，只需将 someNumber = 0 设置即可。

第107.8节：注册（非交互式）推送通知

建议将注册推送通知的逻辑添加到AppDelegate.swift中，因为回调函数（成功、失败）会在那里被调用。注册只需执行以下操作：

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

然后会调用回调函数didRegisterUserNotificationSettings，在这种情况下，你只需这样触发注册：

```
func application(application: UIApplication, didRegisterUserNotificationSettings
notificationSettings: UIUserNotificationSettings) {
application.registerForRemoteNotifications()
}
```

在这种情况下，系统会弹出提示，询问是否允许接收推送通知。以下回调函数之一将被调用：

```
func application(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken
deviceToken: NSData) {
    let tokenChars = UnsafePointer<CChar>(deviceToken.bytes)
    var tokenString = ""

    for i in 0..<deviceToken.length {
tokenString += String(format: "%02.2hhx", arguments: [tokenChars[i]])
    }
```



```
UIApplication.sharedApplication().unregisterForRemoteNotifications()
```

this is similar to going into the setting of your phone and manually switching off Notifications for the application.

NOTE: There may be rare cases where you would need this(eg: when your app no longer supports push notifications)

If you just want to allow the user to temporarily disable Notifications. You should implement a method to remove device token in the database on your server. else if you only disable Notification locally on your device your server will still send messages.

Section 107.7: Setting the application icon badge number

Use the following piece of code to set the badge number from within your application (suppose `someNumber` has been declared before):

Objective-C

```
[UIApplication sharedApplication].applicationIconBadgeNumber = someNumber;
```

Swift

```
UIApplication.shared.applicationIconBadgeNumber = someNumber
```

In order to *remove* the badge completely, just set `someNumber = 0`.

Section 107.8: Registering for (Non Interactive) Push Notification

The logic of registering for push notification is recommended to be added in `AppDelegate.swift` as the callback functions (success, failure) will be called their. To register just do the following:

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

Then the callback function `didRegisterUserNotificationSettings` will be called and in that case you just trigger the register like this:

```
func application(application: UIApplication, didRegisterUserNotificationSettings
notificationSettings: UIUserNotificationSettings) {
    application.registerForRemoteNotifications()
}
```

And in that case and system alert will be shown asking for persmission to receive push notification. One of the following callback functions will be called:

```
func application(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken
deviceToken: NSData) {
    let tokenChars = UnsafePointer<CChar>(deviceToken.bytes)
    var tokenString = ""

    for i in 0..<deviceToken.length {
        tokenString += String(format: "%02.2hhx", arguments: [tokenChars[i]])
    }
```

```
    print("推送令牌: \(tokenString)")
}

func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError
error: NSError) {
    print("注册远程通知失败，错误: \(error)")

}
```

在极少数情况下，既不会调用成功回调函数，也不会调用失败回调函数。这种情况发生在您遇到网络连接问题或APNS沙盒服务宕机时。系统会向APNS发起API调用进行验证，验证失败将导致两个回调函数都不会被调用。请访问Apple系统状态以确认服务是否正常。

第107.9节：处理推送通知

用户点击推送通知后，将调用以下回调函数。您可以解析JSON以获取后端发送的任何特定信息，这将有助于实现深度链接：

> Swift

```
func application(application: UIApplication, didReceiveRemoteNotification userInfo: [NSObject :
AnyObject]) {
    print("收到通知: \(userInfo)")
}
```

> Objective C

```
- (void)application:(UIApplication *)application didReceiveRemoteNotification: (NSDictionary
*)userInfo
{
NSLog(@"收到通知: %@", userInfo);
}
```

> iOS 10

```
#define SYSTEM_VERSION_GREATER_THAN_OR_EQUAL_TO(v)  ([[[UIDevice currentDevice] systemVersion]
compare:v options:NSNumericSearch] != NSOrderedAscending)

-(void) application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary
*)userInfo fetchCompletionHandler:(void
 (^)(UIBackgroundFetchResult))completionHandler
{
    // iOS 10 将通过其他方法处理通知
    NSLog(@"收到通知: %@", userInfo);


    if( SYSTEM_VERSION_GREATER_THAN_OR_EQUAL_TO( @"10.0" ) )
    {
NSLog( @"iOS 版本 >= 10。让 NotificationCenter 处理这个。" );
        // 设置成员变量以告知新代理这是后台
        return;
    }
```



```
    print("Push token: \(tokenString)")
}

func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError
error: NSError) {
    print("didFailToRegisterForRemoteNotificationsWithError: \(error)")

}
```

In very rare cases, neither success or failure callback functions are called. This happens when you have internet connection problems or the APNS Sandbox is down. The system do an API call to APNS to do some verification, failing to do so will lead to none of the two callbacks functions will be called. Visit Apple system status to make sure its fine.

Section 107.9: Handling Push Notification

Once user clicks on a push notification, the following callback function will be called. You can parse the JSON to get any specific info sent from backend that will helps you in deep linking:

> Swift

```
func application(application: UIApplication, didReceiveRemoteNotification userInfo: [NSObject :
AnyObject]) {
    print("Received notification: \(userInfo)")
}
```

> Objective C

```
- (void)application:(UIApplication *)application didReceiveRemoteNotification: (NSDictionary
*)userInfo
{
    NSLog(@"Received notification: %@", userInfo);
}
```

> iOS 10

```
#define SYSTEM_VERSION_GREATER_THAN_OR_EQUAL_TO(v)  ([[[UIDevice currentDevice] systemVersion]
compare:v options:NSNumericSearch] != NSOrderedAscending)

-(void) application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary
*)userInfo fetchCompletionHandler:(void
 (^)(UIBackgroundFetchResult))completionHandler
{
    // iOS 10 will handle notifications through other methods
    NSLog(@"Received notification: %@", userInfo);


    if( SYSTEM_VERSION_GREATER_THAN_OR_EQUAL_TO( @"10.0" ) )
    {
        NSLog( @"iOS version >= 10. Let NotificationCenter handle this one." );
        // set a member variable to tell the new delegate that this is background
        return;
    }
```

```
NSLog( @"处理推送，didReceiveRemoteNotification: %@", userInfo );

    // 自定义代码处理通知内容

    if( [UIApplication sharedApplication].applicationState == UIApplicationStateInactive )
   {
NSLog( @"非活动状态" );
completionHandler( UIBackgroundFetchResultNewData );
   }
   else if( [UIApplication sharedApplication].applicationState == UIApplicationStateBackground )
   {
NSLog( @"后台" );
      completionHandler( UIBackgroundFetchResultNewData );
   }
   else
   {
NSLog( @"FOREGROUND" );
      completionHandler( UIBackgroundFetchResultNewData );
   }
}

- (void)userNotificationCenter:(UNUserNotificationCenter *)center
   willPresentNotification:(UNNotification *)notification
withCompletionHandler:(void (^)(UNNotificationPresentationOptions options))completionHandler
{
NSLog( @"处理前台推送" );
   // 自定义代码处理应用处于前台时的推送
    NSLog(@"%@", notification.request.content.userInfo);
}

- (void)userNotificationCenter:(UNUserNotificationCenter *)center
didReceiveNotificationResponse:(UNNotificationResponse *)response
     withCompletionHandler:(void (^)())completionHandler
{

NSLog( @"处理后台或关闭状态下的推送" );// 如果你在 didReceiveRe
    moteNotification 中设置了成员变量，你将知道这是来自关闭还是后台

NSLog(@"%@", response.notification.request.content.userInfo);
}
```



```
    NSLog( @"HANDLE PUSH, didReceiveRemoteNotification: %@", userInfo );

    // custom code to handle notification content

    if( [UIApplication sharedApplication].applicationState == UIApplicationStateInactive )
   {
      NSLog( @"INACTIVE" );
      completionHandler( UIBackgroundFetchResultNewData );
   }
   else if( [UIApplication sharedApplication].applicationState == UIApplicationStateBackground )
   {
      NSLog( @"BACKGROUND" );
      completionHandler( UIBackgroundFetchResultNewData );
   }
   else
   {
      NSLog( @"FOREGROUND" );
      completionHandler( UIBackgroundFetchResultNewData );
   }
}

- (void)userNotificationCenter:(UNUserNotificationCenter *)center
   willPresentNotification:(UNNotification *)notification
     withCompletionHandler:(void (^)(UNNotificationPresentationOptions options))completionHandler
{
    NSLog( @"Handle push from foreground" );
   // custom code to handle push while app is in the foreground
    NSLog(@"%@", notification.request.content.userInfo);
}

- (void)userNotificationCenter:(UNUserNotificationCenter *)center
didReceiveNotificationResponse:(UNNotificationResponse *)response
     withCompletionHandler:(void (^)())completionHandler
{

    NSLog( @"Handle push from background or closed" );
    // if you set a member variable in didReceiveRemoteNotification, you  will know if this is from
closed or background
    NSLog(@"%@", response.notification.request.content.userInfo);
}
```

第108章：丰富推送的扩展通知 - iOS 10。

iOS 10 为我们带来了UserNotifications.framework，这是本地/远程通知的新API。它支持直接从通知中查看媒体附件或回复消息。

通知内容包括：标题、副标题、正文和附件。附件可以包含最大50MB的图片/GIF/视频。

第108.1节：通知内容扩展

我们为什么需要它？

内容扩展帮助我们在通知展开时创建自定义用户界面。

您可以使用此框架定义一个扩展，该扩展接收通知数据并提供相应的视觉呈现。您的扩展还可以响应与这些通知相关的自定义操作。

第108.2节：实现

1. 在xCode导航器窗口中，进入目标部分。点击添加新目标。
2. 选择通知内容扩展模板：

3. 在您的info.plist文件中，为UNNotificationExtensionCategory键设置标识符：

Chapter 108: Extension for rich Push Notification - iOS 10.

iOS 10 gave us `UserNotifications.framework`, the new API for local/remote notifications. It offers viewing media attachments or responding to messages right from the notification.

Notification content consists of: title, subtitle, body and attachment. Attachment can contain images/gifs/videos up to 50 mb.

Section 108.1: Notification Content Extension

Why do we need it?

Content extension helps us to create custom user interface upon notification expanasion.

You use this framework to define an extension that receives the notification data and provides the corresponding visual representation. Your extension can also respond to custom actions associated with those notifications.

Section 108.2: Implementation

1. In xCode `Navigator` window go to `Targets` section. Press `Add New Target`.
2. Select `Notification Content Extension` template:

3. In your `info.plist` file set the identifier for `UNNotificationExtensionCategory` key:

▼ NSExtension	Dictionary	(3 items)
▼ NSExtensionAttributes	Dictionary	(3 items)
UNNotificationExtensionDefaultContentHidden	Boolean	YES
UNNotificationExtensionCategory	String	customContentIdentifier
UNNotificationExtensionInitialContentSizeRatio	Number	0.7
NSExtensionMainStoryboard	String	MainInterface
NSExtensionPointIdentifier	String	com.apple.usernotifications.content-extension

NSExtensionAttributes：

UNNotificationExtensionCategory（必填）

该键的值是一个字符串或字符串数组。每个字符串包含由应用使用 UNNotification Category 类声明的类别标识符。

`UNNotificationExtensionInitialContentSizeRatio`（必填）

表示视图控制器视图初始大小的数字，按其高度与宽度的比例表示。

UNNotificationExtensionDefaultContentHidden（可选）设置为 YE

S 时，系统仅在通知界面显示您的自定义视图控制器。设置为 NO 时，系统会在显示您的视图控制器内容的同时显示默认通知内容。

UNNotificationExtensionOverridesDefaultTitle（可选）该键的值为布

尔值。设置为 true 时，系统使用您的视图控制器的 title 属性作为通知标题。设置为 false 时，系统将通知标题设置为您的应用名称。如果未指定此键，默认值为 false。

4. 在 NotificationViewController.swift 文件中创建自定义视图
5. 添加新的category key，并将其值设置为我们在 Info.plist（步骤3）中输入的内容：

推送：

```
{
 aps: {
alert: { ... },
类别: 'io.swifting.notification-类别'
 }
}
```

本地:

```
let mutableNotificationContent = UNMutableNotificationContent()
mutableNotificationContent.category = "io.swifting.notification-category"
mutableNotificationContent.title = "Swifting.io 通知"
mutableNotificationContent.subtitle = "Swifting.io 呈现"
mutableNotificationContent.body = "自定义通知"
```

另请查看官方 API 参考：
https://developer.apple.com/reference/usernotificationsui/unnotificationcontentextension?utm_source=swifting.io&utm_medium=web&utm_campaign=blog%20post

▼ NSExtension	Dictionary	(3 items)
▼ NSExtensionAttributes	Dictionary	(3 items)
UNNotificationExtensionDefaultContentHidden	Boolean	YES
UNNotificationExtensionCategory	String	customContentIdentifier
UNNotificationExtensionInitialContentSizeRatio	Number	0.7
NSExtensionMainStoryboard	String	MainInterface
NSExtensionPointIdentifier	String	com.apple.usernotifications.content-extension

NSExtensionAttributes:

`UNNotificationExtensionCategory` (Required)

The value of this key is a string or an array of strings. Each string contains the identifier of a category declared by the app using the UNNotificationCategory class.

`UNNotificationExtensionInitialContentSizeRatio` (Required)

Number that represents the initial size of your view controller's view expressed as a ratio of its height to its width.

`UNNotificationExtensionDefaultContentHidden` (Optional)

When set to YES, the system displays only your custom view controller in the notification interface. When set to NO, the system displays the default notification content in addition to your view controller's content.

`UNNotificationExtensionOverridesDefaultTitle` (Optional)

The value of this key is a Boolean. When set to true, the system uses the title property of your view controller as the title of the notification. When set to false, the system sets the notification's title to the name of your app. If you do not specify this key, the default value is set to false.

4. Create custom view in `NotificationViewController.swift` file
5. Add new `category key` and set its value to what we typed in the Info.plist (step 3):

Push:

```
{
 aps: {
 alert: { ... },
 category: 'io.swifting.notification-category'
 }
}
```

Local:

```
let mutableNotificationContent = UNMutableNotificationContent()
mutableNotificationContent.category = "io.swifting.notification-category"
mutableNotificationContent.title = "Swifting.io Notifications"
mutableNotificationContent.subtitle = "Swifting.io presents"
mutableNotificationContent.body = "Custom notifications"
```

Also check out the official API reference:
https://developer.apple.com/reference/usernotificationsui/unnotificationcontentextension?utm_source=swifting.io&utm_medium=web&utm_campaign=blog%20post

第109章：丰富通知

丰富通知让您可以自定义本地和远程通知在用户设备上出现时的外观。丰富通知主要包括 UNNotificationServiceExtension 和 UNNotificationContentExtension，即以扩展的方式显示普通通知

第109.1节：创建一个简单的 UNNotificationContentExtension

步骤 1

使环境适合通知。确保已启用**后台模式**和**推送通知**

步骤 2：创建 UNNotificationContentExtension

点击底部的+图标，创建一个目标模板并选择通知内容扩展 -> 下一步 -> 为内容扩展创建一个名称 -> 完成

Chapter 109: Rich Notifications

The Rich Notifications lets you customize the appearance of local and remote notifications when they appear on the user's device.Rich notification mostly includes UNNotificationServiceExtension and UNNotificationContentExtension ie displaying the normal notification in an extended manner

Section 109.1: Creating a simple UNNotificationContentExtension

Step 1

Making the environment suitable for Notification. Make sure you enabled **Background Modes** and **Push Notification**

Step 2: Creating an UNNotificationContentExtension

Click on the **+** icon in the bottom which creates a target template and select Notification Content Extention -> next -> create a name for the content extension -> finish

步骤 3：配置已创建扩展的 info.plist 文件

NSExtension 中的字典表示通知内容的显示方式，这些操作在长按收到的通知时执行

- UNNotificationExtensionOverridesDefaultTitle：我们可以为通知自定义标题，默认显示应用程序自身的名称。title = myTitle
- UNNotificationDefaultContentHidden：此布尔值决定是否隐藏通知的默认正文
- UNNotificationCategory：类别在应用程序的UNUserNotificationCenter中创建。这里可以是字符串或字符串数组，因此每个类别可以提供不同类型的数据，我们可以

Step 3:Configuring the info.plist file of the created extension

The dictionary in NSExtension signifies how the notification content is displayed, these are performed on long pressing the received notification

- UNNotificationExtensionOverridesDefaultTitle: We can give custom title for our notification by default it displays the name of the application `self.title = myTitle`
- UNNotificationDefaultContentHidden: This boolean determines whether the default body of the notification is to be hidden or not
- UNNotificationCategory: Category is created in `UNUserNotificationCenter` in your application. Here it can be either a string or an array of strings, so each category can gave different types of data from which we can

创建不同的用户界面。我们发送的负载必须包含类别名称，以便显示此特定的扩展

- UNNotificationExtensionInitialContentSizeRatio：初始内容的大小，即首次显示ContentExtension时，相对于设备宽度的初始大小。这里1表示高度将等于宽度

步骤4：在我们的应用中创建UNNotificationAction和UNNotificationCategory

在应用的AppDelegate.swift中的`didFinishLaunchingWithOptions`函数中添加

```
    let userNotificationAction:UNNotificationAction = UNNotificationAction.init(identifier: "ID1",
标题: "வணக்கம்", 选项: .destructive)
    let userNotificationAction2:UNNotificationAction = UNNotificationAction.init(identifier: "ID2",
标题: "Success", 选项: .destructive)

    let notifCategory:UNNotificationCategory = UNNotificationCategory.init(identifier: "CATID1",
动作: [userNotificationAction,userNotificationAction2], intentIdentifiers: ["ID1","ID2"] ,
选项:.customDismissAction)

UNUserNotificationCenter.current().delegate = self
    UNUserNotificationCenter.current().setNotificationCategories([notifCategory])
    UIApplication.shared.registerForRemoteNotifications()
```

我们创建了两个`UNNotificationAction`，标识符分别为`ID1`和`ID2`，并将这些动作添加到一个`UNNotificationCategory`，标识符为`CATID1`（ContentExtension的info.plist文件中的categoryID相同，我们这里创建的应在负载和plist文件中使用）。我们将该类别设置到应用的`UNUserNotificationCenter`中，下一行我们注册通知，这会调用`didRegisterForRemoteNotificationsWithDeviceToken`函数，在那里我们获取设备令牌

注意：别忘了在AppDelegate.swift中`import UserNotifications`并添加`UNUserNotificationCenterDelegate`

步骤5：NotificationContent的示例负载

```
 'aps': {
'badge': 0,
    'alert': {
'title': "丰富通知",
        'body': "丰富通知的正文",
        },
'sound' : "default",
    'category': "CATID1",
    'mutable-content':"1",
    },
'attachment': "2"
```

步骤6：配置ContentExtension

当执行通知操作时，类别对应的操作会自动显示。
让我们看看代码是如何实现的

```
import UIKit
import UserNotifications
import UserNotificationsUI

class NotificationViewController: UIViewController, UNNotificationContentExtension {

@IBOutlet var imageView: UIImageView?
```


create different UI's. The payload we send must contain the category name in order to display this particular extension

- UNNotificationExtensionInitialContentSizeRatio: The size of the initial content ie when displaying the ContentExtension for the first time the initial size with respect to width of the device. here 1 denotes the height will be equal to the width

Step 4: Creating `UNNotificationAction` and `UNNotificationCategory` in our application

In your app's AppDelegate.swift `didFinishLaunchingWithOptions` function add

```
    let userNotificationAction:UNNotificationAction = UNNotificationAction.init(identifier: "ID1",
title: "வணக்கம்", options: .destructive)
    let userNotificationAction2:UNNotificationAction = UNNotificationAction.init(identifier: "ID2",
title: "Success", options: .destructive)

    let notifCategory:UNNotificationCategory = UNNotificationCategory.init(identifier: "CATID1",
actions: [userNotificationAction,userNotificationAction2], intentIdentifiers: ["ID1","ID2"] ,
options:.customDismissAction)

    UNUserNotificationCenter.current().delegate = self
    UNUserNotificationCenter.current().setNotificationCategories([notifCategory])
    UIApplication.shared.registerForRemoteNotifications()
```

We created two `UNNotificationAction` with identifiers `ID1` and `ID2` and added those actions to a `UNNotificationCategory` with identifier `CATID1` (the categoryID in ContentExtension's info.plist file are same, what we created here should be used in payload and the plist file). We set the category to our application's `UNUserNotificationCenter` and in next line we are registering for the notification which calls the `didRegisterForRemoteNotificationsWithDeviceToken` function where we get the device token

Note: dont forget to `import UserNotifications` in your AppDelegate.swift and add `UNUserNotificationCenterDelegate`

Step 5: Sample payload for the NotificationContent

```
 'aps': {
    'badge': 0,
    'alert': {
        'title': "Rich Notification",
        'body': "Body of RICH NOTIFICATION",
        },
    'sound' : "default",
    'category': "CATID1",
    'mutable-content':"1",
    },
'attachment': "2"
```

Step 6: Configuring the ContentExtension

The corresponding actions for the category is automatically displayed while the notification action is performed. Lets us see the code how its being performed

```
import UIKit
import UserNotifications
import UserNotificationsUI

class NotificationViewController: UIViewController, UNNotificationContentExtension {

@IBOutlet var imageView: UIImageView?
```

```
override func viewDidLoad() {
    super.viewDidLoad()
}

func didReceive(_ notification: UNNotification) {
     self.title = "Koushik"
imageView?.backgroundColor = UIColor.clear
    imageView?.image = #imageLiteral(resourceName: "welcome.jpeg")
}

func didReceive(_ response: UNNotificationResponse, completionHandler completion: @escaping
(UNNotificationContentExtensionResponseOption) -> Void) {

    self.title = "Koushik"
imageView?.image = UIImage.init(named: "Success.jpeg")

    if(response.actionIdentifier == "ID1")
    {
imageView?.image = UIImage.init(named: "Success.jpeg")
    }
    else
    {
imageView?.image = UIImage.init(named: "welcome.jpeg")
    }

    }
}
```

第7步：结果

收到并长按/点击查看通知后，通知显示如下


```
override func viewDidLoad() {
    super.viewDidLoad()
}

func didReceive(_ notification: UNNotification) {
     self.title = "Koushik"
    imageView?.backgroundColor = UIColor.clear
    imageView?.image = #imageLiteral(resourceName: "welcome.jpeg")
}

func didReceive(_ response: UNNotificationResponse, completionHandler completion: @escaping
(UNNotificationContentExtensionResponseOption) -> Void) {

    self.title = "Koushik"
    imageView?.image = UIImage.init(named: "Success.jpeg")

    if(response.actionIdentifier == "ID1")
    {
       imageView?.image = UIImage.init(named: "Success.jpeg")
    }
    else
    {
        imageView?.image = UIImage.init(named: "welcome.jpeg")
    }

    }
}
```

Step 7: Result

After receiving and long press/Clicking View notification, the notification looks like this

标题是“Koushik”，因为我们设置了`self.title = "Koushik"` 并且 `UNNotificationExtensionOverrideDefaultTitle` 为YES。在第3步中我们将 `UNNotificationExtensionDefaultContentHidden` 设置为NO，如果是YES，通知将显示为图3和图4的样子。

The title is "Koushik" since we gave `self.title = "Koushik"` and `UNNotificationExtensionOverrideDefaultTitle` as YES. In step 3 we gave `UNNotificationExtensionDefaultContentHidden` as NO if its YES then the notification will look like images 3 and 4.

第110章：键值编码-键值观察

KVC :- 键值编码

通常实例变量通过属性或访问器访问，但KVC提供了另一种通过字符串访问变量的方式。这样你的类就像一个字典，属性名例如“age”变成了键，该属性持有的值变成了该键对应的值。

```
例如，你有一个员工class，带有"age"属性。通常我们这样访问。
emp.age = @“20”;
NSString age = emp.age;

但KVC的工作方式如下：
[emp valueForKey:@“age”];
[emp setValue:@“25” forKey:@“age”];
```

KVO：键值观察者

当对象的某个属性发生变化时，通知该变化的机制称为KVO。
例如：键盘通知

> 例如，person对象希望在BankAccount对象的accountBalance属性发生变化时收到通知。为此，Person对象必须通过发送addObserver: forKeyPath: options: context:消息，注册为BankAccount对象accountBalance属性的观察者。

第110.1节：观察NSObject子类的属性

大多数KVO和KVC功能默认已在所有NSObject子类中实现。

要开始观察名为firstName的属性，属于名为personObject的对象，在观察类中执行以下操作：

```
[personObject addObserver:self
                forKeyPath:@“firstName”
                     options:NSKeyValueObservingOptionNew
                   context:nil];
```

上述代码中`self`所指的对象将在被观察的键路径发生变化时接收到`observeValueForKeyPath:ofObject:change:context:`消息。

```
- (void)observeValueForKeyPath:(NSString *)keyPath
                      ofObject:(id)object
change:(NSDictionary<NSString *,id> *)change
                       context:(void *)context
{
NSLog(@"new value of %@ is: %@", keyPath, change[NSKeyValueChangeNewKey]);
}
```

“键路径”是KVC的术语。NSObject子类默认实现KVC功能。

名为_firstName的实例变量可以通过@"firstName"键路径访问。

访问@"firstName"键路径时，会调用名为firstName的getter方法，无论是否存在

Chapter 110: Key Value Coding-Key Value Observation

KVC :- Key-Value Coding

Normally instance variables are accessed through properties or accessors but KVC gives another way to access variables in form of strings. In this way your class acts like a dictionary and your property name for example “age” becomes key and value that property holds becomes value for that key.

```
For example, you have employee class with "age" property. Normally we access like this.
emp.age = @”20”;
NSString age = emp.age;

But KVC works like this:
[emp valueForKey:@"age"];
[emp setValue:@"25" forKey:@"age"];
```

KVO :- Key-Value Observer

The mechanism through which objects are notified when there is change in any of property is called KVO.
Ex.:keyboard notification

> For example, person object is interested in getting notification when accountBalance property is changed in BankAccount object. To achieve this, Person Object must register as an observer of the BankAccount’s accountBalance property by sending an addObserver: forKeyPath: options: context: message.

Section 110.1: Observing a property of a NSObject subclass

Most KVO and KVC functionality is already implemented by default on all `NSObject` subclasses.

To start observing a property named `firstName` of an object named `personObject` do this in the observing class:

```
[personObject addObserver:self
                forKeyPath:@"firstName"
                   options:NSKeyValueObservingOptionNew
                   context:nil];
```

The object that `self` in the above code refers to will then receive a `observeValueForKeyPath:ofObject:change:context:` message whenever the observed key path changes.

```
- (void)observeValueForKeyPath:(NSString *)keyPath
                      ofObject:(id)object
                        change:(NSDictionary<NSString *,id> *)change
                       context:(void *)context
{
    NSLog(@"new value of %@ is: %@", keyPath, change[NSKeyValueChangeNewKey]);
}
```

"Key path" is a KVC term. `NSObject` subclasses implement KVC functionality by default.

An instance variable named `_firstName` will be accessible by the `@"firstName"` key path.

A getter method named `firstName` will be called when accessing the `@"firstName"` key path, regardless of there

_firstName实例变量或setFirstNamesetter方法。

第110.2节：KVO观察中上下文的使用

```
-(void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:(NSDictionary<NSString
*,id> *)change context:(void *)context
```

如果你将类交给他人使用，Context（上下文）非常重要。Context 让你的类的观察者能够验证调用的是你的观察者。

不传递观察者的问题是，如果有人继承你的类并为同一个
对象、同一个键注册观察者，但他没有传递上下文，那么超类的观察者可能会被多次调用。

一个唯一且供你内部使用的变量是一个很好的上下文。

更多信息请参见。

重要性和良好的上下文

being a `_firstName` instance variable or `setFirstName` setter method.

Section 110.2: Use of context for KVO Observation

```
-(void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:(NSDictionary<NSString
*,id> *)change context:(void *)context
```

Context is important if you ship your class for others to use.Context lets your class observer verify that its you observer which is being called.

The problem with not passing an observer is, if some one subclass your class and register an observer for the same object,same key and he does not passes a context ,then the super class observer can be called multiple time.

A variable which is unique and internal for your use is a good context.

For more information.

importance and good context

第111章：初始化惯用法

第111.1节：带块的工厂方法

```
internal func Init<Type>(value : Type, block: @noescape (object: Type) -> Void) -> Type
{
block(object: value)
    return value
}
```

用法：

```
Init(UILabel(frame: CGRect.zero)) {
    $0.backgroundColor = UIColor.blackColor()
}
```

第111.2节：设置元组以避免代码重复

通过一行代码设置变量元组，避免构造函数中的代码重复：

```
class 联系人: UIView
{
    private var message: UILabel
    private var phone: UITextView

    required init?(coder aDecoder: NSCoder) {
        (message, phone) = self.dynamicType.setUp()
        super.init(coder: aDecoder)
    }

    override func awakeFromNib() {
        (message, phone) = self.dynamicType.setUp()
        super.awakeFromNib()
    }

    override init(frame: CGRect) {
        (message, phone) = self.dynamicType.setUp()
        super.init(frame: frame)
    }

    private static func setUp(){
        let message = UILabel()  // ...
        let phone = UITextView() // ...
        return (message, phone)
    }
}
```

第111.3节：使用位置常量初始化

```
let mySwitch: UISwitch = {
view.addSubview($0)
$0.addTarget(self, action: "action", forControlEvents: .TouchUpInside)
    return $0
}(UISwitch())
```


Chapter 111: Initialization idioms

Section 111.1: Factory method with block

```
internal func Init<Type>(value : Type, block: @noescape (object: Type) -> Void) -> Type
{
    block(object: value)
    return value
}
```

Usage:

```
Init(UILabel(frame: CGRect.zero)) {
    $0.backgroundColor = UIColor.blackColor()
}
```

Section 111.2: Set to tuples to avoid code repetition

Avoid code repetition in constructors by setting a tuple of variables with a one liner:

```
class Contact: UIView
{
    private var message: UILabel
    private var phone: UITextView

    required init?(coder aDecoder: NSCoder) {
        (message, phone) = self.dynamicType.setUp()
        super.init(coder: aDecoder)
    }

    override func awakeFromNib() {
        (message, phone) = self.dynamicType.setUp()
        super.awakeFromNib()
    }

    override init(frame: CGRect) {
        (message, phone) = self.dynamicType.setUp()
        super.init(frame: frame)
    }

    private static func setUp(){
        let message = UILabel()  // ...
        let phone = UITextView() // ...
        return (message, phone)
    }
}
```

Section 111.3: Initialize with positional constants

```
let mySwitch: UISwitch = {
    view.addSubview($0)
    $0.addTarget(self, action: "action", forControlEvents: .TouchUpInside)
    return $0
}(UISwitch())
```

第111.4节：在didSet中初始化属性

```
@IBOutlet weak var title: UILabel! {
  didSet {
label.textColor = UIColor.redColor()
    label.font = UIFont.systemFontOfSize(20)
    label.backgroundColor = UIColor.blueColor()
  }
}
```

也可以同时设置值并初始化：

```
private var loginButton = UIButton() {
    didSet(oldValue) {
loginButton.addTarget(self, action: #selector(LoginController.didClickLogin),
forControlEvents: .TouchUpInside)
    }
}
```

第111.5节：在自定义NSObject中分组outlet

将每个outlet移动到一个NSObject中。然后从库中拖动一个对象到故事板的控制器场景中，并在那里连接元素。

```
class ContactFormStyle: NSObject
{
    @IBOutlet private weak var message: UILabel! {
      didSet {
message.font = UIFont.systemFontOfSize(12)
        message.textColor = UIColor.blackColor()
      }
    }
}

class ContactFormVC: UIViewController
{
    @IBOutlet private var style: ContactFormStyle!
}
```

第111.6节：使用then初始化

这在语法上类似于使用位置常量初始化的示例，但需要Then扩展，来自https://github.com/devxoul/Then（附在下方）。

```
let label = UILabel().then {
    $0.textAlignment = .Center
    $0.textColor = UIColor.blackColor(
    $0.text = "Hello, World!"
}
```

Then 扩展：

```
import Foundation

public protocol Then {}

extension Then
```


Section 111.4: Initialize attributes in didSet

```
@IBOutlet weak var title: UILabel! {
  didSet {
    label.textColor = UIColor.redColor()
    label.font = UIFont.systemFontOfSize(20)
    label.backgroundColor = UIColor.blueColor()
  }
}
```

It's also possible to both set a value and initialize it:

```
private var loginButton = UIButton() {
    didSet(oldValue) {
        loginButton.addTarget(self, action: #selector(LoginController.didClickLogin),
forControlEvents: .TouchUpInside)
    }
}
```

Section 111.5: Group outlets in a custom NSObject

Move every outlet to an NSObject. Then drag an Object from the library to the controller scene of the storyboard and hook the elements there.

```
class ContactFormStyle: NSObject
{
    @IBOutlet private weak var message: UILabel! {
      didSet {
        message.font = UIFont.systemFontOfSize(12)
        message.textColor = UIColor.blackColor()
      }
    }
}

class ContactFormVC: UIViewController
{
    @IBOutlet private var style: ContactFormStyle!
}
```

Section 111.6: Initialize with then

This is similar in syntax to the example that initializes using positional constants, but requires the Then extension from https://github.com/devxoul/Then (attached below).

```
let label = UILabel().then {
    $0.textAlignment = .Center
    $0.textColor = UIColor.blackColor(
    $0.text = "Hello, World!"
}
```

The Then extension:

```
import Foundation

public protocol Then {}

extension Then
```

```
{
    public func then(@noescape block: inout Self -> Void) -> Self {
        var copy = self
block(&copy)
        return copy
    }
}

extension NSObject: Then {}
```



```
{
    public func then(@noescape block: inout Self -> Void) -> Self {
        var copy = self
        block(&copy)
        return copy
    }
}

extension NSObject: Then {}
```

第112章：Storyboard

通常，Storyboard中的视图控制器会根据Storyboard本身定义的操作自动实例化和创建。然而，你也可以使用Storyboard对象来实例化Storyboard文件中的初始视图控制器，或者实例化你想要以编程方式呈现的其他视图控制器。下面你将看到这两种用例的示例。

第112.1节：初始化

```
//Swift
let storyboard = UIStoryboard(name: "Main", bundle: NSBundle.mainBundle())

//Objective-c
      UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"Main" bundle:[NSBundle mainBundle]];
```

第112.2节：获取初始视图控制器

```
//Swift
let initialScreen = storyboard.instantiateInitialViewController()

//Objective-c
UIViewController *initailScreen = [storyboard instantiateInitialViewController];
```

第112.3节：获取视图控制器

```
//Swift
let viewController = storyboard.instantiateViewControllerWithIdentifier("identifier")

//Objective-c
UIViewController *viewController = [storyboard
instantiateViewControllerWithIdentifier:@"identifier"];
```


Chapter 112: Storyboard

Normally, view controllers in a storyboard are instantiated and created automatically in response to actions defined within the storyboard itself. However, you can use a storyboard object to instantiate the initial view controller in a storyboard file or instantiate other view controllers that you want to present programmatically. Below you will find examples of both use cases.

Section 112.1: Initialize

```
//Swift
let storyboard = UIStoryboard(name: "Main", bundle: NSBundle.mainBundle())

//Objective-c
UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"Main" bundle:[NSBundle mainBundle]];
```

Section 112.2: Fetch Initial ViewController

```
//Swift
let initialScreen = storyboard.instantiateInitialViewController()

//Objective-c
UIViewController *initailScreen = [storyboard instantiateInitialViewController];
```

Section 112.3: Fetch ViewController

```
//Swift
let viewController = storyboard.instantiateViewControllerWithIdentifier("identifier")

//Objective-c
UIViewController *viewController = [storyboard
instantiateViewControllerWithIdentifier:@"identifier"];
```

第113章：后台模式和事件

第113.1节：后台播放音频

在属性列表（.plist）文件中添加一个名为Required background modes的键。。

如下图所示。。

Key	Type	Value
Information Property List	Dictionary	(16 items)
Localization native development r...	String	en
Bundle display name	String	
Executable file	String	${EXECUTABLE_NAME}
Icon files	Array	(14 items)
Bundle identifier	String	
InfoDictionary version	String	6.0
Bundle name	String	${PRODUCT_NAME}
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.1
Bundle creator OS Type code	String	????
Bundle version	String	1.1
Application requires iPhone envir...	Boolean	YES
Required background modes	Array	(1 item)
Item 0	String	App plays audio or streams audio/video using AirPlay
Icon already includes gloss effects	Boolean	YES
Required device capabilities	Array	(1 item)
Supported interface orientations	Array	(1 item)

并在以下文件中添加代码

AppDelegate.h

```
#import <AVFoundation/AVFoundation.h>
#import <AudioToolbox/AudioToolbox.h>
```

AppDelegate.m

在 application didFinishLaunchingWithOptions 中

```
[[AVAudioSession sharedInstance] setDelegate:self];
[[AVAudioSession sharedInstance] setCategory:AVAudioSessionCategoryPlayback error:nil];
[[AVAudioSession sharedInstance] setActive:YES error:nil];
[[UIApplication sharedApplication] beginReceivingRemoteControlEvents];

UInt32 size = sizeof(CFStringRef);
CFStringRef route;
AudioSessionGetProperty(kAudioSessionProperty_AudioRoute, &size, &route);
NSLog(@"route = %@", route);
```

如果你想根据事件进行更改，需要在 AppDelegate.m 中添加以下代码

```
- (void)remoteControlReceivedWithEvent:(UIEvent *)theEvent {
```


Chapter 113: Background Modes and Events

Section 113.1: Play Audio in Background

Add a key named **Required background modes** in property list (.plist) file ..

as following picture..

Key	Type	Value
Information Property List	Dictionary	(16 items)
Localization native development r...	String	en
Bundle display name	String	
Executable file	String	${EXECUTABLE_NAME}
Icon files	Array	(14 items)
Bundle identifier	String	
InfoDictionary version	String	6.0
Bundle name	String	${PRODUCT_NAME}
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.1
Bundle creator OS Type code	String	????
Bundle version	String	1.1
Application requires iPhone envir...	Boolean	YES
Required background modes	Array	(1 item)
Item 0	String	App plays audio or streams audio/video using AirPlay
Icon already includes gloss effects	Boolean	YES
Required device capabilities	Array	(1 item)
Supported interface orientations	Array	(1 item)

And add following code in

AppDelegate.h

```
#import <AVFoundation/AVFoundation.h>
#import <AudioToolbox/AudioToolbox.h>
```

AppDelegate.m

in application didFinishLaunchingWithOptions

```
[[AVAudioSession sharedInstance] setDelegate:self];
[[AVAudioSession sharedInstance] setCategory:AVAudioSessionCategoryPlayback error:nil];
[[AVAudioSession sharedInstance] setActive:YES error:nil];
[[UIApplication sharedApplication] beginReceivingRemoteControlEvents];

UInt32 size = sizeof(CFStringRef);
CFStringRef route;
AudioSessionGetProperty(kAudioSessionProperty_AudioRoute, &size, &route);
NSLog(@"route = %@", route);
```

If you want changes as per events you have to add following code in AppDelegate.m

```
- (void)remoteControlReceivedWithEvent:(UIEvent *)theEvent {
```

```
    if (theEvent.type == UIEventTypeRemoteControl)    {
        switch(theEvent.subtype)        {
            case UIEventSubtypeRemoteControlPlay:
                [[NSNotificationCenter defaultCenter] postNotificationName:@"TogglePlayPause"
object:nil];
                break;
            case UIEventSubtypeRemoteControlPause:
                [[NSNotificationCenter defaultCenter] postNotificationName:@"TogglePlayPause"
object:nil];
                break;
            case UIEventSubtypeRemoteControlStop:
                break;
            case UIEventSubtypeRemoteControlTogglePlayPause:
                [[NSNotificationCenter defaultCenter] postNotificationName:@"TogglePlayPause"
object:nil];
                break;
            default:
                return;
        }
    }
}
```

需要根据通知进行处理。


```
    if (theEvent.type == UIEventTypeRemoteControl)    {
        switch(theEvent.subtype)        {
            case UIEventSubtypeRemoteControlPlay:
                [[NSNotificationCenter defaultCenter] postNotificationName:@"TogglePlayPause"
object:nil];
                break;
            case UIEventSubtypeRemoteControlPause:
                [[NSNotificationCenter defaultCenter] postNotificationName:@"TogglePlayPause"
object:nil];
                break;
            case UIEventSubtypeRemoteControlStop:
                break;
            case UIEventSubtypeRemoteControlTogglePlayPause:
                [[NSNotificationCenter defaultCenter] postNotificationName:@"TogglePlayPause"
object:nil];
                break;
            default:
                return;
        }
    }
}
```

Based on notification have to work on it..

第114章：Fastlane

第114.1节：fastlane工具

fastlane 是一个面向Android和iOS开发者的开源构建自动化工具。它可以减少你的构建生成时间。它是一个使用Ruby的命令行工具，因此你的电脑需要安装Ruby。大多数Mac电脑默认已经安装了Ruby。

安装fastlane

1. 打开终端。
2. 运行 `sudo gem install fastlane --verbose`
3. 如果您还没有安装 Xcode 命令行工具，请运行 xcode-select --install 来安装它们
4. 现在，cd 到你的项目文件夹（输入 cd [末尾带空格] 并将你的项目文件夹拖入终端）
5. 运行 fastlane init 来设置 fastlane。
6. 现在你可以使用所有 Fastlane 工具了：

iOS 工具

- deliver：上传截图、元数据和你的应用到 App Store
- snapshot：自动化在每个设备上为你的 iOS 应用拍摄本地化截图
- frameit：快速将截图放入正确的设备框架中
- pem：自动生成和更新推送通知配置文件
- sigh：因为你更愿意花时间开发而不是处理配置问题
- produce：使用命令行在 iTunes Connect 和开发者门户创建新的 iOS 应用
- cert：自动创建和维护 iOS 代码签名证书
- gym：构建你的 iOS 应用从未如此简单
- match：使用 Git 轻松同步团队的证书和配置文件
- scan：运行 iOS 和 Mac 应用测试的最简单方法
- spaceship：访问 Apple 开发者中心和 iTunes Connect 的 Ruby 库

iOS TestFlight 工具

- pilot：从终端管理您的TestFlight测试人员和构建的最佳方式
- boarding：邀请您的TestFlight测试版测试人员的最简单方法

Android工具

- supply：上传您的Android应用及其元数据到Google Play
- screengrab：自动在每个设备上为您的Android应用拍摄本地化截图

Chapter 114: Fastlane

Section 114.1: fastlane tools

fastlane is an open source build automation tool for Android and iOS for developers. It reduce your build generation time. It is a command line tool that uses Ruby, so you need Ruby on your computer. Most Macs already have Ruby installed by default.

Install fastlane

1. Open a terminal.
2. Run `sudo gem install fastlane --verbose`
3. If you haven't installed the Xcode command-line tools yet, run `xcode-select --install` to install them
4. Now, cd into your project folder (type `cd` [with the space at the end] and drag your project folder into the terminal)
5. Run `fastlane init` to get fastlane setup.
6. Now you can able to use all the Fastlane tools:

iOS Tools

- deliver: Upload screenshots, metadata, and your app to the App Store
- snapshot: Automate taking localized screenshots of your iOS app on every device
- frameit: Quickly put your screenshots into the right device frames
- pem: Automatically generate and renew your push notification profiles
- sigh: Because you would rather spend your time building stuff than fighting provisioning
- produce: Create new iOS apps on iTunes Connect and Dev Portal using the command line
- cert: Automatically create and maintain iOS code signing certificates
- gym: Building your iOS apps has never been easier
- match: Easily sync your certificates and profiles across your team using Git
- scan: The easiest way to run tests for your iOS and Mac apps
- spaceship: Ruby library to access the Apple Dev Center and iTunes Connect

iOS TestFlight Tools

- pilot: The best way to manage your TestFlight testers and builds from your terminal
- boarding: The easiest way to invite your TestFlight beta testers

Android Tools

- supply: Upload your Android app and its metadata to Google Play
- screengrab: Automate taking localized screenshots of your Android app on every device

第115章：CAShapeLayer

第115.1节：绘制矩形

```
CAShapeLayer *mask = [[CAShapeLayer alloc] init];

mask.frame = CGRectMake(50, 50, 100, 100);

CGFloat 宽度 = 100;

CGFloat 高度 = 100;

CGMutablePathRef path = CGPathCreateMutable();

CGPathMoveToPoint(path, nil, 30, 30);

CGPathAddLineToPoint(path, nil, 宽度, 30);

CGPathAddLineToPoint(path, nil, 宽度, 高度);

CGPathAddLineToPoint(path, nil,30, 高度);

CGPathAddLineToPoint(path, nil, 30, 30);

CGPathCloseSubpath(path);

mask.path = path;

CGPathRelease(path);

self.view.layer.mask = mask;
```

第115.2节：绘制圆形

```
CAShapeLayer *circle = [CAShapeLayer layer];

    [circle setPath:[[UIBezierPath bezierPathWithOvalInRect:CGRectMake(100, 100, 150, 150)]
CGPath]];

    [circle setStrokeColor:[[UIColor blueColor] CGColor]];
```


Chapter 115: CAShapeLayer

Section 115.1: Draw Rectangle

```
CAShapeLayer *mask = [[CAShapeLayer alloc] init];

mask.frame = CGRectMake(50, 50, 100, 100);

CGFloat width = 100;

CGFloat height = 100;

CGMutablePathRef path = CGPathCreateMutable();

CGPathMoveToPoint(path, nil, 30, 30);

CGPathAddLineToPoint(path, nil, width, 30);

CGPathAddLineToPoint(path, nil, width, height);

CGPathAddLineToPoint(path, nil,30, height);

CGPathAddLineToPoint(path, nil, 30, 30);

CGPathCloseSubpath(path);

mask.path = path;

CGPathRelease(path);

self.view.layer.mask = mask;
```

Section 115.2: Draw Circle

```
CAShapeLayer *circle = [CAShapeLayer layer];

    [circle setPath:[[UIBezierPath bezierPathWithOvalInRect:CGRectMake(100, 100, 150, 150)]
CGPath]];

    [circle setStrokeColor:[[UIColor blueColor] CGColor]];
```

```
[circle setFillColor:[[UIColor clearColor] CGColor]];

[[self.view layer] addSublayer:circle];
```

第115.3节：CAShapeLayer动画

```
CAShapeLayer *circle = [CAShapeLayer layer];

[circle setPath:[[UIBezierPath bezierPathWithOvalInRect:CGRectMake(100, 100, 150, 150)] CGPath]];

[circle setStrokeColor:[[UIColor blueColor] CGColor]];

[circle setFillColor:[[UIColor clearColor] CGColor]];

[[self.view layer] addSublayer:circle];


CABasicAnimation *pathAnimation = [CABasicAnimation animationWithKeyPath:@"strokeEnd"];

pathAnimation.duration = 1.5f;

pathAnimation.fromValue = [NSNumber numberWithFloat:0.0f];

pathAnimation.toValue = [NSNumber numberWithFloat:1.0f];

pathAnimation.repeatCount = 10;

pathAnimation.autoreverses = YES;

[circle addAnimation:pathAnimation
              forKey:@"strokeEnd"];
```

```
[circle setFillColor:[[UIColor clearColor] CGColor]];

[[self.view layer] addSublayer:circle];
```

Section 115.3: CAShapeLayer Animation


```
CAShapeLayer *circle = [CAShapeLayer layer];

[circle setPath:[[UIBezierPath bezierPathWithOvalInRect:CGRectMake(100, 100, 150, 150)] CGPath]];

[circle setStrokeColor:[[UIColor blueColor] CGColor]];

[circle setFillColor:[[UIColor clearColor] CGColor]];

[[self.view layer] addSublayer:circle];


CABasicAnimation *pathAnimation = [CABasicAnimation animationWithKeyPath:@"strokeEnd"];

pathAnimation.duration = 1.5f;

pathAnimation.fromValue = [NSNumber numberWithFloat:0.0f];

pathAnimation.toValue = [NSNumber numberWithFloat:1.0f];

pathAnimation.repeatCount = 10;

pathAnimation.autoreverses = YES;

[circle addAnimation:pathAnimation
              forKey:@"strokeEnd"];
```

第115.4节：基本CAShapeLayer操作

使用UIBezierPath创建圆形路径ShapeLayer

```
CAShapeLayer *circleLayer = [CAShapeLayer layer];
[circleLayer setPath:[[UIBezierPath bezierPathWithOvalInRect:
CGRectMake(50, 50, 100, 100)] CGPath]];
circleLayer.lineWidth = 2.0;
[circleLayer setStrokeColor:[[UIColor redColor] CGColor]];
[circleLayer setFillColor:[[UIColor clearColor] CGColor]];
circleLayer.lineJoin = kCALineJoinRound;  //4种类型可用于创建线条样式
circleLayer.lineDashPattern = [NSArray arrayWithObjects:
[NSNumber numberWithInt:2],[NSNumber numberWithInt:3 ], nil];
// self.origImage 是父视图
[[self.view layer] addSublayer:circleLayer];
self.currentShapeLayer = circleLayer;  // 用于保存该形状图层引用的公共值

self.view.layer.borderWidth = 1.0f;
                    self.view.layer.borderColor = [[UIColor blueColor]CGColor];  // 这将在主视图中绘制
```

移除 ShapeLayer

保持对该形状图层的引用。例如，你可能有一个属性 currentShapeLayer：现在你有了引用，就可以轻松移除该图层：

类型 1：

```
[self.currentShapeLayer removeFromSuperlayer];
```

类型 2：

```
self.view.layer.sublayers = nil ;              // 移除所有之前的形状
```

其他操作

```
//绘制正方形形状

CAShapeLayer *squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(20, 20, 100, 100);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = nil;
squareLayer.strokeColor = [[UIColor redColor] CGColor];
squareLayer.path = [UIBezierPath bezierPathWithRect:squareLayer.bounds].CGPath;
[[self.view layer] addSublayer:squareLayer];


//绘制圆形形状

CAShapeLayer *circleShape = [CAShapeLayer layer];
circleShape.frame = CGRectMake(160, 20, 120, 120);
circleShape.lineWidth = 2.0;
circleShape.fillColor = nil;
circleShape.strokeColor = [[UIColor redColor] CGColor];
circleShape.path = [UIBezierPath bezierPathWithOvalInRect:circleShape.bounds].CGPath;
[[self.view layer] addSublayer:circleShape];


//子路径
                    //UIBezierPath 可以有任意数量的“路径段”（或子路径），因此你可以有效地绘制为
```


Section 115.4: Basic CAShapeLayer Operation

UIBezierPath using to create a circular path ShapeLayer

```
CAShapeLayer *circleLayer = [CAShapeLayer layer];
[circleLayer setPath:[[UIBezierPath bezierPathWithOvalInRect:
CGRectMake(50, 50, 100, 100)] CGPath]];
circleLayer.lineWidth = 2.0;
[circleLayer setStrokeColor:[[UIColor redColor] CGColor]];
[circleLayer setFillColor:[[UIColor clearColor] CGColor]];
circleLayer.lineJoin = kCALineJoinRound;  //4 types are available to create a line style
circleLayer.lineDashPattern = [NSArray arrayWithObjects:
[NSNumber numberWithInt:2],[NSNumber numberWithInt:3 ], nil];
// self.origImage is parentView
[[self.view layer] addSublayer:circleLayer];
self.currentShapeLayer = circleLayer;  // public value using to keep that reference of the shape
Layer
self.view.layer.borderWidth = 1.0f;
self.view.layer.borderColor = [[UIColor blueColor]CGColor];  // that will plotted in the mainview
```

Remove ShapeLayer

Keep a reference to that shape layer. For example, you might have a property currentShapeLayer: Now that you have a reference, you can easily remove the layer:

Type 1:

```
[self.currentShapeLayer removeFromSuperlayer];
```

Type 2:

```
self.view.layer.sublayers = nil ;                //removed all earlier shapes
```

Other Operation

```
//Draw Square Shape

CAShapeLayer *squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(20, 20, 100, 100);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = nil;
squareLayer.strokeColor = [[UIColor redColor] CGColor];
squareLayer.path = [UIBezierPath bezierPathWithRect:squareLayer.bounds].CGPath;
[[self.view layer] addSublayer:squareLayer];


//Draw Circle Shape

CAShapeLayer *circleShape = [CAShapeLayer layer];
circleShape.frame = CGRectMake(160, 20, 120, 120);
circleShape.lineWidth = 2.0;
circleShape.fillColor = nil;
circleShape.strokeColor = [[UIColor redColor] CGColor];
circleShape.path = [UIBezierPath bezierPathWithOvalInRect:circleShape.bounds].CGPath;
[[self.view layer] addSublayer:circleShape];


//Subpaths
//UIBezierPath can have any number of “path segments” (or subpaths) so you can effectively draw as
```

```
在单个路径对象中绘制任意多的形状或线条

CAShapeLayer *shapeLayer = [CAShapeLayer layer];
shapeLayer.frame = CGRectMake(20, 140, 200, 200);
shapeLayer.lineWidth = 2.0;
shapeLayer.fillColor = nil;
shapeLayer.strokeColor = [[UIColor redColor] CGColor];

CGMutablePathRef combinedPath= CGPathCreateMutableCopy(circleShape.path);
CGPathAddPath(combinedPath, NULL, squareLayer.path);

shapeLayer.path = combinedPath;
[[self.view layer] addSublayer:shapeLayer];

//开放路径
// 路径不需要将终点连接回起点。连接回起点的路径称为闭合路径，不连接回起点的称为开放路径。



shapeLayer = [CAShapeLayer layer];
shapeLayer.frame = CGRectMake(160, 140, 300, 300);
shapeLayer.lineWidth = 2.0;
shapeLayer.fillColor = nil;
shapeLayer.strokeColor = [[UIColor redColor] CGColor];

UIBezierPath *linePath=[UIBezierPath bezierPath];
[linePath moveToPoint:CGPointZero];
[linePath addLineToPoint:CGPointMake(0 , 120)];
[linePath addLineToPoint:CGPointMake(120 , 120)];
[linePath addLineToPoint:CGPointMake(120 , 0)];
shapeLayer.path = linePath.CGPath;
[[self.view layer] addSublayer:shapeLayer];
```

填充概念 //填充颜色

```
CAShapeLayer *squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(20, 30, 100, 100);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = [[UIColor yellowColor]CGColor];
squareLayer.strokeColor = [[UIColor redColor] CGColor];
squareLayer.path = [UIBezierPath bezierPathWithRect:squareLayer.bounds].CGPath;
[[self.view layer] addSublayer:squareLayer];
```



```
many shapes or lines as you want in a single path object

CAShapeLayer *shapeLayer = [CAShapeLayer layer];
shapeLayer.frame = CGRectMake(20, 140, 200, 200);
shapeLayer.lineWidth = 2.0;
shapeLayer.fillColor = nil;
shapeLayer.strokeColor = [[UIColor redColor] CGColor];

CGMutablePathRef combinedPath= CGPathCreateMutableCopy(circleShape.path);
CGPathAddPath(combinedPath, NULL, squareLayer.path);

shapeLayer.path = combinedPath;
[[self.view layer] addSublayer:shapeLayer];

//Open Path
// Paths do not need to connect their end points back to their starting points. A path that
connects back to its starting point is called a closed path, and one that does not is called an
open path.

shapeLayer = [CAShapeLayer layer];
shapeLayer.frame = CGRectMake(160, 140, 300, 300);
shapeLayer.lineWidth = 2.0;
shapeLayer.fillColor = nil;
shapeLayer.strokeColor = [[UIColor redColor] CGColor];

UIBezierPath *linePath=[UIBezierPath bezierPath];
[linePath moveToPoint:CGPointZero];
[linePath addLineToPoint:CGPointMake(0 , 120)];
[linePath addLineToPoint:CGPointMake(120 , 120)];
[linePath addLineToPoint:CGPointMake(120 , 0)];
shapeLayer.path = linePath.CGPath;
[[self.view layer] addSublayer:shapeLayer];
```

Fill Concepts //Fill Color

```
CAShapeLayer *squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(20, 30, 100, 100);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = [[UIColor yellowColor]CGColor];
squareLayer.strokeColor = [[UIColor redColor] CGColor];
squareLayer.path = [UIBezierPath bezierPathWithRect:squareLayer.bounds].CGPath;
[[self.view layer] addSublayer:squareLayer];
```

```
//填充图案颜色
//images.jpeg

squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(140, 30, 100, 100);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = [[UIColor colorWithPatternImage:[UIImage
imageNamed:@"images.jpeg"]]CGColor];
squareLayer.strokeColor = [[UIColor redColor] CGColor];
squareLayer.path = [UIBezierPath bezierPathWithRect:squareLayer.bounds].CGPath;
[[self.view layer] addSublayer:squareLayer];


//填充规则

//类型 1: kCAFillRuleNonZero
squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(0, 140, 150, 150);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = [[UIColor yellowColor]CGColor];
squareLayer.fillRule = kCAFillRuleNonZero;   // 指定规则类型
squareLayer.strokeColor = [[UIColor redColor] CGColor];
UIBezierPath *outerPath = [UIBezierPath bezierPathWithRect:CGRectInset(squareLayer.bounds, 20.0,
20.0)];
UIBezierPath *innerPath = [UIBezierPath bezierPathWithRect:CGRectInset(squareLayer.bounds, 50.0,
50.0)];
CGMutablePathRef combinedPath= CGPathCreateMutableCopy(outerPath.CGPath);
CGPathAddPath(combinedPath, NULL, innerPath.CGPath);
squareLayer.path = combinedPath;
[[self.view layer] addSublayer:squareLayer];


//类型 2: kCAFillRuleEvenOdd
squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(140, 140, 150, 150);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = [[UIColor yellowColor]CGColor];
squareLayer.fillRule = kCAFillRuleEvenOdd;   // 指定规则类型
squareLayer.strokeColor = [[UIColor redColor] CGColor];
outerPath = [UIBezierPath bezierPathWithRect:CGRectInset(squareLayer.bounds, 20.0, 20.0)];
innerPath = [UIBezierPath bezierPathWithRect:CGRectInset(squareLayer.bounds, 50.0, 50.0)];
combinedPath= CGPathCreateMutableCopy(outerPath.CGPath);
CGPathAddPath(combinedPath, NULL, innerPath.CGPath);
squareLayer.path = combinedPath;
[[self.view layer] addSublayer:squareLayer];
```



```
//Fill Pattern Color
//images.jpeg

squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(140, 30, 100, 100);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = [[UIColor colorWithPatternImage:[UIImage
imageNamed:@"images.jpeg"]]CGColor];
squareLayer.strokeColor = [[UIColor redColor] CGColor];
squareLayer.path = [UIBezierPath bezierPathWithRect:squareLayer.bounds].CGPath;
[[self.view layer] addSublayer:squareLayer];


//Fill Rule

//Type 1: kCAFillRuleNonZero
squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(0, 140, 150, 150);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = [[UIColor yellowColor]CGColor];
squareLayer.fillRule = kCAFillRuleNonZero;   // indicate the rule type
squareLayer.strokeColor = [[UIColor redColor] CGColor];
UIBezierPath *outerPath = [UIBezierPath bezierPathWithRect:CGRectInset(squareLayer.bounds, 20.0,
20.0)];
UIBezierPath *innerPath = [UIBezierPath bezierPathWithRect:CGRectInset(squareLayer.bounds, 50.0,
50.0)];
CGMutablePathRef combinedPath= CGPathCreateMutableCopy(outerPath.CGPath);
CGPathAddPath(combinedPath, NULL, innerPath.CGPath);
squareLayer.path = combinedPath;
[[self.view layer] addSublayer:squareLayer];


//Type 2: kCAFillRuleEvenOdd
squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(140, 140, 150, 150);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = [[UIColor yellowColor]CGColor];
squareLayer.fillRule = kCAFillRuleEvenOdd;   // indicate the rule type
squareLayer.strokeColor = [[UIColor redColor] CGColor];
outerPath = [UIBezierPath bezierPathWithRect:CGRectInset(squareLayer.bounds, 20.0, 20.0)];
innerPath = [UIBezierPath bezierPathWithRect:CGRectInset(squareLayer.bounds, 50.0, 50.0)];
combinedPath= CGPathCreateMutableCopy(outerPath.CGPath);
CGPathAddPath(combinedPath, NULL, innerPath.CGPath);
squareLayer.path = combinedPath;
[[self.view layer] addSublayer:squareLayer];
```

列出了访问样式属性

填充颜色
根据绘制的形状填充颜色。

填充规则
填充规则，有两种规则应用于绘制形状。
1. kCAFillRuleNonZero
2. kCAFillRuleEvenOdd

线帽样式
以下类型用于更改线条的样式。
1. kCALineCapButt
2. kCALineCapRound
3. kCALineCapSquare

线段虚线模式
应用于形状路径描边时的虚线模式。
创建虚线样式当你描边该线时。

线段虚线相位
应用于形状路径描边时的虚线相位。可动画。
线段连接方式
形状路径的线段连接样式。以下样式用于绘制线段连接样式。
1. kCALineJoinMiter
2. kCALineJoinRound
3. kCALineJoinBevel

线宽
用于设置线宽。

斜接限制
描边形状路径时使用的斜接限制。可动画。

描边颜色
根据线条路径设置描边颜色。

描边起点
描边开始的位置。

描边终点
描边结束的位置。

Listed the accessing Style properties

```
fillColor
     Fill the color based on the drawed shape.

fillRule
     Fill Rule the there are two rule is applied to draw the shape.
     1.  kCAFillRuleNonZero
     2.  kCAFillRuleEvenOdd

lineCap
     Below type used to change the style of the line.
     1.  kCALineCapButt
     2.  kCALineCapRound
     3.  kCALineCapSquare

lineDashPattern
     The dash pattern applied to the shape's path when stroked.
     Create DashStyle while you will stroke the line.

lineDashPhase
     The dash phase applied to the shape's path when stroked. Animatable.
lineJoin
     Line join style for the shape path.Below style use to draw the line join style.
     1.  kCALineJoinMiter
     2.  kCALineJoinRound
     3.  kCALineJoinBevel

lineWidth
     Which using to set the line width.

miterLimit
     The miter limit used when stroking the shape's path. Animatable.

strokeColor
     Set the stroke color based on the path of the line.

strokeStart
     When the stroke will start.

strokeEnd
    When the stroke will end.
```

第116章：WKWebView

WKWebView是iOS 8和OS X Yosemite中引入的现代WebKit API的核心。它取代了UIKit中的UIWebView和AppKit中的WebView，在两个平台上提供了一致的API。

WKWebView拥有流畅的60fps滚动、内置手势、应用与网页之间的简化通信，以及与Safari相同的JavaScript引擎，是WWDC 2014发布的最重要的内容之一。

第116.1节：添加从应用程序包加载的自定义用户脚本

```
let configuration = WKWebViewConfiguration()

if let path = NSBundle.mainBundle().pathForResource("customUserScript", ofType: "js"),
       source = try? NSString(contentsOfFile: path, encoding: NSUTF8StringEncoding) as String {

        let userScript = WKUserScript(source: source, injectionTime:
WKUserScriptInjectionTime.AtDocumentStart, forMainFrameOnly: false)

        let userContentController = WKUserContentController()
        userContentController.addUserScript(userScript)
        configuration.userContentController = userContentController
    }

let webView = WKWebView(frame: self.view.bounds, configuration: configuration)
```

在WKUserScriptInjectionTime枚举中的任何值都是有效的：.AtDocumentStart，.AtDocumentEnd

第116.2节：从JavaScript发送消息并在本地端处理

可以使用以下代码从JavaScript发送消息

```
window.webkit.messageHandlers.{NAME}.postMessage()
```

以下是创建脚本消息处理器以处理消息的方法：

```
class NotificationScriptMessageHandler: NSObject, WKScriptMessageHandler {
    func userContentController(userContentController: WKUserContentController,
didReceiveScriptMessage message: WKScriptMessage!) {
        if message.name == "{NAME}" {
            // 确保处理正确的消息
            print(message.body)
        }
    }
}
```

以下是在WKWebView中配置脚本消息处理器的方法：

```
let configuration = WKWebViewConfiguration()
let userContentController = WKUserContentController()
let handler = NotificationScriptMessageHandler()
userContentController.addScriptMessageHandler(handler, name: "{NAME}")
configuration.userContentController = userContentController
```


Chapter 116: WKWebView

WKWebView is the centerpiece of the modern WebKit API introduced in iOS 8 & OS X Yosemite. It replaces UIWebView in UIKit and WebView in AppKit, offering a consistent API across the two platforms.

Boasting responsive 60fps scrolling, built-in gestures, streamlined communication between app and webpage, and the same JavaScript engine as Safari, WKWebView is one of the most significant announcements to come out of WWDC 2014.

Section 116.1: Adding custom user script loaded from app bundle

```
let configuration = WKWebViewConfiguration()

if let path = NSBundle.mainBundle().pathForResource("customUserScript", ofType: "js"),
       source = try? NSString(contentsOfFile: path, encoding: NSUTF8StringEncoding) as String {

        let userScript = WKUserScript(source: source, injectionTime:
WKUserScriptInjectionTime.AtDocumentStart, forMainFrameOnly: false)

        let userContentController = WKUserContentController()
        userContentController.addUserScript(userScript)
        configuration.userContentController = userContentController
    }

let webView = WKWebView(frame: self.view.bounds, configuration: configuration)
```

Any value in the `WKUserScriptInjectionTime` enum is valid: `.AtDocumentStart`, `.AtDocumentEnd`

Section 116.2: Send messages from JavaScript and Handle them on the native side

Messages can be sent from JavaScript using the following code

```
window.webkit.messageHandlers.{NAME}.postMessage()
```

Here how to create a script message handler to handle the messages:

```
class NotificationScriptMessageHandler: NSObject, WKScriptMessageHandler {
    func userContentController(userContentController: WKUserContentController,
didReceiveScriptMessage message: WKScriptMessage!) {
        if message.name == "{NAME}" {
            // to be sure of handling the correct message
            print(message.body)
        }
    }
}
```

Here how to configure the script message handler in the WKWebView:

```
let configuration = WKWebViewConfiguration()
let userContentController = WKUserContentController()
let handler = NotificationScriptMessageHandler()
userContentController.addScriptMessageHandler(handler, name: "{NAME}")
configuration.userContentController = userContentController
```

```
let webView = WKWebView(frame: self.view.bounds, configuration: configuration)
```

> **注意：**多次使用addScriptMessageHandler:name:添加相同的"{NAME}"处理器，会导致NSInvalidArgumentException异常。

第116.3节：创建一个简单的网页浏览器

```
import UIKit
import WebKit


        class ViewController: UIViewController, UISearchBarDelegate, WKNavigationDelegate, WKUIDelegate {

    var searchbar: UISearchBar! //所有网页浏览器都有搜索栏。
    var webView: WKWebView! //我们将使用的WKWebView。
    var toolbar: UIToolbar! //底部的工具栏，就像Safari中的一样。
    var activityIndicator: UIActivityIndicatorView! //活动指示器，用于告知用户页面正在加载。


    override func viewDidLoad() {
        super.viewDidLoad()

        self.initControls()
        self.setTheme()
        self.doLayout()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }


    func initControls() {
        self.searchbar = UISearchBar()

        //WKUserContentController 允许我们向 webView 添加 JavaScript 脚本，这些脚本将在页面加载开始时或结束时运行
。

        let configuration = WKWebViewConfiguration()
        let contentController = WKUserContentController()
        configuration.userContentController = contentController

        //使用自定义配置创建 webView。
        self.webView = WKWebView(frame: .zero, configuration: configuration)

        self.toolbar = UIToolbar()
        self.layoutToolbar()

        self.activityIndicator = UIActivityIndicatorView(activityIndicatorStyle: .gray)
        self.activityIndicator.hidesWhenStopped = true
    }

    func setTheme() {
        self.edgesForExtendedLayout = UIRectEdge(rawValue: 0)
        self.navigationController?.navigationBar.barTintColor = UIColor.white()

        //设置键盘和搜索栏的主题。设置代理。
        self.searchbar.delegate = self
        self.searchbar.returnKeyType = .go
```



```
let webView = WKWebView(frame: self.view.bounds, configuration: configuration)
```

> **NOTE:** adding same "{NAME}" handler with addScriptMessageHandler:name: more than once, results in NSInvalidArgumentExceptionException.

Section 116.3: Creating a Simple WebBrowser

```
import UIKit
import WebKit


class ViewController: UIViewController, UISearchBarDelegate, WKNavigationDelegate, WKUIDelegate {

    var searchbar: UISearchBar! //All web-browsers have a search-bar.
    var webView: WKWebView! //The WKWebView we'll use.
    var toolbar: UIToolbar! //Toolbar at the bottom just like in Safari.
    var activityIndicator: UIActivityIndicatorView! //Activity indicator to let the user know the
page is loading.

    override func viewDidLoad() {
        super.viewDidLoad()

        self.initControls()
        self.setTheme()
        self.doLayout()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }


    func initControls() {
        self.searchbar = UISearchBar()

        //WKUserContentController allows us to add Javascript scripts to our webView that will run
either at the beginning of a page load OR at the end of a page load.

        let configuration = WKWebViewConfiguration()
        let contentController = WKUserContentController()
        configuration.userContentController = contentController

        //create the webView with the custom configuration.
        self.webView = WKWebView(frame: .zero, configuration: configuration)

        self.toolbar = UIToolbar()
        self.layoutToolbar()

        self.activityIndicator = UIActivityIndicatorView(activityIndicatorStyle: .gray)
        self.activityIndicator.hidesWhenStopped = true
    }

    func setTheme() {
        self.edgesForExtendedLayout = UIRectEdge(rawValue: 0)
        self.navigationController?.navigationBar.barTintColor = UIColor.white()

        //Theme the keyboard and searchBar. Setup delegates.
        self.searchbar.delegate = self
        self.searchbar.returnKeyType = .go
```

```
        self.searchbar.searchBarStyle = .prominent
        self.searchbar.placeholder = "搜索或输入网站名称"
        self.searchbar.autocapitalizationType = .none
        self.searchbar.autocorrectionType = .no

        //设置 WebView 的代理。
        self.webView.navigationDelegate = self //处理页面导航的代理self.webView.uiDelegate = self //处
        理新标签页、窗口、弹出窗口、布局等的代理


        self.activityIndicator.transform = CGAffineTransform(scaleX: 1.5, y: 1.5)
    }

    func 布局工具栏() {
        //浏览器通常有后退按钮、前进按钮、刷新按钮和
新标签页/新窗口按钮。

        var items = Array<UIBarButtonItem>()

        let space = UIBarButtonItem(barButtonSystemItem: .flexibleSpace, target: nil, action: nil)

items.append(UIBarButtonItem(title: "<", style: .plain, target: self, action:
#selector(onBackButtonPressed)))
items.append(space)
items.append(UIBarButtonItem(title: ">", style: .plain, target: self, action:
#selector(onForwardButtonPressed)))
        items.append(space)
items.append(UIBarButtonItem(barButtonSystemItem: .refresh, target: self, action:
#selector(onRefreshPressed)))
items.append(space)
items.append(UIBarButtonItem(barButtonSystemItem: .organize, target: self, action:
#selector(onTabPressed)))

        self.toolbar.items = items
    }

    func doLayout() {
        //将搜索栏添加到导航栏。
        self.navigationItem.titleView = self.searchbar

        //将所有其他子视图添加到 self.view。
        self.view.addSubview(self.webView)
        self.view.addSubview(self.toolbar)
        self.view.addSubview(self.activityIndicator)

        //设置将要约束的视图。

        let views: [String: AnyObject] = ["webView": self.webView, "toolbar": self.toolbar,
"activityIndicator": self.activityIndicator];
        var constraints = Array<String>();

constraints.append("H:|-0-[webView]-0-|")
        constraints.append("H:|-0-[toolbar]-0-|")
        constraints.append("V:|-0-[webView]-0-[toolbar(50)]-0-|")


        //使用上述视觉约束约束子视图。

        for constraint in constraints {
            self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat: constraint,
options: NSLayoutFormatOptions(rawValue: 0), metrics: nil, views: views))
        }
```



```
        self.searchbar.searchBarStyle = .prominent
        self.searchbar.placeholder = "Search or enter website name"
        self.searchbar.autocapitalizationType = .none
        self.searchbar.autocorrectionType = .no

        //Set the WebView's delegate.
        self.webView.navigationDelegate = self //Delegate that handles page navigation
        self.webView.uiDelegate = self //Delegate that handles new tabs, windows, popups, layout,
etc..

        self.activityIndicator.transform = CGAffineTransform(scaleX: 1.5, y: 1.5)
    }

    func layoutToolbar() {
        //Browsers typically have a back button, forward button, refresh button, and
newTab/newWindow button.

        var items = Array<UIBarButtonItem>()

        let space = UIBarButtonItem(barButtonSystemItem: .flexibleSpace, target: nil, action: nil)

        items.append(UIBarButtonItem(title: "<", style: .plain, target: self, action:
#selector(onBackButtonPressed)))
        items.append(space)
        items.append(UIBarButtonItem(title: ">", style: .plain, target: self, action:
#selector(onForwardButtonPressed)))
        items.append(space)
        items.append(UIBarButtonItem(barButtonSystemItem: .refresh, target: self, action:
#selector(onRefreshPressed)))
        items.append(space)
        items.append(UIBarButtonItem(barButtonSystemItem: .organize, target: self, action:
#selector(onTabPressed)))

        self.toolbar.items = items
    }

    func doLayout() {
        //Add the searchBar to the navigationBar.
        self.navigationItem.titleView = self.searchbar

        //Add all other subViews to self.view.
        self.view.addSubview(self.webView)
        self.view.addSubview(self.toolbar)
        self.view.addSubview(self.activityIndicator)

        //Setup which views will be constrained.

        let views: [String: AnyObject] = ["webView": self.webView, "toolbar": self.toolbar,
"activityIndicator": self.activityIndicator];
        var constraints = Array<String>();

        constraints.append("H:|-0-[webView]-0-|")
        constraints.append("H:|-0-[toolbar]-0-|")
        constraints.append("V:|-0-[webView]-0-[toolbar(50)]-0-|")


        //constrain the subviews using the above visual constraints.

        for constraint in constraints {
            self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat: constraint,
options: NSLayoutFormatOptions(rawValue: 0), metrics: nil, views: views))
        }
```

```
        for view in self.view.subviews {
view.translatesAutoresizingMaskIntoConstraints = false
        }


        //将活动指示器约束到视图的中心。
                self.view.addConstraint(NSLayoutConstraint(item: self.activityIndicator, attribute:
.centerX, relatedBy: .equal, toItem: self.view, attribute: .centerX, multiplier: 1.0, constant:
0.0))
                self.view.addConstraint(NSLayoutConstraint(item: self.activityIndicator, attribute:
.centerY, relatedBy: .equal, toItem: self.view, attribute: .centerY, multiplier: 1.0, constant:
0.0))
    }

    //搜索栏代理方法

    func searchBarSearchButtonClicked(_ searchBar: UISearchBar) {
        self.searchbar.resignFirstResponder()

        if let searchText = self.searchbar.text, url = URL(string: searchText) {
            //从搜索栏获取URL。使用该URL创建一个新的NSURLRequest，并告诉
webView导航到该URL/页面。同时指定超时时间以防页面加载过久。还
处理了cookie/缓存策略。

            let request = URLRequest(url: url, cachePolicy: .useProtocolCachePolicy,
timeoutInterval: 30)
            self.webView.load(request)
        }
    }


    //工具栏代理

    func onBackButtonPressed(button: UIBarButtonItem) {
        if (self.webView.canGoBack) { //允许用户返回上一页。
            self.webView.goBack()
        }
    }

    func onForwardButtonPressed(button: UIBarButtonItem) {
        if (self.webView.canGoForward) { //允许用户前进到下一页。
            self.webView.goForward()
        }
    }

    func onRefreshPressed(button: UIBarButtonItem) {
        self.webView.reload()  //重新加载当前页面。
    }

    func onTabPressed(button: UIBarButtonItem) {
        //TODO: 打开一个新标签页或网页。
    }


    //网页视图代理

    func webView(_ webView: WKWebView, decidePolicyFor navigationAction: WKNavigationAction,
decisionHandler: (WKNavigationActionPolicy) -> Void) {

        decisionHandler(.allow) //允许用户导航到请求的页面。
    }
```



```
        for view in self.view.subviews {
            view.translatesAutoresizingMaskIntoConstraints = false
        }


        //constraint the activity indicator to the center of the view.
        self.view.addConstraint(NSLayoutConstraint(item: self.activityIndicator, attribute:
.centerX, relatedBy: .equal, toItem: self.view, attribute: .centerX, multiplier: 1.0, constant:
0.0))
        self.view.addConstraint(NSLayoutConstraint(item: self.activityIndicator, attribute:
.centerY, relatedBy: .equal, toItem: self.view, attribute: .centerY, multiplier: 1.0, constant:
0.0))
    }

    //Searchbar Delegates

    func searchBarSearchButtonClicked(_ searchBar: UISearchBar) {
        self.searchbar.resignFirstResponder()

        if let searchText = self.searchbar.text, url = URL(string: searchText) {
            //Get the URL from the search bar. Create a new NSURLRequest with it and tell the
webView to navigate to that URL/Page. Also specify a timeout for if the page takes too long. Also
handles cookie/caching policy.

            let request = URLRequest(url: url, cachePolicy: .useProtocolCachePolicy,
timeoutInterval: 30)
            self.webView.load(request)
        }
    }


    //Toolbar Delegates

    func onBackButtonPressed(button: UIBarButtonItem) {
        if (self.webView.canGoBack) { //allow the user to go back to the previous page.
            self.webView.goBack()
        }
    }

    func onForwardButtonPressed(button: UIBarButtonItem) {
        if (self.webView.canGoForward) { //allow the user to go forward to the next page.
            self.webView.goForward()
        }
    }

    func onRefreshPressed(button: UIBarButtonItem) {
        self.webView.reload()  //reload the current page.
    }

    func onTabPressed(button: UIBarButtonItem) {
        //TODO: Open a new tab or web-page.
    }


    //WebView Delegates

    func webView(_ webView: WKWebView, decidePolicyFor navigationAction: WKNavigationAction,
decisionHandler: (WKNavigationActionPolicy) -> Void) {

        decisionHandler(.allow) //allow the user to navigate to the requested page.
    }
```

```
    func webView(_ webView: WKWebView, decidePolicyFor navigationResponse: WKNavigationResponse,
decisionHandler: (WKNavigationResponsePolicy) -> Void) {

        decisionHandler(.allow) //允许 webView 处理响应。
    }

    func webView(_ webView: WKWebView, didStartProvisionalNavigation navigation: WKNavigation!) {
        self.activityIndicator.startAnimating()
    }

    func webView(_ webView: WKWebView, didFailProvisionalNavigation navigation: WKNavigation!,
withError error: NSError) {
        self.activityIndicator.stopAnimating()

        //处理错误。向用户显示一个警告，告诉他们发生了什么。

        let alert = UIAlertController(title: "错误", message: error.localizedDescription,
preferredStyle: .alert)
        let action = UIAlertAction(title: "确定", style: .default) { (action) in
            alert.dismiss(animated: true, completion: nil)
        }
alert.addAction(action)
        self.present(alert, animated: true, completion: nil)
    }

    func webView(_ webView: WKWebView, didFinish navigation: WKNavigation!) {
        self.activityIndicator.stopAnimating()

        //使用网页的最终端点 URL 更新我们的搜索栏。
        if let url = self.webView.url {
            self.searchbar.text = url.absoluteString ?? self.searchbar.text
        }
    }

    func webView(_ webView: WKWebView, didReceiveServerRedirectForProvisionalNavigation navigation:
WKNavigation!) {
        //当 WebView 收到跳转到不同页面或端点时，会调用此方法。
    }

    func webView(_ webView: WKWebView, didCommit navigation: WKNavigation!) {
        //当网页内容开始加载时，会调用此方法。
    }

    func webView(_ webView: WKWebView, didFail navigation: WKNavigation!, withError error: NSError)
{

    }

    func webView(_ webView: WKWebView, didReceive challenge: URLAuthenticationChallenge,
completionHandler: (URLSession.AuthChallengeDisposition, URLCredential?) -> Void) {

        completionHandler(.performDefaultHandling, .none) //默认处理 SSL 连接。
//我们没有进行 SSL 固定或自定义证书处理。

    }


    //WebView 的 UINavigation 代理方法

    //当 WebView 或已加载页面想要打开新窗口/标签页时，会调用此方法。
    func webView(_ webView: WKWebView, createWebViewWith configuration: WKWebViewConfiguration, for
```



```
    func webView(_ webView: WKWebView, decidePolicyFor navigationResponse: WKNavigationResponse,
decisionHandler: (WKNavigationResponsePolicy) -> Void) {

        decisionHandler(.allow) //allow the webView to process the response.
    }

    func webView(_ webView: WKWebView, didStartProvisionalNavigation navigation: WKNavigation!) {
        self.activityIndicator.startAnimating()
    }

    func webView(_ webView: WKWebView, didFailProvisionalNavigation navigation: WKNavigation!,
withError error: NSError) {
        self.activityIndicator.stopAnimating()

        //Handle the error. Display an alert to the user telling them what happened.

        let alert = UIAlertController(title: "Error", message: error.localizedDescription,
preferredStyle: .alert)
        let action = UIAlertAction(title: "OK", style: .default) { (action) in
            alert.dismiss(animated: true, completion: nil)
        }
        alert.addAction(action)
        self.present(alert, animated: true, completion: nil)
    }

    func webView(_ webView: WKWebView, didFinish navigation: WKNavigation!) {
        self.activityIndicator.stopAnimating()

        //Update our search bar with the webPage's final endpoint-URL.
        if let url = self.webView.url {
            self.searchbar.text = url.absoluteString ?? self.searchbar.text
        }
    }

    func webView(_ webView: WKWebView, didReceiveServerRedirectForProvisionalNavigation navigation:
WKNavigation!) {
        //When the webview receives a "Redirect" to a different page or endpoint, this is called.
    }

    func webView(_ webView: WKWebView, didCommit navigation: WKNavigation!) {
        //When the content for the webpage starts arriving, this is called.
    }

    func webView(_ webView: WKWebView, didFail navigation: WKNavigation!, withError error: NSError)
{

    }

    func webView(_ webView: WKWebView, didReceive challenge: URLAuthenticationChallenge,
completionHandler: (URLSession.AuthChallengeDisposition, URLCredential?) -> Void) {

        completionHandler(.performDefaultHandling, .none) //Handle SSL connections by default. We
aren't doing SSL pinning or custom certificate handling.

    }


    //WebView's UINavigation Delegates

    //This is called when a webView or existing loaded page wants to open a new window/tab.
    func webView(_ webView: WKWebView, createWebViewWith configuration: WKWebViewConfiguration, for
```

```
navigationAction: WKNavigationAction, windowFeatures: WKWindowFeatures) -> WKWebView? {

        // 表示新标签页/窗口的视图。该视图左上角会有一个关闭按钮 + 一个 webView。

        let container = UIView()

        // 新标签页需要一个退出按钮。
        let XButton = UIButton()
XButton.addTarget(self, action: #selector(onWebViewExit), for: .touchUpInside)
        XButton.layer.cornerRadius = 22.0

        // 创建新的 webView 窗口。
        let webView = WKWebView(frame: .zero, configuration: configuration)
        webView.navigationDelegate = self
        webView.uiDelegate = self

        // 布局标签页。
container.addSubview(XButton)
        container.addSubview(webView)

        let views: [String: AnyObject] = ["XButton": XButton, "webView": webView];
        var constraints = Array<String>()

约束条件。append("H:|-(-22)-[XButton(44)]")
       约束条件。append("H:|-0-[webView]-0-|")
       约束条件。append("V:|-(-22)-[XButton(44)]-0-[webView]-0-|")


        //约束子视图。
        for constraint in constraints {
容器。addConstraints(NSLayoutConstraint.constraints(withVisualFormat: constraint,
options: NSLayoutFormatOptions(rawValue: 0), metrics: nil, views: views))
        }

        for 视图 in 容器。子视图 {
view.translatesAutoresizingMaskIntoConstraints = false
        }

        //TODO: 将containerView添加到self.view或用新控制器呈现。跟踪标签页..


        返回 webView
    }

    func onWebViewExit(button: UIButton) {
                                //TODO: 销毁标签页。从当前窗口或控制器中移除新标签页。
    }
}
```

在键盘中显示自定义的GO按钮：


```
navigationAction: WKNavigationAction, windowFeatures: WKWindowFeatures) -> WKWebView? {

        //The view that represents the new tab/window. This view will have an X button at the top
left corner + a webView.
        let container = UIView()

        //New tabs need an exit button.
        let XButton = UIButton()
        XButton.addTarget(self, action: #selector(onWebViewExit), for: .touchUpInside)
        XButton.layer.cornerRadius = 22.0

        //Create the new webView window.
        let webView = WKWebView(frame: .zero, configuration: configuration)
        webView.navigationDelegate = self
        webView.uiDelegate = self

        //Layout the tab.
        container.addSubview(XButton)
        container.addSubview(webView)

        let views: [String: AnyObject] = ["XButton": XButton, "webView": webView];
        var constraints = Array<String>()

        constraints.append("H:|-(-22)-[XButton(44)]")
        constraints.append("H:|-0-[webView]-0-|")
        constraints.append("V:|-(-22)-[XButton(44)]-0-[webView]-0-|")


        //constrain the subviews.
        for constraint in constraints {
            container.addConstraints(NSLayoutConstraint.constraints(withVisualFormat: constraint,
options: NSLayoutFormatOptions(rawValue: 0), metrics: nil, views: views))
        }

        for view in container.subviews {
            view.translatesAutoresizingMaskIntoConstraints = false
        }

        //TODO: Add the containerView to self.view or present it with a new controller. Keep track
of tabs..

        return webView
    }

    func onWebViewExit(button: UIButton) {
        //TODO: Destroy the tab. Remove the new tab from the current window or controller.
    }
}
```

Showing the custom GO button in the keyboard:

iPhone SE – iOS 10.0 (14A5297c)
Carrier
10:58 PM
https://stackoverflow.com/
Stack Overflow
sign up
log in
Questions
Tags
Users
Badges
Unanswered
Ask
All Questions
Show
Interesting
0
0
Mysql error, "Specified key was too long; max key length is 767 bytes" need workaround
mysql
6 secs ago rerat
0
Error Code: 1305. FUNCTION or PROCEDURE does not exist
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
Go

iPhone SE – iOS 10.0 (14A5297c)
Carrier
10:58 PM
https://stackoverflow.com/
Stack Overflow
sign up
log in
Questions
Tags
Users
Badges
Unanswered
Ask
All Questions
Show
Interesting
0
0
Mysql error, "Specified key was too long; max key length is 767 bytes" need workaround
mysql
6 secs ago rerat
0
Error Code: 1305. FUNCTION or PROCEDURE does not exist
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
Go

显示工具栏和完全加载的页面。

Showing the toolbar and fully loaded page.

iPhone SE – iOS 10.0 (14A5297c)

Carrier 10:58 PM

https://stackoverflow.com/

Stack Overflow sign up log in

Questions Tags Users Badges Unanswered Ask

All Questions Show Interesting

0 0 Mysql error, "Specified key was too long; max key length is 767 bytes" need workaround

mysql

6 secs ago rerat

0 0 Error Code: 1305. FUNCTION or PROCEDURE does not exist

mysql

8 secs ago neubert

0 0 cqlengine python lib is not recognizing batchquery object

python cqlengine

14 secs ago gotham

iPhone SE – iOS 10.0 (14A5297c)

Carrier 10:58 PM

https://stackoverflow.com/

Stack Overflow sign up log in

Questions Tags Users Badges Unanswered Ask

All Questions Show Interesting

0 0 Mysql error, "Specified key was too long; max key length is 767 bytes" need workaround

mysql

6 secs ago rerat

0 0 Error Code: 1305. FUNCTION or PROCEDURE does not exist

mysql

8 secs ago neubert

0 0 cqlengine python lib is not recognizing batchquery object

python cqlengine

14 secs ago gotham

第117章：UUID（通用唯一标识符）

第117.1节：苹果的IFA与IFV（苹果广告标识符与供应商标识符）

- 您可以使用IFA来衡量广告点击量，使用IFV来衡量应用安装量。
- IFA具有内置的隐私机制，非常适合用于广告。相比之下，IFV供开发者内部使用，用于衡量安装其应用的用户。

IFA

- ASIdentifierManager类提供
  - advertisingIdentifier：UUID：一个唯一的字母数字字符串，专属于每台设备，仅用于投放广告。
  - isAdvertisingTrackingEnabled：一个布尔值，表示用户是否限制了广告追踪。

IFV

- ASIdentifierManager类提供
  - identifierForVendor：UUID：一个字母数字字符串，用于唯一标识设备对应用程序的供应商。

在这里找到您的设备IFA和IFV。

第117.2节：为iOS设备创建UUID字符串

这里我们可以在一行内创建UUID字符串。

表示UUID字符串，可用于唯一标识类型、接口和其他项目。

Swift 3.0

```
print(UUID().uuidString)
```

它对于使用唯一ID识别多个设备非常有用。

第117.3节：生成UUID

随机 UUID

Swift

```
func randomUUID() -> NSString{
    return NSUUID.UUID().UUIDString()
}
```

Objective-C

```
+ (NSString *)randomUUID {
    if(NSClassFromString(@"NSUUID")) { // 仅在 iOS >= 6.0 可用
        return [[NSUUID UUID] UUIDString];
    }
    CFUUIDRef uuidRef = CFUUIDCreate(kCFAllocatorDefault);
```


Chapter 117: UUID (Universally Unique Identifier)

Section 117.1: Apple's IFA vs. IFV (Apple Identifier for Advertisers vs. Identifier for Vendors)

- You can use the IFA for measuring ad clicks and the IFV for measuring app installs.
- IFA has built-in privacy mechanisms that make it perfect for advertising. In contrast, the IFV is for developers to use internally to measure users who install their apps.

IFA

- ASIdentifierManager class provides
  - **advertisingIdentifier: UUID**: An alphanumeric string unique to each device, used only for serving advertisements.
  - **isAdvertisingTrackingEnabled**: A Boolean value that indicates whether the user has limited ad tracking.

IFV

- ASIdentifierManager class provides
  - **identifierForVendor: UUID**: An alphanumeric string that uniquely identifies a device to the app's vendor.

Find your device IFA and IFV here.

Section 117.2: Create UUID String for iOS devices

Here we can create `UUID String` with in one line.

Represents UUID strings, which can be used to uniquely identify types, interfaces, and other items.

Swift 3.0

```
print(UUID().uuidString)
```

It is very useful for identify multiple devices with unique id.

Section 117.3: Generating UUID

Random UUID

Swift

```
func randomUUID() -> NSString{
    return NSUUID.UUID().UUIDString()
}
```

Objective-C

```
+ (NSString *)randomUUID {
    if(NSClassFromString(@"NSUUID")) { // only available in iOS >= 6.0
        return [[NSUUID UUID] UUIDString];
    }
    CFUUIDRef uuidRef = CFUUIDCreate(kCFAllocatorDefault);
```

```
    CFStringRef cfuuid = CFUUIDCreateString(kCFAllocatorDefault, uuidRef);
    CFRelease(uuidRef);
    NSString *uuid = [((__bridge NSString *) cfuuid) copy];
    CFRelease(cfuuid);
    return uuid;
}
```

第117.4节：供应商标识符

版本 ≥ iOS 6

在一行代码中，我们可以获得如下的UUID：

Swift

```
let UDIDString = UIDevice.currentDevice().identifierForVendor?.UUIDString
```

Objective-C

```
NSString *UDIDString = [[[UIDevice currentDevice] identifierForVendor] UUIDString];
```

identifierForVendor 是一个唯一标识符，在单个设备上同一供应商的所有应用中保持不变，除非该设备上该供应商的所有应用都被删除。请参阅 Apple的文档 了解该 UUID 何时会发生变化。


```
    CFStringRef cfuuid = CFUUIDCreateString(kCFAllocatorDefault, uuidRef);
    CFRelease(uuidRef);
    NSString *uuid = [((__bridge NSString *) cfuuid) copy];
    CFRelease(cfuuid);
    return uuid;
}
```

Section 117.4: Identifier for vendor

Version ≥ iOS 6

Within a single line, we can get an UUID like below:

Swift

```
let UDIDString = UIDevice.currentDevice().identifierForVendor?.UUIDString
```

Objective-C

```
NSString *UDIDString = [[[UIDevice currentDevice] identifierForVendor] UUIDString];
```

The `identifierForVendor` is an unique identifier that stays the same for every app of a single vendor on a single device, unless all of the vendor's apps are deleted from this device. See Apple's documentation about when this `UUID` changes.

第118章：类别

第118.1节：创建类别

类别提供了在不进行子类化或更改实际对象的情况下，为对象添加额外功能的能力。

例如，我们想设置一些自定义字体。让我们创建一个为 UIFont 类添加功能的类别。打开你的Xcode项目，点击文件 -> 新建 -> 文件，选择Objective-C文件，点击下一步，输入你的类别名称，比如“CustomFont”，选择文件类型为类别，类为UIFont，然后点击“下一步”，最后点击“创建”。

Chapter 118: Categories

Section 118.1: Create a Category

Categories provide the ability to add some extra functionality to an object without subclassing or changing the actual object.

For example we want to set some custom fonts. Lets create a category that add functionality to `UIFont` class. Open your Xcode project, click on File -> New -> File and choose Objective-C file , click Next enter your category name say "CustomFont" choose file type as Category and Class as UIFont then Click "Next" followed by "Create."

声明类别方法：

点击 "UIFont+CustomFonts.h" 查看新类别的头文件。在接口中添加以下代码以声明该方法。

```
@interface UIFont (CustomFonts)

+(UIFont *)productSansRegularFontWithSize:(CGFloat)size;

@end
```

现在实现类别方法：

点击 "UIFont+CustomFonts.m" 查看类别的实现文件。添加以下代码以创建一个设置 ProductSansRegular 字体的方法。

```
+(UIFont *)productSansRegularFontWithSize:(CGFloat)size{

    return [UIFont fontWithName:@"ProductSans-Regular" size:size];

}
```

导入你的类别

```
#import "UIFont+CustomFonts.h"
```

现在设置标签字体

Declare the Category Method:

Click "UIFont+CustomFonts.h" to view the new category's header file. Add the following code to the interface to declare the method.

```
@interface UIFont (CustomFonts)

+(UIFont *)productSansRegularFontWithSize:(CGFloat)size;

@end
```

Now Implement the Category Method:

Click "UIFont+CustomFonts.m" to view the category's implementation file. Add the following code to create a method that will set ProductSansRegular Font.

```
+(UIFont *)productSansRegularFontWithSize:(CGFloat)size{

    return [UIFont fontWithName:@"ProductSans-Regular" size:size];

}
```

Import your category

```
#import "UIFont+CustomFonts.h"
```

Now set the Label font

```
[self.label setFont:[UIFont productSansRegularFontWithSize:16.0]];
```



```
[self.label setFont:[UIFont productSansRegularFontWithSize:16.0]];
```

第119章：处理URL方案

参数	含义
aUrl	a NSURL实例，存储内置或自定义的方案字符串

第119.1节：使用内置URL方案打开邮件应用

Swift:

```
if let url = URL(string: "mailto://azimov@demo.com") {
    if UIApplication.shared.canOpenURL(url) {
        UIApplication.shared.openURL(url)
    } else {
        print("无法打开URL")
    }
}
```

Objective-C:

```
NSURL *url = [NSURL URLWithString:@"mailto://azimov@demo.com"];
if ([[UIApplication sharedApplication] canOpenURL:url]) {
    [[UIApplication sharedApplication] openURL:url];
} else {
NSLog(@"无法打开URL");
}
```

第119.2节：苹果URL方案

这些是iOS、OS X和watchOS 2及更高版本的原生应用支持的URL方案。

在Safari中打开链接：

Objective-C

```
NSString *stringURL = @"http://stackoverflow.com/";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```
let stringURL = "http://stackoverflow.com/"
if let url = URL(string: stringURL) {
    UIApplication.shared.openURL(url)
}
```

开始电话通话

Objective-C

```
NSString *stringURL = @"tel:1-408-555-5555";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```
let stringURL = "tel:1-408-555-5555"
if let url = URL(string: stringURL) {
    UIApplication.shared.openURL(url)
```


Chapter 119: Handling URL Schemes

Parameter	Meaning
aUrl	a NSURL instance which stores a built-in or custom scheme string

Section 119.1: Using built-in URL scheme to open Mail app

Swift:

```
if let url = URL(string: "mailto://azimov@demo.com") {
    if UIApplication.shared.canOpenURL(url) {
        UIApplication.shared.openURL(url)
    } else {
        print("Cannot open URL")
    }
}
```

Objective-C:

```
NSURL *url = [NSURL URLWithString:@"mailto://azimov@demo.com"];
if ([[UIApplication sharedApplication] canOpenURL:url]) {
    [[UIApplication sharedApplication] openURL:url];
} else {
    NSLog(@"Cannot open URL");
}
```

Section 119.2: Apple URL Schemes

These are URL schemes supported by native apps on iOS, OS X, and watchOS 2 and later.

Opening link in Safari:

Objective-C

```
NSString *stringURL = @"http://stackoverflow.com/";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```
let stringURL = "http://stackoverflow.com/"
if let url = URL(string: stringURL) {
    UIApplication.shared.openURL(url)
}
```

Starting a phone conversation

Objective-C

```
NSString *stringURL = @"tel:1-408-555-5555";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```
let stringURL = "tel:1-408-555-5555"
if let url = URL(string: stringURL) {
    UIApplication.shared.openURL(url)
```

```
}
```

HTML

```
<a href="tel:1-408-555-5555">1-408-555-5555</a>
```

开始FaceTime通话

Objective-C

```
NSString *stringURL = @"facetime:14085551234";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```
let stringURL = "facetime:14085551234"
if let url = URL(string: stringURL) {
    UIApplication.shared.openURL(url)
}
```

HTML

```
<a href="facetime:14085551234">使用FaceTime连接</a>
<a href="facetime:user@example.com">使用FaceTime连接</a>
```

打开信息应用以撰写短信给收件人：

Objective-C

```
NSString *stringURL = @"sms:1-408-555-1212";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```
let stringURL = "sms:1-408-555-1212"
if let url = URL(string: stringURL) {
    UIApplication.shared.openURL(url)
}
```

HTML

```
<a href="sms:">启动信息应用</a>
<a href="sms:1-408-555-1212">新短信</a>
```

打开邮件应用以撰写邮件给收件人：

Objective-C

```
NSString *stringURL = @"mailto:foo@example.com";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Swift:


```
}
```

HTML

```
<a href="tel:1-408-555-5555">1-408-555-5555</a>
```

Starting a FaceTime conversation

Objective-C

```
NSString *stringURL = @"facetime:14085551234";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```
let stringURL = "facetime:14085551234"
if let url = URL(string: stringURL) {
    UIApplication.shared.openURL(url)
}
```

HTML

```
<a href="facetime:14085551234">Connect using FaceTime</a>
<a href="facetime:user@example.com">Connect using FaceTime</a>
```

Opening Messages App to compose an sms to recipient:

Objective-C

```
NSString *stringURL = @"sms:1-408-555-1212";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```
let stringURL = "sms:1-408-555-1212"
if let url = URL(string: stringURL) {
    UIApplication.shared.openURL(url)
}
```

HTML

```
<a href="sms:">Launch Messages App</a>
<a href="sms:1-408-555-1212">New SMS Message</a>
```

Opening Mail app to compose an email to recipient:

Objective-C

```
NSString *stringURL = @"mailto:foo@example.com";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```
let stringURL = "mailto:foo@example.com"
if let url = URL(string: stringURL) {
    UIApplication.shared.openURL(url)
}
```

HTML

```
<a href="mailto:frank@wwdcdemo.example.com">约翰·弗兰克</a>
```

您还可以在“收件人”、“抄送”和“密送”字段中包含主题字段、消息和多个收件人。（在 iOS 中，from 属性会被忽略。）以下示例展示了一个包含多个不同属性的 mailto URL：

```
mailto:foo@example.com?cc=bar@example.com&subject=来自库比蒂诺的问候！&body=希望你在这里！
```

注意： 也可以使用MFMailComposeViewController在应用内呈现撰写邮件对话框。


```
let stringURL = "mailto:foo@example.com"
if let url = URL(string: stringURL) {
    UIApplication.shared.openURL(url)
}
```

HTML

```
<a href="mailto:frank@wwdcdemo.example.com">John Frank</a>
```

You can also include a subject field, a message, and multiple recipients in the To, Cc, and Bcc fields. (In iOS, the from attribute is ignored.) The following example shows a mailto URL that includes several different attributes:

```
mailto:foo@example.com?cc=bar@example.com&subject=Greetings%20from%20Cupertino!&body=Wish%20you%20w
ere%20here!
```

Note: Compose email dialog can also be presented within app using `MFMailComposeViewController`.

第120章：Realm

第120.1节：带主键的RLMObject基类模型 - Objective-C

一个使用主键和一些通用默认属性的RLMObject基类模型示例。
子类随后可以设置其特定需求的元数据。

```
@interface BaseModel : RLMObject

@property NSString *uuid;
@property NSString *metadata;

@end

@implementation BaseModel

+ (NSString *)primaryKey
{
    return @"uuid";
}

+ (NSDictionary *)defaultPropertyValues
{
    NSMutableDictionary *defaultPropertyValues = [NSMutableDictionary
dictionaryWithDictionary:[super defaultPropertyValues]];
    NSString *uuid = [[NSUUID UUID] UUIDString];
    [defaultPropertyValues setValue:@"" forKey:@"metadata"];
    [defaultPropertyValues setValue:uuid forKey:@"uuid"];
    return defaultPropertyValues;
}

+ (NSArray *)ignoredProperties
{
    return @[];
}

@end
```


Chapter 120: Realm

Section 120.1: RLMObject Base Model Class with Primary Key - Objective-C

An example of a RLMObject base model class that uses a primary key and some generic default properties.
Subclasses can then set metadata specific to their needs.

```
@interface BaseModel : RLMObject

@property NSString *uuid;
@property NSString *metadata;

@end

@implementation BaseModel

+ (NSString *)primaryKey
{
    return @"uuid";
}

+ (NSDictionary *)defaultPropertyValues
{
    NSMutableDictionary *defaultPropertyValues = [NSMutableDictionary
dictionaryWithDictionary:[super defaultPropertyValues]];
    NSString *uuid = [[NSUUID UUID] UUIDString];
    [defaultPropertyValues setValue:@"" forKey:@"metadata"];
    [defaultPropertyValues setValue:uuid forKey:@"uuid"];
    return defaultPropertyValues;
}

+ (NSArray *)ignoredProperties
{
    return @[];
}

@end
```

第121章：ARC（自动引用计数）

第121.1节：在文件上启用/禁用ARC

可以通过为每个文件添加-fno-objc-arc编译器标志来禁用单个文件的ARC。相反，也可以在Targets‣Build Phases‣Compile Sources中添加。

Chapter 121: ARC (Automatic Reference Counting)

Section 121.1: Enable/Disable ARC on a file

ARC can be disabled for individual files by adding the **`-fno-objc-arc`** compiler flag for each file. Conversely it can be added in *Targets ‣ Build Phases ‣ Compile Sources*

第122章：动态字体

第122.1节：在WKWebView中匹配动态字体大小

WKWebView会调整网页内容的字体大小，以使完整的网页适应设备的外形尺寸。如果你希望网页文本在纵向和横向模式下的大小都与用户偏好的阅读大小相似，则需要显式设置。

Swift

```
// 构建用于动态字体和响应式设计的HTML头部
func buildHTMLHeader() -> String {

    // 获取HTML样式的首选动态字体大小
    let bodySize = UIFont.preferredFont(forTextStyle: UIFontTextStyle.body).pointSize
    let h1Size = UIFont.preferredFont(forTextStyle: UIFontTextStyle.title1).pointSize
    let h2Size = UIFont.preferredFont(forTextStyle: UIFontTextStyle.title2).pointSize
    let h3Size = UIFont.preferredFont(forTextStyle: UIFontTextStyle.title3).pointSize

    // 在iPad上，横屏文本比首选字体大小更大
    var portraitMultiplier = CGFloat(1.0)
    var landscapeMultiplier = CGFloat(0.5)

    // iPhone上的文本被缩小
    if UIDevice.current.model.range(of: "iPhone") != nil {
        portraitMultiplier = CGFloat(3.0)
        landscapeMultiplier = CGFloat(1.5)
    }

    // 开始HTML头部文本
    let patternText = "<html> <head> <style> "

    // 匹配本页的动态字体。
    + "body { background-color: \(backgroundColor);} "
    + "@media all and (orientation:portrait) {img {max-width: 90%; height: auto;} "
    + "p, li { font: -apple-system-body; font-family: Georgia, serif; font-size:calc(\(bodySize *
portraitMultiplier)px + 1.0vw); font-weight: normal; color: \(fontColor) } "
    + "h1 { font: -apple-system-headine; font-family: Verdana, sans-serif; font-size:calc(\(h1Size
* portraitMultiplier)px + 1.0vw); font-weight: bold; color: \(headFontColor) } "
    + "h2 { font: -apple-system-headine; font-family: Verdana, sans-serif; font-size:calc(\(h2Size
* portraitMultiplier)px + 1.0vw); font-weight: bold; color: \(headFontColor) } "
    + "h3, h4 { font: -apple-system-headine; font-family: Verdana, sans-serif; font-
size:calc(\(h3Size * portraitMultiplier)px + 1.0vw); font-weight: bold; color: \(headFontColor) } }
"
    + "@media all and (orientation:landscape) {img {max-width: 65%; height: auto;}"
    + "p, li { font: -apple-system-body; font-family: Georgia, serif; font-size:calc(\(bodySize *
landscapeMultiplier)px + 1.0vw); font-weight: normal; color: \(fontColor) }"
    + "h1 { font: -apple-system-headine; font-family: Verdana, sans-serif; font-size:calc(\(h1Size
* landscapeMultiplier)px + 1.0vw); font-weight: bold; color: \(headFontColor) } "
    + "h2 { font: -apple-system-headine; font-family: Verdana, sans-serif; font-size:calc(\(h2Size
* landscapeMultiplier)px + 1.0vw); font-weight: bold; color: \(headFontColor) } "
    + "h3, h4 { font: -apple-system-headine; font-family: Verdana, sans-serif; font-
size:calc(\(h3Size * landscapeMultiplier)px + 1.0vw); font-weight: bold; color: \(headFontColor) }
} </style>"
    + "</head><body>"
    + "<meta name=\"viewport\" content=\"width: device-width\">"

    return patternText
```


Chapter 122: Dynamic Type

Section 122.1: Matching Dynamic Type Font Size in WKWebView

WKWebView resizes the fonts on web content so that a full-sized web page will fit on the device's form factor. If you want the web text in both portrait and landscape to be similar in size to the user's preferred reading size, you need to set it explicitly.

Swift

```
// build HTML header for dynamic type and responsive design
func buildHTMLHeader() -> String {

    // Get preferred dynamic type font sizes for html styles
    let bodySize = UIFont.preferredFont(forTextStyle: UIFontTextStyle.body).pointSize
    let h1Size = UIFont.preferredFont(forTextStyle: UIFontTextStyle.title1).pointSize
    let h2Size = UIFont.preferredFont(forTextStyle: UIFontTextStyle.title2).pointSize
    let h3Size = UIFont.preferredFont(forTextStyle: UIFontTextStyle.title3).pointSize

    // On iPad, landscape text is larger than preferred font size
    var portraitMultiplier = CGFloat(1.0)
    var landscapeMultiplier = CGFloat(0.5)

    // iPhone text is shrunken
    if UIDevice.current.model.range(of: "iPhone") != nil {
        portraitMultiplier = CGFloat(3.0)
        landscapeMultiplier = CGFloat(1.5)
    }

    // Start HTML header text
    let patternText = "<html> <head> <style> "

    // Match Dynamic Type for this page.
    + "body { background-color: \(backgroundColor);} "
    + "@media all and (orientation:portrait) {img {max-width: 90%; height: auto;} "
    + "p, li { font: -apple-system-body; font-family: Georgia, serif; font-size:calc(\(bodySize *
portraitMultiplier)px + 1.0vw); font-weight: normal; color: \(fontColor) } "
    + "h1 { font: -apple-system-headine; font-family: Verdana, sans-serif; font-size:calc(\(h1Size
* portraitMultiplier)px + 1.0vw); font-weight: bold; color: \(headFontColor) } "
    + "h2 { font: -apple-system-headine; font-family: Verdana, sans-serif; font-size:calc(\(h2Size
* portraitMultiplier)px + 1.0vw); font-weight: bold; color: \(headFontColor) } "
    + "h3, h4 { font: -apple-system-headine; font-family: Verdana, sans-serif; font-
size:calc(\(h3Size * portraitMultiplier)px + 1.0vw); font-weight: bold; color: \(headFontColor) } }
"
    + "@media all and (orientation:landscape) {img {max-width: 65%; height: auto;}"
    + "p, li { font: -apple-system-body; font-family: Georgia, serif; font-size:calc(\(bodySize *
landscapeMultiplier)px + 1.0vw); font-weight: normal; color: \(fontColor) }"
    + "h1 { font: -apple-system-headine; font-family: Verdana, sans-serif; font-size:calc(\(h1Size
* landscapeMultiplier)px + 1.0vw); font-weight: bold; color: \(headFontColor) } "
    + "h2 { font: -apple-system-headine; font-family: Verdana, sans-serif; font-size:calc(\(h2Size
* landscapeMultiplier)px + 1.0vw); font-weight: bold; color: \(headFontColor) } "
    + "h3, h4 { font: -apple-system-headine; font-family: Verdana, sans-serif; font-
size:calc(\(h3Size * landscapeMultiplier)px + 1.0vw); font-weight: bold; color: \(headFontColor) }
} </style>"
    + "</head><body>"
    + "<meta name=\"viewport\" content=\"width: device-width\">"

    return patternText
```

```
}
```

第122.2节：获取当前内容大小

Swift

```
UIApplication.sharedApplication().preferredContentSizeCategory
```

Objective-C

```
[UIApplication sharedApplication].preferredContentSizeCategory;
```

这将返回一个内容大小类别常量，或一个辅助功能内容大小类别常量。

第122.3节：在iOS 10上无通知情况下处理首选文本大小变化

UILabel, UITextField, & UITextView 类从iOS 10开始新增了一个属性，用于在用户更改首选阅读大小时自动调整字体，名为 adjustsFontForContentSizeCategory。

Swift

```
@IBOutlet var label:UILabel!

if #available(iOS 10.0, *) {
    label.adjustsFontForContentSizeCategory = true
} else {
    // 监听 UIContentSizeCategoryDidChangeNotification 并手动处理
    // 因为 adjustsFontForContentSizeCategory 属性不可用。
}
```

第122.4节：文本大小变化通知

您可以注册设备文本大小更改的通知。

Swift

```
NSNotificationCenter.defaultCenter().addObserver(self, selector: #selector(updateFont), name:
name:UIContentSizeCategoryDidChangeNotification, object: nil)
```

Objective-C

```
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(updateFont)
name:UIContentSizeCategoryDidChangeNotification object:nil];
```

通知的userInfo对象包含了新大小，键为UIContentSizeCategoryNewValueKey。


```
}
```

Section 122.2: Get the Current Content Size

Swift

```
UIApplication.sharedApplication().preferredContentSizeCategory
```

Objective-C

```
[UIApplication sharedApplication].preferredContentSizeCategory;
```

This returns a content size category constant, or an accessibility content size category constant.

Section 122.3: Handling Preferred Text Size Change Without Notifications on iOS 10

`UILabel`, `UITextField`, & `UITextView` classes have a new property starting from iOS 10 for automatically resizing their font when a user changes their preferred reading size named `adjustsFontForContentSizeCategory`.

Swift

```
@IBOutlet var label:UILabel!

if #available(iOS 10.0, *) {
    label.adjustsFontForContentSizeCategory = true
} else {
    // Observe for UIContentSizeCategoryDidChangeNotification and handle it manually
    // since the adjustsFontForContentSizeCategory property isn't available.
}
```

Section 122.4: Text Size Change Notification

You can register for notifications of when the device text size is changed.

Swift

```
NSNotificationCenter.defaultCenter().addObserver(self, selector: #selector(updateFont), name:
name:UIContentSizeCategoryDidChangeNotification, object: nil)
```

Objective-C

```
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(updateFont)
name:UIContentSizeCategoryDidChangeNotification object:nil];
```

The notification `userInfo` object contains the new size under `UIContentSizeCategoryNewValueKey`.

第123章：SWRevealViewController

第123.1节：使用

SWRevealViewController 设置基本应用

使用 Swift 语言创建一个带有单视图应用模板的基础应用程序

添加 `SWRevealViewController.h` 和 `SWRevealViewController.m`

然后点击创建桥接头文件按钮

并添加

```
#import "SWRevealViewController.h"
```

到桥接头文件中

然后在故事板中选择viewController并将类改为SWRevealViewController

然后将文件中的viewController重命名为MainViewController，并添加一个名为

RightViewController的新ViewController

Chapter 123: SWRevealViewController

Section 123.1: Setting up a basic app with SWRevealViewController

Create a basic application with single view application template with swift as language

Add `SWRevealViewController.h` and `SWRevealViewController.m`

then click on Create Bridging Header button

and add

```
#import "SWRevealViewController.h"
```

on the Bridging header

Then select viewController on storyboard and change class to `SWRevealViewController`

Then rename the viewController on files to MainViewController and add new ViewController with RightViewController name

接着我们从SWRevealViewController分别添加两个segue到MainViewController和RightViewController，然后需要选择第一个（从SWRevealViewController到MainViewController）并编辑属性

在标识符中设置 sw_front 在类中设置 SWRevealViewControllerSegueSetController

之后我们需要对segue（从SWRevealViewController到RightViewController）做同样操作

在标识符中设置 sw_rear 在类中设置 SWRevealViewControllerSegueSetController

then we add two segues from SWRevealViewController to MainViewController and from SWRevealViewController to RightViewController, then we need to select the first (from SWRevealViewController to MainViewController) and edit properties

on identifier set `sw_front` on Class set `SWRevealViewControllerSegueSetController`

after this we need to do the same with the segue (from SWRevealViewController to RightViewController)

on identifier set `sw_rear` on Class set `SWRevealViewControllerSegueSetController`

然后在 MainViewController 的 viewDidLoad 方法中添加这一行

```
self.view.addGestureRecognizer(self.revealViewController().panGestureRecognizer());
```

就是这样，你已经有了一个集成了 SWRevealViewController 的基础应用，你可以向右滑动以显示 RightViewController 作为侧边菜单

then on MainViewController add this line on `viewDidLoad` method

```
self.view.addGestureRecognizer(self.revealViewController().panGestureRecognizer());
```

And this is all, you have a basic app with SWRevealViewController integrated, you can swipe to right to show `RightViewController` as lateral menu

第124章：DispatchGroup

相关主题：

Grand Central Dispatch

并发

第124.1节：介绍

假设你有多个线程在运行。每个线程都在执行一个任务。你希望在所有任务线程完成时，主线程或另一个线程能够收到通知。

解决此类问题的最简单方法是使用DispatchGroup。

使用DispatchGroup时，对于每个请求，您需要enter该组，对于每个完成的请求，您需要leave该组。

当组内不再有请求时，您将会被notify（通知）。

用法：

```
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()


        let dispatchGroup = DispatchGroup() //为任务创建一个组。
        let session: URLSession = URLSession.shared

dispatchGroup.enter() //第一个任务进入组。

        let firstTask = session.dataTask(with: URLRequest(url: URL(string:
"https://stackoverflow.com")!)) { (data, response, error) in

            //处理响应..

dispatchGroup.leave() //第一个任务离开组。
        }


dispatchGroup.enter()  //进入第二个任务的组。

        let secondTask = session.dataTask(with: URLRequest(url: URL(string: "https://google.ca")!))
{ (data, response, error) in

            //处理响应..

dispatchGroup.leave()  //离开第二个任务的组。
        }


        //当上述所有任务全部完成时，在主线程/队列上收到通知。
dispatchGroup.notify(queue: DispatchQueue.main) {
```


Chapter 124: DispatchGroup

Related topics:

Grand Central Dispatch

Concurrency

Section 124.1: Introduction

Suppose you have multiple threads running. Each thread is doing one task. You want to get notified either on the mainThread OR another thread, when all the task-threads are completed.

The simplest solution to such a problem is a `DispatchGroup`.

When using a `DispatchGroup`, for each request, you `enter` the group and for each completed request, you `leave` the group.

When there are no longer requests in the group, you will be `notify` (notified).

Usage:

```
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()


        let dispatchGroup = DispatchGroup() //Create a group for the tasks.
        let session: URLSession = URLSession.shared

        dispatchGroup.enter() //Enter the group for the first task.

        let firstTask = session.dataTask(with: URLRequest(url: URL(string:
"https://stackoverflow.com")!)) { (data, response, error) in

            //Process Response..

            dispatchGroup.leave() //Leave the group for the first task.
        }


        dispatchGroup.enter()  //Enter the group for the second task.

        let secondTask = session.dataTask(with: URLRequest(url: URL(string: "https://google.ca")!))
{ (data, response, error) in

            //Process Response..

            dispatchGroup.leave()  //Leave the group for the second task.
        }


        //Get notified on the main thread/queue.. when ALL of the tasks above has been completed.
        dispatchGroup.notify(queue: DispatchQueue.main) {
```

```
            print("所有任务已完成")

        }


        //开始任务。
firstTask.resume()
        secondTask.resume()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }
}
```

通过上述方法，你不必无限期等待所有任务完成。你可以在所有任务开始之前显示加载器，并在所有任务完成后关闭加载器。这样，主线程不会被阻塞，代码也保持整洁。

现在假设你还想让任务有序执行，或者将它们的响应依次添加到数组中。你可以这样做：

```
import UIKit

//锁机制..
func synchronized(_ lock: AnyObject, closure: () -> Void) {
    objc_sync_enter(lock)
closure()
    objc_sync_exit(lock)
}

class ViewController: UIViewController {

    let lock = NSObject() //用于加锁的对象。
    var responseArray = Array<Data?>() //响应数组。

    override func viewDidLoad() {
        super.viewDidLoad()

        let dispatchGroup = DispatchGroup()
        let session: URLSession = URLSession.shared

dispatchGroup.enter() //第一个任务进入组。

        let firstTask = session.dataTask(with: URLRequest(url: URL(string:
"https://stackoverflow.com")!)) { (data, response, error) in

            //处理响应..

synchronized(self.lock, closure: { () -> Void in
                self.responseArray[0] = data ?? nil
            })

dispatchGroup.leave() //第一个任务离开组。
        }


dispatchGroup.enter()  //进入第二个任务的组。

        let secondTask = session.dataTask(with: URLRequest(url: URL(string: "https://google.ca")!))
{ (data, response, error) in
```



```
            print("Every task is complete")

        }


        //Start the tasks.
        firstTask.resume()
        secondTask.resume()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }
}
```

With the above, you don't have to `wait` infinitely until all the tasks are completed. You can display a loader BEFORE all the tasks have started and dismiss the loader AFTER all tasks are completed. This way, your main thread does not get blocked and your code remains clean.

Now suppose you also wanted the tasks to be `ordered` or add their responses to an array sequentially. You could do the following:

```
import UIKit

//Locking mechanism..
func synchronized(_ lock: AnyObject, closure: () -> Void) {
    objc_sync_enter(lock)
    closure()
    objc_sync_exit(lock)
}

class ViewController: UIViewController {

    let lock = NSObject() //Object to lock on.
    var responseArray = Array<Data?>() //Array of responses.

    override func viewDidLoad() {
        super.viewDidLoad()

        let dispatchGroup = DispatchGroup()
        let session: URLSession = URLSession.shared

        dispatchGroup.enter() //Enter the group for the first task.

        let firstTask = session.dataTask(with: URLRequest(url: URL(string:
"https://stackoverflow.com")!)) { (data, response, error) in

            //Process Response..

            synchronized(self.lock, closure: { () -> Void in
                self.responseArray[0] = data ?? nil
            })

            dispatchGroup.leave() //Leave the group for the first task.
        }


        dispatchGroup.enter()  //Enter the group for the second task.

        let secondTask = session.dataTask(with: URLRequest(url: URL(string: "https://google.ca")!))
{ (data, response, error) in
```

```
            //处理响应..

synchronized(self.lock, closure: { () -> Void in
                self.responseArray[1] = data ?? nil
            })

dispatchGroup.leave()  //离开第二个任务的组。
        }


        //在主线程上接收通知.. 当上述所有请求都完成时。
dispatchGroup.notify(queue: DispatchQueue.main) {

            print("所有任务已完成..")

            for i in 0..<self.responseArray.count {

                if self.responseArray[i] == nil {print("请
                    求 #\(i) 失败。")}

                else {
                    print("请求 #\(i) 成功。")}

            }
        }

        //向数组中添加了两个任务。响应在完成之前假定为 nil。

        self.responseArray.append(nil)

        //开始任务。
firstTask.resume()
        secondTask.resume()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }
}
```

注意事项

每个条目在 DispatchGroup 中必须有对应的退出。如果你在 enter 后忘记调用 leave，后果自负。任务完成时你将永远不会收到通知。

enter 的次数必须等于 leave 的次数。


```
        //Process Response..

        synchronized(self.lock, closure: { () -> Void in
            self.responseArray[1] = data ?? nil
        })

        dispatchGroup.leave()  //Leave the group for the second task.
    }


    //Get notified on the main thread.. when ALL of the requests above has been completed.
    dispatchGroup.notify(queue: DispatchQueue.main) {

        print("Every task is complete..")

        for i in 0..<self.responseArray.count {

            if self.responseArray[i] == nil {
                print("Request #\(i) Failed.\n")
            }
            else {
                print("Request #\(i) Succeeded.\n")
            }
        }
    }

    //Two tasks added to the array. Responses are assumed nil until they complete.
    self.responseArray.append(nil)
    self.responseArray.append(nil)

    //Start the tasks.
    firstTask.resume()
    secondTask.resume()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }
}
```

Notes

Every entry must have an exit in a `DispatchGroup`. If you forget to `leave` after `entering`, you are setting yourself up. You will NEVER be notified when the tasks are completed.

The amount of `enter` must equal the amount of `leave`.

第125章：GCD（Grand Central Dispatch）

Grand Central Dispatch (GCD) 是苹果针对多线程的解决方案。它是一个轻量级框架，用于在队列中同步或异步执行任务，并在后台为你管理 CPU 线程。

相关主题：并发

第125.1节：调度信号量

> DispatchSemaphore 提供了传统计数信号量的高效实现，可用于控制多个执行上下文对资源的访问。

使用信号量的场景之一是当你进行文件读写时，如果多个任务同时尝试读写文件，让每个任务按顺序等待可以提高性能，避免过度负担 I/O 控制器。

Swift 3

```
func do2TasksAtATime () {
    print("开始长时间运行的任务（每次2个）")let sem = DispatchSe
    maphore(value: 2)            //该信号量只允许同时运行2个任务（资源计数）

    for i in 0...7 {                                //启动一堆任务        DispatchQueue.globa
l().async {                //在后台线程运行任务            sem.wait()                                  //如果
没有可用资源，则在此等待            sleep(2)                                //执行一些长时间任务，例如文件
访问（这里为了演示，休眠2秒）

            print("长任务 \(i) 完成！\(Date())")
sem.signal()                            //通知信号量该资源现在可用

        }
    }
}
```

示例输出：（注意时间戳）

```
开始长时间运行的任务（每次2个）
长任务0完成！2017-02-16 07:11:53 +0000
长任务1完成！2017-02-16 07:11:53 +0000
长任务2完成！2017-02-16 07:11:55 +0000
长任务3完成！2017-02-16 07:11:55 +0000
长任务5完成！2017-02-16 07:11:57 +0000
长任务4完成！2017-02-16 07:11:57 +0000
长任务6完成！2017-02-16 07:11:59 +0000
长任务7完成！2017-02-16 07:11:59 +0000
```

更多信息，请参阅Apple Docs

第125.2节：Dispatch Group（调度组）

> DispatchGroup允许对工作进行整体同步。您可以使用它们提交多个
> 不同的工作项，并跟踪它们全部完成的时间，即使它们可能在不同的队列上运行。
> 当所有指定任务完成之前无法取得进展时，这种行为非常有用。

Chapter 125: GCD (Grand Central Dispatch)

Grand Central Dispatch (GCD) is Apple's answer to multithreading. It is a lightweight framework for performing tasks synchronously or asynchronously in queues and handles CPU threads for you behind the scenes.

Related Topic: Concurrency

Section 125.1: Dispatch Semaphore

> DispatchSemaphore provides an efficient implementation of a traditional counting semaphore, which can be used to control access to a resource across multiple execution contexts.

A scenario for when to use a semaphore could be if you are doing some file reading/writing, if multiple tasks are trying to read and write from file at the same time, it could increase your performance to make each task wait its turn so as to not overburden the I/O controller.

Swift 3

```
func do2TasksAtATime () {
    print("starting long running tasks (2 at a time)")
    let sem = DispatchSemaphore(value: 2)            //this semaphore only allows 2 tasks to run at
the same time (the resource count)
    for i in 0...7 {                                 //launch a bunch of tasks
        DispatchQueue.global().async {               //run tasks on a background thread
            sem.wait()                               //wait here if no resources available
            sleep(2)                                 //do some long task eg file access (here we
are just sleeping for a 2 seconds for demonstration purposes)
            print("long task \(i) done! \(Date())")
            sem.signal()                             //let the semaphore know this resource is now
available
        }
    }
}
```

Example output: (notice the time stamps)

```
starting long running tasks (2 at a time)
long task 0 done! 2017-02-16 07:11:53 +0000
long task 1 done! 2017-02-16 07:11:53 +0000
long task 2 done! 2017-02-16 07:11:55 +0000
long task 3 done! 2017-02-16 07:11:55 +0000
long task 5 done! 2017-02-16 07:11:57 +0000
long task 4 done! 2017-02-16 07:11:57 +0000
long task 6 done! 2017-02-16 07:11:59 +0000
long task 7 done! 2017-02-16 07:11:59 +0000
```

For more info, refer to the Apple Docs

Section 125.2: Dispatch Group

> DispatchGroup allows for aggregate synchronization of work. You can use them to submit multiple different work items and track when they all complete, even though they might run on different queues. This behavior can be helpful when progress can't be made until all of the specified tasks are complete.

一个适用场景是，如果您有多个网络服务调用都需要完成后才能继续。
例如，您需要下载多组数据，这些数据需要由某个函数进行处理。
您必须等待所有网络服务完成后，才能调用函数处理所有接收到的数据。

Swift 3

```
func doLongTasksAndWait () {
    print("starting long running tasks")
    let group = DispatchGroup()          //创建一个组，用于管理我们即将执行的一系列任务for i in 0...3 {
        //启动一系列任务（例如一堆需要完成后才能继续到下一个视图控制器的网络服务调用）          group.enter()
         //通知组有任务被添加          DispatchQueue.global().async {    //在后台线程运
行任务

sleep(arc4random() % 4)      //执行一些耗时任务，例如网络服务或数据库查询（这里仅为演示目的随机休眠一段时间）

            print("long task \(i) done!")
group.leave()                //通知组该任务已完成
        }
    }
group.wait()                         //阻塞当前线程，直到上述所有任务完成（因此建议不要在主线程使用此函数）

    print("all tasks done!")
}
```

或者，如果你不想等待组内任务全部完成，而是想在所有任务完成后执行某个函数，可以使用 notify 函数替代 group.wait()

```
group.notify(queue: DispatchQueue.main) { //queue: 参数指定该代码块在哪个队列执行，如果需要更新UI，请使用主队列

    print("all tasks done!")              //当所有任务都离开组时执行该代码块
}
```

示例输出：

```
开始长时间运行的任务
长任务 0 完成！
长任务 3 完成！
长任务 1 完成！
长任务 2 完成！
所有任务完成！
```

更多信息，请参阅Apple Docs或相关主题

第125.3节：获取主队列

主队列是执行所有UI更新的调度队列，涉及UI更改的代码都放在该队列中。

您需要获取主队列，以便在异步过程（如）完成时更新UI
`NSURLSession`

主要有两种类型的队列调用：同步和异步。当你同步调用某个操作时，意味着发起该操作的线程会等待任务完成后再继续。异步则表示不会等待。

Objective-C 代码

A Scenario when this could be useful is if you have multiple webservice calls that all need to finish before continuing. For example, you need to download multiple sets of data that needs to be processed by some function. You have to wait for all webservices to complete before calling the function to process all the received data.

Swift 3

```
func doLongTasksAndWait () {
    print("starting long running tasks")
    let group = DispatchGroup()          //create a group for a bunch of tasks we are about to do
    for i in 0...3 {                     //launch a bunch of tasks (eg a bunch of webservice calls
that all need to be finished before proceeding to the next ViewController)
        group.enter()                    //let the group know that something is being added
        DispatchQueue.global().async {   //run tasks on a background thread
            sleep(arc4random() % 4)      //do some long task eg webservice or database lookup (here
we are just sleeping for a random amount of time for demonstration purposes)
            print("long task \(i) done!")
            group.leave()                //let group know that the task is finished
        }
    }
    group.wait()                         //will block whatever thread we are on here until all the
above tasks have finished (so maybe don't use this function on your main thread)
    print("all tasks done!")
}
```

Alternatively, if you do not want to wait for the groups to finish, but instead want to run a function once all the tasks have completed, use the `notify` function in place of the `group.wait()`

```
group.notify(queue: DispatchQueue.main) { //the queue: parameter is which queue this block will run
on, if you need to do UI updates, use the main queue
    print("all tasks done!")              //this will execute when all tasks have left the group
}
```

Example output:

```
starting long running tasks
long task 0 done!
long task 3 done!
long task 1 done!
long task 2 done!
all tasks done!
```

For more info, refer to the Apple Docs or the related topic

Section 125.3: Getting the Main Queue

The main queue is the dispatch queue in which all the UI updates take place and the code involving UI changes are placed.

You need to get to the main queue in order to update UI on completion of an asynchronous process like
`NSURLSession`

There are two types of main queue calls `synchronous` and `asynchronous`. When you invoke something `synchronously`, it means that the thread that initiated that operation will wait for the task to finish before continuing. `Asynchronous` means that it will not wait.

Code Objective-C

同步主队列调用

```
dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
```

异步主队列调用

```
dispatch_async(dispatch_get_main_queue(), ^{
   // 通常在这里执行工作以更新用户界面
});
```

SWIFT 3

异步主队列调用

```
DispatchQueue.main.async {

}
```

同步主队列调用

```
DispatchQueue.main.sync {

}
```

第125.4节：创建一个调度队列

你可以使用 `dispatch_queue_create`

Objective-C 创建你自己的队列

```
dispatch_queue_t queue = dispatch_queue_create("com.example.myqueue",  DISPATCH_QUEUE_SERIAL);
```

Swift

```
// Swift 3 之前
let queue = dispatch_queue_create("com.example.myqueue", DISPATCH_QUEUE_SERIAL)
// Swift 3
let queue = DispatchQueue(label: "com.example.myqueue") //默认是串行队列，除非
.concurrent 被指定为属性
```

第125.5节：串行队列与并发队列

Swift 3

串行队列

```
func serialQueues () {
    let serialQueue = DispatchQueue(label: "com.example.serial") //默认队列类型是串行
队列
    let start = Date ()
    for i in 0...3 {                                        //启动一堆任务        serialQueue.async {
                                    //在后台线程上运行任务，使用我们的串行队列

sleep(2)                                                //执行一些耗时任务，例如网络服务
或数据库查询
            let timeTaken = Date().timeIntervalSince(start)
```


Synchronous Main Queue call

```
dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
```

Asynchronous Main Queue call

```
dispatch_async(dispatch_get_main_queue(), ^{
   // do work here to Usually to update the User Interface
});
```

SWIFT 3

Asynchronous Main Queue call

```
DispatchQueue.main.async {

}
```

Synchronous Main Queue call

```
DispatchQueue.main.sync {

}
```

Section 125.4: Create a dispatch queue

You can create your own queue using `dispatch_queue_create`

Objective-C

```
dispatch_queue_t queue = dispatch_queue_create("com.example.myqueue",  DISPATCH_QUEUE_SERIAL);
```

Swift

```
// Before Swift 3
let queue = dispatch_queue_create("com.example.myqueue", DISPATCH_QUEUE_SERIAL)
// Swift 3
let queue = DispatchQueue(label: "com.example.myqueue") //default is serial queue, unless
.concurrent is specified as an attribute otherwise
```

Section 125.5: Serial vs Concurrent Dispatch Queues

Swift 3

Serial Queue

```
func serialQueues () {
    let serialQueue = DispatchQueue(label: "com.example.serial") //default queue type is a serial
queue
    let start = Date ()
    for i in 0...3 {                                                    //launch a bunch of tasks
        serialQueue.async {                                             //run tasks on a background
thread, using our serial queue
            sleep(2)                                                    //do some long task eg webservice
or database lookup
            let timeTaken = Date().timeIntervalSince(start)
```

```
            print("串行耗时任务 \(i) 完成！总耗时: \(timeTaken)")
        }
    }
}
```

示例输出：

```
串行耗时任务 0 完成！总耗时: 2.07241100072861
串行耗时任务 1 完成！总耗时: 4.16347700357437
串行耗时任务 2 完成！总耗时: 6.23209798336029
串行耗时任务 3 完成！总耗时: 8.30682599544525
```

并发队列

```
func concurrentQueues () {
    let concurrentQueue = DispatchQueue(label: "com.example.concurrent", attributes: .concurrent)
//显式指定队列为并发队列
    let start = Date ()
    for i in 0...3 {            //启动一组任务
        concurrentQueue.async { //在后台线程上运行任务，使用我们的并发队列
            sleep(2)            //执行一些耗时任务，例如网络服务或数据库查询
            let timeTaken = Date().timeIntervalSince(start)
            print("并发耗时任务 \(i) 完成！总耗时: \(timeTaken)")
        }
    }
}
```

示例输出：

```
并发长任务 3 完成！总耗时：2.07092100381851
并发长任务 0 完成！总耗时：2.07087397575378
并发长任务 2 完成！总耗时：2.07086700201035
并发长任务 1 完成！总耗时：2.07089096307755
```

讨论

从上面的例子可以看出，串行队列会按照提交到队列的顺序完成每个任务。每个任务都会等待前一个任务完成后才执行。至于并发队列，每个任务不会等待队列中的其他任务，而是尽快执行；其优点是队列中的所有任务会在不同线程上同时运行，使得并发队列所需时间少于串行队列。

如果任务的执行顺序不重要，始终使用并发队列以获得最佳效率。


```
            print("serial long task \(i) done! total time taken: \(timeTaken)")
        }
    }
}
```

Example output:

```
serial long task 0 done! total time taken: 2.07241100072861
serial long task 1 done! total time taken: 4.16347700357437
serial long task 2 done! total time taken: 6.23209798336029
serial long task 3 done! total time taken: 8.30682599544525
```

Concurrent Queue

```
func concurrentQueues () {
    let concurrentQueue = DispatchQueue(label: "com.example.concurrent", attributes: .concurrent)
//explicitly specify the queue to be a concurrent queue
    let start = Date ()
    for i in 0...3 {            //launch a bunch of tasks
        concurrentQueue.async { //run tasks on a background thread, using our concurrent queue
            sleep(2)            //do some long task eg webservice or database lookup
            let timeTaken = Date().timeIntervalSince(start)
            print("concurrent long task \(i) done! total time taken: \(timeTaken)")
        }
    }
}
```

Example output:

```
concurrent long task 3 done! total time taken: 2.07092100381851
concurrent long task 0 done! total time taken: 2.07087397575378
concurrent long task 2 done! total time taken: 2.07086700201035
concurrent long task 1 done! total time taken: 2.07089096307755
```

Discussion

As we can see from the examples above, a serial queue will complete each task in the order they are submitted to the queue. Each task will wait for the previous task to finish before executing. As for the concurrent queue, each task does not wait on the others in the queue and executes as soon as possible; the advantage is that all tasks on the queue will run at the same time on separate threads, making a concurrent queue take less time than a serial queue.

If order of execution of tasks is not important, always use a concurrent queue for the best efficiency.

第126章：尺寸类别与适应性

第126.1节：特征集合

在iOS应用中，用户界面可以呈现几种不同的一般形状和尺寸。这些通过尺寸类别定义，可以通过视图或视图控制器的特征集合获得。

苹果定义了两种尺寸类别：常规和紧凑。这两种尺寸类别在设备的两个轴线上（水平和垂直）均可用。您的应用在其生命周期中可能处于这四种状态中的任何一种。作为简写，开发者通常通过先写水平轴的尺寸类别，再写垂直轴的尺寸类别来描述尺寸类别组合："紧凑/常规"描述的是水平紧凑但垂直常规的界面。

在您的应用中，使用UITraitEnvironment协议上的方法检查当前的尺寸类别并响应变化：

```
class MyViewController: UIViewController {
    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        print("水平尺寸类别: \(traitCollection.horizontalSizeClass)")
        print("垂直尺寸类别: \(traitCollection.verticalSizeClass)")
    }

    override func traitCollectionDidChange(_ previousTraitCollection: UITraitCollection?) {
        super.traitCollectionDidChange(previousTraitCollection)
        print("特征集合已更改；尺寸类别可能不同。")
    }
}
```

UIView 和 UIViewController 都遵循 UITraitEnvironment，因此你可以查看当前的特征集合并在任一子类中处理更改。

第126.2节：使用特征集合更改更新自动布局

使应用程序自适应——即通过响应尺寸类别的变化来改变布局——通常需要大量自动布局系统的帮助。应用程序变得自适应的主要方式之一是当视图的尺寸类别发生变化时，更新活动的自动布局约束。

例如，考虑一个使用 UIStackView 来排列两个 UILabel 的应用程序。我们可能希望在水平紧凑环境中，这些标签垂直堆叠，而在水平常规环境中有更多空间时，它们并排放置。

```
class ViewController: UIViewController {
    var stackView: UIStackView!

    override func viewDidLoad() {
        super.viewDidLoad()

        stackView = UIStackView()
        for text in ["foo", "bar"] {
            let label = UILabel()
label.translatesAutoresizingMaskIntoConstraints = false
            label.text = text
stackView.addArrangedSubview(label)
        }
```


Chapter 126: Size Classes and Adaptivity

Section 126.1: Trait Collections

In an iOS app, your user interface can take on one of a few different general shapes and sizes. These are defined using **size classes**, which are available through a view or view controller's **trait collection**.

Apple defines two size classes: **regular** and **compact**. Each of these size classes are available on both axes of the device (**horizontal** and **vertical**). Your app may exist in any these four states throughout its lifetime. As a shorthand, developers often describe a size class combination by saying or writing the two size classes, with the horizontal axis first: "Compact/Regular" describes an interface that is horizontally compact but vertically regular.

In your app, use methods on the UITraitEnvironment protocol to check your current size class and respond to changes:

```
class MyViewController: UIViewController {
    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        print("Horizontal size class: \(traitCollection.horizontalSizeClass)")
        print("Vertical size class: \(traitCollection.verticalSizeClass)")
    }

    override func traitCollectionDidChange(_ previousTraitCollection: UITraitCollection?) {
        super.traitCollectionDidChange(previousTraitCollection)
        print("Trait collection changed; size classes may be different.")
    }
}
```

Both UIView and UIViewController conform to UITraitEnvironment, so you can look at your current trait collection and handle changes in subclasses of either.

Section 126.2: Updating Auto Layout with Trait Collection Changes

Making an app **adaptive** – that is, responding to size class changes by changing your layout – often involves lots of help from the Auto Layout system. One of the primary ways apps become adaptive is by updating the active Auto Layout constraints when a view's size class changes.

For example, consider an app that uses a UIStackView to arrange two UILabels. We might want these labels to stack on top of each other in horizontally compact environments, but sit next to each other when we have a little more room in horizontally regular environments.

```
class ViewController: UIViewController {
    var stackView: UIStackView!

    override func viewDidLoad() {
        super.viewDidLoad()

        stackView = UIStackView()
        for text in ["foo", "bar"] {
            let label = UILabel()
            label.translatesAutoresizingMaskIntoConstraints = false
            label.text = text
            stackView.addArrangedSubview(label)
        }
```

```
view.addSubview(stackView)
stackView.translatesAutoresizingMaskIntoConstraints = false
        stackView.centerXAnchor.constraint(equalTo: view.centerXAnchor).isActive = true
        stackView.centerYAnchor.constraint(equalTo: view.centerYAnchor).isActive = true
    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
updateAxis(forTraitCollection: traitCollection)
    }

    override func traitCollectionDidChange(_ previousTraitCollection: UITraitCollection?) {
        super.traitCollectionDidChange(previousTraitCollection)
updateAxis(forTraitCollection: traitCollection)
    }

    private func updateAxis(forTraitCollection traitCollection: UITraitCollection) {
        switch traitCollection.horizontalSizeClass {
        case .regular:
stackView.axis = .水平
        case .紧凑:
stackView.axis = .vertical
        case .unspecified:
            print("未指定的尺寸类别！")
          stackView.axis = .horizontal
        }
    }
}
```

第126.3节：支持iPad上的iOS多任务处理

现代iOS应用适应性的关键部分是支持iPad上的多任务处理。默认情况下，Xcode 7及更高版本创建的应用将配置为支持多任务处理：它们将拥有一个使用自动布局（Auto Layout）的LaunchScreen.storyboard文件。

现有应用启用多任务处理的最简单方法是创建这样的storyboard，然后将其设置为项目的启动屏幕（Launch Screen）：


```
        view.addSubview(stackView)
        stackView.translatesAutoresizingMaskIntoConstraints = false
        stackView.centerXAnchor.constraint(equalTo: view.centerXAnchor).isActive = true
        stackView.centerYAnchor.constraint(equalTo: view.centerYAnchor).isActive = true
    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        updateAxis(forTraitCollection: traitCollection)
    }

    override func traitCollectionDidChange(_ previousTraitCollection: UITraitCollection?) {
        super.traitCollectionDidChange(previousTraitCollection)
        updateAxis(forTraitCollection: traitCollection)
    }

    private func updateAxis(forTraitCollection traitCollection: UITraitCollection) {
        switch traitCollection.horizontalSizeClass {
        case .regular:
            stackView.axis = .horizontal
        case .compact:
            stackView.axis = .vertical
        case .unspecified:
            print("Unspecified size class!")
            stackView.axis = .horizontal
        }
    }
}
```

Section 126.3: Supporting iOS Multitasking on iPad

A key piece of adaptivity in a modern iOS app is supporting multitasking on iPad. By default, apps created in Xcode 7 and newer will be configured to support multitasking: they'll have a LaunchScreen.storyboard file that uses Auto Layout.

The easiest way for existing apps to opt in to multitasking is to create such a storyboard, then set it as the project's Launch Screen:

一旦您的应用支持iPad多任务处理，请审核现有的视图和视图控制器，确保它们使用自动布局并能支持各种尺寸类别组合。

Once your app supports iPad multitasking, audit existing views and view controllers to make sure that they use Auto Layout and can support a variety of size class combinations.

第127章：IBOutlets

第127.1节：在UI元素中使用IBOutlet

一般来说，IBOutlet 用于将用户界面对象连接到另一个对象，在本例中是 UIViewController。该连接用于允许通过代码或事件以编程方式影响该对象。这可以通过使用故事板的助手，按住 Control 键从元素拖动到视图控制器的 .h 属性部分来简单完成，但也可以通过编程方式手动将 IBOutlet 代码连接到右侧实用工具栏中对象的“connections”标签来完成。以下是一个带有标签 outlet 的 UIViewController 的 Objective-C 示例：

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

@interface ViewController : UIViewController

//这是 outlet 的声明
@property (nonatomic, weak) IBOutlet UILabel *myLabel;

@end

//ViewController.m
#import "ViewController.h"

@implementation ViewController

@synthesize myLabel;

-(void) viewDidLoad {

    [super viewDidLoad];
    //编辑 outlet 的属性
   myLabel.text = @"TextHere";

}

@end
```

Swift 版本：

```
import UIKit
class ViewController: UIViewController {
    //这是 outlet 的声明
    @IBOutlet weak var myLabel: UILabel!


    override func viewDidLoad() {
        super.viewDidLoad()
        //编辑出口的属性
       myLabel.text = "TextHere"
    }
}
```

如果.h文件中outlet声明左侧的点被填充，则可以验证storyboard对象与编程对象之间已连接。空心圆表示连接不完整。

Chapter 127: IBOutlets

Section 127.1: Using an IBOutlet in a UI Element

In general, IBOutlets are used to connect an user interface object to another object, in this case a UIViewController. The connection serves to allow for the object to be affected my code or events programmatically. This can be done simply by using the assistant from a storyboard and control-clicking from the element to the view controller's .h property section, but it can also be done programmatically and manually connecting the IBOutlet code to the "connections" tab of the object the utility bar on the right. Here is an objective-c example of a UIViewController with a label outlet:

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

@interface ViewController : UIViewController

//This is the declaration of the outlet
@property (nonatomic, weak) IBOutlet UILabel *myLabel;

@end

//ViewController.m
#import "ViewController.h"

@implementation ViewController

@synthesize myLabel;

-(void) viewDidLoad {

    [super viewDidLoad];
    //Editing the properties of the outlet
    myLabel.text = @"TextHere";

}

@end
```

And swift:

```
import UIKit
class ViewController: UIViewController {
    //This is the declaration of the outlet
    @IBOutlet weak var myLabel: UILabel!


    override func viewDidLoad() {
        super.viewDidLoad()
        //Editing the properties of the outlet
        myLabel.text = "TextHere"
    }
}
```

The connection between the storyboard object, and the programmed object can be verified as connected if the dot to the left of the declaration of the outlet in the .h is filled. An empty circle implied an incomplete connection.

第128章：AWS SDK

第128.1节：使用AWS SDK上传图片或视频到S3

在开始示例之前，我建议创建一个带有代理类成员的单例，这样你可以实现后台上传文件的用例，让用户在文件上传时继续使用你的应用，即使应用处于后台。

首先，我们应该创建一个表示S3配置的枚举：

```
enum S3Configuration : String
{
    case IDENTITY_POOL_ID   = "YourIdentityPoolId"
    case BUCKET_NAME        = "YourBucketName"
    case CALLBACK_KEY       = "YourCustomStringForCallBackWhenUploadingInTheBackground"
    case CONTENT_TYPE_IMAGE = "image/png"
    case CONTENT_TYPE_VIDEO = "video/mp4"
}
```

现在，我们应该在应用首次启动时设置凭证，因此，我们应将它们设置在
在 AppDelegate 的 didFinishLaunchingWithOptions 方法中（注意你应该在 regionType 参数中设置你的区域）：

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject: AnyObject]?) -> Bool
{
   let credentialProvider = AWSCognitoCredentialsProvider(regionType: .EUWest1, identityPoolId:
S3Configuration.IDENTITY_POOL_ID.rawValue)
   let configuration = AWSServiceConfiguration(region: .EUWest1, credentialsProvider:
credentialProvider)
AWSS3TransferUtility.registerS3TransferUtilityWithConfiguration(configuration, forKey:
S3Configuration.CALLBACK_KEY.rawValue)
}
```

由于我们已经在 AppDelegate 中，我们应该实现由 AWS SDK 处理的后台回调：

```
func application(application: UIApplication, handleEventsForBackgroundURLSession identifier:
String, completionHandler: () -> Void)
{
    // 将打印您在枚举中设置的标识符：.CALLBACK_KEY
    print("标识符: " + identifier)
    // 存储完成处理程序。
AWSS3TransferUtility.interceptApplication(application,
                                          handleEventsForBackgroundURLSession: identifier,
                                          completionHandler: completionHandler)
}
```

现在，当用户将应用移到后台时，您的上传将继续实际上传。

为了使用AWS SDK上传文件，我们必须将文件写入设备并提供给SDK实际的
路径。举例来说，假设我们有一个UIImage（也可以是视频..），我们将其写入一个临时
文件夹：

```
// 一些图片....
```


Chapter 128: AWS SDK

Section 128.1: Upload an image or a video to S3 using AWS SDK

Before starting with the example I'd recommend to create a Singleton with a delegate class member so you could achieve a use case of uploading a file in the background and let the user keep using your app while the files are being uploaded even when the app is the background.

Let's start, first, we should create an enum that represent the S3 configuration:

```
enum S3Configuration : String
{
    case IDENTITY_POOL_ID   = "YourIdentityPoolId"
    case BUCKET_NAME        = "YourBucketName"
    case CALLBACK_KEY       = "YourCustomStringForCallBackWhenUploadingInTheBackground"
    case CONTENT_TYPE_IMAGE = "image/png"
    case CONTENT_TYPE_VIDEO = "video/mp4"
}
```

Now, we should set the credentials when your app launch for the first time, thus, we should set them inside the `AppDelegate` at the `didFinishLaunchingWithOptions` method (pay attention that you should set your region at the `regionType` param):

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject: AnyObject]?) -> Bool
{
  let credentialProvider = AWSCognitoCredentialsProvider(regionType: .EUWest1, identityPoolId:
S3Configuration.IDENTITY_POOL_ID.rawValue)
  let configuration = AWSServiceConfiguration(region: .EUWest1, credentialsProvider:
credentialProvider)
  AWSS3TransferUtility.registerS3TransferUtilityWithConfiguration(configuration, forKey:
S3Configuration.CALLBACK_KEY.rawValue)
}
```

Since we are already inside the AppDelegate, we should implement the background callback that is handled by the AWS SDK:

```
func application(application: UIApplication, handleEventsForBackgroundURLSession identifier:
String, completionHandler: () -> Void)
{
    //  Will print the identifer you have set at the enum: .CALLBACK_KEY
    print("Identifier: " + identifier)
    //  Stores the completion handler.
    AWSS3TransferUtility.interceptApplication(application,
                                              handleEventsForBackgroundURLSession: identifier,
                                              completionHandler: completionHandler)
}
```

Now, when the user will move the app to the background your upload will continue the actual upload.

In order to upload the file using the AWS SDK we will have to write the file to the device and give the SDK the actual path. For the sake of the example, imagine we have a UIImage (could be a video also..) and we will write it to a temp folder:

```
// Some image....
```

```
let image = UIImage()
let fileURL = NSURL(fileURLWithPath: NSTemporaryDirectory()).URLByAppendingPathComponent(fileName)
let filePath = fileURL.path!
let imageData = UIImageJPEGRepresentation(image, 1.0)
imageData!.writeToFile(filePath, atomically: true)
```

FileURL 和 fileName 将用于后续的实际上传。

我们需要定义两个由 AWS SDK 提供的闭包，

1. AWSS3TransferUtilityUploadCompletionHandlerBlock - 一个在上传完成时（或未完成时）通知的闭包 not)
2. AWSS3TransferUtilityUploadProgressBlock - 一个通知每个已发送字节的闭包

如果你计划使用单例模式，应该将这些类型定义为类成员。实现应如下所示：

```
var completionHandler : AWSS3TransferUtilityUploadCompletionHandlerBlock? =
    { (task, error) -> Void in

        if ((error) != nil)
        {
          print("上传失败")
        }
        else
        {
          print("文件上传成功")
        }
    }

var progressBlock : AWSS3TransferUtilityUploadProgressBlock? =
    { [unowned self] (task, bytesSent:Int64, totalBytesSent:Int64,  totalBytesExpectedToSend:Int64)
-> Void in

     let progressInPercentage = Float(Double(totalBytesSent) / Double(totalBytesExpectedToSend)) *
100
     print(progressInPercentage)
    }
```

注意： 如果您使用的是单例模式，您可能需要定义一个代理，用于报告进度或文件完成情况。
如果您不使用单例模式，可以创建一个静态方法，包含相关类型：

```
    static func uploadImageToS3(fileURL : NSURL,
                                fileName : String,
                progressFunctionUpdater : Float -> Void,
                            resultBlock : (NSError?) -> Void)
{
 //    实际实现 .....
 //     ...
 //     ...
}
```

1. progressFunctionUpdater - 将通过函数回调报告进度。
2. resultBlock - 如果返回nil，则上传成功，否则返回错误对象。

女士们，先生们，实际上传开始：

```
        let fileData = NSData(contentsOfFile: fileURL.relativePath!)
```



```
let image = UIImage()
let fileURL = NSURL(fileURLWithPath: NSTemporaryDirectory()).URLByAppendingPathComponent(fileName)
let filePath = fileURL.path!
let imageData = UIImageJPEGRepresentation(image, 1.0)
imageData!.writeToFile(filePath, atomically: true)
```

FileURL and fileName will be used for the actual uploading later.

There are 2 closures we will have to define that are provided by the AWS SDK,

1. `AWSS3TransferUtilityUploadCompletionHandlerBlock` - A closure that notifies when the upload is done (or not)
2. `AWSS3TransferUtilityUploadProgressBlock` - A closure that notifies each byte sent

If you plan to have a Singleton you should define those types as class members. The implementation should look like this:

```
var completionHandler : AWSS3TransferUtilityUploadCompletionHandlerBlock? =
    { (task, error) -> Void in

        if ((error) != nil)
        {
          print("Upload failed")
        }
        else
        {
          print("File uploaded successfully")
        }
    }

var progressBlock : AWSS3TransferUtilityUploadProgressBlock? =
    { [unowned self] (task, bytesSent:Int64, totalBytesSent:Int64,  totalBytesExpectedToSend:Int64)
-> Void in

     let progressInPercentage = Float(Double(totalBytesSent) / Double(totalBytesExpectedToSend)) *
100
     print(progressInPercentage)
    }
```

NOTE: If you are using a Singleton you might want to define a delegate that will report back with progress or when the file is done. If you are not using a Singleton you can create a static method that would have the relevant types:

```
    static func uploadImageToS3(fileURL : NSURL,
                                fileName : String,
                progressFunctionUpdater : Float -> Void,
                            resultBlock : (NSError?) -> Void)
{
 //    Actual implementation .....
 //     ...
 //     ...
}
```

1. `progressFunctionUpdater` - will report back to a function with progress.
2. `resultBlock` - If you return nil then upload was successfully else, you send the error object

Ladies and gentlemen, the actual upload:

```
        let fileData = NSData(contentsOfFile: fileURL.relativePath!)
```

```
        let expression = AWSS3TransferUtilityUploadExpression()
        expression.uploadProgress = progressBlock

        let transferUtility =
AWSS3TransferUtility.S3TransferUtilityForKey(S3Configuration.CALLBACK_KEY.rawValue)

        transferUtility?.uploadData(fileData!,
bucket: S3Configuration.BUCKET_NAME.rawValue,
            key: fileName,
contentType: S3Configuration.CONTENT_TYPE_IMAGE.rawData,
            expression: expression,
completionHander: completionHandler).continueWithBlock
             { (task : AWSTask) -> AnyObject? in

                 if let error = task.error
                 {
                     print(error)
                 }
                 if let exception = task.exception
                 {
                     print("异常: " + exception.description)
                 }
                 if let uploadTask = task.result as? AWSS3TransferUtilityUploadTask
                 {
                     print("上传开始...")
                 }

                 return nil
        }
```

愉快的 S3 上传 :)


```
        let expression = AWSS3TransferUtilityUploadExpression()
        expression.uploadProgress = progressBlock

        let transferUtility =
AWSS3TransferUtility.S3TransferUtilityForKey(S3Configuration.CALLBACK_KEY.rawValue)

        transferUtility?.uploadData(fileData!,
            bucket: S3Configuration.BUCKET_NAME.rawValue,
            key: fileName,
            contentType: S3Configuration.CONTENT_TYPE_IMAGE.rawData,
            expression: expression,
            completionHander: completionHandler).continueWithBlock
            { (task : AWSTask) -> AnyObject? in

                if let error = task.error
                {
                    print(error)
                }
                if let exception = task.exception
                {
                    print("Exception: " + exception.description)
                }
                if let uploadTask = task.result as? AWSS3TransferUtilityUploadTask
                {
                    print("Upload started...")
                }

                return nil
        }
```

Happy S3 uploading :)

第129章：调试崩溃

第129.1节：调试 EXC_BAD_ACCESS

EXC_BAD_ACCESS 意味着进程尝试以无效方式访问内存，比如解引用一个 NULL 指针或写入只读内存。这是最难调试的崩溃类型，因为它通常没有错误信息，有些崩溃可能非常难以重现和/或发生在与问题完全无关的代码中。这个错误在 Swift 中非常罕见，但如果发生，通常可以通过降低编译器优化来获得更易调试的崩溃。

大多数 EXC_BAD_ACCESS 错误是由于尝试解引用一个 NULL 指针引起的。如果是这种情况，红色箭头中列出的地址通常是一个低于正常内存地址的十六进制数字，通常是 0x0。在调试器中设置断点或偶尔添加printf/NSLog语句，以找出该指针为何为 NULL。

不太稳定发生或完全没有意义的EXC_BAD_ACCESS可能是内存管理问题导致的。常见导致此问题的原因有：

- 使用已被释放的内存
- 尝试写入超出C数组或其他类型缓冲区末尾的内容
- 使用未初始化的指针

在方案编辑器的诊断部分，Xcode包含一些有用的工具来帮助调试内存问题：

地址消毒器（Address Sanitizer）增加了许多检查，一旦发生内存问题就会停止应用并提供详细说明具体发生了什么的错误信息。僵尸对象（Zombie Objects）检测已释放的Objective-C对象的问题，但开启自动引用计数（Automatic Reference Counting）后通常不会出现这类问题。

Chapter 129: Debugging Crashes

Section 129.1: Debugging EXC_BAD_ACCESS

EXC_BAD_ACCESS means the process tried to access memory in an invalid way, like dereferencing a NULL pointer or writing to read-only memory. This is the hardest kind of crash to debug, because it usually does not have an error message, and some crashes can be *very* difficult to reproduce and/or occur in code completely unrelated to the problem. This error is very rare in Swift, but if it occurs, you can often get easier-to-debug crashes by reducing compiler optimizations.

Most EXC_BAD_ACCESS errors are caused by trying to dereference a NULL pointer. If this is the case, the address listed in the red arrow will usually be a hexadecimal number that is lower than a normal memory address, often 0x0. Set breakpoints in the debugger or add occasional printf/NSLog statements to find out why that pointer is NULL.

An EXC_BAD_ACCESS that occurs less reliably or makes no sense at all could be the result of a memory management problem. Common problems that can cause this are:

- Using memory that has been deallocated
- Trying to write past the end of a C array or other kind of buffer
- Using a pointer which has not been initialized

In the Diagnostics section of the Scheme Editor, Xcode includes a few useful tools to help debug memory problems:

The Address Sanitizer adds lots of checks that will stop the app whenever memory problems occur and provide a helpful error message detailing exactly what happened. Zombie Objects detects problems with deallocated Objective-C objects, but you shouldn't get these kinds of problems with Automatic Reference Counting turned on.

第129.2节：查找崩溃信息

当您的应用崩溃时，Xcode会进入调试器并显示有关崩溃的更多信息：

最重要的部分是：

红色箭头

红色箭头显示了哪一行代码崩溃以及崩溃的原因。

调试器控制台

许多崩溃会将更多信息记录到调试器控制台。应用程序崩溃时，调试器应自动出现，

但如果没有出现，可以通过选择来显示调试器 在Xcode右上角的按钮，显示

通过点击控制台 调试器右下角的按钮。

堆栈跟踪

堆栈跟踪列出了程序在到达崩溃代码之前调用过的函数。

部分堆栈跟踪显示在屏幕左侧的调试导航器中，调试器控件允许您选择一个堆栈帧以在调试器中查看：

Section 129.2: Finding information about a crash

When your app crashes, Xcode will enter the debugger and show you more information about the crash:

The most important parts are:

The red arrow

The red arrow displays which line of code crashed & why it crashed.

The debugger console

Many crashes log more information to the debugger console. It should automatically appear when the app crashes, but if it's not there, show the debugger by selecting the button in the top-right corner of Xcode, and show the console by clicking the button in the bottom-right corner of the debugger.

The stack trace

The stack trace lists the functions the program came from before it got to the code that crashed.

Part of the stack trace is displayed in the Debug Navigator on the left of the screen, and the debugger controls allow you to select a stack frame to view in the debugger:

如果你在调试器的 (lldb) 提示符下输入 bt 命令并按下 返回 ，你将获得一个堆栈跟踪的文本表示，可以复制粘贴：

```
(lldb) bt
* 线程 #1: 线程ID = 0x3aaec5, 0x00007fff91055f06 libsystem_kernel.dylib`__pthread_kill + 10, 队列 =
'com.apple.main-thread', 停止原因 = 信号 SIGABRT
帧 #0: 0x00007fff91055f06 libsystem_kernel.dylib`__pthread_kill + 10
    帧 #1: 0x000000010008142d libsystem_pthread.dylib`pthread_kill + 90
    帧 #2: 0x00007fff96dc76e7 libsystem_c.dylib`abort + 129
帧 #3: 0x00007fff8973bf81 libc++abi.dylib`abort_message + 257
    帧 #4: 0x00007fff89761a47 libc++abi.dylib`default_terminate_handler() + 267
    帧 #5: 0x00007fff94f636ae libobjc.A.dylib`_objc_terminate() + 103
帧 #6: 0x00007fff8975f19e libc++abi.dylib`std::__terminate(void (*)()) + 8
    帧 #7: 0x00007fff8975ec12 libc++abi.dylib`__cxa_throw + 121
帧 #8: 0x00007fff94f6108c libobjc.A.dylib`objc_exception_throw + 318
    帧 #9: 0x00007fff8d067372 CoreFoundation`-[__NSPlaceholderArray initWithObjects:count:] +
290
帧 #        10: 0x00007fff8d0eaa1f CoreFoundation`+[NSArray arrayWithObject:] + 47
  * frame #11: 0x0000000100001b54 test`main(argc=1, argv=0x00007fff5fbff808) + 68 at main.m:15
帧 #12: 0x00007fff8bea05ad libdyld.dylib`start + 1
    帧 #13: 0x00007fff8bea05ad libdyld.dylib`start + 1
```

第129.3节：调试 SIGABRT 和 EXC_BAD_INSTRUCTION 崩溃

一个SIGABRT或EXC_BAD_INSTRUCTION通常意味着应用程序因某些检查失败而故意崩溃。它们应该会在调试器控制台记录一条包含更多信息的消息；请在那里查看更多信息。

许多SIGABRT是由未捕获的Objective-C异常引起的。异常可能被抛出的原因有很多，且它们总是会在控制台记录大量有用的信息。

If you enter the bt command at the (lldb) prompt in the debugger and press return , you will get a textual representation of the stack trace that you can copy and paste:

```
(lldb) bt
* thread #1: tid = 0x3aaec5, 0x00007fff91055f06 libsystem_kernel.dylib`__pthread_kill + 10, queue =
'com.apple.main-thread', stop reason = signal SIGABRT
    frame #0: 0x00007fff91055f06 libsystem_kernel.dylib`__pthread_kill + 10
    frame #1: 0x000000010008142d libsystem_pthread.dylib`pthread_kill + 90
    frame #2: 0x00007fff96dc76e7 libsystem_c.dylib`abort + 129
    frame #3: 0x00007fff8973bf81 libc++abi.dylib`abort_message + 257
    frame #4: 0x00007fff89761a47 libc++abi.dylib`default_terminate_handler() + 267
    frame #5: 0x00007fff94f636ae libobjc.A.dylib`_objc_terminate() + 103
    frame #6: 0x00007fff8975f19e libc++abi.dylib`std::__terminate(void (*)()) + 8
    frame #7: 0x00007fff8975ec12 libc++abi.dylib`__cxa_throw + 121
    frame #8: 0x00007fff94f6108c libobjc.A.dylib`objc_exception_throw + 318
    frame #9: 0x00007fff8d067372 CoreFoundation`-[__NSPlaceholderArray initWithObjects:count:] +
290
    frame #10: 0x00007fff8d0eaa1f CoreFoundation`+[NSArray arrayWithObject:] + 47
  * frame #11: 0x0000000100001b54 test`main(argc=1, argv=0x00007fff5fbff808) + 68 at main.m:15
    frame #12: 0x00007fff8bea05ad libdyld.dylib`start + 1
    frame #13: 0x00007fff8bea05ad libdyld.dylib`start + 1
```

Section 129.3: Debugging SIGABRT and EXC_BAD_INSTRUCTION crashes

A SIGABRT or an EXC_BAD_INSTRUCTION usually means the app crashed itself intentionally because some check failed. These should log a message to the debugger console with more information; check there for more information.

Many SIGABRTs are caused by uncaught Objective-C exceptions. There are a *lot* of reasons exceptions can be thrown, and they will *always* log a lot of helpful information to the console.

- `NSInvalidArgumentException`，表示应用程序向方法传递了无效参数
- `NSRangeException`，表示应用程序试图访问对象（如NSArray或NSString）的越界索引
- NSInternalInconsistencyException表示对象发现自身处于意外状态。
- `NSUnknownKeyException`通常表示你在XIB中有错误的连接。尝试一些对这个问题的解答。

- `NSInvalidArgumentException`, which means the app passed an invalid argument to a method
- `NSRangeException`, which means the app tried to access an out-of-bounds index of an object such as an `NSArray` or an `NSString`
- `NSInternalInconsistencyException` means an object discovered it was in an unexpected state.
- `NSUnknownKeyException` usually means you have a bad connection in an XIB. Try some of the answers to this question.

第130章：CloudKit

第130.1节：注册应用以使用CloudKit

您需要获取一个权限文件，以便应用能够访问您的iCloud并使用CloudKit写入记录。

按照以下步骤授予应用访问iCloud的权限：

1- 在项目导航器中选择项目，然后打开“常规”标签页。

2- 在“身份”部分，将您的开发者Apple ID设置到“团队”下拉菜单中。（如果不可用，请在Xcode菜单 -> 偏好设置 -> 账户中添加。）

3- 进入项目属性中的“功能”标签页，开启iCloud。然后选择“键值存储”和“CloudKit”。

4- 确保这些项目已被勾选：

如果所有项目都已勾选，则您的应用已准备好使用CloudKit。

第130.2节：使用CloudKit仪表盘

所有使用CloudKit相关代码创建的记录都可以在CloudKit仪表盘中预览、编辑甚至删除。要访问CloudKit仪表盘，请前往 here 。

仪表盘中有几个部分：

- 记录类型（稍后将讨论）
- 安全角色（您可以在这里设置数据库为公开或私有）订阅类型（您的应用可
- 以注册苹果推送通知（APNs），以便在记录更改时通知您）

记录类型

在这里，您可以看到应用中所有现有的记录类型列表。当您第一次打开某个应用的CloudKit仪表板时，那里有一个名为Users的记录类型，您可以使用它，也可以删除它并使用您自己的记录类型。

在此页面，您可以手动设置数据类型。当然，在大多数情况下这没有意义，因为iOS SDK

Chapter 130: CloudKit

Section 130.1: Registering app for use with CloudKit

What you need is to get an entitlements file so the app can access your iCloud and write records using CloudKit.

Follow the steps to grant access to iCloud from your app:

1- Select the project in the Project Navigator, and then open the General tab.

2- In the Identity section, set your developer Apple ID to the Team dropdown menu. (If it is not available, add it in Xcode menu -> Preferences -> Accounts.

3- Go to Capabilities tab in the project properties and turn iCloud on. Then, select "Key-Value Storage" and "CloudKit".

4- Make sure these items are checked:

If all of the items are checked, then your app is ready to use CloudKit.

Section 130.2: Using CloudKit Dashboard

All the records create using CloudKit-related code can be previewed, edited and even removed in CloudKit Dashboard. To access CloudKit Dashboard, go here.

There are several parts in the dashboard:

- Record Types (which will be discussed later)
- Security Roles (which is where you can set databases as public or private)
- Subscription Types (which your app could register for Apple Push Notifications (APNs) to notify you when a record is changed)

Record Types

Here, you get a list of all the existing record types in the app. When you first open CloudKit Dashboard for an app, there's a record type called Users there, which you can use it or just delete it and use your own.

In this page, you could make your data typed manually. Of course, in most cases this is pointless, because iOS SDK

比仪表板处理得更好，但如果您愿意，这个功能也存在。此页面最常用的功能是预览类型。

第130.3节：将数据保存到CloudKit

要将数据保存到CloudKit，我们必须创建：

- 一个`CKRecordID`（您唯一记录的键）
- 一个`CKRecord`（包含数据）

制作记录键

为了确保每个新记录标识符都是唯一的，我们使用当前的时间戳，它是唯一的。我们使用NSDate的方法timeIntervalSinceReferenceDate()获取时间戳。它的形式是###.###（#代表数字），我们将使用整数部分。为此，我们将字符串拆分：

Swift

```
let timestamp = String(format: "%f", NSDate.timeIntervalSinceReferenceDate())
let timestampParts = timestamp.componentsSeparatedByString(".")
let recordID = CKRecordID(recordName: timestampParts[0])
```

制作记录

制作记录时，我们应指定记录类型（在使用CloudKit仪表盘中有说明）为Users，ID为刚才制作的内容，以及数据。这里，我们将向记录中添加示例文本、一张图片和当前日期：

Swift

```
let record = CKRecord(recordType: "Users", recordID: recordID)
record.setObject("Some Text", forKey: "text")
record.setObject(CKAsset(fileURL: someValidImageURL), forKey: "image")
record.setObject(NSDate(), forKey: "date")
```

Objective-C

```
CKRecord *record = [[CKRecord alloc] initWithRecordType: "Users" recordID: recordID];
[record setObject: "Some Text" forKey: "text"];
[record setObject: [CKAsset assetWithFileURL: someValidImageURL] forKey: "image"];
[record setObject: [[NSDate alloc] init] forKey: "date"];
```

> **注意**
>
> 这里，我们没有直接将`UIImage`添加到记录中，因为如备注中所述，CloudKit不直接支持图像格式，因此我们将`UIImage`转换成了`CKAsset`。

访问容器

Swift

```
let container = CKContainer.defaultContainer()
let database = container.privateCloudDatabase // 或者 container.publicCloudDatabase
```

将记录保存到 CloudKit 数据库

Swift

```
database.saveRecord(record, completionHandler: { (_, error) -> Void in
    print(error ?? "")
})
```


can handle it way better than the dashboard, but the functionality is also there if you prefer. The most use of this page is for previewing types.

Section 130.3: Saving data to CloudKit

To save date to CloudKit, we must make:

- A `CKRecordID` (the key of your unique record)
- A `CKRecord` (which includes data)

Making a record key

To ensure that every new record identifier is unique, we use the current *timestamp*, which is unique. We get the timestamp using `NSDate`'s method `timeIntervalSinceReferenceDate()`. It is in form of ###.### (# are numbers), which we will use the integer part. To do this, we split the string:

Swift

```
let timestamp = String(format: "%f", NSDate.timeIntervalSinceReferenceDate())
let timestampParts = timestamp.componentsSeparatedByString(".")
let recordID = CKRecordID(recordName: timestampParts[0])
```

Making the record

To make the record, we should specify the record type (explained in Using CloudKit Dashboard) as Users, the ID as the thing we made just now and the data. Here, we will add a sample text, a picture and the current date to the record:

Swift

```
let record = CKRecord(recordType: "Users", recordID: recordID)
record.setObject("Some Text", forKey: "text")
record.setObject(CKAsset(fileURL: someValidImageURL), forKey: "image")
record.setObject(NSDate(), forKey: "date")
```

Objective-C

```
CKRecord *record = [[CKRecord alloc] initWithRecordType: "Users" recordID: recordID];
[record setObject: "Some Text" forKey: "text"];
[record setObject: [CKAsset assetWithFileURL: someValidImageURL] forKey: "image"];
[record setObject: [[NSDate alloc] init] forKey: "date"];
```

> **Note**
>
> Here, we didn't add the `UIImage` directly to the record, because as mentioned in Remarks, image format isn't directly supported in CloudKit, so we have converted `UIImage` into `CKAsset`.

Accessing the container

Swift

```
let container = CKContainer.defaultContainer()
let database = container.privateCloudDatabase // or container.publicCloudDatabase
```

Saving the records to CloudKit database

Swift

```
database.saveRecord(record, completionHandler: { (_, error) -> Void in
    print(error ?? "")
})
```

第131章：GameplayKit

第131.1节：生成随机数

虽然GameplayKit（随 iOS 9 SDK 引入）主要用于实现游戏逻辑，但它也可以用来生成随机数，这在应用和游戏中非常有用。

除了以下章节中使用的GKRandomSource.sharedRandom之外，开箱即用的还有三种额外类型的GKRandomSource。

- GKARC4RandomSource 使用 ARC4 算法GKLinearCongruen
- tialRandomSource 是一种快速但随机性较差的GKRandomSourceGKMersenneTwisterRando
- mSource 实现了梅森旋转算法。它速度较慢但随机性更强。

在接下来的章节中，我们只使用GKRandomSource的nextInt()方法。除此之外，还有 `nextBool() -> Bool`和`nextUniform() -> Float`

生成

首先，导入GameplayKit：

Swift

```
import GameplayKit
```

Objective-C

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

然后，使用以下代码生成一个随机数：

Swift

```
let randomNumber = GKRandomSource.sharedRandom().nextInt()
```

Objective-C

```
int randomNumber = [[GKRandomSource sharedRandom] nextInt];
```

> **注意**
>
> 当nextInt()函数不带参数使用时，会返回一个介于 -2,147,483,648和2,147,483,647之间的随机数，包含这两个数，因此我们不能确定它总是一个正数或非零数。

生成一个从0到n的数字

为实现这一点，您应将 n 传递给 nextIntWithUpperBound() 方法：

Swift

```
let randomNumber = GKRandomSource.sharedRandom().nextInt(upperBound: 10)
```

Objective-C

Chapter 131: GameplayKit

Section 131.1: Generating random numbers

Although `GameplayKit` (which is introduced with iOS 9 SDK) is about implementing game logic, it could also be used to generate random numbers, which is very useful in apps and games.

Beside the `GKRandomSource.sharedRandom` which is used in the following chapters there are three additional types of `GKRandomSource`'s out of the box.

- **GKARC4RandomSource** Which uses the the ARC4 algorithm
- **GKLinearCongruentialRandomSource** Which is a fast but not so random `GKRandomSource`
- **GKMersenneTwisterRandomSource** Which implements a MersenneTwister algorithm. It is slower but more random.

In the following chapter we only use the `nextInt()` method of a `GKRandomSource`. In addition to this there is the `nextBool() -> Bool` and the `nextUniform() -> Float`

Generation

First, import `GameplayKit`:

Swift

```
import GameplayKit
```

Objective-C

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

Then, to generate a random number, use this code:

Swift

```
let randomNumber = GKRandomSource.sharedRandom().nextInt()
```

Objective-C

```
int randomNumber = [[GKRandomSource sharedRandom] nextInt];
```

> **Note**
>
> The nextInt() function, when used without parameters, will return a random number between -2,147,483,648 and 2,147,483,647, including themselves, so we are not sure that it is always a positive or non-zero number.

Generating a number from 0 to n

To achieve this, you should give n to `nextIntWithUpperBound()` method:

Swift

```
let randomNumber = GKRandomSource.sharedRandom().nextInt(upperBound: 10)
```

Objective-C

```
int randomNumber = [[GKRandomSource sharedRandom] nextIntWithUpperBound: 10];
```

这段代码会给我们一个介于0到10之间的数字，包括0和10。

生成从m到n的数字

为此，你需要创建一个带有GKRandomSource的GKRandomDistribution对象，并传入边界值。一个GKRandomDistribution可以用来改变分布行为，比如GKGaussianDistribution或GKShuffledDistribution。

之后，该对象可以像普通的GKRandomSource一样使用，因为它也实现了GKRandom协议。

Swift

```
        let randomizer = GKRandomDistribution(randomSource: GKRandomSource(), lowestValue: 0, highestValue:
6)
let randomNumberInBounds = randomizer.nextInt()
```

Objective-C outdated

```
int randomNumber = [[GKRandomSource sharedRandom] nextIntWithUpperBound: n - m] + m;
```

例如，要生成一个介于3到10之间的随机数，可以使用以下代码：

Swift

```
let randomNumber = GKRandomSource.sharedRandom().nextInt(upperBound: 7) + 3
```

Objective-C outdated

```
int randomNumber = [[GKRandomSource sharedRandom] nextIntWithUpperBound: 7] + 3;
```

第131.2节：GKEntity和GKComponent

实体代表游戏中的一个对象，比如玩家角色或敌人角色。由于这个对象没有手脚，
它本身不能做太多事情，我们可以给它添加组件。为了创建这个系统，苹果提供了GKEntity和GKComponent类。

假设我们在接下来的章节中有以下类：

```
class Player: GKEntity{}
class PlayerSpriteComponent: GKComponent {}
```

GK实体

实体是组件的集合，并提供多个函数来添加、移除和与其组件交互。

虽然我们可以直接使用 GKEntity，但通常会为特定类型的游戏实体对其进行子类化。

重要的是，每个类的组件只能添加一次。如果你添加了第二个相同类的组件，它将覆盖 GKEntity 中已存在的第一个组件。

```
let otherComponent = PlayerSpriteComponent()
var player = Player()
player.addComponent(PlayerSpriteComponent())
player.addComponent(otherComponent)
```



```
int randomNumber = [[GKRandomSource sharedRandom] nextIntWithUpperBound: 10];
```

This code will give us a number between 0 and 10, including themselves.

Generating a number from m to n

To do this you create a GKRandomDistribution object with a GKRandomSource and pass in the bounds. A GKRandomDistribution can be used to change the distribution behaviour like GKGaussianDistribution or GKShuffledDistribution.

After that the object can be used like every regular GKRandomSource since it does implement the GKRandom protocol too.

Swift

```
let randomizer = GKRandomDistribution(randomSource: GKRandomSource(), lowestValue: 0, highestValue:
6)
let randomNumberInBounds = randomizer.nextInt()
```

Objective-C *outdated*

```
int randomNumber = [[GKRandomSource sharedRandom] nextIntWithUpperBound: n - m] + m;
```

For example, to generate a random number between 3 and 10, you use this code:

Swift

```
let randomNumber = GKRandomSource.sharedRandom().nextInt(upperBound: 7) + 3
```

Objective-C *outdated*

```
int randomNumber = [[GKRandomSource sharedRandom] nextIntWithUpperBound: 7] + 3;
```

Section 131.2: GKEntity and GKComponent

An entity represents an object of a game like a player figure or an enemy figure. Since this object does not do much without arms and legs we can add the components to this. To create this system apple has the GKEntity and GKComponent classes.

Lets assume we have the following classe for the following chapters:

```
class Player: GKEntity{}
class PlayerSpriteComponent: GKComponent {}
```

GKEntity

An entity is a collection of components and offers several functions to add, remove and interact with components of it.

While we could just use the GKEntity it is common to Subclass it for a specific type of game entity.

It is important that it is only possible to add a component of a class once. In case you add a second component of the same class it will override the first exsisting component inside of the GKEntity

```
let otherComponent = PlayerSpriteComponent()
var player = Player()
player.addComponent(PlayerSpriteComponent())
player.addComponent(otherComponent)
```

```
print(player.components.count) //将打印 1
print(player.components[0] === otherComponent) // 将打印 true
```

你可能会问为什么。原因是调用了名为component(for: T.Type)的方法，该方法返回实体中特定类型的组件。

```
let component = player.component(ofType: PlayerSpriteComponent.self)
```

除了组件方法外，它还有一个update方法，用于将游戏逻辑的增量时间或当前时间委托给其组件。

```
var player = Player()
player.addComponent(PlayerSpriteComponent())
player.update(deltaTime: 1.0) // 将调用添加到它的 PlayerSpriteComponent 的 update 方法
```

GKComponent

组件表示实体的某个部分，例如视觉组件或逻辑组件。

如果调用实体的 update 方法，它会将此调用委托给所有组件。重写此方法用于操作实体。

```
class PlayerSpriteComponent: GKComponent {
    override func update(deltaTime seconds: TimeInterval) {
        // 根据更新时间移动精灵
    }
}
```

此外，可以重写方法 didAddToEntity 和 willRemoveFromEntity 来通知其他组件其被添加或移除。

要操作组件内的其他组件，可以获取该组件所属的 GKEntity。

```
override func update(deltaTime seconds: TimeInterval) {
    let controller = self.entity?.component(ofType: PlayerControlComponent.self)
    // 调用控制器上的方法
}
```

虽然这是可能的，但这不是一种常见的模式，因为它将两个组件连接在一起。

GK组件系统

虽然我们刚才讨论了使用GKEntity的更新委托机制来更新GKComponents，但还有一种不同的方式来更新GKComponents，称为GKComponentSystem。

当需要一次性更新所有特定类型的组件时，会使用它。

为特定类型的组件创建一个GKComponentSystem。

```
let system = GKComponentSystem(componentClass: PlayerSpriteComponent.self)
```

要添加组件，可以使用add方法：

```
system.addComponent(PlayerSpriteComponent())
```



```
print(player.components.count) //will print 1
print(player.components[0] === otherComponent) // will print true
```

You may ask why. The reason for this is the methods called `component(for: T.Type)` which returns the component of a specific type of the entity.

```
let component = player.component(ofType: PlayerSpriteComponent.self)
```

In addition to the components-methods it has an `update` method which is used to delegate the delta time or current time of the game logic to it's components.

```
var player = Player()
player.addComponent(PlayerSpriteComponent())
player.update(deltaTime: 1.0) // will call the update method of the PlayerSpriteComponent added to it
```

GKComponent

A component represents something of an entity for example the visual component or the logic component.

If an the update method of an entity is called it will delegate this to all of it's components. Overriding this method is used to manipulate an Entity.

```
class PlayerSpriteComponent: GKComponent {
    override func update(deltaTime seconds: TimeInterval) {
        //move the sprite depending on the update time
    }
}
```

In addition to this it is possible to override the method `didAddToEntity` and `willRemoveFromEntity` to inform other components about it's removal or add.

To manipulate a other component inside of a component it is possible to get the GKEntity which the component is added to.

```
override func update(deltaTime seconds: TimeInterval) {
    let controller = self.entity?.component(ofType: PlayerControlComponent.self)
    //call methods on the controller
}
```

While this is possible it is **not** a common pattern since it wires the two components together.

GKComponentSystem

While we just talked about using the update delegate mechanism of the `GKEntity` to update the `GKComponents` there is a different way to update `GKComponents` which is called `GKComponentSystem`.

It is used in case it is needed that all components of a specific type need to be updated in one go.

A `GKComponentSystem` is created for a specific type of component.

```
let system = GKComponentSystem(componentClass: PlayerSpriteComponent.self)
```

To add a component you can use the add method:

```
system.addComponent(PlayerSpriteComponent())
```

但更常见的方式是将创建的实体及其组件传递给GKComponentSystem，它会在实体中找到匹配的组件。

```
system.addComponent(foundIn: player)
```

要更新特定类型的所有组件，请调用更新：

```
system.update(deltaTime: delta)
```

如果您想使用GKComponentSystem而不是基于实体的更新机制，则必须为每个组件拥有一个GKComponentSystem，并对所有系统调用更新。

But a more common way is to pass the created entity with it's components to the GKComponentSystem and it will find a matching component inside of the entity.

```
system.addComponent(foundIn: player)
```

To update all components of a specific type call the update:

```
system.update(deltaTime: delta)
```

In case you want to use the GKComponentSystem instead of a entity based update mechanism you have to have a GKComponentSystem for every component and call the update on all of the systems.

第132章：从命令行进行Xcode构建与归档

选项	描述
-project	构建项目 name.xcodeproj。
-scheme	构建工作区时必需。
-目的地	使用目标设备
-configuration	使用构建配置
-sdk	指定的 SDK

第 132.1 节：构建与归档

构建：

```
xcodebuild -exportArchive -exportFormat ipa \
    -archivePath "/Users/username/Desktop/MyiOSApp.xcarchive" \
    -exportPath "/Users/username/Desktop/MyiOSApp.ipa" \
    -exportProvisioningProfile "MyCompany Distribution Profile"
```

归档：

```
xcodebuild -project <ProjectName.xcodeproj>
    -scheme <ProjectName>
    -sdk iphonesimulator
    -configuration Debug
    -destination "platform=iOS Simulator,name=<Device>,OS=9.3"
   clean build
```


Chapter 132: Xcode Build & Archive From Command Line

Option	Description
-project	Build the project name.xcodeproj.
-scheme	Required if building a workspace.
-destination	Use the destination device
-configuration	Use the build configuration
-sdk	specified SDK

Section 132.1: Build & Archive

Build:

```
xcodebuild -exportArchive -exportFormat ipa \
    -archivePath "/Users/username/Desktop/MyiOSApp.xcarchive" \
    -exportPath "/Users/username/Desktop/MyiOSApp.ipa" \
    -exportProvisioningProfile "MyCompany Distribution Profile"
```

Archive:

```
xcodebuild -project <ProjectName.xcodeproj>
    -scheme <ProjectName>
    -sdk iphonesimulator
    -configuration Debug
    -destination "platform=iOS Simulator,name=<Device>,OS=9.3"
    clean build
```

第133章：XCTest框架 - 单元测试

第133.1节：向Xcode项目添加测试文件

创建项目时

您应该在项目创建对话框中勾选“包含单元测试”。

创建项目后

如果您在创建项目时错过了勾选该项，您仍然可以稍后添加测试文件。操作步骤如下：

1- 进入 Xcode 中的项目设置

2- 进入“Targets”（目标）

3- 点击“Add Target”（添加目标）

4- 在“Other”（其他）下，选择“Cocoa Touch Unit Test Testing Bundl

e”最后，您应该会有一个名为[您的应用名称]Tests.swift的文件。若是 Objective-C，则应有两个文件，分别命名为[您的应用名称]Te sts.h和[您的应用名称]Tests.m。

[您的应用名称]Tests.swift或.m文件默认包含：

- 一个XCTest模块导入
- 一个继承自XCTestCase的[您的应用名称]Tests类，包含setUp、tea
- rDown、testExample、testPerformanceExample方法

Swift

```
import XCTest

class MyProjectTests: XCTestCase {

override func setUp() {
    super.setUp()
    // 在此处放置初始化代码。此方法在类中每个测试方法调用之前被调用。

}

override func tearDown() {
    // 在此处放置清理代码。此方法在类中每个测试方法调用之后被调用。

    super.tearDown()
}
```


Chapter 133: XCTest framework - Unit Testing

Section 133.1: Adding Test Files to Xcode Project

When creating the project

You should check "Include Unit Tests" in the project creation dialog.

After creating the project

If you missed checking that item while creating your project, you could always add test files later. To do so:

1- Go to your project settings in Xcode

2- Go to "Targets"

3- Click "Add Target"

4- Under "Other", select "Cocoa Touch Unit Test Testing Bundle"

At the end, you should have a file named `[Your app name]Tests.swift`. In Objective-C, you should have two files named `[Your app name]Tests.h` and `[Your app name]Tests.m` instead.

`[Your app name]Tests.swift or .m` file will include by default :

- A `XCTest` module import
- A `[Your app name]Tests` class which extends `XCTestCase`
- `setUp`, `tearDown`, `testExample`, `testPerformanceExample` methods

Swift

```
import XCTest

class MyProjectTests: XCTestCase {

override func setUp() {
    super.setUp()
    // Put setup code here. This method is called before the invocation of each test method in the
class.
}

override func tearDown() {
    // Put teardown code here. This method is called after the invocation of each test method in
the class.
    super.tearDown()
}
```

```
func testExample() {
    // 这是一个功能测试用例的示例。
    // 使用 XCTAssert 及相关函数验证测试是否产生正确结果。

}

func testPerformanceExample() {
    // 这是一个性能测试用例的示例。
    self.measure {
        // 在这里放置你想要测量时间的代码。
    }
}

}
```

Objective-C

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

@interface MyProjectTests : XCTestCase

@end

@implementation MyProjectTests

- (void)setUp {
    [super setUp];
// 在此处放置初始化代码。此方法在类中每个测试方法调用之前被调用。

}

- (void)tearDown {
// 在这里放置清理代码。此方法在每个测试方法调用后执行。
class。
    [super tearDown];
}

- (void)testExample {
// 这是一个功能测试用例的示例。
// 使用 XCTAssert 及相关函数验证测试是否产生正确结果。
}

- (void)testPerformanceExample {
// 这是一个性能测试用例的示例。
    [self measureBlock:^{
    // 在这里放置你想要测量时间的代码。
    }];
}

@end
```

第133.2节：添加测试方法

根据苹果公司：

> **测试方法**
>
> 测试方法是测试类的实例方法，以前缀test开头，不带参数，返回void，例如(void)testColorIsRed()。测试方法执行项目中的代码，如果代码未产生预期结果，则使用一组断言API报告失败。


```
func testExample() {
    // This is an example of a functional test case.
    // Use XCTAssert and related functions to verify your tests produce the correct results.

}

func testPerformanceExample() {
    // This is an example of a performance test case.
    self.measure {
        // Put the code you want to measure the time of here.
    }
}

}
```

Objective-C

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

@interface MyProjectTests : XCTestCase

@end

@implementation MyProjectTests

- (void)setUp {
    [super setUp];
// Put setup code here. This method is called before the invocation of each test method in the
class.
}

- (void)tearDown {
// Put teardown code here. This method is called after the invocation of each test method in the
class.
    [super tearDown];
}

- (void)testExample {
// This is an example of a functional test case.
// Use XCTAssert and related functions to verify your tests produce the correct results.
}

- (void)testPerformanceExample {
// This is an example of a performance test case.
    [self measureBlock:^{
    // Put the code you want to measure the time of here.
    }];
}

@end
```

Section 133.2: Adding test methods

According to Apple:

> **Test Methods**
>
> A test method is an instance method of a test class that begins with the prefix test, takes no parameters, and returns void, for example, (void)testColorIsRed(). A test method exercises code in your project and, if that code does not produce the expected result, reports failures using a set of assertion APIs. For

> 例如，函数的返回值可能会与预期值进行比较，或者测试可能断言对某个类的方法的不当使用会抛出异常。

因此，我们使用“test”作为方法前缀添加测试方法，如：

Swift

```
func testSomething() {

}
```

Objective-C

```
- (void)testSomething {

}
```

为了实际测试结果，我们使用XCTAssert()方法，该方法接受一个布尔表达式，如果为真，则标记测试为成功，否则标记为失败。

假设我们在视图控制器类中有一个名为 sum() 的方法，用于计算两个数字的和。为了测试它，
我们使用这个方法：

Swift

```
func testSum(){
    let result = viewController.sum(4, and: 5)
    XCTAssertEqual(result, 9)
}
```

Objective-C

```
- (void)testSum {
int result = [viewController sum:4 and:5];
    XCTAssertEqual(result, 9);
}
```

> **注意**
>
> 默认情况下，如果视图控制器类中的标签、文本框或其他UI元素最初是在Storyboard文件中创建的，那么测试类无法访问它们。这是因为它们是在视图控制器类的loadView()方法中初始化的，而该方法在测试时不会被调用。调用loadView()及所有其他所需方法的最佳方式是访问我们viewController属性的view属性。你应该在测试UI元素之前添加这行代码：

```
XCTAssertNotNil(viewController.view)
```

第133.3节：编写测试类

```
import XCTest
@testable import PersonApp

class PersonTests: XCTestCase {
    func test_completeName() {
        let person = Person(firstName: "Josh", lastName: "Brown")
        XCTAssertEqual(person.completeName(), "Josh Brown")
    }
```


> example, a function's return value might be compared against an expected value or your test might assert that improper use of a method in one of your classes throws an exception.

So we add a test method using "test" as the prefix of the method, like:

Swift

```
func testSomething() {

}
```

Objective-C

```
- (void)testSomething {

}
```

To actually test the results, we use `XCTAssert()` method, which takes a boolean expression, and if true, marks the test as succeeded, else it will mark it as failed.

Let's say we have a method in View Controller class called `sum()` which calculates sum of two numbers. To test it, we use this method:

Swift

```
func testSum(){
    let result = viewController.sum(4, and: 5)
    XCTAssertEqual(result, 9)
}
```

Objective-C

```
- (void)testSum {
    int result = [viewController sum:4 and:5];
    XCTAssertEqual(result, 9);
}
```

> **Note**
>
> By default, you can't access label, text box or other UI items of the View Controller class from test class if they are first made in Storyboard file. This is because they are initialized in `loadView()` method of the View Controller class, and this will not be called when testing. The best way to call `loadView()` and all other required methods is accessing the `view` property of our `viewController` property. You should add this line before testing UI elements:

```
XCTAssertNotNil(viewController.view)
```

Section 133.3: Writing a test class

```
import XCTest
@testable import PersonApp

class PersonTests: XCTestCase {
    func test_completeName() {
        let person = Person(firstName: "Josh", lastName: "Brown")
        XCTAssertEqual(person.completeName(), "Josh Brown")
    }
```

```
}
```

现在让我们来讨论这里发生了什么。 import XCTest 这一行允许我们扩展 XCTestCase 并使用XCTAssertEqual（以及其他断言）。扩展 XCTestCase 并且测试函数名前缀为 test 将确保Xcode 在运行项目测试时（⌘U 或 Product > Test）自动执行该测试。 @testableimport PersonApp 这一行将导入我们的 PersonApp 目标，以便我们可以测试并使用其中的类，例如上面示例中的 Person。最后，XCTAssertEqual 将确保 person.completeName() 等于字符串 "Josh Brown"。

第133.4节：将Storyboard和视图控制器作为实例添加到测试文件中

要开始单元测试，这将在测试文件中进行，并且将测试视图控制器和Storyboard，我们应当将这两个文件引入测试文件中。

定义视图控制器

Swift

```
var viewController : ViewController!
```

引入Storyboard并初始化视图控制器

将以下代码添加到 setUp() 方法中：

Swift

```
let storyboard = UIStoryboard(name: "Main", bundle: nil)
viewController = storyboard.instantiateInitialViewController() as! ViewController
```

Objective-C

```
UIStoryboard *storyboard = [UIStoryboard storyboardWithName:"Main" bundle:nil];
viewController = (ViewController *) [storyboard instantiateInitialViewController];
```

这样，你可以编写测试方法，它们就知道去哪里检查错误。在这种情况下，有视图控制器和故事板。

第133.5节：导入可测试的模块

类、结构体、枚举及其所有方法默认都是internal的。这意味着它们只能从同一个模块访问。测试用例在不同的目标中，这意味着它们在不同的模块中。为了能够访问你想测试的方法，你需要使用@testable关键字导入要测试的模块。

假设我们有一个名为ToDo的主模块，我们想为它编写测试。我们会这样导入该模块：

```
@testable import ToDo
```

在包含此导入语句的文件中的所有测试方法现在都可以访问ToDo模块中所有internal的类、结构体、枚举及其所有internal方法。

你绝不应该将包含你想测试的元素的文件添加到测试目标中，因为这可能导致难以调试的错误。


```
}
```

Now let's discuss what's going on here. The `import XCTest` line will allow us to extend `XCTestCase` and use `XCTAssertEqual` (among other assertions). Extending `XCTestCase` and prefixing our test name with `test` will ensure that Xcode automatically runs this test when running the tests in the project (⌘**U** or **Product** > **Test**). The `@testable import PersonApp` line will import our `PersonApp` target so we can test and use classes from it, such as the `Person` in our example above. And finally, our `XCTAssertEqual` will ensure that `person.completeName()` is equal to the string `"Josh Brown"`.

Section 133.4: Adding Storyboard and View Controller as instances to test file

To get started with unit testing, which will be done in the tests file and will be testing the View Controller and Storyboard, we should introduce these two files to the test file.

Defining the View Controller

Swift

```
var viewController : ViewController!
```

Introducing the Storyboard and initializing the View Controller

Add this code to the `setUp()` method:

Swift

```
let storyboard = UIStoryboard(name: "Main", bundle: nil)
viewController = storyboard.instantiateInitialViewController() as! ViewController
```

Objective-C

```
UIStoryboard *storyboard = [UIStoryboard storyboardWithName:"Main" bundle:nil];
viewController = (ViewController *) [storyboard instantiateInitialViewController];
```

This way, you could write test methods, and they will know where to check for errors. In this case, there are View Controller and the Storyboard.

Section 133.5: Import a module that it can be tested

Classes, structs, enums and all their methods are `internal` by default. This means they can be only accessed from the same module. The test cases are in a different target an this means they are in a different module. To be able to access the method you want to test, you need to import the module to be tested using the `@testable` keyword.

Let's say we have a main module called `ToDo` and we want to write tests for it. We would import that module like this:

```
@testable import ToDo
```

All test methods in the file with this import statement can now access all `internal` classes, structs, enums and all their `internal` methods of the `ToDo` module.

You should never add the files with the elements you want to test to the test target because that can lead to hard to debug errors.

第133.6节：触发视图加载和显示

视图加载

在视图控制器的测试中，有时你需要触发loadView()或viewDidLoad()的执行。这可以通过访问视图来实现。假设你在测试中有一个名为sut（被测试系统）的视图控制器实例，那么代码如下：

```
XCTAssertNotNil(sut.view)
```

视图显示

你也可以通过添加以下代码来触发viewWillAppear(_:)和viewDidAppear(_:)方法：

```
sut.beginAppearanceTransition(true, animated: true)
sut.endAppearanceTransition()
```

第133.7节：开始测试

测试特定方法

要测试特定方法，点击方法定义旁边的方框。

测试所有方法

要测试所有方法，请点击类定义旁边的方框。

查看测试结果

如果定义旁边有绿色勾号，测试已成功。

如果定义旁边有红色叉号，测试失败。

运行所有测试

```
产品 -> 测试 或 Cmd + U
```

它将运行所有测试目标中的所有测试！

Section 133.6: Trigger view loading and appearance

View loading

In a test for a view controller you want sometimes to trigger the execution of `loadView()` or `viewDidLoad()`. This can be done by accessing the view. Let's say you have view controller instance in your test called `sut` (system under test), then the code would look like this:

```
XCTAssertNotNil(sut.view)
```

View appearance

You can also trigger the methods `viewWillAppear(_:)` and `viewDidAppear(_:)` by adding the following code:

```
sut.beginAppearanceTransition(true, animated: true)
sut.endAppearanceTransition()
```

Section 133.7: Start Testing

Testing a specific method

To test a specific method, click the square next to the method definition.

Testing all methods

To test all methods, click the square next to the class definition.

See the testing result

If there is a green check next to the definition, the test has succeeded.

If there is a red cross next to the definition, the test has failed.

Running all tests

```
Product -> Test OR Cmd + U
```

It will run all the tests from all the test targets!

第134章：AVPlayer和AVPlayerViewController

第134.1节：使用AVPlayer和AVPlayerLayer播放媒体

Objective C

```
NSURL *url = [NSURL URLWithString:@"YOUR URL"];
AVPlayer *player = [AVPlayer playerWithURL:videoURL];
AVPlayerLayer *playerLayer = [AVPlayerLayer playerLayerWithPlayer:player];
playerLayer.frame = self.view.bounds;
[self.view.layer 添加子图层:playerLayer];
[player 播放];
```

Swift

```
let url = NSURL(string: "YOUR URL")
let player = AVPlayer(URL: videoURL!)
let playerLayer = AVPlayerLayer(player: player)
playerLayer.frame = self.view.bounds
self.view.layer.addSublayer(playerLayer)
player.play()
```

第134.2节：使用AVPlayerViewController播放媒体

Objective-C

```
NSURL *url = [[NSURL alloc] initWithString:@"YOUR URL"]; // url可以是远程或本地的

AVPlayer *player = [AVPlayer playerWithURL:url];
// 创建播放器视图控制器

AVPlayerViewController *controller = [[AVPlayerViewController alloc] init];
[self presentViewController:controller animated:YES completion:nil];
controller.player = player;
[player 播放];
```

Swift

```
let player = AVPlayer(URL: url) // url可以是远程或本地的

let playerViewController = AVPlayerViewController()
// 创建播放器视图控制器
playerViewController.player = player
self.presentViewController(playerViewController, animated: true) {

    playerViewController.player!.play()
}
```

第134.3节：AVPlayer示例

AVPlayer *avPlayer = [AVPlayer playerWithURL:[NSURL URLWithString:@"YOUR URL"]];

```
AVPlayerViewController *avPlayerCtrl = [[AVPlayerViewController alloc] init];
    avPlayerCtrl.view.frame = self.view.frame;
avPlayerCtrl.player = avPlayer;
    avPlayerCtrl.delegate = self;
```


Chapter 134: AVPlayer and AVPlayerViewController

Section 134.1: Playing Media using AVPlayer and AVPlayerLayer

Objective C

```
NSURL *url = [NSURL URLWithString:@"YOUR URL"];
AVPlayer *player = [AVPlayer playerWithURL:videoURL];
AVPlayerLayer *playerLayer = [AVPlayerLayer playerLayerWithPlayer:player];
playerLayer.frame = self.view.bounds;
[self.view.layer addSublayer:playerLayer];
[player play];
```

Swift

```
let url = NSURL(string: "YOUR URL")
let player = AVPlayer(URL: videoURL!)
let playerLayer = AVPlayerLayer(player: player)
playerLayer.frame = self.view.bounds
self.view.layer.addSublayer(playerLayer)
player.play()
```

Section 134.2: Playing Media Using AVPlayerViewController

Objective-C

```
NSURL *url = [[NSURL alloc] initWithString:@"YOUR URL"]; // url can be remote or local

AVPlayer *player = [AVPlayer playerWithURL:url];
// create a player view controller

AVPlayerViewController *controller = [[AVPlayerViewController alloc] init];
[self presentViewController:controller animated:YES completion:nil];
controller.player = player;
[player play];
```

Swift

```
let player = AVPlayer(URL: url) // url can be remote or local

let playerViewController = AVPlayerViewController()
// creating a player view controller
playerViewController.player = player
self.presentViewController(playerViewController, animated: true) {

    playerViewController.player!.play()
}
```

Section 134.3: AVPlayer Example

AVPlayer *avPlayer = [AVPlayer playerWithURL:[NSURL URLWithString:@"YOUR URL"]];

```
    AVPlayerViewController *avPlayerCtrl = [[AVPlayerViewController alloc] init];
    avPlayerCtrl.view.frame = self.view.frame;
    avPlayerCtrl.player = avPlayer;
    avPlayerCtrl.delegate = self;
```

```
    [avPlayer play];
    [self presentViewController:avPlayerCtrl animated:YES completion:nil
```



```
    [avPlayer play];
    [self presentViewController:avPlayerCtrl animated:YES completion:nil
```

第135章：iOS中的深度链接

第135.1节：为自己的应用添加URL方案

假设你正在开发一个名为MyTasks的应用，你想允许通过传入的URL来创建一个带有标题和正文的新任务。你设计的URL可能看起来像这样：

```
mytasks://create?title=hello&body=world
```

（当然，text 和 body 参数用于填充我们正在创建的任务！）

将此 URL 方案添加到您的项目中的主要步骤如下：

1. 在应用的 Info.plist 文件中注册一个 URL 方案，这样系统才能知道何时将 URL 路由到您的应用。
2. 向您的 UIApplicationDelegate 添加一个函数，用于接收和处理传入的 URL。
3. 执行打开该 URL 时需要进行的任何任务。

第一步：在 Info.plist 中注册 URL 方案：

首先，我们需要在 Info.plist 文件中添加一个"URL Types"条目。点击这里的（+）按钮：

...然后输入您的应用的唯一标识符，以及您想使用的 URL 方案。要具体！您不想让 URL 方案与其他应用的实现冲突。这里宁可长一点也不要太短！

Chapter 135: Deep Linking in iOS

Section 135.1: Adding a URL scheme to your own app

Let's say you're working on an app called `MyTasks`, and you want to allow inbound URLs to create a new task with a title and a body. The URL you're designing might look something like this:

```
mytasks://create?title=hello&body=world
```

(Of course, the `text` and `body` parameters are used to populate our task that we're creating!)

Here are the Big Steps to adding this URL scheme to your project:

1. Register a URL scheme in your app's `Info.plist` file, so the system knows when to route a URL to your app.
2. Add a function to your `UIApplicationDelegate` that accepts and handles incoming URLs.
3. Perform whatever task needs to occur when that URL is opened.

Step One: Register a URL scheme in Info.plist:

First, we need to add a "URL Types" entry to our Info.plist file. Click the (+) button here:

...then enter a unique identifier for your app, as well as the URL scheme you want to use. Be specific! You don't want the URL scheme to conflict with another app's implementation. Better to be too-long here than too-short!

步骤二：处理 UIApplicationDelegate 中的 URL

我们需要在我们的UIApplicationDelegate上实现application:openURL:options:。我们将检查传入的URL，看看是否有可以执行的操作！

一种实现方式如下：

```
func application(app: UIApplication, openURL url: NSURL, options: [String : AnyObject]) -> Bool {
    if url.scheme == "mytasks" && url.host == "create" {
        let title = // 使用你选择的方法从URL的查询中获取标题
        let body = // 使用你选择的方法从URL的查询中获取正文
        self.rootViewController.createTaskWithTitle(title, body: body)
        return true
    }

    返回 false
}
```

第三步：根据URL执行任务。

当用户通过URL打开你的应用时，他们可能期望*某些*事情发生。也许是导航到某个内容，也许是创建一个新项目——在这个例子中，我们将创建一个新的任务！

在上面的代码中，我们可以看到调用了self.rootViewController.createTaskWithTitle(:body:)——所以假设你的AppDelegate有一个指向其根视图控制器的指针，并且该控制器正确实现了该函数，那么你就准备好了！

第135.2节：基于URL scheme打开应用

要使用定义的URL schemeodolist://:打开应用

Objective-C

```
NSURL *myURL = [NSURL URLWithString:@"todolist://there/is/something/to/do"];
[[UIApplication sharedApplication] openURL:myURL];
```

Swift

```
let stringURL = "todolist://there/is/something/to/do"
if let url = NSURL(string: stringURL) {
    UIApplication.shared().openURL(url)
}
```

HTML

```
<a href="todolist://there/is/something/to/do">New SMS Message</a>
```

> 注意： 检查链接是否可以打开很有用，否则可以向用户显示适当的信息。

Step Two: Handle the URL in the UIApplicationDelegate

We need to implement `application:openURL:options:` on our `UIApplicationDelegate`. We'll inspect the incoming `URL` and see if there's an action we can take!

One implementation would be this:

```
func application(app: UIApplication, openURL url: NSURL, options: [String : AnyObject]) -> Bool {
    if url.scheme == "mytasks" && url.host == "create" {
        let title = // get the title out of the URL's query using a method of your choice
        let body = // get the title out of the URL's query using a method of your choice
        self.rootViewController.createTaskWithTitle(title, body: body)
        return true
    }

    return false
}
```

Step Three: Perform a task depending on the URL.

When a user opens your app via a URL, they probably expected *something* to happen. Maybe that's navigating to a piece of content, maybe that's creating a new item - in this example, we're going to create a new task in the app!

In the above code, we can see a call to `self.rootViewController.createTaskWithTitle(:body:)` - so assuming that your `AppDelegate` has a pointer to its root view controller which implements the function properly, you're all set!

Section 135.2: Opening an app based on its URL scheme

To open an app with defined URL scheme `todolist://`:

Objective-C

```
NSURL *myURL = [NSURL URLWithString:@"todolist://there/is/something/to/do"];
[[UIApplication sharedApplication] openURL:myURL];
```

Swift

```
let stringURL = "todolist://there/is/something/to/do"
if let url = NSURL(string: stringURL) {
    UIApplication.shared().openURL(url)
}
```

HTML

```
<a href="todolist://there/is/something/to/do">New SMS Message</a>
```

> **Note:** It's useful to check if link can be opened to otherwise display an appropriate message to the user.

> 这可以使用 canOpenURL: 方法来完成。

第135.3节：为您的应用设置深度链接

为您的应用设置深度链接很简单。您只需要一个用于打开应用的小型URL。

按照以下步骤为您的应用设置深度链接。

1.让我们创建一个项目并命名为 DeepLinkPOC。
2. 现在选择你的项目目标。
3. 选择目标后，点击“信息”标签。
4. 向下滚动到底部，直到看到“URL 类型”选项
5. 点击“+”选项。
6. 你会看到URL 方案，添加一个字符串，用于打开你的应用。我们添加“DeepLinking”到 URL 方案中。

因此，要打开你的应用，可以在 Safari 中输入"DeepLinking://"来启动。你的深度链接字符串格式如下。

```
[scheme]://[host]/[path]  --> DeepLinking://path/Page1
```

其中，Scheme（方案）："DeepLinking" Host（主机）："path" path（路径）："Page1"

注意：即使不添加主机和路径，也能启动应用，所以不用担心。但你可以添加主机和路径，以便应用启动后额外重定向到特定页面。

7. 现在将以下方法添加到你的 AppDelegate 中。

Swift:

```
 func application(application: UIApplication, openURL url: NSURL, sourceApplication: String?,
annotation: AnyObject) -> Bool
```

Objective-c：

```
-(BOOL)application:(UIApplication *)application
           openURL:(NSURL *)url
sourceApplication:(NSString *)sourceApplication
         annotation:(id)annotation
```

每当你的应用通过你为其设置的深度链接字符串启动时，都会调用上述方法。

8. 现在是安装你的应用的时候了，但在你直接点击运行按钮之前，先对方案的应用启动方法做一个小改动。

- 选择并编辑你的方案，如下所示

> This can be done using `canOpenURL:` method.

Section 135.3: Setting up deeplink for your app

Setting up deep-linking for your app is easy.You just need a small url using which you want to open your app.

Follow the steps to set up deep-linking for your app.

1. Lets create a project and name it DeepLinkPOC.
2. Now select your project target.
3. After selecting target,select the 'info' tab.
4. Scroll down to the bottom until you see an option of **URL Types**
5. Click '+' option.
6. You will see **URL schemes** add a string using which you want to open your app.Lets add "**DeepLinking**" in URL schemes.

So, to open your app you can launch it by typing **"DeepLinking://"** into your safari. Your deep-linking string has following format.

```
[scheme]://[host]/[path]  --> DeepLinking://path/Page1
```

where, Scheme : "DeepLinking" Host : "path" path : "Page1"

Note : Even if don't add host and path it will launch the app,so no worries.But you can add host and path to additionally redirect to particular page after application launch.

7. Now add following method to your appdelegate.

Swift:

```
 func application(application: UIApplication, openURL url: NSURL, sourceApplication: String?,
annotation: AnyObject) -> Bool
```

Objective-c:

```
-(BOOL)application:(UIApplication *)application
          openURL:(NSURL *)url
          sourceApplication:(NSString *)sourceApplication
          annotation:(id)annotation
```

The above method is called whenever your app is launched using a deep-linking string you set for your app.

8. Now is the time to install your app but wait before you directly jump to run button.Lets do a small change in scheme's app-launch method.

- Select and edit your scheme as

- 更改其启动类型并关闭

9. 现在点击运行按钮（如果需要，可以在 didFinishLaunchingWithOptions 和 openURL 方法中添加断点以观察值）
10. 你会看到一条消息“等待 DeepLinkPOC（或你的应用名称）启动”。
11. 打开 Safari，在搜索栏输入“DeepLinking://”，这将显示提示“在 DeepLinkPOC”中打开此页面，点击打开以启动你的应用。

希望你已经了解了如何为你的应用设置深度链接 :)

- change its launch type and close

9. Now click the Run button (if you want you can add breakpoint to your didFinishLaunchingWithOptions and openURL methods to observe values)
10. You'll see a message "Waiting for DeepLinkPOC(or your app name) to launch".
11. Open safari and type in "**DeepLinking://**" into the search bar this will show prompt "open this page in DeepLinkPOC" click open to launch your app.

Hope you got to know how to set up deep-linking for your app :)

第136章：核心图形

第136.1节：创建核心图形上下文

> **核心图形上下文**
>
> 核心图形上下文是一个画布，我们可以在上面绘图并设置一些属性，比如线条粗细。

创建上下文

要创建上下文，我们使用UIGraphicsBeginImageContextWithOptions() C函数。然后，当绘制完成后，调用UIGraphicsEndImageContext()来结束上下文：

Swift

```
let size = CGSize(width: 256, height: 256)

UIGraphicsBeginImageContextWithOptions(size, false, 0)

let context = UIGraphicsGetCurrentContext()

// 在这里绘制代码

UIGraphicsEndImageContext()
```

Objective-C

```
CGSize size = [CGSize width:256 height:256];

UIGraphicsBeginImageContextWithOptions(size, NO, 0);

CGContext *context = UIGraphicsGetCurrentContext();

// 在这里绘制代码

UIGraphicsEndImageContext();
```

在上面的代码中，我们向UIGraphicsBeginImageContextWithOptions()函数传递了3个参数：

1. 一个CGSize对象，用于存储上下文（画布）的整体大小
2. 一个布尔值，如果为true，上下文将是不透明的
3. 一个整数值，用于设置缩放比例（1表示非视网膜屏幕，2表示视网膜屏幕，3表示视网膜高清屏幕）。如果设置为0，系统会根据目标设备自动处理缩放比例。

第136.2节：向用户呈现绘制的画布

Swift

```
let image = UIGraphicsGetImageFromCurrentImageContext()
imageView.image = image //假设 imageView 是一个有效的 UIImageView 对象
```

Objective-C

```
UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
```


Chapter 136: Core Graphics

Section 136.1: Creating a Core Graphics Context

> **Core Graphics context**
>
> A Core Graphics context is a canvas which we can draw in it and set some properties like the line thickness.

Making a context

To make a context, we use the `UIGraphicsBeginImageContextWithOptions()` C function. Then, when we are done with drawing, we just call `UIGraphicsEndImageContext()` to end the context:

Swift

```
let size = CGSize(width: 256, height: 256)

UIGraphicsBeginImageContextWithOptions(size, false, 0)

let context = UIGraphicsGetCurrentContext()

// drawing code here

UIGraphicsEndImageContext()
```

Objective-C

```
CGSize size = [CGSize width:256 height:256];

UIGraphicsBeginImageContextWithOptions(size, NO, 0);

CGContext *context = UIGraphicsGetCurrentContext();

// drawing code here

UIGraphicsEndImageContext();
```

In the code above, we passed 3 parameters to the `UIGraphicsBeginImageContextWithOptions()` function:

1. A `CGSize` object which stores the whole size of the context (the canvas)
2. A boolean value which if it is true, the context will be opaque
3. An integer value which sets the scale (1 for non-retina, 2 for retina and 3 for retina HD screens). If set to 0, the system automatically handles the scale based on the target device.

Section 136.2: Presenting the Drawn Canvas to User

Swift

```
let image = UIGraphicsGetImageFromCurrentImageContext()
imageView.image = image //assuming imageView is a valid UIImageView object
```

Objective-C

```
UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
```

```
imageView.image = image; //假设 imageView 是一个有效的 UIImageView 对象
```



```
imageView.image = image; //assuming imageView is a valid UIImageView object
```

第137章：Segue（界面跳转）

第137.1节：使用 Segue 在导航栈中向后导航

回退 Segue

> 回退 Segue 让你可以"回退"导航栈并指定返回的目标。
> 此函数的签名是 Interface Builder 识别它的关键。**它必须有一个返回值为 IBAction，且带有一个参数 UIStoryboardSegue**。函数名无关紧要。
> 实际上，该函数甚至不必执行任何操作。它只是作为一个标记，指明哪个 UIViewController 是回退 Segue 的目标。[source][1]

解除segue所需的签名

Objective C：

```
-(IBAction)prepareForUnwind:(UIStoryboardSegue *)segue {
}
```

Swift:

```
@IBAction func prepareForUnwind(segue: UIStoryboardSegue) {
}
```

第137.2节：概述

摘自苹果文档：

> UIStoryboardSegue对象负责执行两个视图控制器之间的视觉过渡。此外，segue对象用于准备从一个视图控制器到另一个视图控制器的过渡。segue对象包含有关过渡中涉及的视图控制器的信息。
>
> 当segue被触发但视觉过渡尚未发生时，故事板运行时会调用当前视图控制器的prepareForSegue:sender:方法，以便将任何所需数据传递给即将显示的视图控制器。

属性

Swift

```
sourceViewController: UIViewController {get}
destinationViewController: UIViewController {get}
identifier: String? {get}
```

参考文献：

- UIViewController 类参考
- UIStoryBoardSegue 类参考

Chapter 137: Segues

Section 137.1: Using Segues to navigate backwards in the navigation stack

Unwind Segues

> Unwind Segues give you a way to "unwind" the navigation stack and specify a destination to go back to. The signature of this function is key to Interface Builder recognizing it. **It must have a return value of IBAction and take one parameter of UIStoryboardSegue**. The name of the function does not matter. In fact, the function does not even have to do anything. It's just there as a marker of which UIViewController is the destination of the Unwind Segue. [source][1]

Required signature of an unwind segue

Objective C:

```
-(IBAction)prepareForUnwind:(UIStoryboardSegue *)segue {
}
```

Swift:

```
@IBAction func prepareForUnwind(segue: UIStoryboardSegue) {
}
```

Section 137.2: An Overview

From the Apple documentation:

> A UIStoryboardSegue object is responsible for **performing the visual transition between two view controllers**. In addition, segue objects are used to prepare for the transition from one view controller to another. **Segue objects contain information about the view controllers involved in a transition**. When a segue is triggered, but before the visual transition occurs, the storyboard runtime calls the current view controller's prepareForSegue:sender: method so that it can pass any needed data to the view controller that is about to be displayed.

Attributes

Swift

```
sourceViewController: UIViewController {get}
destinationViewController: UIViewController {get}
identifier: String? {get}
```

References:

- UIViewController Class Reference
- UIStoryBoardSegue Class Reference

第137.3节：在触发Segue之前准备你的视图控制器

PrepareForSegue:

```
func prepareForSegue(_ segue:UIStoryboardSegue, sender sender:AnyObject?)
```

> 通知视图控制器即将执行segue

参数

segue: segue对象。

sender: 初始化segue的对象。

Swift示例

如果segue的标识符是"SomeSpecificIdentifier"，则执行某个任务

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "SomeSpecificIdentifier" {
        //- 执行特定任务
    }
}
```

第137.4节：决定是否执行调用的Segue

ShouldPerformSegueWithIdentifier:

```
func shouldPerformSegueWithIdentifier(_ identifier:String, sender sender:AnyObject?) -> Bool
```

> 确定是否应该执行具有指定标识符的segue。

参数

Identifier: 触发segue的标识字符串

Sender: 初始化segue的对象。

Swift示例

仅当标识符为"SomeSpecificIdentifier"时执行segue

```
override func shouldPerformSegueWithIdentifier(identifier:String, sender:AnyObject?) -> Bool {
    if identifier == "SomeSpecificIdentifier" {
        return true
    }
    返回 false
}
```

第137.5节：以编程方式触发Segue

PerformSegueWithIdentifier:

Section 137.3: Preparing your view controller before a triggering a Segue

PrepareForSegue:

```
func prepareForSegue(_ segue:UIStoryboardSegue, sender sender:AnyObject?)
```

> Notifies the view controller that a segue is about to be performed

Parameters

segue: The segue object.

sender: The object that initialized the segue.

Example in Swift

Perform a task if the identifier of the segue is "SomeSpecificIdentifier"

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "SomeSpecificIdentifier" {
        //- Do specific task
    }
}
```

Section 137.4: Deciding if an invoked Segue should be performed

ShouldPerformSegueWithIdentifier:

```
func shouldPerformSegueWithIdentifier(_ identifier:String, sender sender:AnyObject?) -> Bool
```

> Determines whether the segue with the specified identifier should be performed.

Parameters

Identifier: String that identifies the triggered segue

Sender: The object that initialized the segue.

Example in Swift

Only perform segue if the identifier is "SomeSpecificIdentifier"

```
override func shouldPerformSegueWithIdentifier(identifier:String, sender:AnyObject?) -> Bool {
    if identifier == "SomeSpecificIdentifier" {
        return true
    }
    return false
}
```

Section 137.5: Trigger Segue Programmatically

PerformSegueWithIdentifier:

```
func performSegueWithIdentifier(_ identifier:String, sender sender:AnyObject?)
```

> 从当前视图控制器的故事板文件中启动具有指定标识符的segue

参数

标识符: 用于识别触发的segue的字符串

发送者: 将启动segue的对象。

Swift示例

从表视图行选择执行标识符为"SomeSpecificIdentifier"的segue：

```
func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
    performSegueWithIdentifier("SomeSpecificIdentifier", sender: indexPath.item)
}
```



```
func performSegueWithIdentifier(_ identifier:String, sender sender:AnyObject?)
```

> Initiates the segue with the specified identifier from the current view controller's storyboard file

Parameters

Identifier: String that identifies the triggered segue

Sender: The object that will initiate the segue.

Example in Swift

Performing a segue with identifier "SomeSpecificIdentifier" from a table view row selection:

```
func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
    performSegueWithIdentifier("SomeSpecificIdentifier", sender: indexPath.item)
}
```

第138章：EventKit

第138.1节：访问不同类型的日历

访问日历数组

要访问EKCalendar数组，我们使用calendarsForEntityType方法：

Swift

```
let calendarsArray = eventStore.calendarsForEntityType(EKEntityType.Event) as! [EKCalendar]
```

遍历日历

只需使用一个简单的for循环：

Swift

```
for calendar in calendarsArray{
    //...
}
```

访问日历标题和颜色

Swift

```
let calendarColor = UIColor(CGColor: calendar.CGColor)
let calendarTitle = calendar.title
```

Objective-C

```
UIColor *calendarColor = [UIColor initWithCGColor: calendar.CGColor];
NSString *calendarTitle = calendar.title;
```

第138.2节：请求权限

您的应用程序在未获得权限的情况下无法访问您的提醒事项和日历。相反，它必须向用户显示警告，请求用户授予应用访问事件的权限。

首先，导入EventKit框架：

Swift

```
import EventKit
```

Objective-C

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

创建一个EKEventStore

然后，我们创建一个EKEventStore对象。这个对象用于访问日历和提醒事项数据：

Swift

```
let eventStore = EKEventStore()
```

Objective-C

```
EKEventStore *eventStore = [[EKEventStore alloc] init];
```

> **注意**

Chapter 138: EventKit

Section 138.1: Accessing different types of calendars

Accessing the array of calendars

To access the array of `EKCalendars`, we use the `calendarsForEntityType` method:

Swift

```
let calendarsArray = eventStore.calendarsForEntityType(EKEntityType.Event) as! [EKCalendar]
```

Iterating through calendars

Just use a simple `for` loop:

Swift

```
for calendar in calendarsArray{
    //...
}
```

Accessing the calendar title and color

Swift

```
let calendarColor = UIColor(CGColor: calendar.CGColor)
let calendarTitle = calendar.title
```

Objective-C

```
UIColor *calendarColor = [UIColor initWithCGColor: calendar.CGColor];
NSString *calendarTitle = calendar.title;
```

Section 138.2: Requesting Permission

Your app can't access your reminders and your calendar without permission. Instead, it must show an alert to user, requesting him/her to grant access to events for the app.

To get started, import the `EventKit` framework:

Swift

```
import EventKit
```

Objective-C

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

Making an `EKEventStore`

Then, we make an `EKEventStore` object. This is the object from which we can access calendar and reminders data:

Swift

```
let eventStore = EKEventStore()
```

Objective-C

```
EKEventStore *eventStore = [[EKEventStore alloc] init];
```

> **Note**

> 每次需要访问日历时都创建一个EKEventStore对象效率不高。尽量只创建一次并在代码中到处使用它。

检查可用性

可用性有三种不同状态：已授权、已拒绝和未确定。未确定表示应用程序需要请求访问权限。

要检查可用性，我们使用EKEventStore对象的authorizationStatusForEntityType()方法：

Swift

```
switch EKEventStore.authorizationStatusForEntityType(EKEntityTypeEvent){
    case .Authorized: //...
    case .Denied: //...
    case .NotDetermined: //...
    default: break
}
```

Objective-C

```
switch ([EKEventStore authorizationStatusForEntityType:EKEntityTypeEvent]){
    case EKAuthorizationStatus.Authorized:
        //...
        break;
    case EKAuthorizationStatus.Denied:
        //...
        break;
    case EKAuthorizationStatus.NotDetermined:
        //...
        break;
    default:
        break;
}
```

请求权限

将以下代码放入NotDetermined情况：

Swift

```
eventStore.requestAccessToEntityType(EKEntityTypeEvent, completion: { [weak self]
(userGrantedAccess, _) -> Void in
    if userGrantedAccess{
        //访问日历
    }
}
```

第138.3节：添加事件

创建事件对象

Swift

```
var event = EKEvent(eventStore: eventStore)
```

Objective-C

```
EKEvent *event = [EKEvent initWithEventStore:eventStore];
```

设置相关日历、标题和日期

Swift

> Making an `EKEventStore` object every time we need to access calendar is not efficient. Try to make it once and use it everywhere in your code.

Checking Availability

Availability has three different status: Authorized, Denied and Not Determined. Not Determined means the app needs to grant access.

To check availability, we use `authorizationStatusForEntityType()` method of the `EKEventStore` object:

Swift

```
switch EKEventStore.authorizationStatusForEntityType(EKEntityTypeEvent){
    case .Authorized: //...
    case .Denied: //...
    case .NotDetermined: //...
    default: break
}
```

Objective-C

```
switch ([EKEventStore authorizationStatusForEntityType:EKEntityTypeEvent]){
    case EKAuthorizationStatus.Authorized:
        //...
        break;
    case EKAuthorizationStatus.Denied:
        //...
        break;
    case EKAuthorizationStatus.NotDetermined:
        //...
        break;
    default:
        break;
}
```

Requesting Permission

Put the following code in `NotDetermined` case:

Swift

```
eventStore.requestAccessToEntityType(EKEntityTypeEvent, completion: { [weak self]
(userGrantedAccess, _) -> Void in
    if userGrantedAccess{
        //access calendar
    }
}
```

Section 138.3: Adding an event

Creating the event object

Swift

```
var event = EKEvent(eventStore: eventStore)
```

Objective-C

```
EKEvent *event = [EKEvent initWithEventStore:eventStore];
```

Setting related calendar, title and dates

Swift

```
event.calendar = calendar
event.title = "事件标题"
event.startDate = startDate //假设startDate是有效的NSDate对象
event.endDate = endDate //假设endDate是有效的NSDate对象
```

将事件添加到日历

Swift

```
try {
    do eventStore.saveEvent(event, span: EKSpan.ThisEvent)
} catch let error as NSError {
    //错误
}
```

Objective-C

```
NSError *error;
BOOL *result = [eventStore saveEvent:event span:EKSpanThisEvent error:&error];
if (result == NO){
    //错误
}
```



```
event.calendar = calendar
event.title = "Event Title"
event.startDate = startDate //assuming startDate is a valid NSDate object
event.endDate = endDate //assuming endDate is a valid NSDate object
```

Adding event to calendar

Swift

```
try {
    do eventStore.saveEvent(event, span: EKSpan.ThisEvent)
} catch let error as NSError {
    //error
}
```

Objective-C

```
NSError *error;
BOOL *result = [eventStore saveEvent:event span:EKSpanThisEvent error:&error];
if (result == NO){
    //error
}
```

第139章：SiriKit

第139.1节：向应用添加Siri扩展

要在您的应用中集成Siri功能，您应当像创建iOS 10
小组件（旧版今日视图扩展）或自定义键盘时那样添加扩展。

增加能力

1- 在项目设置中，选择你的 iOS 应用目标并进入功能（Capabilities）标签页

2- 启用 Siri 功能

添加扩展

1- 进入 文件 -> 新建 -> 目标...

2- 从左侧面板选择 iOS -> 应用扩展

3- 双击右侧的 Intents 扩展

> **根据苹果公司：**
>
> Intents 扩展模板构建了一个 Intents 扩展，允许你的应用处理由系统服务（如 Siri 和地图）发出的意图。

Chapter 139: SiriKit

Section 139.1: Adding Siri Extension to App

To integrate Siri capabilities in your app, you should add an extensions as you would do while creating an iOS 10 Widget (old Today View Extension) or a custom keyboard.

Adding capability

1- In the project settings, select your iOS app target and go to Capabilities tab

2- Enable the Siri capability

Adding the extension

1- Go to File -> New -> Target...

2- Select iOS -> Application Extension from the left pane

3- Double-click Intents Extension from right

> **According to Apple:**
>
> Intents Extension template builds an Intents extension that allows your app to handle intents issued by system services like Siri and Maps.

4- 选择一个名称，并确保勾选“包含 UI 扩展”

完成这些步骤后，会创建两个新目标（Intents 扩展和 UI 扩展），默认包含锻炼意图代码。有关不同类型的 Siri 请求，请参见备注。

> **注意**
>
> 每当你想调试扩展时，只需从可用方案中选择 Intent 方案。

> **注意**
>
> 你无法在模拟器中测试 SiriKit 应用。你需要一台真实设备。

4- Choose a name, and be sure to check "Include UI Extension"

By doing this steps, two new targets (Intents Extension and UI Extension) are created, and by default they contain Workout Intent code. For different types of Siri requests, see Remarks.

> **Note**
>
> Anytime you want to debug your extension, just select the Intent scheme from the available schemes.

> **Note**
>
> You can't test SiriKit apps in the Simulator. Instead, you need a real device.

第140章：联系人框架

第140.1节：添加联系人

Swift

```
import Contacts

// 创建一个可变对象以添加联系人
let contact = CNMutableContact()

contact.imageData = NSData() // 作为NSData对象的头像图片

contact.givenName = "约翰"
contact.familyName = "苹果种子"

let homeEmail = CNLabeledValue(label:CNLabelHome, value:"john@example.com")
let workEmail = CNLabeledValue(label:CNLabelWork, value:"j.appleseed@icloud.com")
contact.emailAddresses = [homeEmail, workEmail]

contact.phoneNumbers = [CNLabeledValue(
    label:CNLabelPhoneNumberiPhone,
     value:CNPhoneNumber(stringValue:"(408) 555-0126"))]

let homeAddress = CNMutablePostalAddress()
homeAddress.street = "无限循环路1号"
homeAddress.city = "库比蒂诺"
homeAddress.state = "CA"
homeAddress.postalCode = "95014"
contact.postalAddresses = [CNLabeledValue(label:CNLabelHome, value:homeAddress)]

let birthday = NSDateComponents()
birthday.day = 1
birthday.month = 4
birthday.year = 1988  // 你可以省略年份值以表示无年份生日
contact.birthday = birthday

// 保存新创建的联系人
let store = CNContactStore()
let saveRequest = CNSaveRequest()
saveRequest.addContact(contact, toContainerWithIdentifier:nil)
try! store.executeSaveRequest(saveRequest)
```

第140.2节：授权联系人访问权限

导入框架

Swift

```
import Contacts
```

Objective-C

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

检查无障碍

Swift

```
switch CNContactStore.authorizationStatusForEntityType(CNEntityType.Contacts){
case .Authorized: //访问联系人
case .Denied, .NotDetermined: //请求权限
default: break
```


Chapter 140: Contacts Framework

Section 140.1: Adding a Contact

Swift

```
import Contacts

// Creating a mutable object to add to the contact
let contact = CNMutableContact()

contact.imageData = NSData() // The profile picture as a NSData object

contact.givenName = "John"
contact.familyName = "Appleseed"

let homeEmail = CNLabeledValue(label:CNLabelHome, value:"john@example.com")
let workEmail = CNLabeledValue(label:CNLabelWork, value:"j.appleseed@icloud.com")
contact.emailAddresses = [homeEmail, workEmail]

contact.phoneNumbers = [CNLabeledValue(
    label:CNLabelPhoneNumberiPhone,
    value:CNPhoneNumber(stringValue:"(408) 555-0126"))]

let homeAddress = CNMutablePostalAddress()
homeAddress.street = "1 Infinite Loop"
homeAddress.city = "Cupertino"
homeAddress.state = "CA"
homeAddress.postalCode = "95014"
contact.postalAddresses = [CNLabeledValue(label:CNLabelHome, value:homeAddress)]

let birthday = NSDateComponents()
birthday.day = 1
birthday.month = 4
birthday.year = 1988  // You can omit the year value for a yearless birthday
contact.birthday = birthday

// Saving the newly created contact
let store = CNContactStore()
let saveRequest = CNSaveRequest()
saveRequest.addContact(contact, toContainerWithIdentifier:nil)
try! store.executeSaveRequest(saveRequest)
```

Section 140.2: Authorizing Contact Access

Importing the framework

Swift

```
import Contacts
```

Objective-C

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

Checking accessibility

Swift

```
switch CNContactStore.authorizationStatusForEntityType(CNEntityType.Contacts){
case .Authorized: //access contacts
case .Denied, .NotDetermined: //request permission
default: break
```

```
}
```

Objective-C

```
switch ([CNContactStore authorizationStatusForEntityType:CNEntityType.Contacts]){
case CNAuthorizationStatus.Authorized:
    //访问联系人
    break;
case CNAuthorizationStatus.Denied:
    //请求权限
    break;
case CNAuthorizationStatus.NotDetermined:
    //请求权限
    break;
}
```

请求权限

Swift

```
var contactStore = CKContactStore()
contactStore.requestAccessForEntityType(CKEntityType.Contacts, completionHandler: { (ok, _) -> Void
in
    if access{
        //访问联系人
    }
}
```

第140.3节：访问联系人

应用过滤器

要访问联系人，我们应该对之前在授权联系人访问示例中定义的contactStore变量应用类型为NSPredicate的过滤器。例如，这里我们想筛选出姓名与我们自己匹配的联系人：

Swift

```
let predicate = CNContact.predicateForContactsMatchingName("Some Name")
```

Objective-C

```
NSPredicate *predicate = [CNContact predicateForContactsMatchingName:@"Some Name"];
```

指定要获取的键

这里，我们想获取联系人的名字、姓氏和头像：

Swift

```
let keys = [CNContactGivenNameKey, CNContactFamilyNameKey, CNContactImageDataKey]
```

获取联系人

Swift

```
do {
    let contacts = try contactStore.unifiedContactsMatchingPredicate(predicate, keysToFetch: keys)
} catch let error as NSError {
    //...
}
```

访问联系人详情

Swift

```
print(contacts[0].givenName)
```



```
}
```

Objective-C

```
switch ([CNContactStore authorizationStatusForEntityType:CNEntityType.Contacts]){
case CNAuthorizationStatus.Authorized:
    //access contacts
    break;
case CNAuthorizationStatus.Denied:
    //request permission
    break;
case CNAuthorizationStatus.NotDetermined:
    //request permission
    break;
}
```

Requesting Permission

Swift

```
var contactStore = CKContactStore()
contactStore.requestAccessForEntityType(CKEntityType.Contacts, completionHandler: { (ok, _) -> Void
in
    if access{
        //access contacts
    }
}
```

Section 140.3: Accessing Contacts

Applying a filter

To access contacts, we should apply a filter of type `NSPredicate` to our contactStore variable which we defined it in Authorizing Contact Access example. For example, here we want to sort out contacts with name matching with our own:

Swift

```
let predicate = CNContact.predicateForContactsMatchingName("Some Name")
```

Objective-C

```
NSPredicate *predicate = [CNContact predicateForContactsMatchingName:@"Some Name"];
```

Specifying keys to fetch

Here, we want to fetch the contact's first name, last name and profile image:

Swift

```
let keys = [CNContactGivenNameKey, CNContactFamilyNameKey, CNContactImageDataKey]
```

Fetching contacts

Swift

```
do {
    let contacts = try contactStore.unifiedContactsMatchingPredicate(predicate, keysToFetch: keys)
} catch let error as NSError {
    //...
}
```

Accessing contact details

Swift

```
print(contacts[0].givenName)
```

```
print(contacts[1].familyName)
let image = contacts[2].imageData
```



```
print(contacts[1].familyName)
let image = contacts[2].imageData
```

第141章：iOS 10语音识别API

第141.1节：语音转文本：识别捆绑包中包含的音频录音的语音

```
//import Speech
//import AVFoundation

// 创建一个文本框以显示语音输出
@IBOutlet weak var transcriptionTextField: UITextView!
// 我们需要这个音频播放器来播放音频
var audioPlayer: AVAudioPlayer!

override func viewDidLoad()
{
    super.viewDidLoad()
}

// 该函数用于在音频播放完成时停止音频，否则会重复播放同一音频

func audioPlayerDidFinishPlaying(_ player: AVAudioPlayer, successfully flag: Bool)
{
player.stop()
}

// 该函数用于获取语音识别器，之后向语音识别器发起请求

func requestSpeechAuth()
{
SFSpeechRecognizer.requestAuthorization { authStatus in
        if authStatus == SFSpeechRecognizerAuthorizationStatus.authorized {
            if let path = Bundle.main.url(forResource: "mpthreetest", withExtension: "m4a") {
                do {
                    let sound = try AVAudioPlayer(contentsOf: path)
                    self.audioPlayer = sound
                    self.audioPlayer.delegate = self
                    sound.play()
                }  catch {
                    print("error")
                }

                let recognizer = SFSpeechRecognizer()
                let request = SFSpeechURLRecognitionRequest(url:path)
                recognizer?.recognitionTask(with: request) { (result, error) in
                    if let error = error {
                    print("there is a error\(error)")
                    } else {
// here you are printing out the audio output basically showing it on uitext field
                        self.transcriptionTextField.text =
result?.bestTranscription.formattedString
                    }
                }
            }
        }
    }
}

// 这里你在 UIButton 按下时调用 requestSpeechAuth 函数
@IBAction func playButtonPress(_ sender: AnyObject)
{
```


Chapter 141: iOS 10 Speech Recognition API

Section 141.1: Speech to text: Recognize speech from a bundle contained audio recording

```
//import Speech
//import AVFoundation

// create a text field to show speech output
@IBOutlet weak var transcriptionTextField: UITextView!
// we need this audio player to play audio
var audioPlayer: AVAudioPlayer!

override func viewDidLoad()
{
    super.viewDidLoad()
}

// this function is required to stop audio on audio completion otherwise it will play same audio
again and again
func audioPlayerDidFinishPlaying(_ player: AVAudioPlayer, successfully flag: Bool)
{
    player.stop()
}

// this function is required to get a speech recognizer and after that make and request to speech
recognizer
func requestSpeechAuth()
{
    SFSpeechRecognizer.requestAuthorization { authStatus in
        if authStatus == SFSpeechRecognizerAuthorizationStatus.authorized {
            if let path = Bundle.main.url(forResource: "mpthreetest", withExtension: "m4a") {
                do {
                    let sound = try AVAudioPlayer(contentsOf: path)
                    self.audioPlayer = sound
                    self.audioPlayer.delegate = self
                    sound.play()
                }  catch {
                    print("error")
                }

                let recognizer = SFSpeechRecognizer()
                let request = SFSpeechURLRecognitionRequest(url:path)
                recognizer?.recognitionTask(with: request) { (result, error) in
                    if let error = error {
                    print("there is a error\(error)")
                    } else {
// here you are printing out the audio output basically showing it on uitext field
                        self.transcriptionTextField.text =
result?.bestTranscription.formattedString
                    }
                }
            }
        }
    }
}

// here you are calling requestSpeechAuth function on UIButton press
@IBAction func playButtonPress(_ sender: AnyObject)
{
```

```
requestSpeechAuth()
}
```



```
    requestSpeechAuth()
}
```

第142章：StoreKit

第142.1节：从App

Store获取本地化的产品信息

使用SKProductsRequest从一组产品标识符字符串中获取本地化的产品信息：

```
import StoreKit

let productIdentifierSet = Set(["yellowSubmarine", "pennyLane"])
let productsRequest = SKProductsRequest(productIdentifiers: productIdentifierSet)
```

为了处理来自productsRequest的产品，我们需要为请求分配一个代理来处理响应。该代理需要遵循SKProductsRequestDelegate协议，这意味着它必须继承自NSObject（即任何Foundation对象）并实现productsRequest方法：

```
class PaymentManager: NSObject, SKProductsRequestDelegate {

    var products: [SKProduct] = []

    func productsRequest(request: SKProductsRequest,
                         didReceiveResponse response: SKProductsResponse) {

        products = response.products

    }

}
```

为了启动productsRequest，我们将PaymentManager设置为产品请求的代理，并调用请求的start()方法：

```
let paymentManager = PaymentManager()
productsRequest.delegate = paymentManager
productsRequest.start()
```

如果请求成功，产品将会在paymentManager.products中。

Chapter 142: StoreKit

Section 142.1: Get localized product information from the App Store

Get localized product information from a set of product identifier strings using `SKProductsRequest`:

```
import StoreKit

let productIdentifierSet = Set(["yellowSubmarine", "pennyLane"])
let productsRequest = SKProductsRequest(productIdentifiers: productIdentifierSet)
```

In order to process the products from the `productsRequest`, we need to assign a delegate to the request that handles the response. The delegate needs to conform to the `SKProductsRequestDelegate` protocol, which means that it must inherit from `NSObject` (i.e. any `Foundation` object) and implement the `productsRequest` method:

```
class PaymentManager: NSObject, SKProductsRequestDelegate {

    var products: [SKProduct] = []

    func productsRequest(request: SKProductsRequest,
                         didReceiveResponse response: SKProductsResponse) {

        products = response.products

    }

}
```

To initiate the `productsRequest` we assign `PaymentManager` as the products-request's delegate, and call the `start()` method on the request:

```
let paymentManager = PaymentManager()
productsRequest.delegate = paymentManager
productsRequest.start()
```

If the requests succeeds the products will be in `paymentManager.products`.

第143章：代码签名

第143.1节：配置描述文件

为了在XCode中构建IPA文件，您需要使用证书和配置描述文件对应用进行签名。这些可以在 https://developer.apple.com/account/ios/profile/create 创建

配置描述文件类型

配置描述文件分为两种类型，开发和发布：

开发

- iOS 应用开发 / tvOS 应用开发 - 用于开发过程中将您的应用安装到测试设备上。

分发

- App Store / tvOS App Store - 用于为您的应用签名以便上传到应用商店。
- 企业内部分发 - 用于企业内部将您的应用分发到公司内的设备。
- Ad Hoc / tvOS Ad Hoc - 用于将您的应用分发到有限数量的特定设备（例如，您需要知道想要安装应用的设备的 UDID）。

Chapter 143: Code signing

Section 143.1: Provisioning Profiles

In order to build an IPA file in XCode, you require to sign your application with a certificate and provisioning profile. These can be created at https://developer.apple.com/account/ios/profile/create

Provisioning Profile Types

Provisioning Profiles are split into two types, Development, and Distribution:

Development

- iOS App Development / tvOS App Development - used in development in order to install your app onto a test device.

Distribution

- App Store / tvOS App Store - used to sign your application for app store upload.
- In House - Used for Enterprise distribution of your app, to devices within your business.
- Ad Hoc / tvOS Ad Hoc - Used to distribute your app to a limited number of specific devices (e.g. you will need to know the UDIDs of the devices you want to install your app to).

第144章：使用 Applicationloader 创建上传到 App Store 的 .ipa 文件

第144.1节：使用

Application Loader 创建上传到 App Store 的 .ipa 文件

如果您想上传 .ipa 文件到 iTunes Connect而不想在 Xcode 中集成开发者账号，并且想使用Application Loader，那么您可以使用 iTunes 生成 .ipa 文件。

步骤 1： 选择设备而非模拟器。

步骤 2： 进入产品 -> 选择归档

Chapter 144: Create .ipa File to upload on appstore with Applicationloader

Section 144.1: create .ipa file to upload app to appstore with Application Loader

If you want to upload .ipa file to itunesconnect **without integrating developer account in Xcode** and you want to use **application loader**. then you can **generate .ipa with iTunes** .

Step 1: Select device inplace of simulator.

Step 2: Go to Product -> select Archive

步骤 3： 过程完成后，右键点击你的归档 -> 选择在 Finder 中显示

步骤 4： 点击"在 Finder 中显示"后，你将被重定向到归档文件夹，界面如下所示

步骤 5： 右键点击 .xarchive 文件 -> 选择"在 Finder 中显示"选项。

Step 3: After complited process right click to your Archive -> and select show in Finder

Step 4: when you click on show in finder you will redirect to Archive folder, looks like this

Step 5: Right click on .xarchive file -> select Show in finder option.

步骤 6： 进入产品文件夹 -> 应用程序文件夹 -> 你会找到 yourprojectname.app

步骤 7： 现在将 .app 转换为 .ipa，只需将其拖放到 iTunes 中。请查看下图，

Step 6: Go to Product Folder -> Application Folder -> You will find yourprojectname.app

Step 7: Now to convert .app to .ipa just drag and drop into itunes . check below image ,

步骤8： 现在将此 .ipa 文件放在安全的地方，上传时使用应用程序加载器。

注意： 如果你想了解如何使用应用加载器上传应用，请查看此内容，

使用应用加载器上传应用

编辑：

警告： 不要通过将扩展名从 .aap 改为 .zip 再从 .zip 改为 .ipa 来制作 .ipa 文件。

我在许多回答中看到，有人建议先压缩 .app 文件，然后将扩展名从 .zip 改为 .ipa 。这种方法现在已经不行了。使用这种方法你会遇到如下错误，

> IPA 无效，未包含 payload 目录。

Step 8: Now put this .ipa file in safe place and use when upload with application loader .

Note: if you want to know how to upload app with application loader then check this ,

Upload app with application Loader

EDIT:

WARNING: Don't make .ipa with changing extension from .aap to .zip and .zip to .ipa.

I have seen in many answer that , they have suggest compress .app file and then change the extension from .zip to .ipa . It is not working now . By this method you will get Error like ,

> IPA is invalid, it does not include a payload directory.

第145章：尺寸类别和自适应

第145.1节：通过Storyboard实现尺寸类别和自适应

我们可以为任何 UIView 的子类添加自适应，该子类是在 nib 文件中添加到视图控制器上的。
让我们以使用尺寸类别为视图添加自适应为例。

1. 在视图控制器中添加一个视图，如下所示：

2. 现在我们需要将该视图固定到其父视图，以使用约束来确定其大小和位置，如下所示：

Chapter 145: Size Classes and Adaptivity

Section 145.1: Size Classes and Adaptivity through Storyboard

We can add adaptivity to any subclass of `UIView` which we add on view controller in nib file.
Lets take an example of adding adaptivity using size classes to a view.

1. Add a view on view controller as:

2. Now we need to pin this view to it's superview for fixing it's size and position using **constraints** as:

3. 我们可以看到添加的约束如下：

3. We can see the added constraints as:

这些约束定义了添加的视图将在其父视图中放置为

```
CGRect(20, 0, superview.width - 20, superview.height - 20)
```

4. 要在屏幕上查看这些添加的约束的预览，我们可以使用辅助编辑器，如下所示；

These constraints defines that the added view will be placed in it's superview as

```
CGRect(20, 0, superview.width - 20, superview.height - 20)
```

4. To see the preview on screen of these added constraints we can use **Assistant Editor** as;

5. 我们可以添加更多屏幕以查看预览，如下所示：

5. We can add more screen to see preview like:

我们还可以通过将鼠标移动到设备名称上并点击旋转按钮来查看横屏模式的预览，如下所示：

We can also see the preview with landscape mode by moving mouse on the name of device and clicking the rotation button as:

iPhone 4-inch

iPhone 4-inch

第146章：MKDistanceFormatter

第146.1节：从距离生成字符串

给定一个CLLocationDistance（简单来说是表示米数的Double），输出一个用户可读的字符串：

```
let distance = CLLocationDistance(42)
let formatter = MKDistanceFormatter()
let answer = formatter.stringFromDistance(distance)
// answer = "150 feet"
```

Objective-C

```
CLLocationDistance distance=42;
MKDistanceFormatter *formatter=[[MKDistanceFormatter alloc]init];
NSString *answer=[formatter stringFromDistance:distance];
// answer = "150 feet"
```

默认情况下，这会遵循用户的区域设置。

第146.2节：距离单位

import Mapkit 将 units 设置为以下之一.Default，.Metric，.Imperial，.ImperialWithYards：

```
formatter.units = .Metric
var answer = formatter.stringFromDistance(distance)
// "40 m"

formatter.units = .ImperialWithYards
answer = formatter.stringFromDistance(distance)
// "50 yards"
```

Objective-C

```
MKDistanceFormatter *formatter=[[MKDistanceFormatter alloc]init];
formatter.units=MKDistanceFormatterUnitsMetric;
NSString *answer=[formatter stringFromDistance:distance];
//40 m

formatter.units=MKDistanceFormatterUnitsImperialWithYards;
NSString *answer=[formatter stringFromDistance:distance];
//50 yards
```

第146.3节：单位样式

将unitStyle设置为以下之一：.Default、.Abbreviated、.Full：

```
formatter.unitStyle = .Full
var answer = formatter.stringFromDistance(distance)
// "150 英尺"

formatter.unitStyle = .Abbreviated
answer = formatter.stringFromDistance(distance)
// "150 英尺"
```


Chapter 146: MKDistanceFormatter

Section 146.1: String from distance

Given a `CLLocationDistance` (simply a `Double` representing meters), output a user-readable string:

```
let distance = CLLocationDistance(42)
let formatter = MKDistanceFormatter()
let answer = formatter.stringFromDistance(distance)
// answer = "150 feet"
```

Objective-C

```
CLLocationDistance distance=42;
MKDistanceFormatter *formatter=[[MKDistanceFormatter alloc]init];
NSString *answer=[formatter stringFromDistance:distance];
// answer = "150 feet"
```

By default, this respects the user's locale.

Section 146.2: Distance units

`import Mapkit` Set `units` to one of `.Default`, `.Metric`, `.Imperial`, `.ImperialWithYards`:

```
formatter.units = .Metric
var answer = formatter.stringFromDistance(distance)
// "40 m"

formatter.units = .ImperialWithYards
answer = formatter.stringFromDistance(distance)
// "50 yards"
```

Objective-C

```
MKDistanceFormatter *formatter=[[MKDistanceFormatter alloc]init];
formatter.units=MKDistanceFormatterUnitsMetric;
NSString *answer=[formatter stringFromDistance:distance];
//40 m

formatter.units=MKDistanceFormatterUnitsImperialWithYards;
NSString *answer=[formatter stringFromDistance:distance];
//50 yards
```

Section 146.3: Unit style

Set `unitStyle` to one of `.Default`, `.Abbreviated`, `.Full`:

```
formatter.unitStyle = .Full
var answer = formatter.stringFromDistance(distance)
// "150 feet"

formatter.unitStyle = .Abbreviated
answer = formatter.stringFromDistance(distance)
// "150 ft"
```

Objective-C

```
formatter.unitStyle=MKDistanceFormatterUnitStyleFull;
NSString *answer=[formatter stringFromDistance:distance];
// "150 英尺"

formatter.unitStyle=MKDistanceFormatterUnitStyleAbbreviated;
 NSString *answer=[formatter stringFromDistance:distance];
// "150 英尺"
```


Objective-C

```
formatter.unitStyle=MKDistanceFormatterUnitStyleFull;
NSString *answer=[formatter stringFromDistance:distance];
// "150 feet"

 formatter.unitStyle=MKDistanceFormatterUnitStyleAbbreviated;
 NSString *answer=[formatter stringFromDistance:distance];
// "150 ft"
```

第147章：3D触控

第147.1节：使用Swift的3D触控

3D触控首次出现在iPhone 6s Plus中。这个新的界面层增加了两种交互行为：Peek和Pop。

Peek 和 Pop 简要说明

Peek - 用力按压

Pop - 更用力按压

检查是否支持 3D 触控

你应该检查设备是否支持 3D 触控。你可以通过检查 UITraitCollection 对象的 forceTouchCapability 属性值来实现。UITraitCollection 描述了你的应用的 iOS 界面环境。

```
if (traitCollection.forceTouchCapability == .Available) {
    registerForPreviewingWithDelegate(self, sourceView: view)
}
```

实现代理

你需要在你的类中实现 UIViewControllerPreviewingDelegate 的两个方法。其中一个方法用于 peek，另一个用于 pop 行为。

用于 peek 的方法是 previewingContext。

```
func previewingContext(previewingContext: UIViewControllerPreviewing, viewControllerForLocation
location: CGPoint) -> UIViewController? {

guard let indexPath = self.tableView.indexPathForRowAtPoint(location), cell =
self.tableView.cellForRowAtIndexPath(indexPath) as? <YourTableViewCell> else {
        return nil
    }

guard let datailVC =
storyboard?.instantiateViewControllerWithIdentifier("<YourViewControllerIdentifier>") as?
<YourViewController> else {
        return nil
    }

datailVC.peekActive = true
    previewingContext.sourceRect = cell.frame

    // 执行操作

    return datailVC

}
```

实现pop方法的是previewingContext。:)

```
func previewingContext(previewingContext: UIViewControllerPreviewing, commitViewController
viewControllerToCommit: UIViewController) {
```


Chapter 147: 3D Touch

Section 147.1: 3D Touch with Swift

3D touch has been introduced with iPhone 6s Plus. There are two behaviors added with this new interface layer: Peek and Pop.

Peek and Pop in a nutshell

Peek - Press hard

Pop - Press really hard

Checking for 3D support

You should check if the device has a 3D touch support. You can do this by checking the value of *forceTouchCapability* property of a *UITraitCollection* object. UITraitCollection describes the iOS interface environment for your app.

```
if (traitCollection.forceTouchCapability == .Available) {
    registerForPreviewingWithDelegate(self, sourceView: view)
}
```

Implementing the delegate

You need to implement the two methods of *UIViewControllerPreviewingDelegate* in your class. One of the methods is for *peek* and the other one is for *pop* behavior.

The method to be implemented for the *peek* is *previewingContext*.

```
func previewingContext(previewingContext: UIViewControllerPreviewing, viewControllerForLocation
location: CGPoint) -> UIViewController? {

    guard let indexPath = self.tableView.indexPathForRowAtPoint(location), cell =
self.tableView.cellForRowAtIndexPath(indexPath) as? <YourTableViewCell> else {
        return nil
    }

    guard let datailVC =
storyboard?.instantiateViewControllerWithIdentifier("<YourViewControllerIdentifier>") as?
<YourViewController> else {
        return nil
    }

    datailVC.peekActive = true
    previewingContext.sourceRect = cell.frame

    // Do the stuff

    return datailVC

}
```

The method to be implemented for the *pop* is *previewingContext*. :)

```
func previewingContext(previewingContext: UIViewControllerPreviewing, commitViewController
viewControllerToCommit: UIViewController) {
```

```
    let balanceViewController = viewControllerToCommit as! <YourViewController>

    // 执行操作

navigationController?.pushViewController(balanceViewController, animated: true)

}
```

如你所见，它们是重载方法。你可以通过实现这些方法以任何方式使用3D触控。

Objective-C

```
// 检查3D Touch是否可用
if ([self.traitCollection respondsToSelector:@selector(forceTouchCapability)] &&
        (self.traitCollection.forceTouchCapability == UIForceTouchCapabilityAvailable))
    {
        [self registerForPreviewingWithDelegate:self sourceView:self.view];
    }
// 预览（Peek）
- (UIViewController *)previewingContext:(id<UIViewControllerPreviewing>)previewingContext
              viewControllerForLocation:(CGPoint)location {

    NSIndexPath *indexPath = [self.tableView indexPathForRowAtPoint:location];
    Country *country = [self countryForIndexPath:indexPath];
    if (country) {
CountryCell *cell = [self.tableView cellForRowAtIndexPath:indexPath];
        if (cell) {
previewingContext.sourceRect = cell.frame;
            UINavigationController *navController = [self.storyboard
instantiateViewControllerWithIdentifier:@"UYLCountryNavController"];
            [self configureNavigationController:navController withCountry:country];
            return navController;
        }
    }
    return nil;
}
// 弹出（Pop）
- (void)previewingContext:(id<UIViewControllerPreviewing>)previewingContext
commitViewController:(UIViewController *)viewControllerToCommit {

    [self showDetailViewController:viewControllerToCommit sender:self];
}
```

第147.2节：3D Touch Objective-C示例

Objective-C

```
// 检查3D Touch是否可用
if ([self.traitCollection respondsToSelector:@selector(forceTouchCapability)] &&
        (self.traitCollection.forceTouchCapability == UIForceTouchCapabilityAvailable))
    {
        [self registerForPreviewingWithDelegate:self sourceView:self.view];
    }
// 预览（Peek）
- (UIViewController *)previewingContext:(id<UIViewControllerPreviewing>)previewingContext
              viewControllerForLocation:(CGPoint)location {

    NSIndexPath *indexPath = [self.tableView indexPathForRowAtPoint:location];
    Country *country = [self countryForIndexPath:indexPath];
    if (country) {
```



```
    let balanceViewController = viewControllerToCommit as! <YourViewController>

    // Do the stuff

    navigationController?.pushViewController(balanceViewController, animated: true)

}
```

As you can see they are overloaded methods. You can use 3D touch in any way implementing these methods.

Objective-C

```
//Checking for 3-D Touch availability
if ([self.traitCollection respondsToSelector:@selector(forceTouchCapability)] &&
        (self.traitCollection.forceTouchCapability == UIForceTouchCapabilityAvailable))
    {
        [self registerForPreviewingWithDelegate:self sourceView:self.view];
    }
//Peek
- (UIViewController *)previewingContext:(id<UIViewControllerPreviewing>)previewingContext
              viewControllerForLocation:(CGPoint)location {

    NSIndexPath *indexPath = [self.tableView indexPathForRowAtPoint:location];
    Country *country = [self countryForIndexPath:indexPath];
    if (country) {
        CountryCell *cell = [self.tableView cellForRowAtIndexPath:indexPath];
        if (cell) {
            previewingContext.sourceRect = cell.frame;
            UINavigationController *navController = [self.storyboard
instantiateViewControllerWithIdentifier:@"UYLCountryNavController"];
            [self configureNavigationController:navController withCountry:country];
            return navController;
        }
    }
    return nil;
}
//Pop
- (void)previewingContext:(id<UIViewControllerPreviewing>)previewingContext
commitViewController:(UIViewController *)viewControllerToCommit {

    [self showDetailViewController:viewControllerToCommit sender:self];
}
```

Section 147.2: 3 D Touch Objective-C Example

Objective-C

```
//Checking for 3-D Touch availability
if ([self.traitCollection respondsToSelector:@selector(forceTouchCapability)] &&
        (self.traitCollection.forceTouchCapability == UIForceTouchCapabilityAvailable))
    {
        [self registerForPreviewingWithDelegate:self sourceView:self.view];
    }
//Peek
- (UIViewController *)previewingContext:(id<UIViewControllerPreviewing>)previewingContext
              viewControllerForLocation:(CGPoint)location {

    NSIndexPath *indexPath = [self.tableView indexPathForRowAtPoint:location];
    Country *country = [self countryForIndexPath:indexPath];
    if (country) {
```

```
CountryCell *cell = [self.tableView cellForRowAtIndexPath:indexPath];
        if (cell) {
previewingContext.sourceRect = cell.frame;
            UINavigationController *navController = [self.storyboard
instantiateViewControllerWithIdentifier:@"UYLCountryNavController"];
            [self configureNavigationController:navController withCountry:country];
            return navController;
        }
    }
    return nil;
}
// 弹出（Pop）
- (void)previewingContext:(id<UIViewControllerPreviewing>)previewingContext
commitViewController:(UIViewController *)viewControllerToCommit {

    [self showDetailViewController:viewControllerToCommit sender:self];
}
```



```
        CountryCell *cell = [self.tableView cellForRowAtIndexPath:indexPath];
        if (cell) {
            previewingContext.sourceRect = cell.frame;
            UINavigationController *navController = [self.storyboard
instantiateViewControllerWithIdentifier:@"UYLCountryNavController"];
            [self configureNavigationController:navController withCountry:country];
            return navController;
        }
    }
    return nil;
}
//Pop
- (void)previewingContext:(id<UIViewControllerPreviewing>)previewingContext
commitViewController:(UIViewController *)viewControllerToCommit {

    [self showDetailViewController:viewControllerToCommit sender:self];
}
```

第148章：GameCenter排行榜

第148.1节：GameCenter排行榜

先决条件：

1.Apple开发者账号
2.通过iTunesConnect设置GameCenter排行榜

设置GameCenter排行榜：

1. *登录 iTunesConnect*
2. 进入我的应用。为您的项目创建一个应用，然后进入功能。
3. *点击游戏中心*
4. 点击排行榜旁边的加号。
5. 选择单一排行榜作为排行榜类型。
6. 创建一个排行榜参考名称以供参考。
7. 为您的应用创建一个排行榜ID，用于提交分数时引用。
8. 将分数格式设置为整数
9. 分数提交方式为最佳分数
10. 点击添加语言并填写条目。

复制你创建的LeaderboardID，然后我们前往Xcode。

使用Xcode

我们将使用4个函数。

1.导入框架并设置协议

2.检查用户是否已登录GameCenter

3.向GameCenter报告分数

4.查看排行榜

5. 导入GameKitimportGameKit协议GKGameCenterControllerDelegate

6.现在我们要检查用户是否已登录GameCenter

```
func authenticateLocalPlayer() {

        let localPlayer = GKLocalPlayer.localPlayer()
        localPlayer.authenticateHandler = { (viewController, error) -> Void in

            if viewController != nil {
                //如果用户尚未登录GameCenter，我们让他们登录
                let vc:UIViewController = self.view!.window!.rootViewController!
vc.presentViewController(viewController!, animated: true, completion: nil)

            } else {

                //如果需要，可以在这里执行其他操作
            }
        }
    }
```


Chapter 148: GameCenter Leaderboards

Section 148.1: GameCenter Leaderboards

Prerequisites:

1. Apple Developers Account
2. Setup GameCenter Leaderboards with iTunesConnect

Setting up GameCenter Leaderboards:

1. Sign in to *iTunesConnect*
2. Go to *My Apps*. Create an app for your project then go to *Features*.
3. Click on *Game Center*
4. Click the plus sign next to Leaderboards.
5. Choose *Single Leaderboard* for Leaderboard types.
6. Create a *Leaderboard Reference Name* for your reference.
7. Create a *Leaderboard ID* for your app to refer to when reporting scores.
8. Set score format to *Integer*
9. Score Submission will be *Best Score*
10. Click *Add language* and fill the entries.

Copy your `LeaderboardID` that you made and lets head over to Xcode.

Working with Xcode

There are 4 functions that we will be working with.

1. Importing the framework and setting up the protocols
2. Checking if the user is signed in to GameCenter
3. Reporting the scores to GameCenter
4. Viewing leaderboards
5. Import GameKit `import GameKit` Protocols `GKGameCenterControllerDelegate`
6. Now we want to check if the user is signed in to GameCenter

```
func authenticateLocalPlayer() {

        let localPlayer = GKLocalPlayer.localPlayer()
        localPlayer.authenticateHandler = { (viewController, error) -> Void in

            if viewController != nil {
                //If the user is not signed in to GameCenter, we make them sign in
                let vc:UIViewController = self.view!.window!.rootViewController!
                vc.presentViewController(viewController!, animated: true, completion: nil)

            } else {

                //Do something here if you want
            }
        }
    }
```

3.现在用户正在使用应用，突然用户获得了新的高分，我们通过调用下面的函数来报告高分。

下面的函数有两个参数。

Identifier 是一个字符串，用于输入你在iTunesConnect中创建的排行榜ID。

score 是一个整数，表示用户提交到iTunesConnect的分数。

```
func saveHighScore(identifier:String, score:Int) {

        if GKLocalPlayer.localPlayer().authenticated {

            let scoreReporter = GKScore(leaderboardIdentifier: identifier)

scoreReporter.value = Int64(score)

            let scoreArray:[GKScore] = [scoreReporter]

GKScore.reportScores(scoreArray, withCompletionHandler: {
                error -> Void in

                if error != nil {
                    print("Error")
                } else {


                }
            })
        }
    }
```

4.现在如果用户想查看排行榜，调用下面的函数

```
//如果调用此函数，将显示GameCenter排行榜和成就。
    func showGameCenter() {

        let gameCenterViewController = GKGameCenterViewController()
        gameCenterViewController.gameCenterDelegate = self

        let vc:UIViewController = self.view!.window!.rootViewController!
vc.presentViewController(gameCenterViewController, animated: true, completion:nil)

    }

    //此函数在显示后关闭游戏中心。
    func gameCenterViewControllerDidFinish(gameCenterViewController: GKGameCenterViewController) {

gameCenterViewController.dismissViewControllerAnimated(true, completion: nil)
        self.gameCenterAchievements.removeAll()

    }
```


3. Now the user is using the app and suddenly the user has a new high score, we report the high score by calling the function below.

The function below hols 2 parameters.

`Identifier` which is defined as a string and used to enter your leaderboardID that you made in iTunesConnect.

`score` which is defined as an Int which will be the users score to submit to iTunesConnect

```
func saveHighScore(identifier:String, score:Int) {

        if GKLocalPlayer.localPlayer().authenticated {

            let scoreReporter = GKScore(leaderboardIdentifier: identifier)

            scoreReporter.value = Int64(score)

            let scoreArray:[GKScore] = [scoreReporter]

            GKScore.reportScores(scoreArray, withCompletionHandler: {
                error -> Void in

                if error != nil {
                    print("Error")
                } else {


                }
            })
        }
    }
```

4. Now if the user wants to view leaderboards, call the function below

```
//This function will show GameCenter leaderboards and Achievements if you call this function.
    func showGameCenter() {

        let gameCenterViewController = GKGameCenterViewController()
        gameCenterViewController.gameCenterDelegate = self

        let vc:UIViewController = self.view!.window!.rootViewController!
        vc.presentViewController(gameCenterViewController, animated: true, completion:nil)

    }

    //This function closes gameCenter after showing.
    func gameCenterViewControllerDidFinish(gameCenterViewController: GKGameCenterViewController) {

        gameCenterViewController.dismissViewControllerAnimated(true, completion: nil)
        self.gameCenterAchievements.removeAll()

    }
```

第149章：钥匙串

第149.1节：向钥匙串添加密码

每个钥匙串项目通常表示为一个CFDictionary。但你也可以在Objective-C中简单使用NSDictionary并利用桥接，或者在Swift中使用Dictionary并显式转换为CFDictionary。

你可以用以下字典构造一个密码：

Swift

```
var dict = [String : AnyObject]()
```

首先，您需要一个键/值对，让钥匙串知道这是一个密码。请注意，因为我们的字典键是一个字符串所以我们必须在 Swift 3 中显式地将任何CFString转换为String。CFString 不能用作 Swift 字典的键，因为它不可哈希。

Swift

```
dict[kSecClass as String] = kSecClassGenericPassword
```

接下来，我们的密码可能有一系列属性来描述它并帮助我们以后找到它。以下是通用密码的属性列表。

Swift

```
// 密码仅在设备解锁时可访问
dict[kSecAttrAccessible as String] = kSecAttrAccessibleWhenUnlocked
// 标签可能帮助您以后找到它
dict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// 用户名
dict[kSecAttrAccount as String] = "我的名字" as CFString
// 服务名称
dict[kSecAttrService as String] = "MyService" as CFString
```

最后，我们需要实际的私有数据。请确保不要在内存中保留太久。这必须是CFData。

Swift

```
dict[kSecValueData as String] = "my_password!!".data(using: .utf8) as! CFData
```

最后，钥匙串服务的添加函数需要知道它应该如何返回新构建的钥匙串项目。由于你不应该在内存中长时间持有数据，下面是如何只返回属性的方法：

Swift

```
dict[kSecReturnAttributes as String] = kCFBooleanTrue
```

现在我们已经构建了项目。让我们添加它：

Swift

```
var result: AnyObject?
let status = withUnsafeMutablePointer(to: &result) {
    SecItemAdd(dict as CFDictionary, UnsafeMutablePointer($0))
}
let newAttributes = result as! Dictionary<String, AnyObject>
```


Chapter 149: Keychain

Section 149.1: Adding a Password to the Keychain

Every Keychain Item is most often represented as a CFDictionary. You can, however, simply use NSDictionary in Objective-C and take advantage of bridging, or in Swift you may use Dictionary and explicitly cast to CFDictionary.

You could construct a password with the following dictionary:

Swift

```
var dict = [String : AnyObject]()
```

First, you need a key/value pair that lets the Keychain know this is a password. Note that because our dict key is a String we must cast any CFString to a String explicitly in Swift 3. CFString may not be used as the key to a Swift Dictionary because it is not Hashable.

Swift

```
dict[kSecClass as String] = kSecClassGenericPassword
```

Next, our password may have a series of attributes to describe it and help us find it later. Here's a list of attributes for generic passwords.

Swift

```
// The password will only be accessible when the device is unlocked
dict[kSecAttrAccessible as String] = kSecAttrAccessibleWhenUnlocked
// Label may help you find it later
dict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// Username
dict[kSecAttrAccount as String] = "My Name" as CFString
// Service name
dict[kSecAttrService as String] = "MyService" as CFString
```

Finally, we need our actual private data. Be sure not to keep this around in memory for too long. This must be CFData.

Swift

```
dict[kSecValueData as String] = "my_password!!".data(using: .utf8) as! CFData
```

Finally, the Keychain Services add function wants to know how it should return the newly constructed keychain item. Since you shouldn't be holding on to the data very long in memory, here's how you could only return the attributes:

Swift

```
dict[kSecReturnAttributes as String] = kCFBooleanTrue
```

Now we have constructed our item. Let's add it:

Swift

```
var result: AnyObject?
let status = withUnsafeMutablePointer(to: &result) {
    SecItemAdd(dict as CFDictionary, UnsafeMutablePointer($0))
}
let newAttributes = result as! Dictionary<String, AnyObject>
```

这将新的属性字典放入result中。SecItemAdd接收我们构建的字典，以及一个指向我们希望存放结果的位置的指针。该函数随后返回一个OSStatus，指示成功或错误代码。结果代码在此处有描述。

第149.2节：钥匙串访问控制（带密码回退的TouchID）

钥匙串允许保存带有特殊 SecAccessControl 属性的项目，该属性仅在用户通过 Touch ID（或如果允许此类备用方案，则通过密码）进行身份验证后才能从钥匙串中获取项目。应用程序仅会收到身份验证是否成功的通知，整个用户界面由 iOS 管理。

首先，应创建 SecAccessControl 对象：

Swift

```
let error: Unmanaged<CFError>?

guard let accessControl = SecAccessControlCreateWithFlags(kCFAllocatorDefault,
kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly, .userPresence, &error) else {
    fatalError("Something went wrong")
}
```

接下来，将其添加到字典中，使用 kSecAttrAccessControl 键（该键与您在其他示例中使用的 kSecAttrAccessible 键互斥）：

Swift

```
var dictionary = [String : Any]()

dictionary[kSecClass as String] = kSecClassGenericPassword
dictionary[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
dictionary[kSecAttrAccount as String] = "My Name" as CFString
dictionary[kSecValueData as String] = "new_password!!".data(using: .utf8) as! CFData
dictionary[kSecAttrAccessControl as String] = accessControl
```

然后像之前一样保存它：

Swift

```
let lastResultCode = SecItemAdd(query as CFDictionary, nil)
```

要访问存储的数据，只需查询钥匙串中的密钥。钥匙串服务将向用户显示身份验证对话框，并根据是否提供了合适的指纹或匹配了密码返回数据或 nil。

可选地，可以指定提示字符串：

Swift

```
var query = [String: Any]()

query[kSecClass as String] = kSecClassGenericPassword
query[kSecReturnData as String] = kCFBooleanTrue
query[kSecAttrAccount as String] = "我的名字" as CFString
query[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
query[kSecUseOperationPrompt as String] = "请将手指放在那个按钮上" as CFString

var queryResult: AnyObject?
let status = withUnsafeMutablePointer(to: &queryResult) {
    SecItemCopyMatching(query as CFDictionary, UnsafeMutablePointer($0))
```


This places the new attributes dict inside result. SecItemAdd takes in the dictionary we constructed, as well as a pointer to where we would like our result. The function then returns an OSStatus indicating success or an error code. Result codes are described here.

Section 149.2: Keychain Access Control (TouchID with password fallback)

Keychain allows to save items with special SecAccessControl attribute which will allow to get item from Keychain only after user will be authenticated with Touch ID (or passcode if such fallback is allowed). App is only notified whether the authentication was successful or not, whole UI is managed by iOS.

First, SecAccessControl object should be created:

Swift

```
let error: Unmanaged<CFError>?

guard let accessControl = SecAccessControlCreateWithFlags(kCFAllocatorDefault,
kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly, .userPresence, &error) else {
    fatalError("Something went wrong")
}
```

Next, add it to the dictionary with kSecAttrAccessControl key (which is mutually exclusive with kSecAttrAccessible key you've been using in other examples):

Swift

```
var dictionary = [String : Any]()

dictionary[kSecClass as String] = kSecClassGenericPassword
dictionary[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
dictionary[kSecAttrAccount as String] = "My Name" as CFString
dictionary[kSecValueData as String] = "new_password!!".data(using: .utf8) as! CFData
dictionary[kSecAttrAccessControl as String] = accessControl
```

And save it as you've done before:

Swift

```
let lastResultCode = SecItemAdd(query as CFDictionary, nil)
```

To access stored data, just query Keychain for a key. Keychain Services will present authentication dialog to the user and return data or nil depending on whether suitable fingerprint was provided or passcode matched.

Optionally, prompt string can be specified:

Swift

```
var query = [String: Any]()

query[kSecClass as String] = kSecClassGenericPassword
query[kSecReturnData as String] = kCFBooleanTrue
query[kSecAttrAccount as String] = "My Name" as CFString
query[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
query[kSecUseOperationPrompt as String] = "Please put your fingers on that button" as CFString

var queryResult: AnyObject?
let status = withUnsafeMutablePointer(to: &queryResult) {
    SecItemCopyMatching(query as CFDictionary, UnsafeMutablePointer($0))
```

```
}
```

注意，若用户拒绝、取消或授权失败，status 将为 err。

Swift

```
if status == noErr {
    let password = String(data: queryResult as! Data, encoding: .utf8)!
    print("密码: \(password)")
} else {
    print("授权未通过")
}
```

第149.3节：在钥匙串中查找密码

要构造查询，我们需要将其表示为一个`CFDictionary`。你也可以在Objective-C中使用`NSDictionary`，或在Swift中使用`Dictionary`并转换为`CFDictionary`。

我们需要一个类键：

Swift

```
var dict = [String : AnyObject]()
dict[kSecClass as String] = kSecClassGenericPassword
```

接下来，我们可以指定属性以缩小搜索范围：

Swift

```
// 标签
dict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// 用户名
dict[kSecAttrAccount as String] = "我的名字" as CFString
// 服务名称
dict[kSecAttrService as String] = "MyService" as CFString
```

我们还可以指定此处描述的特殊搜索修饰键。

最后，我们需要说明希望如何返回数据。下面，我们将请求仅返回私有密码本身，作为一个CFData对象：

Swift

```
dict[kSecReturnData as String] = kCFBooleanTrue
```

现在，开始搜索：

Swift

```
var queryResult: AnyObject?
let status = withUnsafeMutablePointer(to: &queryResult) {
    SecItemCopyMatching(dict as CFDictionary, UnsafeMutablePointer($0))
}
// 不要长时间将此保存在内存中！！
let password = String(data: queryResult as! Data, encoding: .utf8)!
```

这里，SecItemCopyMatching 接受一个查询字典和一个指向你希望结果存放位置的指针。它返回一个OSStatus结果代码。以下是可能的结果。


```
}
```

Pay attention that `status` will be `err` if user declined, canceled or failed authorization.

Swift

```
if status == noErr {
    let password = String(data: queryResult as! Data, encoding: .utf8)!
    print("Password: \(password)")
} else {
    print("Authorization not passed")
}
```

Section 149.3: Finding a Password in the Keychain

To construct a query, we need to represent it as a `CFDictionary`. You may also use `NSDictionary` in Objective-C or `Dictionary` in Swift and cast to `CFDictionary`.

We need a class key:

Swift

```
var dict = [String : AnyObject]()
dict[kSecClass as String] = kSecClassGenericPassword
```

Next, we can specify attributes to narrow down our search:

Swift

```
// Label
dict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// Username
dict[kSecAttrAccount as String] = "My Name" as CFString
// Service name
dict[kSecAttrService as String] = "MyService" as CFString
```

We can also specify special search modifier keys described here.

Finally, we need to say how we'd like our data returned. Below, we'll request that just the private password itself be returned as a `CFData` object:

Swift

```
dict[kSecReturnData as String] = kCFBooleanTrue
```

Now, let's search:

Swift

```
var queryResult: AnyObject?
let status = withUnsafeMutablePointer(to: &queryResult) {
    SecItemCopyMatching(dict as CFDictionary, UnsafeMutablePointer($0))
}
// Don't keep this in memory for long!!
let password = String(data: queryResult as! Data, encoding: .utf8)!
```

Here, `SecItemCopyMatching` takes in a query dictionary and a pointer to where you'd like the result to go. It returns an `OSStatus` with a result codes. Here are the possibilities.

第149.4节：在钥匙串中更新密码

和往常一样，我们首先需要一个CFDictionary来表示我们想要更新的项目。这个字典必须包含该项目的所有旧值，包括旧的私有数据。然后它接受一个CFDictionary，包含你想要更改的任何属性或数据本身。

所以首先，让我们构建一个类键和一个属性列表。这些属性可以缩小我们的搜索范围，但如果你要更改它们，必须包含任何属性及其旧值。

Swift

```
var dict = [String : AnyObject]()
dict[kSecClass as String] = kSecClassGenericPassword
// 标签
dict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// 用户名
dict[kSecAttrAccount as String] = "My Name" as CFString
```

现在我们必须添加旧数据：

Swift

```
dict[kSecValueData as String] = "my_password!!".data(using: .utf8) as! CFData
```

现在让我们创建相同的属性，但使用不同的密码：

Swift

```
var newDict = [String : AnyObject]()
newDict[kSecClass as String] = kSecClassGenericPassword
// 标签
newDict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// 用户名
newDict[kSecAttrAccount as String] = "我的名字" as CFString
// 新密码
newDict[kSecValueData as String] = "new_password!!".data(using: .utf8) as! CFData
```

现在，我们只需将其传递给钥匙串服务：

Swift

```
let status = SecItemUpdate(dict as CFDictionary, newDict as CFDictionary)
```

SecItemUpdate 返回一个状态码。结果描述见 here。

第149.5节：从钥匙串中移除密码

我们只需要一件事来从钥匙串中删除一个项目：一个包含描述要删除项目属性的 CFDictionary。任何匹配查询字典的项目都会被永久删除，因此如果你只打算删除单个项目，请确保查询足够具体。和往常一样，我们可以在Objective-C中使用 NSDictionary，或者在Swift中使用 Dictionary 然后转换为 CFDictionary。

在此上下文中，查询字典仅包含一个类键来描述项目类型和描述项目信息的属性。不允许包含诸如 kSecMatchCaseInsensitive 之类的搜索限制。

Swift

```
var dict = [String : AnyObject]()
dict[kSecClass as String] = kSecClassGenericPassword
```


Section 149.4: Updating a Password in the Keychain

As usual, we first need a `CFDictionary` to represent the item we want to update. This must contain all of the old values for the item, including the old private data. Then it takes a `CFDictionary` of any attributes or the data itself that you would like to change.

So first, let's construct a class key and a list of attributes. These attributes can narrow our search but you must include any attributes and there old values if you will be changing them.

Swift

```
var dict = [String : AnyObject]()
dict[kSecClass as String] = kSecClassGenericPassword
// Label
dict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// Username
dict[kSecAttrAccount as String] = "My Name" as CFString
```

Now we must add the old data:

Swift

```
dict[kSecValueData as String] = "my_password!!".data(using: .utf8) as! CFData
```

Now let's create the same attributes but a different password:

Swift

```
var newDict = [String : AnyObject]()
newDict[kSecClass as String] = kSecClassGenericPassword
// Label
newDict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// Username
newDict[kSecAttrAccount as String] = "My Name" as CFString
// New password
newDict[kSecValueData as String] = "new_password!!".data(using: .utf8) as! CFData
```

Now, we just pass it to Keychain Services:

Swift

```
let status = SecItemUpdate(dict as CFDictionary, newDict as CFDictionary)
```

`SecItemUpdate` returns a status code. Results are described here.

Section 149.5: Removing a Password from the Keychain

We need only one thing in order to delete an item from the Keychain: a `CFDictionary` with attributes describing the items to be deleted. Any items that match the query dictionary will be deleted permanently, so if you are only intending to delete a single item be sure to be specific with your query. As always, we can use an `NSDictionary` in Objective-C or in Swift we can use a `Dictionary` and then cast to `CFDictionary`.

A query dictionary, in this context exclusively includes a class key to describe what the item is and attributes to describe information about the item. Inclusion of search restrictions such as `kSecMatchCaseInsensitive` is not allowed.

Swift

```
var dict = [String : AnyObject]()
dict[kSecClass as String] = kSecClassGenericPassword
```

```
// 标签
dict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// 用户名
dict[kSecAttrAccount as String] = "My Name" as CFString
```

现在我们可以简单地将其移除：

Swift

```
let status = SecItemDelete(dict as CFDictionary)
```

SecItemDelete 返回一个 OSStatus。结果代码描述见 here。

第149.6节：使用一个文件进行钥匙串的添加、更新、删除和查找操作

Keychain.h

```
#import <Foundation/Foundation.h>
typedef void (^KeychainOperationBlock)(BOOL successfulOperation, NSData *data, OSStatus status);

@interface Keychain : NSObject

-(id) initWithService:(NSString *) service_ withGroup:(NSString*)group_;

-(void)insertKey:(NSString *)key withData:(NSData *)data
withCompletion:(KeychainOperationBlock)completionBlock;
-(void)updateKey:(NSString*)key withData:(NSData*) data
withCompletion:(KeychainOperationBlock)completionBlock;
-(void)removeDataForKey:(NSString*)key withCompletionBlock:(KeychainOperationBlock)completionBlock;
-(void)findDataForKey:(NSString*)key withCompletionBlock:(KeychainOperationBlock)completionBlock;

@end
```

Keychain.m

```
#import "Keychain.h"
#import <Security/Security.h>

@implementation Keychain

{
    NSString * keychainService;
    NSString * keychainGroup;
}

-(id) initWithService:(NSString *)service withGroup:(NSString*)group
{
    self =[super init];
    if(self) {
keychainService = [NSString stringWithString:service];
        if(group) {
keychainGroup = [NSString stringWithString:group];
        }
    }

    return  self;
}

-(void)insertKey:(NSString *)key
```



```
// Label
dict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// Username
dict[kSecAttrAccount as String] = "My Name" as CFString
```

And now we can simply remove it:

Swift

```
let status = SecItemDelete(dict as CFDictionary)
```

`SecItemDelete` returns an `OSStatus`. Result codes are described here.

Section 149.6: Keychain Add, Update, Remove and Find operations using one file

Keychain.h

```
#import <Foundation/Foundation.h>
typedef void (^KeychainOperationBlock)(BOOL successfulOperation, NSData *data, OSStatus status);

@interface Keychain : NSObject

-(id) initWithService:(NSString *) service_ withGroup:(NSString*)group_;

-(void)insertKey:(NSString *)key withData:(NSData *)data
withCompletion:(KeychainOperationBlock)completionBlock;
-(void)updateKey:(NSString*)key withData:(NSData*) data
withCompletion:(KeychainOperationBlock)completionBlock;
-(void)removeDataForKey:(NSString*)key withCompletionBlock:(KeychainOperationBlock)completionBlock;
-(void)findDataForKey:(NSString*)key withCompletionBlock:(KeychainOperationBlock)completionBlock;

@end
```

Keychain.m

```
#import "Keychain.h"
#import <Security/Security.h>

@implementation Keychain

{
    NSString * keychainService;
    NSString * keychainGroup;
}

-(id) initWithService:(NSString *)service withGroup:(NSString*)group
{
    self =[super init];
    if(self) {
        keychainService = [NSString stringWithString:service];
        if(group) {
            keychainGroup = [NSString stringWithString:group];
        }
    }

    return  self;
}

-(void)insertKey:(NSString *)key
```

```
withData:(NSData *)data
  withCompletion:(KeychainOperationBlock)completionBlock
{
    NSMutableDictionary * dict =[self prepareDict:key];
    [dict setObject:data forKey:(__bridge id)kSecValueData];
    [dict setObject:keychainService forKey:(id)kSecAttrService];

OSStatus status = SecItemAdd((__bridge CFDictionaryRef)dict, NULL);
    if(errSecSuccess != status) {
DLog(@"Unable add item with key =%@ error:%d",key,(int)status);
        if (completionBlock) {
completionBlock(errSecSuccess == status, nil, status);
        }
    }
    if (status == errSecDuplicateItem) {
                [self updateKey:key withData:data withCompletion:^(BOOL successfulOperation, NSData
*updateData, OSStatus updateStatus) {
            if (completionBlock) {
completionBlock(successfulOperation, updateData, updateStatus);
            }
DLog(@"发现重复项 -- 使用数据更新键");
        }];
    }
}

-(void)findDataForKey:(NSString *)key
  withCompletionBlock:(KeychainOperationBlock)completionBlock
{
    NSMutableDictionary *dict = [self prepareDict:key];
    [dict setObject:(__bridge id)kSecMatchLimitOne forKey:(__bridge id)kSecMatchLimit];
    [dict setObject:keychainService forKey:(id)kSecAttrService];
    [dict setObject:(id)kCFBooleanTrue forKey:(__bridge id)kSecReturnData];
    CFTypeRef result = NULL;
OSStatus status = SecItemCopyMatching((__bridge CFDictionaryRef)dict,&result);

    if( status != errSecSuccess) {
DLog(@"无法获取键 %@ 的项目，错误码:%d",key,(int)status);
        if (completionBlock) {
completionBlock(errSecSuccess == status, nil, status);
        }
    } else {
        if (completionBlock) {
completionBlock(errSecSuccess == status, (__bridge NSData *)result, status);
        }
    }
}

-(void)updateKey:(NSString *)key
        withData:(NSData *)data
  withCompletion:(KeychainOperationBlock)completionBlock
{
    NSMutableDictionary * dictKey =[self prepareDict:key];

    NSMutableDictionary * dictUpdate =[[NSMutableDictionary alloc] init];
    [dictUpdate setObject:data forKey:(__bridge id)kSecValueData];
    [dictUpdate setObject:keychainService forKey:(id)kSecAttrService];
    OSStatus status = SecItemUpdate((__bridge CFDictionaryRef)dictKey, (__bridge
CFDictionaryRef)dictUpdate);
    if( status != errSecSuccess) {
DLog(@"无法删除键 %@ 的项目，错误码:%d",key,(int)status);
    }
    if (completionBlock) {
```



```
        withData:(NSData *)data
  withCompletion:(KeychainOperationBlock)completionBlock
{
    NSMutableDictionary * dict =[self prepareDict:key];
    [dict setObject:data forKey:(__bridge id)kSecValueData];
    [dict setObject:keychainService forKey:(id)kSecAttrService];

    OSStatus status = SecItemAdd((__bridge CFDictionaryRef)dict, NULL);
    if(errSecSuccess != status) {
        DLog(@"Unable add item with key =%@ error:%d",key,(int)status);
        if (completionBlock) {
            completionBlock(errSecSuccess == status, nil, status);
        }
    }
    if (status == errSecDuplicateItem) {
        [self updateKey:key withData:data withCompletion:^(BOOL successfulOperation, NSData
*updateData, OSStatus updateStatus) {
            if (completionBlock) {
                completionBlock(successfulOperation, updateData, updateStatus);
            }
            DLog(@"Found duplication item -- updating key with data");
        }];
    }
}

-(void)findDataForKey:(NSString *)key
  withCompletionBlock:(KeychainOperationBlock)completionBlock
{
    NSMutableDictionary *dict = [self prepareDict:key];
    [dict setObject:(__bridge id)kSecMatchLimitOne forKey:(__bridge id)kSecMatchLimit];
    [dict setObject:keychainService forKey:(id)kSecAttrService];
    [dict setObject:(id)kCFBooleanTrue forKey:(__bridge id)kSecReturnData];
    CFTypeRef result = NULL;
    OSStatus status = SecItemCopyMatching((__bridge CFDictionaryRef)dict,&result);

    if( status != errSecSuccess) {
        DLog(@"Unable to fetch item for key %@ with error:%d",key,(int)status);
        if (completionBlock) {
            completionBlock(errSecSuccess == status, nil, status);
        }
    } else {
        if (completionBlock) {
            completionBlock(errSecSuccess == status, (__bridge NSData *)result, status);
        }
    }
}

-(void)updateKey:(NSString *)key
        withData:(NSData *)data
  withCompletion:(KeychainOperationBlock)completionBlock
{
    NSMutableDictionary * dictKey =[self prepareDict:key];

    NSMutableDictionary * dictUpdate =[[NSMutableDictionary alloc] init];
    [dictUpdate setObject:data forKey:(__bridge id)kSecValueData];
    [dictUpdate setObject:keychainService forKey:(id)kSecAttrService];
    OSStatus status = SecItemUpdate((__bridge CFDictionaryRef)dictKey, (__bridge
CFDictionaryRef)dictUpdate);
    if( status != errSecSuccess) {
        DLog(@"Unable to remove item for key %@ with error:%d",key,(int)status);
    }
    if (completionBlock) {
```

```
completionBlock(errSecSuccess == status, nil, status);
    }
}

-(void)removeDataForKey:(NSString *)key
    withCompletionBlock:(KeychainOperationBlock)completionBlock {
    NSMutableDictionary *dict = [self prepareDict:key];
OSStatus status = SecItemDelete((__bridge CFDictionaryRef)dict);
    if( status != errSecSuccess) {
DLog(@"无法删除键 %@ 的项目，错误码:%d",key,(int)status);
    }
    if (completionBlock) {
completionBlock(errSecSuccess == status, nil, status);
    }
}

#pragma mark 内部方法

-(NSMutableDictionary*) prepareDict:(NSString *) key {

    NSMutableDictionary *dict = [[NSMutableDictionary alloc] init];
    [dict setObject:(__bridge id)kSecClassGenericPassword forKey:(__bridge id)kSecClass];

    NSData *encodedKey = [key dataUsingEncoding:NSUTF8StringEncoding];
    [dict setObject:encodedKey forKey:(__bridge id)kSecAttrGeneric];
    [dict setObject:encodedKey forKey:(__bridge id)kSecAttrAccount];
    [dict setObject:keychainService forKey:(__bridge id)kSecAttrService];
    [dict setObject:(__bridge id)kSecAttrAccessibleAlwaysThisDeviceOnly forKey:(__bridge
id)kSecAttrAccessible];

    //这是用于跨应用共享数据
    if(keychainGroup != nil) {
        [dict setObject:keychainGroup forKey:(__bridge id)kSecAttrAccessGroup];
    }

    return  dict;
}

@end
```



```
        completionBlock(errSecSuccess == status, nil, status);
    }
}

-(void)removeDataForKey:(NSString *)key
    withCompletionBlock:(KeychainOperationBlock)completionBlock {
    NSMutableDictionary *dict = [self prepareDict:key];
    OSStatus status = SecItemDelete((__bridge CFDictionaryRef)dict);
    if( status != errSecSuccess) {
        DLog(@"Unable to remove item for key %@ with error:%d",key,(int)status);
    }
    if (completionBlock) {
        completionBlock(errSecSuccess == status, nil, status);
    }
}

#pragma mark Internal methods

-(NSMutableDictionary*) prepareDict:(NSString *) key {

    NSMutableDictionary *dict = [[NSMutableDictionary alloc] init];
    [dict setObject:(__bridge id)kSecClassGenericPassword forKey:(__bridge id)kSecClass];

    NSData *encodedKey = [key dataUsingEncoding:NSUTF8StringEncoding];
    [dict setObject:encodedKey forKey:(__bridge id)kSecAttrGeneric];
    [dict setObject:encodedKey forKey:(__bridge id)kSecAttrAccount];
    [dict setObject:keychainService forKey:(__bridge id)kSecAttrService];
    [dict setObject:(__bridge id)kSecAttrAccessibleAlwaysThisDeviceOnly forKey:(__bridge
id)kSecAttrAccessible];

    //This is for sharing data across apps
    if(keychainGroup != nil) {
        [dict setObject:keychainGroup forKey:(__bridge id)kSecAttrAccessGroup];
    }

    return  dict;
}

@end
```

第150章：使用宏处理多环境

第150.1节：使用多个目标和宏处理多环境

例如，我们有两个环境：CI - 预发布，并希望为每个环境添加一些自定义设置。这里我将尝试自定义服务器URL、应用名称。

首先，我们通过复制主目标创建两个环境的两个目标：

对于每个目标，我们将定义一个自定义宏。这里我将在目标 CI 的构建设置中定义名为“CI”的宏，在目标 Staging 中定义名为“STAGING”的宏。

开发目标（MultipleEnvironment 目标）：

Chapter 150: Handle Multiple Environment using Macro

Section 150.1: Handle multiple environment using multiple target and macro

For example, we have two environments: CI - Staging and want to add some customizations for each environment. Here I will try to customize server URL, app name.

First, we create two targets for 2 environments by duplicating the main target:

For each target, we will define a custom macro. Here I will define macro named "CI" in build settings of target CI, macro named "STAGING" for target Staging.

The development target (MultipleEnvironment target):

目标 CI：

目标 Staging：

Target CI:

Target Staging:

为每个目标创建方案：

我们将创建一个头文件来定义服务器 URL，如下所示：

Create scheme for each target:

We will create a header file to define SERVER URL as below:

这意味着，

- 如果我们使用默认目标（MultipleEnvironment）进行运行/归档，SERVER_URL 是 http://192.168.10.10:8080/
- 如果我们使用 CI 目标运行/归档，SERVER_URL 是 http://ci.api.example.com/
- 如果我们使用 STAGING 目标运行/归档，SERVER_URL 是 http://stg.api.example.com/

如果你想做更多自定义，例如：为每个目标更改应用名称：

It means,

- If we run/archive using the default target (MultipleEnvironment), the SERVER_URL is http://192.168.10.10:8080/
- If we run/archive using CI target, the SERVER_URL is http://ci.api.example.com/
- If we run/archive using STAGING target, the SERVER_URL is http://stg.api.example.com/

If you want to do more customize, for example: Change app name for each target:

快完成了。现在我们想在主屏幕显示当前的 SERVER_URL：

Almost done. Now we want to show current SERVER_URL to main screen:

现在，让我们看看如果使用默认目标（MultipleEnvironment）运行应用会怎样

Now, let's see if we run the app with the default target (MultipleEnvironment)

http://192.168.10.10:8080/

http://192.168.10.10:8080/

Carrier
12:03 AM
MultipleEn...
Safari

Carrier
12:03 AM
MultipleEn...
Safari

CI 目标：

CI target:

http://ci.api.example.com/

http://ci.api.example.com/

Carrier
12:05 AM
App Cl
Safari

Carrier
12:05 AM
App Cl
Safari

预发布目标:

Staging target:

Carrier 12:07 AM

http://stg.api.example.com/

Carrier 12:07 AM

http://stg.api.example.com/

Carrier
12:06 AM
App STG
Safari

Carrier
12:06 AM
App STG
Safari

如您所见，SERVER_URL 和应用名称的值会根据不同目标而变化 :)

As you can see, value of SERVER_URL and app name is changed for each target :)

第151章：设置视图背景

第151.1节：填充UIView的背景图像

Objective-C

```
UIGraphicsBeginImageContext(self.view.frame.size);
 [[UIImage imageNamed:@"image.png"] drawInRect:self.view.bounds];
 UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
 UIGraphicsEndImageContext();
 self.view.backgroundColor = [UIColor colorWithPatternImage:image];
```

第151.2节：创建渐变背景视图

要创建带有渐变的背景，可以使用CAGradientLayer类：

Swift 3.1：

```
func createGradient() {
    let caLayer = CAGradientLayer()
caLayer.colors = [UIColor.white, UIColor.green, UIColor.blue]
    caLayer.locations = [0, 0.5, 1]
caLayer.bounds = self.bounds
    self.layer.addSublayer(caLayer)
}
```

这可以在viewDidLoad()中这样调用：

```
override func viewDidLoad() {
    super.viewDidLoad()
createGradient()
}
```

CAGradientLayer 的 locations 和 bounds 变量可以接受多个值，以创建包含任意多种颜色的渐变层。根据文档说明：

> 默认情况下，颜色均匀分布在图层上，但你也可以选择性地指定 locations，以控制渐变中颜色的位置。

第151.3节：使用图片设置视图背景

```
self.view.backgroundColor = [UIColor colorWithPatternImage:[UIImage imageNamed:@"Background.png"]];
```

第151.4节：设置视图背景

Objective C：

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

Swift:

```
view.backgroundColor! = UIColor.redColor()
```


Chapter 151: Set View Background

Section 151.1: Fill background Image of a UIView

Objective-C

```
UIGraphicsBeginImageContext(self.view.frame.size);
[[UIImage imageNamed:@"image.png"] drawInRect:self.view.bounds];
UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
UIGraphicsEndImageContext();
self.view.backgroundColor = [UIColor colorWithPatternImage:image];
```

Section 151.2: Creating a gradient background view

To create a background with a gradient you can use the CAGradientLayer class:

Swift 3.1:

```
func createGradient() {
    let caLayer = CAGradientLayer()
    caLayer.colors = [UIColor.white, UIColor.green, UIColor.blue]
    caLayer.locations = [0, 0.5, 1]
    caLayer.bounds = self.bounds
    self.layer.addSublayer(caLayer)
}
```

This can be called on viewDidLoad() like so:

```
override func viewDidLoad() {
    super.viewDidLoad()
    createGradient()
}
```

The CAGradientLayer locations and bounds variables can take multiple values to create a gradient layer with how ever many colors you desire. From the documentation:

> By default, the colors are spread uniformly across the layer, but you can optionally specify locations for control over the color positions through the gradient.

Section 151.3: Set View backround with image

```
self.view.backgroundColor = [UIColor colorWithPatternImage:[UIImage imageNamed:@"Background.png"]];
```

Section 151.4: Set View background

Objective C:

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

Swift:

```
view.backgroundColor! = UIColor.redColor()
```

Swift 3

```
view.backgroundColor = UIColor.redColor
```


Swift 3

```
view.backgroundColor = UIColor.redColor
```

第152章：块

第152.1节：自定义方法的自定义完成块

1- 定义你自己的自定义块

```
typedef void(^myCustomCompletion)(BOOL);
```

2- 创建一个自定义方法，该方法以你的自定义完成块作为参数。

```
-(void) customMethodName:(myCustomCompletion) compblock{
    // 执行操作
    // 检查完成块是否存在；如果不检查会抛出异常
    if(complblock)
compblock(YES);
  }
```

3- 如何在你的方法中使用块

```
[self customMethodName:^(BOOL finished) {
if(finished){
NSLog(@"success");
}
}];
```

第152.2节：UIView动画

```
[UIView animateWithDuration:1.0
    animations:^{
someView.alpha = 0;
        otherView.alpha = 1;
    }
completion:^(BOOL finished) {
    [someView removeFromSuperview];
}];
```

插入符号“^”字符定义了一个代码块。例如，^{ ... }就是一个代码块。更具体地说，它是一个返回“void”且不接受任何参数的代码块。它相当于一个类似于“- (void)something;”的方法，但代码块本身没有内在的名称。

定义一个可以接受参数的代码块的方式非常相似。要向代码块传递参数，你可以这样定义代码块：^(BOOL someArg , NSString someStr) { ... }*。当你使用支持代码块的API调用时，尤其是在动画代码块或NSURLConnection代码块中，如上例所示，你会编写类似这样的代码块。

第152.3节：修改捕获变量

Block 会捕获出现在同一词法作用域中的变量。通常这些变量会被捕获为“const”值：

```
int val = 10;
void (^blk)(void) = ^{
val = 20; // 错误！val 是一个常量值，不能被修改！
};
```


Chapter 152: Block

Section 152.1: Custom completion block for Custom Methods

1- Define Your own custom Block

```
typedef void(^myCustomCompletion)(BOOL);
```

2- Create custom method which takes your custom completion block as a parameter.

```
-(void) customMethodName:(myCustomCompletion) compblock{
    //do stuff
    // check if completion block exist; if we do not check it will throw an exception
    if(complblock)
       compblock(YES);
  }
```

3- How to use block in your Method

```
[self customMethodName:^(BOOL finished) {
if(finished){
    NSLog(@"success");
}
}];
```

Section 152.2: UIView Animations

```
[UIView animateWithDuration:1.0
    animations:^{
        someView.alpha = 0;
        otherView.alpha = 1;
    }
    completion:^(BOOL finished) {
    [someView removeFromSuperview];
}];
```

The carat “^” character defines a block. For example, ^{ ... } is a block. More specifically, it is a block that returns “void” and accepts no arguments. It is equivalent to a method such like: “- (void)something;” but there is no inherent name associated with the code block.

Define a block that can accept arguments work very similarly. To supply an argument to a block, you define the block like so: *^(BOOL someArg, NSString someStr) { ... }*. When you use API calls that support blocks, you'll be writing blocks that look similar to this, especially for animation blocks or NSURLConnection blocks as shown in the above example.

Section 152.3: Modify captured variable

Block will capture variables that appeared in the same lexical scope. Normally these variables are captured as "const" value:

```
int val = 10;
void (^blk)(void) = ^{
    val = 20; // Error! val is a constant value and cannot be modified!
};
```

为了修改变量，需要使用 __block 存储类型修饰符。

```
__block int val = 10;
void (^blk)(void) = ^{
val = 20; // 正确！val 现在可以像普通变量一样被修改。
};
```


In order to modify the variable, you need to use the __block storage type modifier.

```
__block int val = 10;
void (^blk)(void) = ^{
    val = 20; // Correct! val now can be modified as an ordinary variable.
};
```

第153章：自动布局中的内容拥抱/内容压缩

第153.1节：定义：固有内容尺寸

在自动布局出现之前，你总是需要告诉按钮和其他控件它们应该有多大，要么通过设置它们的frame或bounds属性，要么通过在界面构建器中调整它们的大小。但事实证明，大多数控件完全能够根据其内容确定它们需要多少空间。

一个标签知道它的宽度和高度，因为它知道设置在其上的文本长度，以及该文本的字体大小。按钮也是如此，按钮可能将文本与背景图像和一些内边距结合起来。

分段控件、进度条和大多数其他控件也是如此，尽管有些控件可能只有预定的高度，但宽度未知。

这被称为固有内容尺寸，这是自动布局中的一个重要概念。自动布局会询问你的控件它们需要多大空间，并基于这些信息来布局屏幕。

通常你会想使用固有内容尺寸，但在某些情况下你可能不想这样做。你可以通过在控件上设置明确的宽度或高度约束来防止这种情况。

想象一下，当你在UIImageView上设置一张比屏幕大得多的图片时会发生什么。除非你想让视图调整到图片的尺寸，否则通常你会给图片视图一个固定的宽度和高度，并缩放内容。

参考：https://www.raywenderlich.com/115444/auto-layout-tutorial-in-ios-9-part-2-constraints

Chapter 153: Content Hugging/Content Compression in Autolayout

Section 153.1: Definition: Intrinsic Content Size

Before Auto Layout, you always had to tell buttons and other controls how big they should be, either by setting their frame or bounds properties or by resizing them in Interface Builder. But it turns out that most controls are perfectly capable of determining how much space they need, based on their content.

A **label** knows how wide and tall it is because it knows the length of the text that has been set on it, as well as the font size for that text. Likewise for a **button**, which might combine the text with a background image and some padding.

The same is true for segmented controls, progress bars, and most other controls, although some may only have a predetermined height but an unknown width.

This is known as the intrinsic content size, and it is an important concept in Auto Layout. Auto Layout asks your controls how big they need to be and lays out the screen based on that information.

Usually you want to use the `intrinsic content size`, but there are some cases where you may not want to do that. You can prevent this by setting an explicit Width or Height constraint on a control.

Imagine what happens when you set an image on a UIImageView if that image is much larger than the screen. You usually want to give image views a fixed width and height and scale the content, unless you want the view to resize to the dimensions of the image.

Reference: https://www.raywenderlich.com/115444/auto-layout-tutorial-in-ios-9-part-2-constraints

第154章：iOS谷歌地点API

第154.1节：从当前位置获取附近地点

前提条件

1. 在项目中安装pods
2. 安装GooglePlaces SDK
3. 启用定位服务

首先，我们需要通过获取用户当前的经度和纬度来获取用户的位置。

1. 导入GooglePlaces和GooglePlacePicker

```
import GooglePlaces
import GooglePlacePicker
```

2. 添加CLLocationManagerDelegate协议

```
class ViewController: UIViewController, CLLocationManagerDelegate {

}
```

3. 创建你的CLLocationManager()

```
var currentLocation = CLLocationManager()
```

4. 请求授权

```
currentLocation = CLLocationManager()
currentLocation.请求始终授权()
```

5. 创建一个按钮以调用 GooglePlacePicker 方法

@IBAction func placePickerAction(sender: AnyObject) {

```
if CLLocationManager.authorizationStatus() == .AuthorizedAlways {

        let center = CLLocationCoordinate2DMake((currentLocation.location?.coordinate.latitude)!,
(currentLocation.location?.coordinate.longitude)!)
        let northEast = CLLocationCoordinate2DMake(center.latitude + 0.001, center.longitude +
0.001)
        let southWest = CLLocationCoordinate2DMake(center.latitude - 0.001, center.longitude -
0.001)
        let viewport = GMSCoordinateBounds(coordinate: northEast, coordinate: southWest)
        let config = GMSPlacePickerConfig(viewport: viewport)
placePicker = GMSPlacePicker(config: config)

        placePicker?.pickPlaceWithCallback({ (place: GMSPlace?, error: NSError?) -> Void in
            if let error = error {
                print("选择地点错误: \(error.localizedDescription)")
                return
            }

            if let place = place {
               print("地点名称: \(place.name)")
                print("地址: \(place.formattedAddress)")
```


Chapter 154: iOS Google Places API

Section 154.1: Getting Nearby Places from Current Location

Prerequisites

1. Install pods in your project
2. Install the GooglePlaces SDK
3. Enable location services

First we need to get the users location by getting their current longitude and latitude.

1. Import GooglePlaces and GooglePlacePicker

```
import GooglePlaces
import GooglePlacePicker
```

2. Add the `CLLOcationManagerDelegate` protocol

```
class ViewController: UIViewController, CLLocationManagerDelegate {

}
```

3. create your CLLocationManager()

```
var currentLocation = CLLocationManager()
```

4. Request authorization

```
currentLocation = CLLocationManager()
currentLocation.requetAlwayAuthorization()
```

5. Create a button to call the GooglePlacePicker method

@IBAction func placePickerAction(sender: AnyObject) {

```
if CLLOcationManager.authorizationStatues() == .AuthorizedAlways {

        let center = CLLocationCoordinate2DMake((currentLocation.location?.coordinate.latitude)!,
(currentLocation.location?.coordinate.longitude)!)
        let northEast = CLLocationCoordinate2DMake(center.latitude + 0.001, center.longitude +
0.001)
        let southWest = CLLocationCoordinate2DMake(center.latitude - 0.001, center.longitude -
0.001)
        let viewport = GMSCoordinateBounds(coordinate: northEast, coordinate: southWest)
        let config = GMSPlacePickerConfig(viewport: viewport)
        placePicker = GMSPlacePicker(config: config)

        placePicker?.pickPlaceWithCallback({ (place: GMSPlace?, error: NSError?) -> Void in
            if let error = error {
                print("Pick Place error: \(error.localizedDescription)")
                return
            }

            if let place = place {
               print("Place name: \(place.name)")
                print("Address: \(place.formattedAddress)")
```

```
            } else {
                print("地点名称: nil")
                 print("地址: nil")
            }
        })
    }
}
```



```
            } else {
                print("Place name: nil")
                 print("Address: nil")
            }
        })
    }
}
```

第155章：导航栏

第155.1节：SWIFT示例

```
navigationController?.navigationBar.titleTextAttributes = [NSForegroundColorAttributeName:
UIColor.white, NSFontAttributeName:UIFont(name: "HelveticaNeue-CondensedBold", size: 17)!,]
navigationController?.navigationBar.tintColor = .white
navigationController?.navigationBar.barTintColor = .red
navigationController?.navigationBar.isTranslucent = false
navigationController?.navigationBar.barStyle = .black
```

第155.2节：自定义默认导航栏外观

```
// 应用程序中默认的 UINavigationBar 外观
[[UINavigationBar appearance] setTitleTextAttributes:@{NSForegroundColorAttributeName: [UIColor
whiteColor],
NSFontAttributeName : [UIFont
fontWithName:@"HelveticaNeue-CondensedBold" size:17],
                                                        }];

[[UINavigationBar appearance] setTintColor:[UIColor whiteColor]];
[[UINavigationBar appearance] setBarTintColor:[UIColor KNGRed]];
[[UINavigationBar appearance] setTranslucent:NO];
[[UINavigationBar appearance] setBarStyle:UIBarStyleBlack];
[[UIBarButtonItem appearanceWhenContainedIn: [UISearchBar class], nil] setTintColor:[UIColor
KNGGray]];
```


Chapter 155: Navigation Bar

Section 155.1: SWIFT Example

```
navigationController?.navigationBar.titleTextAttributes = [NSForegroundColorAttributeName:
UIColor.white, NSFontAttributeName:UIFont(name: "HelveticaNeue-CondensedBold", size: 17)!,]
navigationController?.navigationBar.tintColor = .white
navigationController?.navigationBar.barTintColor = .red
navigationController?.navigationBar.isTranslucent = false
navigationController?.navigationBar.barStyle = .black
```

Section 155.2: Customize default navigation bar appearance

```
// Default UINavigationBar appearance throughout the app
[[UINavigationBar appearance] setTitleTextAttributes:@{NSForegroundColorAttributeName: [UIColor
whiteColor],
                                                        NSFontAttributeName : [UIFont
fontWithName:@"HelveticaNeue-CondensedBold" size:17],
                                                        }];

[[UINavigationBar appearance] setTintColor:[UIColor whiteColor]];
[[UINavigationBar appearance] setBarTintColor:[UIColor KNGRed]];
[[UINavigationBar appearance] setTranslucent:NO];
[[UINavigationBar appearance] setBarStyle:UIBarStyleBlack];
[[UIBarButtonItem appearanceWhenContainedIn: [UISearchBar class], nil] setTintColor:[UIColor
KNGGray]];
```

第156章：应用范围内的操作

第156.1节：获取最顶层的 UIViewController

获取最顶层的`UIViewController`的常用方法是获取活动`UIWindow`的RootViewController。我写了一个扩展来实现这个功能：

```
extension UIApplication {

func topViewController(_ base: UIViewController? =
UIApplication.shared.keyWindow?.rootViewController) -> UIViewController {

    if let nav = base as? UINavigationController {
        return topViewController(nav.visibleViewController)
    }

    if let tab = base as? UITabBarController {
        if let selected = tab.selectedViewController {
            return topViewController(selected)
        }
    }

    if let presented = base?.presentedViewController {
        return topViewController(presented)
    }

    return base!
}
```

第156.2节：拦截系统事件

使用iOS的NotificationCenter，它非常强大，您可以拦截某些全应用范围的事件：

```
NotificationCenter.default.addObserver(
        self,
selector: #selector(ViewController.do(_:)),
        name: NSNotification.Name.UIApplicationDidBecomeActive,
        object: nil)
```

您可以注册更多事件，详情请查看
https://developer.apple.com/reference/foundation/nsnotification.name。

Chapter 156: App wide operations

Section 156.1: Get the top most UIViewController

A common approach to get the top most `UIViewController` is to get the RootViewController of your active `UIWindow`. I wrote an extension for this:

```
extension UIApplication {

func topViewController(_ base: UIViewController? =
UIApplication.shared.keyWindow?.rootViewController) -> UIViewController {

    if let nav = base as? UINavigationController {
        return topViewController(nav.visibleViewController)
    }

    if let tab = base as? UITabBarController {
        if let selected = tab.selectedViewController {
            return topViewController(selected)
        }
    }

    if let presented = base?.presentedViewController {
        return topViewController(presented)
    }

    return base!
}
```

Section 156.2: Intercept System Events

Using the NotificationCenter of iOS, which can be very powerful, you are able to intercept certain app-wide events:

```
NotificationCenter.default.addObserver(
        self,
        selector: #selector(ViewController.do(_:)),
        name: NSNotification.Name.UIApplicationDidBecomeActive,
        object: nil)
```

You can register for a lot of more events, just take a look at
https://developer.apple.com/reference/foundation/nsnotification.name.

第157章：CoreImage滤镜

第157.1节：Core Image滤镜示例

Objective-C

只需记录此日志，查看如何使用特定滤镜

```
NSArray *properties = [CIFilter filterNamesInCategory:kCICategoryBuiltIn];

for (NSString *filterName in properties)
{
CIFilter *fltr = [CIFilter filterWithName:filterName];
NSLog(@"%@", [fltr attributes]);
}
```

以CISepiaTone为例，系统日志如下

```
CIAttributeFilterDisplayName = "Sepia Tone";
CIAttributeFilterName = CISepiaTone;
    CIAttributeReferenceDocumentation =
"http://developer.apple.com/cgi-bin/apple_ref.cgi?apple_ref=//apple_ref/doc/filter/ci/CISepiaTone";
   inputImage =     {
        CIAttributeClass = CIImage;
CIAttributeDescription = “用作输入图像的图像。对于也使用
背景图像的滤镜，这是前景图像。”;
CIAttributeDisplayName = 图像；
       CIAttributeType = CIAttributeTypeImage；
    };
    inputIntensity =     {
CIAttributeClass = NSNumber；
CIAttributeDefault = 1；
CIAttributeDescription = “棕褐色效果的强度。值为1.0时创建单色棕褐色图像。值为0.0时对图像无影响。”；

CIAttributeDisplayName = 强度；
        CIAttributeIdentity = 0；
CIAttributeMin = 0；
        CIAttributeSliderMax = 1；
        CIAttributeSliderMin = 0；
       CIAttributeType = CIAttributeTypeScalar；
    };
}
```

使用上述系统日志，我们按如下方式设置滤镜：

```
CIImage *beginImage = [CIImage imageWithCGImage:[myImageView.image CGImage]]；
    CIContext *context = [CIContext contextWithOptions:nil]；
    //选择滤镜名称和强度
CIFilter *filter = [CIFilter filterWithName:@"CISepiaTone" keysAndValues: kCIInputImageKey,
beginImage, @"inputIntensity", [NSNumber numberWithFloat:0.8], nil];
CIImage *outputImage = [filter outputImage];

    CGImageRef cgimg = [context createCGImage:outputImage fromRect:[outputImage extent]];
    UIImage *newImg = [UIImage imageWithCGImage:cgimg];

    [myImageView1 setImage:newImg];

    CGImageRelease(cgimg);
```


Chapter 157: CoreImage Filters

Section 157.1: Core Image Filter Example

Objective-C

Just log this see how to use a particular filter

```
NSArray *properties = [CIFilter filterNamesInCategory:kCICategoryBuiltIn];

for (NSString *filterName in properties)
{
CIFilter *fltr = [CIFilter filterWithName:filterName];
NSLog(@"%@", [fltr attributes]);
}
```

In case of CISepiaTone the system log is as follows

```
CIAttributeFilterDisplayName = "Sepia Tone";
CIAttributeFilterName = CISepiaTone;
    CIAttributeReferenceDocumentation =
"http://developer.apple.com/cgi-bin/apple_ref.cgi?apple_ref=//apple_ref/doc/filter/ci/CISepiaTone";
    inputImage =     {
        CIAttributeClass = CIImage;
        CIAttributeDescription = "The image to use as an input image. For filters that also use a
background image, this is the foreground image.";
        CIAttributeDisplayName = Image;
        CIAttributeType = CIAttributeTypeImage;
    };
    inputIntensity =     {
        CIAttributeClass = NSNumber;
        CIAttributeDefault = 1;
        CIAttributeDescription = "The intensity of the sepia effect. A value of 1.0 creates a
monochrome sepia image. A value of 0.0 has no effect on the image.";
        CIAttributeDisplayName = Intensity;
        CIAttributeIdentity = 0;
        CIAttributeMin = 0;
        CIAttributeSliderMax = 1;
        CIAttributeSliderMin = 0;
        CIAttributeType = CIAttributeTypeScalar;
    };
}
```

Using the above system log we set up the filter as below:

```
    CIImage *beginImage = [CIImage imageWithCGImage:[myImageView.image CGImage]];
    CIContext *context = [CIContext contextWithOptions:nil];
    //select Filter Name and Intensity
    CIFilter *filter = [CIFilter filterWithName:@"CISepiaTone" keysAndValues: kCIInputImageKey,
beginImage, @"inputIntensity", [NSNumber numberWithFloat:0.8], nil];
    CIImage *outputImage = [filter outputImage];

    CGImageRef cgimg = [context createCGImage:outputImage fromRect:[outputImage extent]];
    UIImage *newImg = [UIImage imageWithCGImage:cgimg];

    [myImageView1 setImage:newImg];

    CGImageRelease(cgimg);
```

以上代码生成的图像

设置滤镜的另一种方法

```
UIImageView *imageView1=[[UIImageView alloc]initWithFrame:CGRectMake(0, 0,
self.view.frame.size.width, self.view.frame.size.height/2)];
UIImageView *imageView2=[[UIImageView alloc]initWithFrame:CGRectMake(0,
self.view.frame.size.height/2, self.view.frame.size.width, self.view.frame.size.height/2)];
imageView1.image=[UIImage imageNamed:@"image.png"];

CIImage *beginImage = [CIImage imageWithCGImage:[imageView1.image CGImage]];
CIContext *context = [CIContext contextWithOptions:nil];
//选择滤镜名称和强度

CIFilter *filter = [CIFilter filterWithName:@"CIColorPosterize"];
[filter setValue:beginImage forKey:kCIInputImageKey];
[filter setValue:[NSNumber numberWithFloat:8.0] forKey:@"inputLevels"];
CIImage *outputImage = [filter outputImage];

CGImageRef cgimg = [context createCGImage:outputImage fromRect:[outputImage extent]];
UIImage *newImg = [UIImage imageWithCGImage:cgimg];

[imageView2 setImage:newImg];

CGImageRelease(cgimg);
[self.view addSubview:imageView1];
```


Image generated by the above code

An alternative way to set up a filter

```
UIImageView *imageView1=[[UIImageView alloc]initWithFrame:CGRectMake(0, 0,
self.view.frame.size.width, self.view.frame.size.height/2)];
UIImageView *imageView2=[[UIImageView alloc]initWithFrame:CGRectMake(0,
self.view.frame.size.height/2, self.view.frame.size.width, self.view.frame.size.height/2)];
imageView1.image=[UIImage imageNamed:@"image.png"];

CIImage *beginImage = [CIImage imageWithCGImage:[imageView1.image CGImage]];
CIContext *context = [CIContext contextWithOptions:nil];
//select Filter Name and Intensity

CIFilter *filter = [CIFilter filterWithName:@"CIColorPosterize"];
[filter setValue:beginImage forKey:kCIInputImageKey];
[filter setValue:[NSNumber numberWithFloat:8.0] forKey:@"inputLevels"];
CIImage *outputImage = [filter outputImage];

CGImageRef cgimg = [context createCGImage:outputImage fromRect:[outputImage extent]];
UIImage *newImg = [UIImage imageWithCGImage:cgimg];

[imageView2 setImage:newImg];

CGImageRelease(cgimg);
[self.view addSubview:imageView1];
```

```
[self.view addSubview:imageView2];
```

由此代码生成的图像

所有可用滤镜如下

```
/* CIAccordionFoldTransition,
    CIAdditionCompositing,
    CIAffineClamp,
    CIAffineTile,
    CIAffineTransform,
    CIAreaAverage,
    CIAreaHistogram,
    CIAreaMaximum,
    CIAreaMaximumAlpha,
    CIAreaMinimum,
CIAreaMinimumAlpha,
    CIAztecCodeGenerator,
    CIBarsSwipeTransition,
    CIBlendWithAlphaMask,
    CIBlendWithMask,
CIBloom,
CIBoxBlur,
CIBumpDistortion,
    CIBumpDistortionLinear,
    CICheckerboardGenerator,
    CICircleSplashDistortion,
    CICircularScreen,
CICircularWrap,
    CICMYKHalftone,
    CICode128BarcodeGenerator,
```



```
[self.view addSubview:imageView2];
```

Image generated from this code

All Available Filters are as below

```
/* CIAccordionFoldTransition,
    CIAdditionCompositing,
    CIAffineClamp,
    CIAffineTile,
    CIAffineTransform,
    CIAreaAverage,
    CIAreaHistogram,
    CIAreaMaximum,
    CIAreaMaximumAlpha,
    CIAreaMinimum,
    CIAreaMinimumAlpha,
    CIAztecCodeGenerator,
    CIBarsSwipeTransition,
    CIBlendWithAlphaMask,
    CIBlendWithMask,
    CIBloom,
    CIBoxBlur,
    CIBumpDistortion,
    CIBumpDistortionLinear,
    CICheckerboardGenerator,
    CICircleSplashDistortion,
    CICircularScreen,
    CICircularWrap,
    CICMYKHalftone,
    CICode128BarcodeGenerator,
```

CIColorBlendMode，
CIColorBurnBlendMode，
CIColorClamp，
CIColorControls，
CIColorCrossPolynomial，
CIColorCube，
CIColorCubeWithColorSpace，
CIColorDodgeBlendMode，
CIColorInvert，
CIColorMap，
CIColorMatrix，
CIColorMonochrome，
CIColorPolynomial，
CIColorPosterize，
CIColumnAverage，
CIComicEffect，
CIConstantColorGenerator，
CIConvolution3X3，
CIConvolution5X5,
CIConvolution7X7,
CIConvolution9Horizontal,
CIConvolution9Vertical,
CICopyMachineTransition,
CICrop,
CICrystallize（晶体化），
CIDarkenBlendMode（变暗混合模式），
CIDepthOfField（景深），
CIDifferenceBlendMode（差值混合模式），
CIDiscBlur（圆盘模糊），
CIDisintegrateWithMaskTransition（带蒙版过渡的瓦解），
CIDisplacementDistortion（位移扭曲），
CIDissolveTransition（溶解过渡），
CIDivideBlendMode（除法混合模式），
CIDotScreen（点状网屏），
CIDroste（德罗斯特效应），
CIEdges（边缘），
CIEdgeWork（边缘处理），
CIEightfoldReflectedTile（八重反射平铺），
CIExclusionBlendMode（排除混合模式），
CIExposureAdjust（曝光调整），
CIFalseColor（伪彩色），
CIFlashTransition（闪光过渡），
CIFourfoldReflectedTile（四重反射平铺），
CIFourfoldRotatedTile（四重旋转平铺），
CIFourfoldTranslatedTile（四重平移平铺），
CIGammaAdjust（伽马调整），
CIGaussianBlur（高斯模糊），
CIGaussianGradient（高斯渐变），
CIGlassDistortion（玻璃扭曲），
CIGlassLozenge（玻璃菱形），
CIGlideReflectedTile（滑动反射平铺），
CIGloom（阴影），
CIHardLightBlendMode（强光混合模式），
CIHatchedScreen（阴影线网），
CIHeightFieldFromMask（从蒙版生成高度场），
CIHexagonalPixellate（六边形像素化），
CIHighlightShadowAdjust（高光阴影调整），
CIHistogramDisplayFilter（直方图显示滤镜），
CIHoleDistortion（孔洞扭曲），
CIHueAdjust（色相调整），
CIHueBlendMode（色相混合模式），
CIKaleidoscope（万花筒），


```
CIColorBlendMode,
CIColorBurnBlendMode,
CIColorClamp,
CIColorControls,
CIColorCrossPolynomial,
CIColorCube,
CIColorCubeWithColorSpace,
CIColorDodgeBlendMode,
CIColorInvert,
CIColorMap,
CIColorMatrix,
CIColorMonochrome,
CIColorPolynomial,
CIColorPosterize,
CIColumnAverage,
CIComicEffect,
CIConstantColorGenerator,
CIConvolution3X3,
CIConvolution5X5,
CIConvolution7X7,
CIConvolution9Horizontal,
CIConvolution9Vertical,
CICopyMachineTransition,
CICrop,
CICrystallize,
CIDarkenBlendMode,
CIDepthOfField,
CIDifferenceBlendMode,
CIDiscBlur,
CIDisintegrateWithMaskTransition,
CIDisplacementDistortion,
CIDissolveTransition,
CIDivideBlendMode,
CIDotScreen,
CIDroste,
CIEdges,
CIEdgeWork,
CIEightfoldReflectedTile,
CIExclusionBlendMode,
CIExposureAdjust,
CIFalseColor,
CIFlashTransition,
CIFourfoldReflectedTile,
CIFourfoldRotatedTile,
CIFourfoldTranslatedTile,
CIGammaAdjust,
CIGaussianBlur,
CIGaussianGradient,
CIGlassDistortion,
CIGlassLozenge,
CIGlideReflectedTile,
CIGloom,
CIHardLightBlendMode,
CIHatchedScreen,
CIHeightFieldFromMask,
CIHexagonalPixellate,
CIHighlightShadowAdjust,
CIHistogramDisplayFilter,
CIHoleDistortion,
CIHueAdjust,
CIHueBlendMode,
CIKaleidoscope,
```

CILanczosScaleTransform（兰氏缩放变换），
CILenticularHaloGenerator（透镜光晕生成器），
CILightenBlendMode（变亮混合模式），
CILightTunnel（光隧道），
CILinearBurnBlendMode（线性加深混合模式），
CILinearDodgeBlendMode（线性减淡混合模式），
CILinearGradient（线性渐变），
CILinearToSRGBToneCurve（线性到sRGB色调曲线），
CILineOverlay（线条叠加），
CILineScreen（线条网点），
CILuminosityBlendMode（亮度混合模式），
CIMaskedVariableBlur（带蒙版的可变模糊），
CIMaskToAlpha（蒙版转Alpha），
CIMaximumComponent（最大分量），
CIMaximumCompositing（最大合成），
CIMedianFilter（中值滤波），
CIMinimumComponent（最小分量），
CIMinimumCompositing（最小合成），
CIModTransition（模运算过渡），
CIMotionBlur（运动模糊），
CIMultiplyBlendMode（正片叠底混合模式），
CIMultiplyCompositing（正片叠底合成），
CINoiseReduction（降噪），
CIOpTile（操作平铺），
CIOverlayBlendMode（叠加混合模式），
CIPageCurlTransition（翻页过渡），
CIPageCurlWithShadowTransition（带阴影的翻页过渡），
CIParallelogramTile（平行四边形平铺），
CIPDF417条码生成器，
CIPerspective校正，
CIPerspective平铺，
CIPerspectiveTransform,
CIPerspectiveTransformWithExtent,
CIPhotoEffectChrome,
CIPhotoEffectFade,
CIPhotoEffectInstant,
CIPhotoEffectMono,
CIPhotoEffectNoir,
CIPhotoEffectProcess,
CIPhotoEffectTonal,
CIPhotoEffectTransfer,
CIPinchDistortion,
CIPinLightBlendMode,
CIPixellate,
CIPointillize,
CIQRCodeGenerator,
CIRadialGradient,
CIRandomGenerator,
CIRippleTransition,
CIRowAverage,
CISaturationBlendMode,
CIScreenBlendMode,
CISepiaTone,
CIShadedMaterial,
CISharpenLuminance,
CISixfoldReflectedTile,
CISixfoldRotatedTile,
CISmoothLinearGradient,
CISoftLightBlendMode,
CISourceAtopCompositing,
CISourceInCompositing,
CISourceOutCompositing,


```
CILanczosScaleTransform,
CILenticularHaloGenerator,
CILightenBlendMode,
CILightTunnel,
CILinearBurnBlendMode,
CILinearDodgeBlendMode,
CILinearGradient,
CILinearToSRGBToneCurve,
CILineOverlay,
CILineScreen,
CILuminosityBlendMode,
CIMaskedVariableBlur,
CIMaskToAlpha,
CIMaximumComponent,
CIMaximumCompositing,
CIMedianFilter,
CIMinimumComponent,
CIMinimumCompositing,
CIModTransition,
CIMotionBlur,
CIMultiplyBlendMode,
CIMultiplyCompositing,
CINoiseReduction,
CIOpTile,
CIOverlayBlendMode,
CIPageCurlTransition,
CIPageCurlWithShadowTransition,
CIParallelogramTile,
CIPDF417BarcodeGenerator,
CIPerspectiveCorrection,
CIPerspectiveTile,
CIPerspectiveTransform,
CIPerspectiveTransformWithExtent,
CIPhotoEffectChrome,
CIPhotoEffectFade,
CIPhotoEffectInstant,
CIPhotoEffectMono,
CIPhotoEffectNoir,
CIPhotoEffectProcess,
CIPhotoEffectTonal,
CIPhotoEffectTransfer,
CIPinchDistortion,
CIPinLightBlendMode,
CIPixellate,
CIPointillize,
CIQRCodeGenerator,
CIRadialGradient,
CIRandomGenerator,
CIRippleTransition,
CIRowAverage,
CISaturationBlendMode,
CIScreenBlendMode,
CISepiaTone,
CIShadedMaterial,
CISharpenLuminance,
CISixfoldReflectedTile,
CISixfoldRotatedTile,
CISmoothLinearGradient,
CISoftLightBlendMode,
CISourceAtopCompositing,
CISourceInCompositing,
CISourceOutCompositing,
```

```
CISourceOverCompositing,
    CISpotColor,
CISpotLight,
    CISRGBToneCurveToLinear,
    CIStarShineGenerator,
    CIStraightenFilter,
    CIStretchCrop,
CIStripesGenerator,
    CISubtractBlendMode,
    CISunbeamsGenerator,
    CISwipeTransition,
    CITemperatureAndTint,
    CIToneCurve,
CITorusLensDistortion,
    CITriangleKaleidoscope,
    CITriangleTile,
CITwelvefoldReflectedTile,
    CITwirlDistortion,
CIUnsharpMask,
    CIVibrance,
    CIVignette,
    CIVignetteEffect,
    CIVortexDistortion,
    CIWhitePointAdjust,
    CIZoomBlur*/
```



```
    CISourceOverCompositing,
    CISpotColor,
    CISpotLight,
    CISRGBToneCurveToLinear,
    CIStarShineGenerator,
    CIStraightenFilter,
    CIStretchCrop,
    CIStripesGenerator,
    CISubtractBlendMode,
    CISunbeamsGenerator,
    CISwipeTransition,
    CITemperatureAndTint,
    CIToneCurve,
    CITorusLensDistortion,
    CITriangleKaleidoscope,
    CITriangleTile,
    CITwelvefoldReflectedTile,
    CITwirlDistortion,
    CIUnsharpMask,
    CIVibrance,
    CIVignette,
    CIVignetteEffect,
    CIVortexDistortion,
    CIWhitePointAdjust,
    CIZoomBlur*/
```

第158章：使用CoreImage/OpenCV进行人脸检测

第158.1节：人脸与特征检测

Objective-C

导入以下内容到你的视图控制器

```
#import <CoreImage/CoreImage.h>
#import <CoreImage/CoreImage.h>
#import <QuartzCore/QuartzCore.h>
```

调用函数

```
[self faceDetector];
```

函数定义：

```
-(void)faceDetector
{
    // 加载用于人脸检测的图片
    UIImageView* image = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@"download.jpeg"]];

    // 绘制人脸检测图片
    [self.view addSubview:image];

    // 在后台执行用于标记人脸的方法
    [self performSelectorInBackground:@selector(markFaces:) withObject:image];

    // 在y轴翻转图片以匹配Core Image使用的坐标系
    [image setTransform:CGAffineTransformMakeScale(1, -1)];

    // 翻转整个窗口使所有内容正立
    [self.view setTransform:CGAffineTransformMakeScale(1, -1)];


}
```

标记人脸函数

```
// 添加人脸方框及眼睛和嘴巴的颜色遮罩
-(void)markFaces:(UIImageView *)facePicture
{
    // 使用之前加载的人脸检测图片绘制CI图像
    CIImage* image = [CIImage imageWithCGImage:facePicture.image.CGImage];

    // 创建人脸检测器 - 由于速度不是问题，我们将使用高精度
    // 检测器
CIDetector* detector = [CIDetector detectorOfType:CIDetectorTypeFace
                                          context:nil options:[NSDictionary
dictionaryWithObject:CIDetectorAccuracyHigh forKey:CIDetectorAccuracy]];

    // 创建一个数组，包含检测器检测到的所有人脸
    NSArray* features = [detector featuresInImage:image];
NSLog(@"人脸数量 %d",[features count]);
```


Chapter 158: Face Detection Using CoreImage/OpenCV

Section 158.1: Face and Feature Detection

Objective-C

Import the following to your ViewController

```
#import <CoreImage/CoreImage.h>
#import <CoreImage/CoreImage.h>
#import <QuartzCore/QuartzCore.h>
```

Call the function

```
[self faceDetector];
```

Function definition:

```
-(void)faceDetector
{
    // Load the picture for face detection
    UIImageView* image = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@"download.jpeg"]];

    // Draw the face detection image
    [self.view addSubview:image];

    // Execute the method used to markFaces in background
    [self performSelectorInBackground:@selector(markFaces:) withObject:image];

    // flip image on y-axis to match coordinate system used by core image
    [image setTransform:CGAffineTransformMakeScale(1, -1)];

    // flip the entire window to make everything right side up
    [self.view setTransform:CGAffineTransformMakeScale(1, -1)];


}
```

Mark Face Function

```
//Adds face squares and color masks to eyes and mouth
-(void)markFaces:(UIImageView *)facePicture
{
    // draw a CI image with the previously loaded face detection picture
    CIImage* image = [CIImage imageWithCGImage:facePicture.image.CGImage];

    // create a face detector - since speed is not an issue we'll use a high accuracy
    // detector
    CIDetector* detector = [CIDetector detectorOfType:CIDetectorTypeFace
                                              context:nil options:[NSDictionary
dictionaryWithObject:CIDetectorAccuracyHigh forKey:CIDetectorAccuracy]];

    // create an array containing all the detected faces from the detector
    NSArray* features = [detector featuresInImage:image];
    NSLog(@"Number of faces %d",[features count]);
```

```
    // 我们将遍历每一个检测到的人脸。CIFaceFeature为我们提供了// 整张脸的宽度，以及每只眼
    睛和嘴巴的坐标（如果检测到的话）。// 还提供了眼睛和嘴巴的BOOL值，以便我们检查它们
    是否存在。

    //    for (features in image)
    //    {
    for(CIFaceFeature* faceFeature in features)
    {
        // 获取人脸的宽度
        CGFloat faceWidth = faceFeature.bounds.size.width;

        // 使用人脸的边界创建一个UIView
        UIView* faceView = [[UIView alloc] initWithFrame:faceFeature.bounds];

        // 在新创建的UIView周围添加边框
faceView.layer.borderWidth = 1;
        faceView.layer.borderColor = [[UIColor redColor] CGColor];

        // 将新视图添加到视图中以在脸部周围创建一个框
        [self.view addSubview:faceView];

        if(faceFeature.hasLeftEyePosition)
        {
            // 根据脸部宽度创建一个UIView
            UIView* leftEyeView = [[UIView alloc]
initWithFrame:CGRectMake(faceFeature.leftEyePosition.x-faceWidth*0.15,
faceFeature.leftEyePosition.y-faceWidth*0.15, faceWidth*0.3, faceWidth*0.3)];
            // 更改眼睛视图的背景颜色
            [leftEyeView setBackgroundColor:[[UIColor blueColor] colorWithAlphaComponent:0.3]];
            // 根据脸部位置设置leftEyeView的位置
            [leftEyeView setCenter:faceFeature.leftEyePosition];
            // 圆角处理
leftEyeView.layer.cornerRadius = faceWidth*0.15;
            // 将视图添加到窗口
            [self.view addSubview:leftEyeView];
        }

        if(faceFeature.hasRightEyePosition)
        {
            // 根据脸部宽度创建一个UIView
            UIView* leftEye = [[UIView alloc]
initWithFrame:CGRectMake(faceFeature.rightEyePosition.\tx-faceWidth*0.15,
faceFeature.rightEyePosition.y-faceWidth*0.15, faceWidth*0.3, faceWidth*0.3)];
            // 更改眼睛视图的背景颜色
            [leftEye setBackgroundColor:[[UIColor blueColor] colorWithAlphaComponent:0.3]];
            // 根据脸部设置 rightEyeView 的位置
            [leftEye setCenter:faceFeature.rightEyePosition];
            // 圆角处理
leftEye.layer.cornerRadius = faceWidth*0.15;
            // 将新视图添加到窗口
            [self.view addSubview:leftEye];
        }

        if(faceFeature.hasMouthPosition)
        {
            // 创建一个 UIView，大小基于脸宽
            UIView* mouth = [[UIView alloc] initWithFrame:CGRectMake(faceFeature.mouthPosition.x-
faceWidth*0.2, faceFeature.mouthPosition.y-faceWidth*0.2, faceWidth*0.4, faceWidth*0.4)];
            // 将嘴部背景色改为绿色
            [mouth setBackgroundColor:[[UIColor greenColor] colorWithAlphaComponent:0.3]];
            // 根据脸部设置 mouthView 的位置
            [mouth setCenter:faceFeature.mouthPosition];
```



```
    // we'll iterate through every detected face.  CIFaceFeature provides us
    // with the width for the entire face, and the coordinates of each eye
    // and the mouth if detected.  Also provided are BOOL's for the eye's and
    // mouth so we can check if they already exist.
    //    for (features in image)
    //    {
    for(CIFaceFeature* faceFeature in features)
    {
        // get the width of the face
        CGFloat faceWidth = faceFeature.bounds.size.width;

        // create a UIView using the bounds of the face
        UIView* faceView = [[UIView alloc] initWithFrame:faceFeature.bounds];

        // add a border around the newly created UIView
        faceView.layer.borderWidth = 1;
        faceView.layer.borderColor = [[UIColor redColor] CGColor];

        // add the new view to create a box around the face
        [self.view addSubview:faceView];

        if(faceFeature.hasLeftEyePosition)
        {
            // create a UIView with a size based on the width of the face
            UIView* leftEyeView = [[UIView alloc]
initWithFrame:CGRectMake(faceFeature.leftEyePosition.x-faceWidth*0.15,
faceFeature.leftEyePosition.y-faceWidth*0.15, faceWidth*0.3, faceWidth*0.3)];
            // change the background color of the eye view
            [leftEyeView setBackgroundColor:[[UIColor blueColor] colorWithAlphaComponent:0.3]];
            // set the position of the leftEyeView based on the face
            [leftEyeView setCenter:faceFeature.leftEyePosition];
            // round the corners
            leftEyeView.layer.cornerRadius = faceWidth*0.15;
            // add the view to the window
            [self.view addSubview:leftEyeView];
        }

        if(faceFeature.hasRightEyePosition)
        {
            // create a UIView with a size based on the width of the face
            UIView* leftEye = [[UIView alloc]
initWithFrame:CGRectMake(faceFeature.rightEyePosition.x-faceWidth*0.15,
faceFeature.rightEyePosition.y-faceWidth*0.15, faceWidth*0.3, faceWidth*0.3)];
            // change the background color of the eye view
            [leftEye setBackgroundColor:[[UIColor blueColor] colorWithAlphaComponent:0.3]];
            // set the position of the rightEyeView based on the face
            [leftEye setCenter:faceFeature.rightEyePosition];
            // round the corners
            leftEye.layer.cornerRadius = faceWidth*0.15;
            // add the new view to the window
            [self.view addSubview:leftEye];
        }

        if(faceFeature.hasMouthPosition)
        {
            // create a UIView with a size based on the width of the face
            UIView* mouth = [[UIView alloc] initWithFrame:CGRectMake(faceFeature.mouthPosition.x-
faceWidth*0.2, faceFeature.mouthPosition.y-faceWidth*0.2, faceWidth*0.4, faceWidth*0.4)];
            // change the background color for the mouth to green
            [mouth setBackgroundColor:[[UIColor greenColor] colorWithAlphaComponent:0.3]];
            // set the position of the mouthView based on the face
            [mouth setCenter:faceFeature.mouthPosition];
```

```
            // 圆角处理
嘴部。layer.cornerRadius = faceWidth*0.2;
            // 将新视图添加到窗口
            [self.view 添加子视图:mouth];
        }
    }

    // }
}
```

功能的模拟器截图

```
            // round the corners
            mouth.layer.cornerRadius = faceWidth*0.2;
            // add the new view to the window
            [self.view addSubview:mouth];
        }
    }

    // }
}
```

The Simulator ScreenShot for the Function

第159章：MPMediaPickerDelegate

第159.1节：使用MPMediaPickerControllerDelegate加载音乐并用AVAudioPlayer播放

按照以下步骤进行：

- 在Info.plist中添加“NSAppleMusicUsageDescription”以获取隐私权限。
- 确保你的音乐已存储在iPhone中。模拟器中无法使用。

版本 ≥ iOS 10.0.1

```
import UIKit
import AVFoundation
import MediaPlayer

class ViewController: UIViewController, MPMediaPickerControllerDelegate {

    var avMusicPlayer: AVAudioPlayer!
    var mpMediapicker: MPMediaPickerController!
    var mediaItems = [MPMediaItem]()
    let currentIndex = 0

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    func audioPlayerDidFinishPlaying(_ player: AVAudioPlayer, successfully flag: Bool){
        //该怎么办？
    }

    func mediaPicker(_ mediaPicker: MPMediaPickerController, didPickMediaItems mediaItemCollection:
MPMediaItemCollection) {
        mediaItems = mediaItemCollection.items
       updatePlayer()
        self.dismiss(animated: true, completion: nil)
    }

    func updatePlayer(){
        let item = mediaItems[currentIndex]
        // 尝试使用路径设置 AVAudioPlayer，如果成功，则设置 AVMusicPlayer 和歌曲相关值。

        if let path: NSURL = item.assetURL as NSURL? {
            do
            {
avMusicPlayer = try AVAudioPlayer(contentsOf: path as URL)
                avMusicPlayer.enableRate = true
avMusicPlayer.rate = 1.0
                avMusicPlayer.numberOfLoops = 0
                avMusicPlayer.currentTime = 0
            }
            catch
            {
avMusicPlayer = nil
            }
        }
    }

    @IBAction func Play(_ sender: AnyObject) {
```


Chapter 159: MPMediaPickerDelegate

Section 159.1: Load music with MPMediaPickerControllerDelegate and play it with AVAudioPlayer

Go through the steps:

- Add 'NSAppleMusicUsageDescription' to your Info.plist for the privacy authority.
- Make sure your music is available in your iPhone. It will not work in the simulator.

Version ≥ iOS 10.0.1

```
import UIKit
import AVFoundation
import MediaPlayer

class ViewController: UIViewController, MPMediaPickerControllerDelegate {

    var avMusicPlayer: AVAudioPlayer!
    var mpMediapicker: MPMediaPickerController!
    var mediaItems = [MPMediaItem]()
    let currentIndex = 0

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    func audioPlayerDidFinishPlaying(_ player: AVAudioPlayer, successfully flag: Bool){
        //What to do?
    }

    func mediaPicker(_ mediaPicker: MPMediaPickerController, didPickMediaItems mediaItemCollection:
MPMediaItemCollection) {
        mediaItems = mediaItemCollection.items
        updatePlayer()
        self.dismiss(animated: true, completion: nil)
    }

    func updatePlayer(){
        let item = mediaItems[currentIndex]
        // DO-TRY-CATCH try to setup AVAudioPlayer with the path, if successful, sets up the
AVMusicPlayer, and song values.
        if let path: NSURL = item.assetURL as NSURL? {
            do
            {
                avMusicPlayer = try AVAudioPlayer(contentsOf: path as URL)
                avMusicPlayer.enableRate = true
                avMusicPlayer.rate = 1.0
                avMusicPlayer.numberOfLoops = 0
                avMusicPlayer.currentTime = 0
            }
            catch
            {
                avMusicPlayer = nil
            }
        }
    }

    @IBAction func Play(_ sender: AnyObject) {
```

```
        //AVMusicPlayer.deviceCurrentTime
        avMusicPlayer.play()
    }

    @IBAction func Stop(_ sender: AnyObject) {
        avMusicPlayer.stop()
    }

    @IBAction func picker(_ sender: AnyObject) {
        mpMediapicker = MPMediaPickerController.self(mediaTypes:MPMediaType.music)
        mpMediapicker.allowsPickingMultipleItems = false
mpMediapicker.delegate = self
        self.present(mpMediapicker, animated: true, completion: nil)
    }

}
```



```
        //AVMusicPlayer.deviceCurrentTime
        avMusicPlayer.play()
    }

    @IBAction func Stop(_ sender: AnyObject) {
        avMusicPlayer.stop()
    }

    @IBAction func picker(_ sender: AnyObject) {
        mpMediapicker = MPMediaPickerController.self(mediaTypes:MPMediaType.music)
        mpMediapicker.allowsPickingMultipleItems = false
        mpMediapicker.delegate = self
        self.present(mpMediapicker, animated: true, completion: nil)
    }

}
```

第160章：图表（Coreplot）

第160.1节：使用CorePlot制作图表

Core Plot提供了podspec，因此您可以使用cocoapods作为您的库管理器，这将使安装和更新更加简单

在您的系统上安装cocoapods

在项目目录中通过输入`pod init`添加一个名为Podfile的文本文件到您的项目中

在Podfile中添加这一行 pod 'CorePlot', '~> 1.6'

在终端中，切换到您的项目目录并运行pod install

Cocoapods将生成一个xcworkspace文件，您应使用该文件启动项目（.xcodeproj文件将不包含pod库）

打开由 CocoaPods 生成的 .xcworkspace

在 ViewController.h 文件中

```
#import <CorePlot/ios/CorePlot.h>
//#import "CorePlot-CocoaTouch.h" 或上述导入语句
@interface ViewController : UIViewController<CPTPlotDataSource>
```

在 ViewController.m 文件中

```
-(void)loadView
{
    [super loadView];
    // 我们需要一个 hostview，你可以在 IB 中创建一个（并创建一个 outlet），或者直接这样做：
        CPTGraphHostingView* hostView = [[CPTGraphHostingView alloc] initWithFrame:CGRectMake(10, 40,
300, 400)];
hostView.backgroundColor=[UIColor whiteColor];
    self.view.backgroundColor=[UIColor blackColor];
    [self.view addSubview: hostView];
    // 创建一个 CPTGraph 对象并添加到 hostView
   CPTGraph* graph = [[CPTXYGraph alloc] initWithFrame:CGRectMake(10, 40, 300, 400)];
   hostView.hostedGraph = graph;
    // 从图表中获取（默认）plotspace，以便我们设置其 x/y 范围
   CPTXYPlotSpace *plotSpace = (CPTXYPlotSpace *) graph.defaultPlotSpace;
    // 注意这些 CPTPlotRange 是由起点和长度定义的（不是起点和终点）！！
    [plotSpace setYRange: [CPTPlotRange plotRangeWithLocation:CPTDecimalFromFloat( 0 )
length:CPTDecimalFromFloat( 20 )]];
    [plotSpace setXRange: [CPTPlotRange plotRangeWithLocation:CPTDecimalFromFloat( -4 )
length:CPTDecimalFromFloat( 8 )]];
    // 创建图表（我们尚未定义实际的 x/y 值，这些将由数据源提供...）

CPTScatterPlot* plot = [[CPTScatterPlot alloc] initWithFrame:CGRectZero];
    // 我们保持简单，让此类充当数据源（因此我们实现了
<CPTPlotDataSource>）
plot.dataSource = self;
    // 最后，将创建的图表添加到我们之前创建的 CPTGraph 对象的默认绘图空间中

    [graph addPlot:plot toPlotSpace:graph.defaultPlotSpace];
}
// 该方法存在是因为此类也作为我们图表的数据源
```


Chapter 160: Graph (Coreplot)

Section 160.1: Making graphs with CorePlot

Core Plot provides a podspec, so you can use cocoapods as your library manager which should make installing and updating much simpler

Install cocoapods on your system

In the project directory add a text file to your project called Podfile by typing `pod init` in your project directory

In the Podfile add the line pod 'CorePlot', '~> 1.6'

In the terminal, cd to your project directory and run pod install

Cocoapods will generate a xcworkspace file, which you should use for launching your project (the .xcodeproj file will not include the pod libraries)

Open the .xcworkspace generated by CocoaPods

In the ViewController.h File

```
#import <CorePlot/ios/CorePlot.h>
//#import "CorePlot-CocoaTouch.h" or the above import statement
@interface ViewController : UIViewController<CPTPlotDataSource>
```

In the ViewController.m File

```
-(void)loadView
{
    [super loadView];
    // We need a hostview, you can create one in IB (and create an outlet) or just do this:
    CPTGraphHostingView* hostView = [[CPTGraphHostingView alloc] initWithFrame:CGRectMake(10, 40,
300, 400)];
    hostView.backgroundColor=[UIColor whiteColor];
    self.view.backgroundColor=[UIColor blackColor];
    [self.view addSubview: hostView];
    // Create a CPTGraph object and add to hostView
    CPTGraph* graph = [[CPTXYGraph alloc] initWithFrame:CGRectMake(10, 40, 300, 400)];
    hostView.hostedGraph = graph;
    // Get the (default) plotspace from the graph so we can set its x/y ranges
    CPTXYPlotSpace *plotSpace = (CPTXYPlotSpace *) graph.defaultPlotSpace;
    // Note that these CPTPlotRange are defined by START and LENGTH (not START and END) !!
    [plotSpace setYRange: [CPTPlotRange plotRangeWithLocation:CPTDecimalFromFloat( 0 )
length:CPTDecimalFromFloat( 20 )]];
    [plotSpace setXRange: [CPTPlotRange plotRangeWithLocation:CPTDecimalFromFloat( -4 )
length:CPTDecimalFromFloat( 8 )]];
    // Create the plot (we do not define actual x/y values yet, these will be supplied by the
datasource...)
    CPTScatterPlot* plot = [[CPTScatterPlot alloc] initWithFrame:CGRectZero];
    // Let's keep it simple and let this class act as datasource (therefore we implemtn
<CPTPlotDataSource>)
    plot.dataSource = self;
    // Finally, add the created plot to the default plot space of the CPTGraph object we created
before
    [graph addPlot:plot toPlotSpace:graph.defaultPlotSpace];
}
// This method is here because this class also functions as datasource for our graph
```

```
// 因此此类实现了 CPTPlotDataSource 协议
-(NSUInteger)numberOfRecordsForPlot:(CPTPlot *)plotnumberOfRecords
{
        return 9; // 我们的示例图包含9个“点”
}
// 该方法存在的原因是此类也作为我们图表的数据源
// 因此此类实现了CPTPlotDataSource协议
-(NSNumber *)numberForPlot:(CPTPlot *)plot field:(NSUInteger)fieldEnum
recordIndex:(NSUInteger)index
{
    // 我们需要为每个索引提供一个X或Y值（此方法将为每个调用一次）
   int x = index - 4;
    // 实际上此方法对图中的每个点调用两次，一次用于X值，一次用于Y值

    if(fieldEnum == CPTScatterPlotFieldX)
    {
    // 返回x值，根据索引，x值将在-4到4之间
    return [NSNumber numberWithInt: x];
    } else
    {
    // 返回y值，此示例中我们绘制的是y = x * x
        return [NSNumber numberWithInt: x * x];
    }
}
```

生成的输出如下所示：

```
// Therefore this class implements the CPTPlotDataSource protocol
-(NSUInteger)numberOfRecordsForPlot:(CPTPlot *)plotnumberOfRecords
{
        return 9; // Our sample graph contains 9 'points'
}
// This method is here because this class also functions as datasource for our graph
// Therefore this class implements the CPTPlotDataSource protocol
-(NSNumber *)numberForPlot:(CPTPlot *)plot field:(NSUInteger)fieldEnum
recordIndex:(NSUInteger)index
{
    // We need to provide an X or Y (this method will be called for each) value for every index
    int x = index - 4;
    // This method is actually called twice per point in the plot, one for the X and one for the Y
value
    if(fieldEnum == CPTScatterPlotFieldX)
    {
    // Return x value, which will, depending on index, be between -4 to 4
    return [NSNumber numberWithInt: x];
    } else
    {
    // Return y value, for this example we'll be plotting y = x * x
        return [NSNumber numberWithInt: x * x];
    }
}
```

The generated Output is as given below:

第161章：Swift中的FCM消息传递

第161.1节：在Swift中初始化FCM

按照以下步骤将FCM添加到您的Swift项目中

1- 如果您还没有Xcode项目，请现在创建一个。如果没有Podfile，请创建一个：

```
$ cd your-project 目录
$ pod init
```

2- 添加您想安装的pods。您可以在Podfile中这样包含一个Pod：

```
pod 'Firebase/Core'
pod 'Firebase/Messaging'
```

3- 安装pods并打开.xcworkspace文件以在Xcode中查看项目。

```
$ pod install
$ open your-project.xcworkspace
```

4- 从plist下载 GoogleService-Info.plist 文件并将其包含在您的应用中。

5- 将APNs证书上传到Firebase。 APN证书

6- 在项目的appDelegate文件中添加 "import Firebase"

7- 在你的 "application:didFinishLaunchingWithOptions" 中添加 "FIRApp.configure()"

8- 注册远程通知

```
  if #available(iOS 10.0, *) {
  let authOptions : UNAuthorizationOptions = [.Alert, .Badge, .Sound]
  UNUserNotificationCenter.currentNotificationCenter().requestAuthorizationWithOptions(
    authOptions,
completionHandler: {_,_ in })

  // 对于iOS 10显示通知（通过APNS发送）
  UNUserNotificationCenter.currentNotificationCenter().delegate = self
  // 对于iOS 10数据消息（通过FCM发送）
FIRMessaging.messaging().remoteMessageDelegate = self

} else {
  let settings: UIUserNotificationSettings =
  UIUserNotificationSettings(forTypes: [.Alert, .Badge, .Sound], categories: nil)
  application.registerUserNotificationSettings(settings)
}

application.registerForRemoteNotifications()
```

9- 获取注册令牌使用

Chapter 161: FCM Messaging in Swift

Section 161.1: Initialize FCM in Swift

follow below step to add FCM in your swift Project

1- If you don't have an Xcode project yet, create one now. Create a Podfile if you don't have one:

```
$ cd your-project directory
$ pod init
```

2- Add the pods that you want to install. You can include a Pod in your Podfile like this:

```
pod 'Firebase/Core'
pod 'Firebase/Messaging'
```

3- Install the pods and open the .xcworkspace file to see the project in Xcode.

```
$ pod install
$ open your-project.xcworkspace
```

4- Download a GoogleService-Info.plist file from plist and include it in your app.

5- Upload APNs certificate to Firebase. APN Cert

6- add "import Firebase" in your appDelegate file of project

7- add this "FIRApp.configure()" in your "application:didFinishLaunchingWithOptions"

8- register for remote notification

```
  if #available(iOS 10.0, *) {
  let authOptions : UNAuthorizationOptions = [.Alert, .Badge, .Sound]
  UNUserNotificationCenter.currentNotificationCenter().requestAuthorizationWithOptions(
    authOptions,
    completionHandler: {_,_ in })

  // For iOS 10 display notification (sent via APNS)
  UNUserNotificationCenter.currentNotificationCenter().delegate = self
  // For iOS 10 data message (sent via FCM)
  FIRMessaging.messaging().remoteMessageDelegate = self

} else {
  let settings: UIUserNotificationSettings =
  UIUserNotificationSettings(forTypes: [.Alert, .Badge, .Sound], categories: nil)
  application.registerUserNotificationSettings(settings)
}

application.registerForRemoteNotifications()
```

9- to get register token use

```
let token = FIRInstanceID.instanceID().token()!
```

10- 如果你想监控 token 变化，请在 appDelegate 文件中使用以下代码

```
func tokenRefreshNotification(notification: NSNotification) {
if let refreshedToken = FIRInstanceID.instanceID().token() {
    print("InstanceID token: \(refreshedToken)")
  }

  // 连接到 FCM，因为在获取 token 之前尝试连接可能失败。
connectToFcm()
}
```

11- 要接收来自 FCM 的消息，请在 appDelegate 中添加以下代码

```
func connectToFcm() {
FIRMessaging.messaging().connectWithCompletion { (error) in
    if (error != nil) {
      print("无法连接到 FCM。\(error)")
    } else {
      print("已连接到 FCM。")
    }
  }
}
```

12-和用于断开连接

```
func applicationDidEnterBackground(application: UIApplication) {
  FIRMessaging.messaging().disconnect()
  print("已断开与 FCM 的连接。")
}
```

在你的 appDelegate 中。

初始化完成，客户端已准备好接收来自 FCM 面板的消息或通过第三方服务器的令牌发送的消息


```
let token = FIRInstanceID.instanceID().token()!
```

10- and if u want monitor for token change use below code in appDelegate file

```
func tokenRefreshNotification(notification: NSNotification) {
if let refreshedToken = FIRInstanceID.instanceID().token() {
    print("InstanceID token: \(refreshedToken)")
  }

  // Connect to FCM since connection may have failed when attempted before having a token.
  connectToFcm()
}
```

11- to recieve message from fcm add below code in appDelegate

```
func connectToFcm() {
  FIRMessaging.messaging().connectWithCompletion { (error) in
    if (error != nil) {
      print("Unable to connect with FCM. \(error)")
    } else {
      print("Connected to FCM.")
    }
  }
}
```

12- and for disconnect use

```
func applicationDidEnterBackground(application: UIApplication) {
  FIRMessaging.messaging().disconnect()
  print("Disconnected from FCM.")
}
```

in your appDelegate.

the initialization complete and client ready to recieve message from fcm panel or send by token from third party server

第162章：在 iOS 中创建自定义框架

第162.1节：用 Swift 创建框架

按照以下步骤在 Swift-iOS 中创建自定义框架：

1. 在 Xcode 中创建一个新项目
2. 选择 iOS/Framework & Library/Cocoa Touch Framework 来创建一个新框架
3. 点击下一步并设置产品名称
4. 点击下一步并选择目录以在那里创建项目
5. 向创建的项目添加代码和资源

框架创建成功

要将创建的框架添加到另一个项目，首先应创建一个工作区
将“目标项目”和“框架项目”添加到工作区，然后：

1. 转到目标项目的常规选项卡
2. 将框架项目的产品文件夹中的“*.framework”文件拖到“嵌入式二进制文件”部分
3. 要在任何视图控制器或类中使用，只需在每个文件中导入框架

Chapter 162: Create a Custom framework in iOS

Section 162.1: Create Framework in Swift

follow these step to creating Custom Framework in Swift-IOS:

1. Create a new project. In Xcode
2. Choose iOS/Framework & Library/Cocoa Touch Framework to create a new framework
3. click next and set the productName
4. click next and choose directory to create Project there
5. add code and resources to created project

Framework created successfully

to add created framework to another project, first you should create a workspace
add "target project" and "framework project" to workspace, then :

1. go to the general tab of target project
2. drag the "*.framework" file in product folder of framework project to "Embedded Binaries" section
3. to use in any ViewController or class just import framework at each file

第163章：自定义键盘

第163.1节：自定义键盘示例

Objective-C 和 Xib

向现有的XCode项目添加目标

在"添加目标"中选择"自定义键盘"

Chapter 163: Custom Keyboard

Section 163.1: Custom KeyBoard Example

Objective-C and Xib

Add a target to an existing XCode project

In the Add Target select Custom KeyBoard

像这样添加目标：

Add the target like this:

你的项目文件目录应该看起来像这样

这里 myKeyBoard 是添加的目标名称

添加一个类型为 UIView 的新的 Cocoatouch 文件，并添加一个接口文件

Your project file directory should look something like this

Here myKeyBoard is the name of the added Target

Add new Cocoatouch file of type of type UIView and add an interface file

最后你的项目目录应该看起来像这样

将keyBoardView.xib设置为keyBoardView的子类

Finally your project directory should look like this

make the `keyBoardView.xib` a subclass of `keyBoardView`

在keyBoardView.xib文件中创建接口

将连接从keyBoardView.xib到keyBoardView.h文件

keyBoardView.h应如下所示

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

@interface keyBoardView : UIView

@property (weak, nonatomic) IBOutlet UIButton *deleteKey;
// 删除键的IBOutlet
@property (weak, nonatomic) IBOutlet UIButton *globe;
// 标题为globe的键的Outlet，用于切换键盘类型
@property (strong, nonatomic) IBOutletCollection(UIButton) NSArray *keys;
// 包含所有按键的集合，包括'0到9' '+' '-' 和 '.'

@end
```

在keyBoardViewController.h文件中导入#import "keyBoardView.h"

声明一个键盘属性@property (strong, nonatomic)keyBoardView *keyboard;

注释掉

```
@property (nonatomic, strong) UIButton *nextKeyboardButton 及其相关的所有代码
```

KeyboardViewController.m文件中的viewDidLoad()函数应如下所示

Make interface in the keyBoardView.xib file

Make connections to from the keyBoardView.xib to keyBoardView.h file

keyBoardView.h should look like

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

@interface keyBoardView : UIView

@property (weak, nonatomic) IBOutlet UIButton *deleteKey;
//IBOutlet for the delete Key
@property (weak, nonatomic) IBOutlet UIButton *globe;
//Outlet for the key with title globe which changes the keyboard type
@property (strong, nonatomic) IBOutletCollection(UIButton) NSArray *keys;
//Contains a colloection of all the keys '0 to 9' '+' '-' and '.'

@end
```

In the keyBoardViewController.h file import #import "keyBoardView.h"

Declare a property for keyboard @property (strong, nonatomic)keyBoardView *keyboard;

Comment out the

```
@property (nonatomic, strong) UIButton *nextKeyboardButton and all the code associated with it
```

The KeyboardViewController.m file's viewDidLoad() function should look like this

```
- (void)viewDidLoad {
    [super viewDidLoad];
    self.keyboard=[[[NSBundle mainBundle]loadNibNamed:@"keyBoardView" owner:nil
options:nil]objectAtIndex:0];
      self.inputView=self.keyboard;
    [self 添加手势到键盘];

    // 在此执行自定义UI设置
//    self.nextKeyboardButton = [UIButton buttonWithType:UIButtonTypeSystem];
//
//    [self.nextKeyboardButton setTitle:NSLocalizedString(@"Next Keyboard", @"'下一个键盘'按钮的标题") forState:UI
ControlStateNormal];
//    [self.nextKeyboardButton sizeToFit];
//    self.nextKeyboardButton.translatesAutoresizingMaskIntoConstraints = NO;
//
//    [self.nextKeyboardButton addTarget:self action:@selector(advanceToNextInputMode)
forControlEvents:UIControlEventTouchUpInside];
//
//    [self.view addSubview:self.nextKeyboardButton];
//
//    [self.nextKeyboardButton.leftAnchor constraintEqualToAnchor:self.view.leftAnchor].active =
YES;
//    [self.nextKeyboardButton.bottomAnchor constraintEqualToAnchor:self.view.bottomAnchor].active
= YES;
}
```

函数addGestureToKeyboard、pressDeleteKey、keyPressed定义如下

```
-(void) addGestureToKeyboard
{
    [self.keyboard.deleteKey 添加目标：self 动作：@selector(pressDeleteKey)
事件：UIControlEventTouchUpInside];
    [self.keyboard.globe 添加目标：self 动作：@selector(advanceToNextInputMode)
事件：UIControlEventTouchUpInside];

    for (UIButton *key 在 self.keyboard.keys中)
    {
        [key 添加目标：self 动作：@selector(keyPressed:)
事件：UIControlEventTouchUpInside];
    }


}
-(void) pressDeleteKey
{
    [self.textDocumentProxy 向后删除];
}

-(void)keyPressed:(UIButton *)key
{
    [self.textDocumentProxy 插入文本：[key currentTitle]];
}
```

运行主应用程序，进入设置->通用->键盘->添加新键盘->从第三方键盘部分添加键盘（显示的键盘名称为 keyBoardCustom）

键盘名称可以通过添加一个名为Bundle display name的键来更改，在值字符串中输入主项目键盘的所需名称。


```
- (void)viewDidLoad {
    [super viewDidLoad];
    self.keyboard=[[[NSBundle mainBundle]loadNibNamed:@"keyBoardView" owner:nil
options:nil]objectAtIndex:0];
      self.inputView=self.keyboard;
    [self addGestureToKeyboard];

    // Perform custom UI setup here
//    self.nextKeyboardButton = [UIButton buttonWithType:UIButtonTypeSystem];
//
//    [self.nextKeyboardButton setTitle:NSLocalizedString(@"Next Keyboard", @"Title for 'Next
Keyboard' button") forState:UIControlStateNormal];
//    [self.nextKeyboardButton sizeToFit];
//    self.nextKeyboardButton.translatesAutoresizingMaskIntoConstraints = NO;
//
//    [self.nextKeyboardButton addTarget:self action:@selector(advanceToNextInputMode)
forControlEvents:UIControlEventTouchUpInside];
//
//    [self.view addSubview:self.nextKeyboardButton];
//
//    [self.nextKeyboardButton.leftAnchor constraintEqualToAnchor:self.view.leftAnchor].active =
YES;
//    [self.nextKeyboardButton.bottomAnchor constraintEqualToAnchor:self.view.bottomAnchor].active
= YES;
}
```

The functions `addGestureToKeyboard`, `pressDeleteKey`, `keyPressed` are defined below

```
-(void) addGestureToKeyboard
{
    [self.keyboard.deleteKey addTarget:self action:@selector(pressDeleteKey)
forControlEvents:UIControlEventTouchUpInside];
    [self.keyboard.globe addTarget:self action:@selector(advanceToNextInputMode)
forControlEvents:UIControlEventTouchUpInside];

    for (UIButton *key in self.keyboard.keys)
    {
        [key addTarget:self action:@selector(keyPressed:)
forControlEvents:UIControlEventTouchUpInside];
    }


}
-(void) pressDeleteKey
{
    [self.textDocumentProxy deleteBackward];
}

-(void)keyPressed:(UIButton *)key
{
    [self.textDocumentProxy insertText:[key currentTitle]];
}
```

Run the Main Application and go to Settings->General->Keyboard->Add New Keyboard-> and add keyboard from the third party keyboard section (The displayed keyboardName would be keyBoardCustom)

The keyboard name can be changed by adding a key called `Bundle display name` and in the Value String Value enter the desired name for the keyboard of the main Project.

你也可以观看这个Youtube视频

You can also watch this Youtube Video

第164章：AirDrop

第164.1节：AirDrop

Objective-C

AirDrop可以通过UIActivityViewController使用。UIActivityViewController类是一个标准视图控制器，提供多种标准服务，例如将项目复制到剪贴板、分享到社交媒体网站、通过信息发送项目、AirDrop以及一些第三方应用程序。

在这种情况下，我们将通过UIActivityViewController发送一张图片

```
UIImage*hatImage = [UIImage imageNamed:@"logo.png"];
if (hatImage)//检查图片文件是否不为nil
{
//初始化一个UIActivityViewController
UIActivityViewController*controller = [[UIActivityViewController alloc]
initWithActivityItems:@[hatImage] applicationActivities:nil];
//从UIActivityViewController菜单中排除以下选项
NSArray*excludeActivities = @[UIActivityTypePostToWeibo,UIActivityTypePrint,
UIActivityTypeMail,UIActivityTypeMessage,UIActivityTypePostToTwitter,UIActivityTypePostToFacebook,

UIActivityTypeCopyToPasteboard,UIActivityTypeAssignToContact,

UIActivityTypeSaveToCameraRoll,UIActivityTypeAddToReadingList,

UIActivityTypePostToFlickr,UIActivityTypePostToVimeo,
UIActivityTypePostToTencentWeibo];
controller.excludedActivityTypes = excludeActivities;
[self presentViewController:controller animated:YES completion:nil];
}
```

Swift

```
if ((newImage) != nil)
        {
            let activityVC = UIActivityViewController(activityItems: [newImage],
applicationActivities: nil)
activityVC.excludedActivityTypes =[UIActivityTypeAddToReadingList]
            self.presentViewController(activityVC, animated: true, completion: nil)

        }
```


Chapter 164: AirDrop

Section 164.1: AirDrop

Objective-C

Airdrop can be used from `UIActivityViewController`. The `UIActivityViewController` class is a standard view controller that provides several standard services, such as copying items to the clipboard, sharing content to social media sites, sending items via Messages, AirDrop and some third party applications.

In this case we would be sending an image via `UIActivityViewController`

```
UIImage *hatImage = [UIImage imageNamed:@"logo.png"];
if (hatImage)//checks if the image file is not nil
{
//Initialise a UIActivityViewController
UIActivityViewController *controller = [[UIActivityViewController alloc]
initWithActivityItems:@[hatImage] applicationActivities:nil];
//Excludes following options from the UIActivityViewController menu
NSArray *excludeActivities = @[UIActivityTypePostToWeibo,UIActivityTypePrint,
UIActivityTypeMail,UIActivityTypeMessage,UIActivityTypePostToTwitter,UIActivityTypePostToFacebook,

UIActivityTypeCopyToPasteboard,UIActivityTypeAssignToContact,

UIActivityTypeSaveToCameraRoll,UIActivityTypeAddToReadingList,

UIActivityTypePostToFlickr,UIActivityTypePostToVimeo,
                                                    UIActivityTypePostToTencentWeibo];
controller.excludedActivityTypes = excludeActivities;
[self presentViewController:controller animated:YES completion:nil];
}
```

Swift

```
if ((newImage) != nil)
        {
            let activityVC = UIActivityViewController(activityItems: [newImage],
applicationActivities: nil)
            activityVC.excludedActivityTypes =[UIActivityTypeAddToReadingList]
            self.presentViewController(activityVC, animated: true, completion: nil)

        }
```

第165章：SLComposeViewController

第165.1节：用于Twitter、Facebook、新浪微博和腾讯微博的SLComposeViewController

Objective-C

首先将Social Framework添加到XCode项目中。

导入`#import "Social/Social.h"`类到所需的ViewController

带文本、图片和链接的Twitter

```
//- - 在推特上分享文本 - -
if([SLComposeViewController isAvailableForServiceType:SLServiceTypeTwitter])
{
        //推文
SLComposeViewController *twitterVC=[SLComposeViewController
composeViewControllerForServiceType:SLServiceTypeTwitter];
        //发送链接和文本
        [twitterVC addURL:[NSURL URLWithString:@"https://twitter.com/IbrahimH_ss_n"]];
        //为链接添加图片
        [twitterVC addImage:[UIImage imageNamed:@"image"]];
        //发送带链接和图片的推文
        [twitterVC setInitialText:text];
        /* 在推文中添加链接和图片时，推文的有效长度即
用户可输入的字符数会减少。
推文的默认最大长度为140个字符*/
        [self presentViewController:twitterVC animated:YES completion:nil];
}
else
{//如果未登录Twitter则显示警告
       UIAlertController *alertCont=[UIAlertController alertControllerWithTitle:@"社交分享"
message:@"您尚未登录Twitter。"preferredStyle:UIAlertControllerStyleAlert];
        [self presentViewController:alertCont animated:YES completion:nil];
        UIAlertAction *okay=[UIAlertAction actionWithTitle:@"确定" style:UIAlertActionStyleDefault
handler:nil];
        [alertCont addAction:okay];
}
}
```

带文本、图片和链接的Facebook分享

```
if([SLComposeViewController isAvailableForServiceType:SLServiceTypeFacebook])
{
SLComposeViewController *fbVC=[SLComposeViewController
composeViewControllerForServiceType:SLServiceTypeFacebook];
    [fbVC setInitialText:text];
    //发送链接和文本
    [fbVC addURL:[NSURL URLWithString:@"https://twitter.com/IbrahimH_ss_n"]];
    //添加链接的照片
    [fbVC addImage:[UIImage imageNamed:@"image"]];
    [self presentViewController:fbVC animated:YES completion:nil];
}
else
{//如果未登录Twitter则显示警告
        UIAlertController *alertCont=[UIAlertController alertControllerWithTitle:@"SocialShare"
message:@"您尚未登录推特。"preferredStyle:UIAlertControllerStyleAlert];
```


Chapter 165: SLComposeViewController

Section 165.1: SLComposeViewController for Twitter, facebook, SinaWelbo and TencentWelbo

Objective-C

First add the `Social Framework` to the XCode project.

Import the `#import "Social/Social.h"` class to the required ViewController

Twitter with text, image and link

```
//- - To Share text on twitter - -
if([SLComposeViewController isAvailableForServiceType:SLServiceTypeTwitter])
{
        //Tweet
        SLComposeViewController *twitterVC=[SLComposeViewController
composeViewControllerForServiceType:SLServiceTypeTwitter];
        //To send link together with text
        [twitterVC addURL:[NSURL URLWithString:@"https://twitter.com/IbrahimH_ss_n"]];
        //To add a photo to a link
        [twitterVC addImage:[UIImage imageNamed:@"image"]];
        //Sending link and Image with the tweet
        [twitterVC setInitialText:text];
        /* While adding link and images in a tweet the effective length of a tweet i.e.
        the number of characters which can be entered by the user decreases.
        The default maximum length of a tweet is 140 characters*/
        [self presentViewController:twitterVC animated:YES completion:nil];
}
else
{//Shows alert if twitter is not signed in
       UIAlertController *alertCont=[UIAlertController alertControllerWithTitle:@"SocialShare"
message:@"You are not signed in to twitter."preferredStyle:UIAlertControllerStyleAlert];
        [self presentViewController:alertCont animated:YES completion:nil];
        UIAlertAction *okay=[UIAlertAction actionWithTitle:@"Okay" style:UIAlertActionStyleDefault
handler:nil];
        [alertCont addAction:okay];
}
}
```

Facebook with Text, Image and Link

```
if([SLComposeViewController isAvailableForServiceType:SLServiceTypeFacebook])
{
    SLComposeViewController *fbVC=[SLComposeViewController
composeViewControllerForServiceType:SLServiceTypeFacebook];
    [fbVC setInitialText:text];
    //To send link together with text
    [fbVC addURL:[NSURL URLWithString:@"https://twitter.com/IbrahimH_ss_n"]];
    //To add a photo to a link
    [fbVC addImage:[UIImage imageNamed:@"image"]];
    [self presentViewController:fbVC animated:YES completion:nil];
}
else
{//Shows alert if twitter is not signed in
       UIAlertController *alertCont=[UIAlertController alertControllerWithTitle:@"SocialShare"
message:@"You are not signed in to twitter."preferredStyle:UIAlertControllerStyleAlert];
```

```
        [self presentViewController:alertCont animated:YES completion:nil];
        UIAlertAction *okay=[UIAlertAction actionWithTitle:@"确定" style:UIAlertActionStyleDefault
handler:nil];
        [alertCont addAction:okay];
}
```

新浪微博

```
//- - 新浪微博 - -
if([SLComposeViewController isAvailableForServiceType:SLServiceTypeSinaWeibo]){

    SLComposeViewController *SinaWeiboVC=[SLComposeViewController
composeViewControllerForServiceType:SLServiceTypeSinaWeibo];
    [SinaWeiboVC setInitialText:text];

    [self presentViewController:SinaWeiboVC animated:YES completion:nil];
}
else
{
    UIAlertController *alertCont=[UIAlertController alertControllerWithTitle:@"社交分享"
message:@"您尚未登录新浪微博。"preferredStyle:UIAlertControllerStyleAlert];
    [self presentViewController:alertCont animated:YES completion:nil];
    UIAlertAction *okay=[UIAlertAction actionWithTitle:@"确定" style:UIAlertActionStyleDefault
handler:nil];
    [alertCont addAction:okay];
}
```

腾讯微博

```
//- -腾讯微博文本分享
if([SLComposeViewController isAvailableForServiceType:SLServiceTypeTencentWeibo])
{
SLComposeViewController *tencentWeiboVC=[SLComposeViewController
composeViewControllerForServiceType:SLServiceTypeTencentWeibo];
    [tencentWeibo setInitialText:text];
    [self presentViewController:tencentWeibo animated:YES completion:nil];
}
else
{
    UIAlertController *alertCont=[UIAlertController alertControllerWithTitle:@"社交分享"
message:@"您尚未登录新浪微博。"preferredStyle:UIAlertControllerStyleAlert];
    [self presentViewController:alertCont animated:YES completion:nil];
    UIAlertAction *okay=[UIAlertAction actionWithTitle:@"确定" style:UIAlertActionStyleDefault
handler:nil];
    [alertCont addAction:okay];
}
```



```
        [self presentViewController:alertCont animated:YES completion:nil];
        UIAlertAction *okay=[UIAlertAction actionWithTitle:@"Okay" style:UIAlertActionStyleDefault
handler:nil];
        [alertCont addAction:okay];
}
```

SinaWeibo

```
//- - SinaWeibo - -
if([SLComposeViewController isAvailableForServiceType:SLServiceTypeSinaWeibo]){

    SLComposeViewController *SinaWeiboVC=[SLComposeViewController
composeViewControllerForServiceType:SLServiceTypeSinaWeibo];
    [SinaWeiboVC setInitialText:text];

    [self presentViewController:SinaWeiboVC animated:YES completion:nil];
}
else
{
    UIAlertController *alertCont=[UIAlertController alertControllerWithTitle:@"SocialShare"
message:@"You are not signed in to SinaWeibo."preferredStyle:UIAlertControllerStyleAlert];
    [self presentViewController:alertCont animated:YES completion:nil];
    UIAlertAction *okay=[UIAlertAction actionWithTitle:@"Okay" style:UIAlertActionStyleDefault
handler:nil];
    [alertCont addAction:okay];
}
```

TencentWeibo

```
//- -TencentWeibo text share
if([SLComposeViewController isAvailableForServiceType:SLServiceTypeTencentWeibo])
{
    SLComposeViewController *tencentWeiboVC=[SLComposeViewController
composeViewControllerForServiceType:SLServiceTypeTencentWeibo];
    [tencentWeibo setInitialText:text];
    [self presentViewController:tencentWeibo animated:YES completion:nil];
}
else
{
    UIAlertController *alertCont=[UIAlertController alertControllerWithTitle:@"SocialShare"
message:@"You are not signed in to SinaWeibo."preferredStyle:UIAlertControllerStyleAlert];
    [self presentViewController:alertCont animated:YES completion:nil];
    UIAlertAction *okay=[UIAlertAction actionWithTitle:@"Okay" style:UIAlertActionStyleDefault
handler:nil];
    [alertCont addAction:okay];
}
```

第166章：iOS中的AirPrint教程

第166.1节：AirPrint打印横幅文本

Objective-C

向ViewController.h文件添加代理和文本格式化器

```
@interface ViewController : UIViewController <UIPrintInteractionControllerDelegate> {
    UISimpleTextPrintFormatter *_textFormatter;
}
```

在ViewController.m文件中定义以下常量

```
#define DefaultFontSize 48
#define PaddingFactor 0.1f
```

打印文本的函数如下：

```
-(IBAction)print:(id)sender;
{
    /* 获取 UIPrintInteractionController，这是一个共享对象 */
    UIPrintInteractionController *controller = [UIPrintInteractionController
sharedPrintController];
    if(!controller){
NSLog(@"无法获取共享的 UIPrintInteractionController！");
        return;
    }

    /* 将此对象设置为代理，以便使用
printInteractionController:cutLengthForPaper: 代理方法 */
    controller.delegate = self;

    UIPrintInfo *printInfo = [UIPrintInfo printInfo];
    printInfo.outputType = UIPrintInfoOutputGeneral;

    /* 对于横幅使用横向打印方向，使文本沿纸张长边打印。
*/
printInfo.orientation = UIPrintInfoOrientationLandscape;

    printInfo.jobName = self.textField.text;
controller.printInfo = printInfo;

    /* 使用用户在文本字段中提供的文本创建 UISimpleTextPrintFormatter
*/
_textFormatter = [[UISimpleTextPrintFormatter alloc] initWithText:self.textField.text];

    /* 根据用户选择设置文本格式化器的颜色和字体属性 */
    _textFormatter.color = [self chosenColor];
    _textFormatter.font = [self chosenFontWithSize:DefaultFontSize];

    /* 在控制器上设置此 UISimpleTextPrintFormatter */
    controller.printFormatter = _textFormatter;

    /* 设置完成处理程序块。如果打印任务在排队前出现错误，将在此处处理。 */

void (^completionHandler)(UIPrintInteractionController *, BOOL, NSError *) =
^(UIPrintInteractionController *printController, BOOL completed, NSError *error) {
```


Chapter 166: AirPrint tutorial in iOS

Section 166.1: AirPrint printing Banner Text

Objective-C

Add the delegate and a text-formatter to the `ViewController.h` file

```
@interface ViewController : UIViewController <UIPrintInteractionControllerDelegate> {
    UISimpleTextPrintFormatter *_textFormatter;
}
```

In the `ViewController.m` file define the following constants

```
#define DefaultFontSize 48
#define PaddingFactor 0.1f
```

The function which prints the text is as follows:

```
-(IBAction)print:(id)sender;
{
    /* Get the UIPrintInteractionController, which is a shared object */
    UIPrintInteractionController *controller = [UIPrintInteractionController
sharedPrintController];
    if(!controller){
        NSLog(@"Couldn't get shared UIPrintInteractionController!");
        return;
    }

    /* Set this object as delegate so you can  use the
printInteractionController:cutLengthForPaper: delegate */
    controller.delegate = self;

    UIPrintInfo *printInfo = [UIPrintInfo printInfo];
    printInfo.outputType = UIPrintInfoOutputGeneral;

    /* Use landscape orientation for a banner so the text  print along the long side of the paper.
*/
    printInfo.orientation = UIPrintInfoOrientationLandscape;

    printInfo.jobName = self.textField.text;
    controller.printInfo = printInfo;

    /* Create the UISimpleTextPrintFormatter with the text supplied by the user in the text field
*/
    _textFormatter = [[UISimpleTextPrintFormatter alloc] initWithText:self.textField.text];

    /* Set the text formatter's color and font properties based on what the user chose */
    _textFormatter.color = [self chosenColor];
    _textFormatter.font = [self chosenFontWithSize:DefaultFontSize];

    /* Set this UISimpleTextPrintFormatter on the controller */
    controller.printFormatter = _textFormatter;

    /* Set up a completion handler block.  If the print job has an error before spooling, this is
where it's handled. */
    void (^completionHandler)(UIPrintInteractionController *, BOOL, NSError *) =
^(UIPrintInteractionController *printController, BOOL completed, NSError *error) {
```

```
        if(completed && error)
NSLog( @"打印失败，错误域为 %@，错误代码为 %lu。 本地化描述：%@，失败原因：%@", error.domain, (long)error.c
ode,
error.localizedDescription, error.localizedFailureReason );
    };

    if (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad)
        [controller presentFromRect:self.printButton.frame inView:self.view animated:YES
completionHandler:completionHandler];
    else
        [controller presentAnimated:YES completionHandler:completionHandler];  // iPhone
}
```

设置打印页面的代理函数：

```
- (CGFloat)printInteractionController:(UIPrintInteractionController *)printInteractionController
cutLengthForPaper:(UIPrintPaper *)paper {

    /* 创建一个任意大小的字体，以便计算文本高度的每屏幕点大致对应的字体点数。 */

    UIFont *font = _textFormatter.font;
    CGSize size = [self.textField.text sizeWithAttributes:@{NSFontAttributeName: font}];

float approximateFontPointPerScreenPoint = font.pointSize / size.height;

    /* 使用一个能填满纸张宽度的大小创建新字体 */
   font = [self chosenFontWithSize: paper.printableRect.size.width *
approximateFontPointPerScreenPoint];

    /* 使用最终字体大小计算文本的高度和宽度 */
    CGSize finalTextSize = [self.textField.text sizeWithAttributes:@{NSFontAttributeName: font}];

    /* 将UISimpleTextFormatter的字体设置为计算出的字体大小 */
_textFormatter.font = font;

    /* 计算卷筒的边距。卷筒打印机在切割前后可能有不可打印区域。        我们必须将其加到切割长度中
，以确保可打印区域有足够的空间容纳我们的文本。 */

    CGFloat lengthOfMargins = paper.paperSize.height - paper.printableRect.size.height;

    /* 切割长度是文本宽度，加上边距，再加上一些填充 */
    return finalTextSize.width + lengthOfMargins + paper.printableRect.size.width * PaddingFactor;
}
```



```
        if(completed && error)
            NSLog( @"Printing failed due to error in domain %@ with error code %lu. Localized
description: %@, and failure reason: %@", error.domain, (long)error.code,
error.localizedDescription, error.localizedFailureReason );
    };

    if (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad)
        [controller presentFromRect:self.printButton.frame inView:self.view animated:YES
completionHandler:completionHandler];
    else
        [controller presentAnimated:YES completionHandler:completionHandler];  // iPhone
}
```

The delegate function which sets up the print page:

```
- (CGFloat)printInteractionController:(UIPrintInteractionController *)printInteractionController
cutLengthForPaper:(UIPrintPaper *)paper {

    /* Create a font with arbitrary size so that you can calculate the approximate
       font points per screen point for the height of the text. */
    UIFont *font = _textFormatter.font;
    CGSize size = [self.textField.text sizeWithAttributes:@{NSFontAttributeName: font}];

    float approximateFontPointPerScreenPoint = font.pointSize / size.height;

    /* Create a new font using a size  that will fill the width of the paper */
    font = [self chosenFontWithSize: paper.printableRect.size.width *
approximateFontPointPerScreenPoint];

    /* Calculate the height and width of the text with the final font size */
    CGSize finalTextSize = [self.textField.text sizeWithAttributes:@{NSFontAttributeName: font}];

    /* Set the UISimpleTextFormatter font to the font with the size calculated */
    _textFormatter.font = font;

    /* Calculate the margins of the roll. Roll printers may have unprintable areas
       before and after the cut.  We must add this to our cut length to ensure the
       printable area has enough room for our text. */
    CGFloat lengthOfMargins = paper.paperSize.height - paper.printableRect.size.height;

    /* The cut length is the width of the text, plus margins, plus some padding */
    return finalTextSize.width + lengthOfMargins + paper.printableRect.size.width * PaddingFactor;
}
```

第167章：Carthage iOS 设置

第167.1节：Mac 上的 Carthage 安装

Carthage 设置

从给定链接下载最新版本的 Carthage 下载链接

在下载部分下载**Carthage.pkg**文件。

下载完成后，双击下载的 pkg 文件进行安装。

要检查是否下载成功，请在终端运行以下命令 carthage version 应显示安装的版本，如0.18-19-g743fa0f

Chapter 167: Carthage iOS Setup

Section 167.1: Carthage Installation Mac

Carthage Setup

Download the latest version of Carthage from the given link Download Link

Down in the Download section download the **Carthage.pkg** file.

Once the download is complete install it by double clinking on the download pkg file.

To check for successful download run the following command in your Terminal carthage version This should give the version installed like `0.18-19-g743fa0f`

第168章：Healthkit

第168.1节：HealthKit

Objective-C

首先进入Target->Capabilities并启用HealthKit。这将设置info.plist条目。

新建一个CocoaClass，类型为NSObject。我给的文件名是GSHealthKitManager，头文件如下所示

GSHealthKitManager.h

```
#import <Foundation/Foundation.h>
#import <HealthKit/HealthKit.h>
@interface GSHealthKitManager : NSObject

+ (GSHealthKitManager *)sharedManager;

- (void)requestAuthorization;

- (NSDate *)readBirthDate;
- (void)writeWeightSample:(double)weight;
- (NSString *)readGender;

@end
```

GSHealthKitManager.m

```
#import "GSHealthKitManager.h"
#import <HealthKit/HealthKit.h>


@interface GSHealthKitManager ()

@property (nonatomic, retain) HKHealthStore *healthStore;

@end


@implementation GSHealthKitManager

+ (GSHealthKitManager *)sharedManager {
    static dispatch_once_t pred = 0;
    static GSHealthKitManager *instance = nil;
   dispatch_once(&pred, ^{
instance = [[GSHealthKitManager alloc] init];
        instance.healthStore = [[HKHealthStore alloc] init];
    });
    return instance;
}

- (void)requestAuthorization {

    if ([HKHealthStore isHealthDataAvailable] == NO) {
        // 如果我们的设备不支持HealthKit -> 返回。
        return;
    }
```


Chapter 168: Healthkit

Section 168.1: HealthKit

Objective-C

First go the `Target->Capabilities` and enable `HealthKit`. This would setup the info.plist entry.

Make a new `CocoaClass` of type `NSObject` The filename I gave is `GSHealthKitManager` and the header file is as shown below

GSHealthKitManager.h

```
#import <Foundation/Foundation.h>
#import <HealthKit/HealthKit.h>
@interface GSHealthKitManager : NSObject

+ (GSHealthKitManager *)sharedManager;

- (void)requestAuthorization;

- (NSDate *)readBirthDate;
- (void)writeWeightSample:(double)weight;
- (NSString *)readGender;

@end
```

GSHealthKitManager.m

```
#import "GSHealthKitManager.h"
#import <HealthKit/HealthKit.h>


@interface GSHealthKitManager ()

@property (nonatomic, retain) HKHealthStore *healthStore;

@end


@implementation GSHealthKitManager

+ (GSHealthKitManager *)sharedManager {
    static dispatch_once_t pred = 0;
    static GSHealthKitManager *instance = nil;
    dispatch_once(&pred, ^{
        instance = [[GSHealthKitManager alloc] init];
        instance.healthStore = [[HKHealthStore alloc] init];
    });
    return instance;
}

- (void)requestAuthorization {

    if ([HKHealthStore isHealthDataAvailable] == NO) {
        // If our device doesn't support HealthKit -> return.
        return;
    }
```

```
    NSArray *readTypes = @[[HKObjectType characteristicTypeForIdentifier:HKCharacteristicTypeIdentifierDateOfBirth],[HKObjectType characteristicTypeForIdentifier:HKCharacteristicTypeIdentifierBiologicalSex]];


    [self.healthStore requestAuthorizationToShareTypes:nil readTypes:[NSSet setWithArray:readTypes] completion:nil];
}

- (NSDate *)readBirthDate {
    NSError *error;
    NSDate *dateOfBirth = [self.healthStore dateOfBirthWithError:&error];   // HKHealthStore 的便捷方法，直接获取出生日期。

    if (!dateOfBirth) {
NSLog(@"获取用户年龄信息时发生错误，或者尚未存储任何信息。在您的应用中，请尝试优雅地处理此情况。");

    }

    return dateOfBirth;
}

- (NSString *)readGender
{
    NSError *error;
HKBiologicalSexObject *gen=[self.healthStore biologicalSexWithError:&error];
    if (gen.biologicalSex==HKBiologicalSexMale)
    {
        return(@"男");
    }
    else if (gen.biologicalSex==HKBiologicalSexFemale)
    {
        return (@"女");
    }
    else if (gen.biologicalSex==HKBiologicalSexOther)
    {
        return (@"其他");
    }
    else{
        return (@"未设置");
    }
}



@end
```

从视图控制器调用

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

    [[GSHealthKitManager sharedManager] requestAuthorization];
    NSDate *birthDate = [[GSHealthKitManager sharedManager] readBirthDate];
        NSLog(@"出生日期 %@", birthDate);
NSLog(@"gender 2131321 %@", [[GSHealthKitManager sharedManager] readGender]);


}
```

日志输出


```
    NSArray *readTypes = @[[HKObjectType characteristicTypeForIdentifier:HKCharacteristicTypeIdentifierDateOfBirth],[HKObjectType characteristicTypeForIdentifier:HKCharacteristicTypeIdentifierBiologicalSex]];


    [self.healthStore requestAuthorizationToShareTypes:nil readTypes:[NSSet setWithArray:readTypes] completion:nil];
}

- (NSDate *)readBirthDate {
    NSError *error;
    NSDate *dateOfBirth = [self.healthStore dateOfBirthWithError:&error];   // Convenience method of HKHealthStore to get date of birth directly.

    if (!dateOfBirth) {
        NSLog(@"Either an error occurred fetching the user's age information or none has been stored yet. In your app, try to handle this gracefully.");
    }

    return dateOfBirth;
}

- (NSString *)readGender
{
    NSError *error;
    HKBiologicalSexObject *gen=[self.healthStore biologicalSexWithError:&error];
    if (gen.biologicalSex==HKBiologicalSexMale)
    {
        return(@"Male");
    }
    else if (gen.biologicalSex==HKBiologicalSexFemale)
    {
        return (@"Female");
    }
    else if (gen.biologicalSex==HKBiologicalSexOther)
    {
        return (@"Other");
    }
    else{
        return (@"Not Set");
    }
}



@end
```

Calling from ViewController

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

    [[GSHealthKitManager sharedManager] requestAuthorization];
    NSDate *birthDate = [[GSHealthKitManager sharedManager] readBirthDate];
        NSLog(@"birthdate %@", birthDate);
        NSLog(@"gender 2131321 %@", [[GSHealthKitManager sharedManager] readGender]);


}
```

Log Output

```
2016-10-13 14:41:39.568 random[778:26371] 出生日期 1992-11-29 18:30:00 +0000
2016-10-13 14:41:39.570 random[778:26371] 性别 2131321 男性
```



```
2016-10-13 14:41:39.568 random[778:26371] birthdate 1992-11-29 18:30:00 +0000
2016-10-13 14:41:39.570 random[778:26371] gender 2131321 Male
```

第169章：iOS中的核心聚焦（Core SpotLight）

第169.1节：核心聚焦（Core-Spotlight）

Objective-C

1. 创建一个新的iOS项目，并将CoreSpotlight和MobileCoreServices框架添加到项目中。

2. 创建实际的CSSearchableItem，并关联uniqueIdentifier、domainIdentifier和attributeSet。最后，使用[[CSSearchableIndex defaultSearchableIndex]...]对CSSearchableItem进行索引，如下所示。

Chapter 169: Core SpotLight in iOS

Section 169.1: Core-Spotlight

Objective-C

1. Create a new iOS project and add *CoreSpotlight* and *MobileCoreServices* framework to your project.

2. Create the actual CSSearchableItem and associating the uniqueIdentifier, domainIdentifier and the attributeSet. Finally index the CSSearchableItem using [[CSSearchableIndex defaultSearchableIndex]...] as show below.

```
//
//  Created by Mayqiyue on 7/10/15.
//  Copyright © 2015 mayqiyue. All rights reserved.
//

#import "ViewController.h"
#import <CoreSpotlight/CoreSpotlight.h>
#import <MobileCoreServices/MobileCoreServices.h>

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    [self setupCoreSpotlightSearch];
}

- (void)setupCoreSpotlightSearch {
    CSSearchableItemAttributeSet *attributeSet = [[CSSearchableItemAttributeSet alloc] initWithItemContentType:
        (NSString *)kUTTypeImage];
    attributeSet.title = @"My First Spotlight Search";
    attributeSet.contentDescription = @"";
    attributeSet.keywords = [NSArray arrayWithObjects:@"Hello", @"Welcome", @"Spotlight", nil];
    UIImage *image = [UIImage imageNamed:@"l"];
    NSData *imageData = [NSData dataWithData:UIImagePNGRepresentation(image)];
    attributeSet.thumbnailData = imageData;

    CSSearchableItem *item = [[CSSearchableItem alloc]initWithUniqueIdentifier:@"com.deeplink"
        domainIdentifier:@"spotlight.sample" attributeSet:attributeSet];

    [[CSSearchableIndex defaultSearchableIndex] indexSearchableItems:@[item] completionHandler: ^(NSError * __nullable
        error) {
        if (!error)
            NSLog(@"Search item indexed");
    }];
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
}
```

3. 好的！测试索引！

```
//
//  Created by Mayqiyue on 7/10/15.
//  Copyright © 2015 mayqiyue. All rights reserved.
//

#import "ViewController.h"
#import <CoreSpotlight/CoreSpotlight.h>
#import <MobileCoreServices/MobileCoreServices.h>

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    [self setupCoreSpotlightSearch];
}

- (void)setupCoreSpotlightSearch {
    CSSearchableItemAttributeSet *attributeSet = [[CSSearchableItemAttributeSet alloc] initWithItemContentType:
        (NSString *)kUTTypeImage];
    attributeSet.title = @"My First Spotlight Search";
    attributeSet.contentDescription = @"";
    attributeSet.keywords = [NSArray arrayWithObjects:@"Hello", @"Welcome", @"Spotlight", nil];
    UIImage *image = [UIImage imageNamed:@"l"];
    NSData *imageData = [NSData dataWithData:UIImagePNGRepresentation(image)];
    attributeSet.thumbnailData = imageData;

    CSSearchableItem *item = [[CSSearchableItem alloc]initWithUniqueIdentifier:@"com.deeplink"
        domainIdentifier:@"spotlight.sample" attributeSet:attributeSet];

    [[CSSearchableIndex defaultSearchableIndex] indexSearchableItems:@[item] completionHandler: ^(NSError * __nullable
        error) {
        if (!error)
            NSLog(@"Search item indexed");
    }];
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
}
```


3. OK!Test the index!

第170章：核心运动

第170.1节：访问气压计以获取相对高度

Swift

导入Core Motion库：

```
import CoreMotion
```

接下来，我们需要创建一个CMAltimeter对象，但一个常见的错误是在viewDidLoad()中创建它。如果那样做，气压计在我们需要调用其方法时将无法访问。不过，先在viewDidLoad()之前创建你的CMAltimeter对象：

```
let altimeter = CMAltimeter()
```

现在：

1. 我们需要使用以下方法检查relativeAltitude是否可用：`CMAltimeter.isRelativeAltitudeAvailable.`
2. 如果返回true，则可以开始使用`startRelativeAltitudeUpdatesToQueue`
3. 如果没有错误，您应该能够从relativeAltitude和压力属性中获取数据。

下面是一个按钮动作的定义，用于开始使用我们的气压计进行监测。

```
@IBAction func start(sender: AnyObject){
if CMAltimeter.isRelativeAltitudeAvailable() {
    // 2
altimeter.startRelativeAltitudeUpdatesToQueue(NSOperationQueue.mainQueue(), withHandler: {
data, error in
        // 3
        if (error == nil) {
            println("相对高度: \(data.relativeAltitude)")
            println("压力: \(data.pressure)")
        }
    })
}
```


Chapter 170: Core Motion

Section 170.1: Accessing Barometer to get relative altitude

Swift

Import the Core Motion library:

```
import CoreMotion
```

Next, we need to create a `CMAltimeter` object, but a common pitfall is to create it in the `viewDidLoad()`. If done that way, the altimeter won't be accessible when we need to call a method on it. Nevertheless, go ahead and create your `CMAltimeter` object just before the `viewDidLoad()`:

```
let altimeter = CMAltimeter()
```

Now:

1. We need to check if `relativeAltitude` is even available with the following method: `CMAltimeter.isRelativeAltitudeAvailable`.
2. If that returns `true`, you can then begin monitoring altitude change with `startRelativeAltitudeUpdatesToQueue`
3. If there are no errors, you should be able to retrieve data from the `relativeAltitude` and pressure properties.

Given below is the definition of a button action to begin monitoring with our barometer.

```
@IBAction func start(sender: AnyObject){
if CMAltimeter.isRelativeAltitudeAvailable() {
    // 2
    altimeter.startRelativeAltitudeUpdatesToQueue(NSOperationQueue.mainQueue(), withHandler: {
data, error in
        // 3
        if (error == nil) {
            println("Relative Altitude: \(data.relativeAltitude)")
            println("Pressure: \(data.pressure)")
        }
    })
}
```

第171章：二维码扫描器

二维码（Quick Response，快速响应）码是广泛应用于机器可读光学标签的二维条码。iOS从7版本开始提供了使用AVFoundation框架读取二维码的方法。该框架提供了一套API，用于设置/打开摄像头并从摄像头视频流中读取二维码。

第171.1节：使用AVFoundation框架扫描二维码

在iOS 7之前，如果想扫描二维码，可能需要依赖第三方框架或库，如zBar或zXing。但苹果从iOS 7开始引入了`AVCaptureMetaDataOutput`用于读取条形码。

要使用`AVFoundation`读取二维码，需要设置/创建`AVCaptureSession`并使用`captureOutput:didOutputMetadataObjects:fromConnection:`代理方法。

步骤 1

导入AVFoundation框架并遵循AVCaptureMetadataOutputObjectsDelegate协议

```
import AVFoundation
class ViewController: UIViewController, AVCaptureMetadataOutputObjectsDelegate
```

步骤2

二维码读取完全基于视频捕捉。因此，为了捕捉连续视频，需要创建一个AVCaptureSession并设置设备输入和输出。在视图控制器的viewDidLoad方法中添加以下代码

```
// 创建一个 AVCaptureDevice 实例，并将视频作为媒体类型参数提供。
let captureDevice = AVCaptureDevice.defaultDevice(withMediaType: AVMediaTypeVideo)

do {
    // 使用设备对象创建 AVCaptureDeviceInput 类的实例并初始化
捕获会话
    let input = try AVCaptureDeviceInput(device: captureDevice)
   captureSession = AVCaptureSession()
captureSession?.addInput(input)

    // 创建 AVCaptureMetadataOutput 对象的实例并将其设置为捕获会话的输出设备。

    let captureMetadataOutput = AVCaptureMetadataOutput()
   captureSession?.addOutput(captureMetadataOutput)
    // 使用默认的调度队列设置代理
   captureMetadataOutput.setMetadataObjectsDelegate(self, queue: DispatchQueue.main)
    // 设置元数据对象类型为二维码，这里也可以添加多种类型
   captureMetadataOutput.metadataObjectTypes = [AVMetadataObjectTypeQRCode]


    // 初始化视频预览图层并将其作为子图层添加到视图控制器视图的图层中。

videoPreviewLayer = AVCaptureVideoPreviewLayer(session: captureSession)
    videoPreviewLayer?.videoGravity = AVLayerVideoGravityResizeAspectFill
    videoPreviewLayer?.frame = view.layer.bounds
view.layer.addSublayer(videoPreviewLayer!)

    // 启动捕获会话。
captureSession?.startRunning()
} catch {
                                // 如果发生任何错误，通知用户。示例中仅打印错误信息
```


Chapter 171: QR Code Scanner

QR (Quick Response) codes are two-dimensional barcodes which are widely used on machine-readable optical labels. iOS do provide a way to read the QR codes by using `AVFoundation` framework from iOS 7 onwards. This framework provides set of API's to setup/open the camera and read QR codes from the camera feed.

Section 171.1: Scanning QR code with AVFoudation framework

Prior to iOS 7 when you want to scan a QR code, we might need to rely on third party frameworks or libraries like zBar or zXing. But Apple introduced `AVCaptureMetaDataOutput` from iOS 7 for reading barcodes.

To read QR code using `AVFoundation` we need to setup/create `AVCaptureSession` and use `captureOutput:didOutputMetadataObjects:fromConnection:` delegate method.

Step 1

Import `AVFoundation` framework and confirm to `AVCaptureMetadataOutputObjectsDelegate` protocol

```
import AVFoundation
class ViewController: UIViewController, AVCaptureMetadataOutputObjectsDelegate
```

Step 2

QR code reading is totally based on video capture. So to capture continuous video create an `AVCaptureSession` and set up device input and output. Add the below code in view controller `viewDidLoad` method

```
// Create an instance of the AVCaptureDevice and provide the video as the media type parameter.
let captureDevice = AVCaptureDevice.defaultDevice(withMediaType: AVMediaTypeVideo)

do {
    // Create an instance of the AVCaptureDeviceInput class using the device object and intialise
capture session
    let input = try AVCaptureDeviceInput(device: captureDevice)
    captureSession = AVCaptureSession()
    captureSession?.addInput(input)

    // Create a instance of AVCaptureMetadataOutput object and set it as the output device the
capture session.
    let captureMetadataOutput = AVCaptureMetadataOutput()
    captureSession?.addOutput(captureMetadataOutput)
    // Set delegate with a default dispatch queue
    captureMetadataOutput.setMetadataObjectsDelegate(self, queue: DispatchQueue.main)
    //set meta data object type as QR code, here we can add more then one type as well
    captureMetadataOutput.metadataObjectTypes = [AVMetadataObjectTypeQRCode]


    // Initialize the video preview layer and add it as a sublayer to the viewcontroller view's
layer.
    videoPreviewLayer = AVCaptureVideoPreviewLayer(session: captureSession)
    videoPreviewLayer?.videoGravity = AVLayerVideoGravityResizeAspectFill
    videoPreviewLayer?.frame = view.layer.bounds
    view.layer.addSublayer(videoPreviewLayer!)

    // Start capture session.
    captureSession?.startRunning()
} catch {
    // If any error occurs, let the user know. For the example purpose just print out the error
```

```
    print(error)
    return
}
```

步骤 3

实现 AVCaptureMetadataOutputObjectsDelegate 委托方法以读取二维码

```
func captureOutput(_ captureOutput: AVCaptureOutput!, didOutputMetadataObjects metadataObjects:
[Any]!, from connection: AVCaptureConnection!) {

    // 检查 metadataObjects 数组是否至少包含一个对象。如果没有，则视频捕获中没有二维码

    if metadataObjects == nil || metadataObjects.count == 0 {
        // 未检测到二维码。
        返回
    }

    // 获取元数据对象并将其转换为 `AVMetadataMachineReadableCodeObject`
    let metadataObj = metadataObjects[0] as! AVMetadataMachineReadableCodeObject

    if metadataObj.type == AVMetadataObjectTypeQRCode {
        // 如果找到的元数据等于二维码元数据，则从元数据中获取字符串值

                let barCodeObject = videoPreviewLayer?.transformedMetadataObject(for: metadataObj)

        if metadataObj.stringValue != nil {
            // metadataObj.stringValue 是我们的二维码
        }
    }
}
```

这里元数据对象还可以为你提供摄像头画面中读取到的二维码的边界。要获取边界，只需将元数据对象传递给videoPreviewLayer的transformedMetadataObject方法，如下所示。

```
let barCodeObject = videoPreviewLayer?.transformedMetadataObject(for: metadataObj)
        qrCodeFrameView?.frame = barCodeObject!.bounds
```

第171.2节：UIViewController 扫描二维码并显示视频输入

```
import AVFoundation
class QRScannerViewController: UIViewController,
    AVCaptureMetadataOutputObjectsDelegate {

    func viewDidLoad() {
        self.initCaptureSession()
    }

    private func initCaptureSession() {
        let captureDevice = AVCaptureDevice
            .defaultDevice(withMediaType: AVMediaTypeVideo)
        do {
            let input = try AVCaptureDeviceInput(device: captureDevice)
            let captureMetadataOutput = AVCaptureMetadataOutput()
            self.captureSession?.addOutput(captureMetadataOutput)
            captureMetadataOutput.setMetadataObjectsDelegate(self,
                 queue: DispatchQueue.main)
            captureMetadataOutput
.metadataObjectTypes = [AVMetadataObjectTypeQRCode]
```



```
    print(error)
    return
}
```

Step 3

Implement `AVCaptureMetadataOutputObjectsDelegate` delegate method to read the QR code

```
func captureOutput(_ captureOutput: AVCaptureOutput!, didOutputMetadataObjects metadataObjects:
[Any]!, from connection: AVCaptureConnection!) {

    // Check if the metadataObjects array contains at least one object. If not no QR code is in our
video capture
    if metadataObjects == nil || metadataObjects.count == 0 {
        // NO QR code is being detected.
        return
    }

    // Get the metadata object and cast it to `AVMetadataMachineReadableCodeObject`
    let metadataObj = metadataObjects[0] as! AVMetadataMachineReadableCodeObject

    if metadataObj.type == AVMetadataObjectTypeQRCode {
        // If the found metadata is equal to the QR code metadata then get the string value from
meta data
        let barCodeObject = videoPreviewLayer?.transformedMetadataObject(for: metadataObj)

        if metadataObj.stringValue != nil {
           // metadataObj.stringValue is our QR code
        }
    }
}
```

here metadata object can give you the bounds of the QR code read on the camera feed as well. To get the bounds simply pass the metadata object to `videoPreviewLayer' s transformedMetadataObject` method like below.

```
let barCodeObject = videoPreviewLayer?.transformedMetadataObject(for: metadataObj)
        qrCodeFrameView?.frame = barCodeObject!.bounds
```

Section 171.2: UIViewController scanning for QR and displaying video input

```
import AVFoundation
class QRScannerViewController: UIViewController,
     AVCaptureMetadataOutputObjectsDelegate {

    func viewDidLoad() {
        self.initCaptureSession()
    }

    private func initCaptureSession() {
        let captureDevice = AVCaptureDevice
            .defaultDevice(withMediaType: AVMediaTypeVideo)
        do {
            let input = try AVCaptureDeviceInput(device: captureDevice)
            let captureMetadataOutput = AVCaptureMetadataOutput()
            self.captureSession?.addOutput(captureMetadataOutput)
            captureMetadataOutput.setMetadataObjectsDelegate(self,
                 queue: DispatchQueue.main)
            captureMetadataOutput
                .metadataObjectTypes = [AVMetadataObjectTypeQRCode]
```

```
            self.videoPreviewLayer =
AVCaptureVideoPreviewLayer(session: self.captureSession)
            self.videoPreviewLayer?
.videoGravity = AVLayerVideoGravityResizeAspectFill
            self.videoPreviewLayer?.frame =
                self.view.layer.bounds

            self._viewController?.view.layer
.addSublayer(videoPreviewLayer!)
            self.captureSession?.startRunning()
        } catch {
            //TODO: 处理输入打开错误
        }
    }
    private func 关闭捕获会话() {
        if let running = self.captureSession?.isRunning, running {
            self.captureSession?.stopRunning()
        }
        self.captureSession = nil
        self.videoPreviewLayer?.removeFromSuperLayer()
        self.videoPreviewLayer = nil
    }

    func captureOutput(_ captureOutput: AVCaptureOutput,
        didOutputMetadataObjects metadataObjects: [Any]!,
       from connection: AVCaptureConnection) {
guard metadataObjects != nil && metadataObjects.count != 0 else {
            //未捕获到任何内容
            返回
        }

        if let metadataObj =
metadataObjects[0] as? AVMetadataMachineReadableCodeObject {
            guard metadataObj.type == AVMetadataObjectTypeQRCode else {
                return
            }

            let barCodeObject = videoPreviewLayer?
.transformedMetadataObject(for:
metadataObj as AVMetadataMachineReadableCodeObject)
                 as! AVMetadataMachineReadableCodeObject

            if let qrValue = metadataObj.stringValue {
                self.handleQRRead(value: qrValue)
            }
        }
    }

    private handleQRRead(value: String) {
        //TODO: 处理读取的二维码
    }
    private captureSession: AVCaptureSession?
    private videoPreviewLayer: AVCaptureVideo
}
```

`handleQRRead` - 在扫描成功时调用 `initCaptureSession` - 初始化二维码扫描和摄像头输入
`dismissCaptureSession` - 隐藏摄像头输入并停止扫描


```
        self.videoPreviewLayer =
            AVCaptureVideoPreviewLayer(session: self.captureSession)
        self.videoPreviewLayer?
            .videoGravity = AVLayerVideoGravityResizeAspectFill
        self.videoPreviewLayer?.frame =
            self.view.layer.bounds

        self._viewController?.view.layer
            .addSublayer(videoPreviewLayer!)
        self.captureSession?.startRunning()
    } catch {
        //TODO: handle input open error
    }
}
private func dismissCaptureSession() {
    if let running = self.captureSession?.isRunning, running {
        self.captureSession?.stopRunning()
    }
    self.captureSession = nil
    self.videoPreviewLayer?.removeFromSuperLayer()
    self.videoPreviewLayer = nil
}

func captureOutput(_ captureOutput: AVCaptureOutput,
    didOutputMetadataObjects metadataObjects: [Any]!,
    from connection: AVCaptureConnection) {
    guard metadataObjects != nil && metadataObjects.count != 0 else {
        //Nothing captured
        return
    }

    if let metadataObj =
        metadataObjects[0] as? AVMetadataMachineReadableCodeObject {
        guard metadataObj.type == AVMetadataObjectTypeQRCode else {
            return
        }

        let barCodeObject = videoPreviewLayer?
            .transformedMetadataObject(for:
                metadataObj as AVMetadataMachineReadableCodeObject)
             as! AVMetadataMachineReadableCodeObject

        if let qrValue = metadataObj.stringValue {
            self.handleQRRead(value: qrValue)
        }
    }
}

private handleQRRead(value: String) {
    //TODO: Handle the read qr
}
private captureSession: AVCaptureSession?
private videoPreviewLayer: AVCaptureVideo
}
```

`handleQRRead` - will be called on a successful scan `initCaptureSession` - initialize scanning for QR and camera input
`dismissCaptureSession` - hide the camera input and stop scanning

第172章：plist iOS

Plist用于iOS应用中的数据存储。Plist以数组和字典的形式保存数据。在plist中我们可以保存数据为：1. 应用中使用的静态数据。2. 来自服务器的数据。

第172.1节：示例：

1. 应用中使用的静态数据。

保存静态数据到plist请遵循以下方法：

a) 添加一个新文件

b) 点击资源中的属性列表（Property list）

Chapter 172: plist iOS

Plist is used for storage of data in iOS app. Plist save data in form of Array and Dictionaries. In plist we can save data as: 1. Static data to be used in app. 2. Data that will be coming from server.

Section 172.1: Example:

1. Static data to be used in app.

To save static data in plist follow these methods:

a) Add a new file

b) Click Property list in Resources

c) 命名属性列表，系统将创建一个文件（此处为data.plist）

d) 你可以创建包含数组和字典的plist，如下：

c) Name the propertylist and a file will be created as(data.plist here)

d) You can create a plist of Arrays and Dictionaries as:

// 从bundle中读取plist并获取根字典

```
NSDictionary *dictRoot = [NSDictionary dictionaryWithContentsOfFile:[[NSBundle mainBundle]
pathForResource:@"Data" ofType:@"plist"]];
```

// 你的字典包含一个字典数组 // 现在从中取出一个数组。

```
NSArray *arrayList = [NSArray arrayWithArray:[dictRoot objectForKey:@"Object1"]];

for(int i=0; i< [arrayList count]; i++)
{
    NSMutableDictionary *details=[arrayList objectAtIndex:i];
}
```

第172.2节：从Plist保存和编辑/删除数据

您已经创建了一个plist。该plist在应用中将保持不变。如果您想编辑此plist中的数据，添加新的数据或从plist中删除数据，您无法直接修改此文件。

为此，您需要将plist存储在文档目录中。您可以编辑保存在文档目录中的plist。

将plist保存到文档目录，如下所示：

```
NSString *filePath = [[NSBundle mainBundle] pathForResource:@"Data" ofType:@"plist"];

NSDictionary *dict = [[NSDictionary alloc] initWithContentsOfFile:filePath];

NSDictionary *plistDict = dict;
```

// Read plist from bundle and get Root Dictionary out of it

```
NSDictionary *dictRoot = [NSDictionary dictionaryWithContentsOfFile:[[NSBundle mainBundle]
pathForResource:@"Data" ofType:@"plist"]];
```

// Your dictionary contains an array of dictionary // Now pull an Array out of it.

```
NSArray *arrayList = [NSArray arrayWithArray:[dictRoot objectForKey:@"Object1"]];

for(int i=0; i< [arrayList count]; i++)
{
    NSMutableDictionary *details=[arrayList objectAtIndex:i];
}
```

Section 172.2: Save and edit/delete data from Plist

You have already created a plist. This plist will remain same in app. If you want to edit the data in this plist, add new data in plist or remove data from plist, you can't make changes in this file.

For this purpose you will have to store your plist in Document Directory. You can edit your plist saved in document directory.

Save plist in document directory as:

```
NSString *filePath = [[NSBundle mainBundle] pathForResource:@"Data" ofType:@"plist"];

NSDictionary *dict = [[NSDictionary alloc] initWithContentsOfFile:filePath];

NSDictionary *plistDict = dict;
```

```
NSFileManager *fileManager = [NSFileManager defaultManager];

NSString *error = nil;

NSData *plistData = [NSPropertyListSerialization dataFromPropertyList:plistDict
format:NSPropertyListXMLFormat_v1_0 errorDescription:&error];

if (![fileManager fileExistsAtPath: plistPath]) {

    if(plistData)
    {
        [plistData writeToFile:plistPath atomically:YES];
    }
}
else
{

}
```

从 Plist 中检索数据如下：

```
    NSArray *paths = NSSearchPathForDirectoriesInDomains (NSDocumentDirectory, NSUserDomainMask,
YES);
    NSString *documentsPath = [paths objectAtIndex:0];
    NSString *plistPath = [documentsPath stringByAppendingPathComponent:@"Data.plist"];
    NSDictionary *dict = [[NSDictionary alloc] initWithContentsOfFile:plistPath];

    NSArray *usersArray = [dict objectForKey:@"Object1"];
```

您可以根据需要编辑、删除、添加新数据，然后将 plist 重新保存到文档目录。


```
NSFileManager *fileManager = [NSFileManager defaultManager];

NSString *error = nil;

NSData *plistData = [NSPropertyListSerialization dataFromPropertyList:plistDict
format:NSPropertyListXMLFormat_v1_0 errorDescription:&error];

if (![fileManager fileExistsAtPath: plistPath]) {

    if(plistData)
    {
        [plistData writeToFile:plistPath atomically:YES];
    }
}
else
{

}
```

Retrieve data from Plist as:

```
    NSArray *paths = NSSearchPathForDirectoriesInDomains (NSDocumentDirectory, NSUserDomainMask,
YES);
    NSString *documentsPath = [paths objectAtIndex:0];
    NSString *plistPath = [documentsPath stringByAppendingPathComponent:@"Data.plist"];
    NSDictionary *dict = [[NSDictionary alloc] initWithContentsOfFile:plistPath];

    NSArray *usersArray = [dict objectForKey:@"Object1"];
```

You can edit remove, add new data as per your requirement and save the plist again to Document Directory.

第173章：WCSessionDelegate

WCSessionDelegate 适用于使用 WatchConnectivity 的 watch OS2 及以上版本。var watchSession : WCSession? func startWatchSession(){ if(WCSession.isSupported()){ watchSession = WCSession.default() watchSession!.delegate = self watchSession!.activate() } } 实现必需的方法：-didReceiveApplicationContext

第173.1节：Watch kit 控制器（WKInterfaceController）

```
import WatchConnectivity

var watchSession : WCSession?

    override func awake(withContext context: Any?) {
        super.awake(withContext: context)
        // 在此配置界面对象。
startWatchSession()
    }

func startWatchSession(){

        if(WCSession.isSupported()){
            watchSession = WCSession.default()
            watchSession!.delegate = self
            watchSession!.activate()
        }
    }

// 当 iOS 应用触发事件时，下面的代理方法回调
func session(_ session: WCSession, didReceiveApplicationContext applicationContext: [String : Any])
{
        print("did ReceiveApplicationContext at watch")
    }
```


Chapter 173: WCSessionDelegate

`WCSessionDelegate` works with watch OS2 + using WatchConnectivity. var watchSession : WCSession? func startWatchSession(){ if(WCSession.isSupported()){ watchSession = WCSession.default() watchSession!.delegate = self watchSession!.activate() } } Implement the required method:- `didReceiveApplicationContext`

Section 173.1: Watch kit controller (WKInterfaceController)

```
import WatchConnectivity

var watchSession : WCSession?

    override func awake(withContext context: Any?) {
        super.awake(withContext: context)
        // Configure interface objects here.
        startWatchSession()
    }

func startWatchSession(){

        if(WCSession.isSupported()){
            watchSession = WCSession.default()
            watchSession!.delegate = self
            watchSession!.activate()
        }
    }

//Callback in below delegate method when iOS app triggers event
func session(_ session: WCSession, didReceiveApplicationContext applicationContext: [String : Any])
{
        print("did ReceiveApplicationContext at watch")
    }
```

第174章：AppDelegate

AppDelegate 是一个协议，定义了由单例 UIApplication 对象在应用程序生命周期中的重要事件发生时调用的方法。

通常用于在应用启动时执行任务（设置应用环境，分析（例如：Mixpanel/GoogleAnalytics/Crashlitics)、数据库堆栈等）以及关闭（例如：保存数据库上下文）、处理 URL 打开请求和类似的全应用程序任务。

第174.1节：通过AppDelegate方法的所有应用状态

要获取更新或在应用上线前执行某些操作，您可以使用以下方法。

AppDidFinishLaunching

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions {
    // 在应用启动前编写代码
    return YES;
}
```

当应用进入前台时：

```
- (void)applicationWillEnterForeground:(UIApplication *)application {
    // 当应用从后台切换到前台激活状态时调用；在这里可以撤销进入后台时所做的许多更改。

}
```

当应用启动以及从后台切换到前台时，会调用以下方法：

```
- (void)applicationDidBecomeActive:(UIApplication *)application {
    // 重新启动在应用处于非活动状态时暂停（或尚未开始）的任何任务。
如果应用之前处于后台状态，可以选择刷新用户界面。
}
```

当应用进入后台时：

```
- (void)applicationDidEnterBackground:(UIApplication *)application {
    // 使用此方法释放共享资源，保存用户数据，失效定时器，并存储足够的应用状态信息，以便在应用被终止后恢复到当前状
态。

    // 如果您的应用支持后台执行，当用户退出时，将调用此方法，而不是applicationWillTerminate:。

}
```

当应用即将失去激活状态时

```
- (void)applicationWillResignActive:(UIApplication *)application {
    // 当应用即将从激活状态转为非激活状态时发送。此情况可能发生在某些临时中断（如来电或短信）时，或用户退出应用并开始
切换到后台状态时。

    // 使用此方法暂停正在进行的任务，禁用定时器，并使图形渲染回调失效。游戏应使用此方法暂停游戏。

}
```


Chapter 174: AppDelegate

AppDelegate is a protocol which defines methods that are called by the singleton UIApplication object in response to important events in the lifetime of an app.

Normally used to perform tasks on application startup (setup app environment, analitycs (ex.: Mixpanel/GoogleAnalytics/Crashlitics), DB stack etc.) and shutdown (ex.: save DB context), handling URL open requests and similar application-wide tasks.

Section 174.1: All States of Application through AppDelegate methods

For Getting updated or to do something before app goes live to user you can use below method.

AppDidFinishLaunching

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions {
    // Write your code before app launch
    return YES;
}
```

While App Enter in Foreground:

```
- (void)applicationWillEnterForeground:(UIApplication *)application {
    // Called as part of the transition from the background to the active state; here you can undo
many of the changes made on entering the background.
}
```

When App Launching and also background to Foreground hit below method:

```
- (void)applicationDidBecomeActive:(UIApplication *)application {
    // Restart any tasks that were paused (or not yet started) while the application was inactive.
If the application was previously in the background, optionally refresh the user interface.
}
```

While App Enter in background:

```
- (void)applicationDidEnterBackground:(UIApplication *)application {
    // Use this method to release shared resources, save user data, invalidate timers, and store
enough application state information to restore your application to its current state in case it is
terminated later.
    // If your application supports background execution, this method is called instead of
applicationWillTerminate: when the user quits.
}
```

While app resign active

```
- (void)applicationWillResignActive:(UIApplication *)application {
    // Sent when the application is about to move from active to inactive state. This can occur for
certain types of temporary interruptions (such as an incoming phone call or SMS message) or when
the user quits the application and it begins the transition to the background state.
    // Use this method to pause ongoing tasks, disable timers, and invalidate graphics rendering
callbacks. Games should use this method to pause the game.
}
```

应用终止时：

```
- (void)applicationWillTerminate:(UIApplication *)application {
    // 当应用即将终止时调用。如有必要，保存数据。另见
applicationDidEnterBackground:。
}
```

第174.2节：AppDelegate的角色：

- AppDelegate包含您的应用启动代码。
- 它响应应用状态的关键变化。具体来说，它响应临时中断以及应用执行状态的变化，例如应用从前台切换到后台时。
- 它响应来自应用外部的通知，例如远程通知（也称为推送通知）、低内存警告、下载完成通知等。
- 它决定是否应进行状态保存和恢复，并在需要时协助保存和恢复过程。
- 它响应针对应用本身的事件，而非特定于应用视图或视图控制器的事件。您可以使用它来存储应用的核心数据对象或任何没有所属视图控制器的内容。

第174.3节：打开指定URL的资源

请求代理打开由 URL 指定的资源，并提供一个启动选项字典。

使用示例：

```
func application(_ app: UIApplication, open url: URL, options: [UIApplicationOpenURLOptionsKey :
Any] = [:]) -> Bool {
        return SomeManager.shared.handle(
            url,
sourceApplication: options[.sourceApplication] as? String,
            annotation: options[.annotation]
        )
    }
```

第174.4节：处理本地和远程通知

使用示例：

```
/* 你自定义的 APNs/本地通知管理器实例 */
private var pushManager: AppleNotificationManager!
```

注册：

```
func application(application: UIApplication, didRegisterUserNotificationSettings
notificationSettings: UIUserNotificationSettings) {
    // 调用以告知代理可以使用哪些类型的通知来获取用户的注意力

pushManager.didRegisterSettings(notificationSettings)
}

func application(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken
deviceToken: NSData) {
    // 告诉代理应用已成功注册到苹果推送通知服务（Apple Push Notification service）
```


While app terminate:

```
- (void)applicationWillTerminate:(UIApplication *)application {
    // Called when the application is about to terminate. Save data if appropriate. See also
applicationDidEnterBackground:.
}
```

Section 174.2: AppDelegate Roles:

- AppDelegate contains your app's `startup code`.
- It responds to `key changes` in the `state` of your app. Specifically, it responds to both temporary interruptions and to changes in the execution state of your app, such as when your app transitions from the foreground to the background.
- It `responds to notifications` originating from outside the app, such as remote notifications (also known as push notifications), low-memory warnings, download completion notifications, and more.
- It `determines` whether `state preservation` and `restoration` should occur and assists in the preservation and restoration process as needed.
- It `responds to events` that target the app itself and are not specific to your app's views or view controllers. You can use it to store your app's central data objects or any content that does not have an owning view controller.

Section 174.3: Opening a URL-Specified Resource

Asks the delegate to open a resource specified by a URL, and provides a dictionary of launch options.

Example of usage:

```
func application(_ app: UIApplication, open url: URL, options: [UIApplicationOpenURLOptionsKey :
Any] = [:]) -> Bool {
        return SomeManager.shared.handle(
            url,
            sourceApplication: options[.sourceApplication] as? String,
            annotation: options[.annotation]
        )
    }
```

Section 174.4: Handling Local and Remote Notifications

Example of usage:

```
/* Instance of your custom APNs/local notification manager */
private var pushManager: AppleNotificationManager!
```

Registration:

```
func application(application: UIApplication, didRegisterUserNotificationSettings
notificationSettings: UIUserNotificationSettings) {
    // Called to tell the delegate the types of notifications that can be used to get the user's
attention
    pushManager.didRegisterSettings(notificationSettings)
}

func application(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken
deviceToken: NSData) {
    // Tells the delegate that the app successfully registered with Apple Push Notification service
```

```
(APNs)
pushManager.didRegisterDeviceToken(deviceToken)
}

func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError
error: NSError) {
    // 当苹果推送通知服务无法成功完成注册过程时发送给代理

pushManager.didFailToRegisterDeviceToken(error)
}
```

远程通知处理：

```
func application(application: UIApplication, didReceiveRemoteNotification userInfo: [NSObject :
AnyObject]) {
    // 远程通知到达，有数据需要获取
    // 进行处理
pushManager.handleNotification(userInfo,
background: application.applicationState == .Background
    )
}
```

本地通知处理：

```
func application(application: UIApplication, didReceiveLocalNotification notification:
UILocalNotification) {
pushManager.handleLocalNotification(notification, background: false)
}
```

处理操作（已弃用）：

```
func application(application: UIApplication, handleActionWithIdentifier identifier: String?,
forRemoteNotification userInfo: [NSObject : AnyObject],
completionHandler: () -> Void) {
    pushManager.handleInteractiveRemoteNotification(userInfo, actionIdentifier: identifier,
completion: completionHandler)
}
```



```
(APNs)
    pushManager.didRegisterDeviceToken(deviceToken)
}

func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError
error: NSError) {
    // Sent to the delegate when Apple Push Notification service cannot successfully complete the
registration process.
    pushManager.didFailToRegisterDeviceToken(error)
}
```

Remote notifications handling:

```
func application(application: UIApplication, didReceiveRemoteNotification userInfo: [NSObject :
AnyObject]) {
    // Remote notification arrived, there is data to be fetched
    // Handling it
    pushManager.handleNotification(userInfo,
                                   background: application.applicationState == .Background
    )
}
```

Local notifications handling:

```
func application(application: UIApplication, didReceiveLocalNotification notification:
UILocalNotification) {
    pushManager.handleLocalNotification(notification, background: false)
}
```

Handling action (deprecated):

```
func application(application: UIApplication, handleActionWithIdentifier identifier: String?,
forRemoteNotification userInfo: [NSObject : AnyObject],
                     completionHandler: () -> Void) {
    pushManager.handleInteractiveRemoteNotification(userInfo, actionIdentifier: identifier,
completion: completionHandler)
}
```

第175章：应用提交流程

本教程涵盖了上传 iOS 应用到 App Store 所需的所有必要步骤。

第175.1节：设置配置描述文件

在以前的版本中，配置描述文件的设置是手动完成的。您需要生成分发描述文件，下载它，然后分发您的应用。这必须针对每台开发机器进行，非常耗时。然而，在大多数情况下，现在的 Xcode 8 会为您完成大部分工作。确保您使用将用于分发应用的账户登录，然后只需在“Targets -> General”中选择“自动管理代码签名”。

第175.2节：归档代码

配置描述文件全部设置完成后，提交应用的下一步是归档您的代码。
从设备和模拟器的下拉菜单中选择“通用 iOS 设备”选项。然后，在“Product”菜单下选择“归档”选项。

如果提交的是对商店中现有应用的更新，请确保构建号高于当前版本且版本号不同。例如，当前应用的构建号为30，版本标签为1.0。下一次更新应至少为构建号31，版本标签为1.0.1。在大多数情况下，您应在版本号中添加第三位小数以应对紧急的错误修复或小补丁，第二位小数主要用于功能更新，而第一位小数在重大应用更新时递增。

第175.3节：导出 IPA 文件

完成后，您可以在 Xcode 归档管理器中找到您的归档文件。这里保存并管理着您所有之前的版本和归档构建，除非您删除它们。您会立即注意到一个大蓝色按钮写着“上传到 App Store...”，但在90%的情况下，由于各种原因（主要是 Xcode 的错误），此按钮无法正常工作。解决方法是导出归档文件，然后使用另一个名为 Application Loader 的 Xcode 工具上传。
但是，由于 Application Loader 是上传 IPA 文件到 App Store 的工具，归档文件需要导出为正确的格式。这是一个简单的任务，大约需要半小时。点击右侧面板中的“导出”按钮。

Chapter 175: App Submission Process

This tutorial covers all the necessary steps required to upload an iOS app to the App Store.

Section 175.1: Setup provisioning profiles

In previous versions, setting up provisioning profiles was done manually. You generate distribution profile, download it and then distribute your app. This had to be done for every development machine which was extremely time consuming. However, in most situations nowadays, Xcode 8 will do most of this work for you. Make sure you sign in with the account which will be used for distributing the app and then simply select "Automatically manage code signing" in Targets -> General.

Section 175.2: Archive the code

Once the provisioning profiles are all set, next step in the process of submitting the app is to archive your code. From the dropdown of devices and simulators select option "Generic iOS device". Then, under "Product" menu select option "Archive".

In case where the submission is an update to the existing app on the store, make sure that the build number is higher than the current and that the version number is different. For example, current application has build number 30 and version label 1.0. The next update should have at least build number 31 and version label 1.0.1. In most cases, you should add third decimal to your version in case of some urgent bug fixes or small patches, second decimal is mostly reserved for feature updates while first decimal is incremented in case of a major app update.

Section 175.3: Export IPA file

Once it's done, you can find your archive in the Xcode organizer. This is where all your previous versions and archive builds are saved and organized in case you do not delete them. You will immediately notice a large blue button saying "Upload to App Store..." however in 9/10 cases this will not work due to various reasons (Xcode bugs mostly). Workaround is to export your archive and upload it using another Xcode tool called Application Loader. However, since Application loader uploads IPA files to the App Store, archive needs to be exported to the correct format. This is a trivial task which might take ~ half an hour. Click on the "Export" button in the right side panel.

如果您要将应用上传到App Store，选择第一个选项并点击下一步。再次登录并验证您的代码，然后去喝杯咖啡。导出过程完成后，系统会询问您保存生成的IPA文件的位置。通常，桌面是最方便的选择。

第175.4节：使用Application Loader上传IPA文件

IPA文件生成后，打开Xcode，导航到开发者工具并打开Application Loader。

如果您的Xcode中有多个账户，系统会要求您选择。自然地选择您在第一步中用于代码签名的账户。选择“交付您的应用”并上传代码。上传完成后，可能需要最多一小时才能在iTunes Connect的构建列表中显示。

If you are uploading app to the App Store, select the first option and click Next. Sign in and validate your code once again and go grab a cup of coffee. Once the exporting process is done, you will be asked where to save the generated IPA file. Usually, desktop is the most convenient choice.

Section 175.4: Upload IPA file using Application Loader

Once the IPA file is generated, open Xcode, navigate to developer tools and open Application Loader.

If you have multiple accounts in your Xcode, you will be asked to choose. Naturally pick the one you used for code signing in the first step. Pick "Deliver your app" and upload the code. Once the upload is done it may take up to one

可能需要最多一小时才能在iTunes Connect的构建列表中显示。

hour to appear in your build list on iTunes Connect.

第176章：FileHandle

从文档目录分块读取文件

第176.1节：从文档目录分块读取文件

我从文档目录获取文件路径，并以1024字节为块读取该文件，然后保存（追加）到NSMutableData对象，或者你也可以直接写入套接字。

```
// MARK: - 获取文件数据块的方法。
func getFileDataInChunks() {

    let documentDirectoryPath = NSSearchPathForDirectoriesInDomains(.documentDirectory,
.userDomainMask, true)[0] as NSString
    let filePath = documentDirectoryPath.appendingPathComponent("video.mp4")


    // 检查路径下文件是否存在。
    if FileManager.default.fileExists(atPath: filePath) {

        let chunkSize = 1024 // 将数据分成1 KB

        // 创建NSMutableData对象以保存读取的数据。
        let ReadData = NSMutableData()

        do {

            //打开文件以供读取。
            outputFileHandle = try FileHandle(forReadingFrom: URL(fileURLWithPath: filePath))

            //获取第一个数据块
            var datas = outputFileHandle?.readData(ofLength: chunkSize)

            //检查下一个数据块是否为空。
            while !(datas?.isEmpty)! {

                //这里我将数据块写入ReadData，或者你也可以直接写入socket。
ReadData.append(datas!)

                //获取下一个数据块
datas = outputFileHandle?.readData(ofLength: chunkSize)

                print("Running: \(ReadData.length)")
            }

            //读取数据完成后关闭outputFileHandle。
outputFileHandle?.closeFile()

            print("文件读取完成")

        }catch let error as NSError {
            print("错误 : \(error.localizedDescription)")
        }
    }
}
```

文件读取完成后，你将在ReadData变量中获得文件数据，这里outputFileHandle是FileHandle的一个对象

Chapter 176: FileHandle

Read file in chunks from document directory

Section 176.1: Read file from document directory in chunks

I get the file path from document directory and read that file in chunks of 1024 and save (append) to `NSMutableData` object or you can directly write to socket.

```
// MARK: - Get file data as chunks Methode.
func getFileDataInChunks() {

    let doumentDirectoryPath = NSSearchPathForDirectoriesInDomains(.documentDirectory,
.userDomainMask, true)[0] as NSString
    let filePath = doumentDirectoryPath.appendingPathComponent("video.mp4")


    //Check file exits at path or not.
    if FileManager.default.fileExists(atPath: filePath) {

        let chunkSize = 1024 // divide data into 1 kb

        //Create NSMutableData object to save read data.
        let ReadData = NSMutableData()

        do {

            //open file for reading.
            outputFileHandle = try FileHandle(forReadingFrom: URL(fileURLWithPath: filePath))

            // get the first chunk
            var datas = outputFileHandle?.readData(ofLength: chunkSize)

            //check next chunk is empty or not.
            while !(datas?.isEmpty)! {

                //here I write chunk data to ReadData or you can directly write to socket.
                ReadData.append(datas!)

                // get the next chunk
                datas = outputFileHandle?.readData(ofLength: chunkSize)

                print("Running: \(ReadData.length)")
            }

            //close outputFileHandle after reading data complete.
            outputFileHandle?.closeFile()

            print("File reading complete")

        }catch let error as NSError {
            print("Error : \(error.localizedDescription)")
        }
    }
}
```

After file reading complete you will get file Data in `ReadData` variable Here outputFileHandle is a object of `FileHandle`

```
var outputFileHandle:FileHandle?
```



```
var outputFileHandle:FileHandle?
```

第177章：基础文本文件输入输出

第177.1节：从文档文件夹读取和写入

Swift 3

```
import UIKit

// 保存字符串到文件
let fileName = "TextFile"
let documentDirectory = try FileManager.default.urlForDirectory(.documentDirectory, in:
.userDomainMask, appropriateFor: nil, create: true)

var fileURL = try documentDirectory.appendingPathComponent(fileName).appendingPathExtension("txt")

print("文件路径: \(fileURL.path)")

var toFileString = "要写入的文本"
do {
  // 写入文件
try toFileString.writeToURL(fileURL, atomically: true, encoding: NSUTF8StringEncoding)
} catch let error as NSError {
  print("写入URL失败: \(fileURL), 错误:\(error.localizedDescription)")
}

// 读取
var fromFileString = ""
do {
fromFileString = try String(contentsOfURL: fileURL)
} catch let error as NSError {
  print("从URL读取失败: \(fileURL), 错误: " + error.localizedDescription)
}
print("来自文件的文本输入: \(fromFileString)")
```

Swift 2

```
import UIKit

// 保存字符串到文件
let fileName = "TextFile"
let DocumentDirectoryURL = try! NSFileManager.defaultManager().URLForDirectory(.DocumentDirectory,
inDomain: .UserDomainMask, appropriateForURL: nil, create: true)

let fileURL =
DocumentDirectoryURL.URLByAppendingPathComponent(fileName).URLByAppendingPathExtension("txt")
print("文件路径: \(fileURL.path)")

var toFileString = "要写入的文本"
do {
  // 写入文件
try toFileString.writeToURL(fileURL, atomically: true, encoding: NSUTF8StringEncoding)
} catch let error as NSError {
  print("写入URL失败: \(fileURL), 错误:\(error.localizedDescription)")
}

// 读取
var fromFileString = ""
do {
fromFileString = try String(contentsOfURL: fileURL)
```


Chapter 177: Basic text file I/O

Section 177.1: Read and write from Documents folder

Swift 3

```
import UIKit

// Save String to file
let fileName = "TextFile"
let documentDirectory = try FileManager.default.urlForDirectory(.documentDirectory, in:
.userDomainMask, appropriateFor: nil, create: true)

var fileURL = try documentDirectory.appendingPathComponent(fileName).appendingPathExtension("txt")

print("FilePath: \(fileURL.path)")

var toFileString = "Text to write"
do {
  // Write to file
  try toFileString.writeToURL(fileURL, atomically: true, encoding: NSUTF8StringEncoding)
} catch let error as NSError {
  print("Failed writing to URL: \(fileURL), Error:\(error.localizedDescription)")
}

// Reading
var fromFileString = ""
do {
  fromFileString = try String(contentsOfURL: fileURL)
} catch let error as NSError {
  print("Failed reading from URL: \(fileURL), Error: " + error.localizedDescription)
}
print("Text input from file: \(fromFileString)")
```

Swift 2

```
import UIKit

// Save String to file
let fileName = "TextFile"
let DocumentDirectoryURL = try! NSFileManager.defaultManager().URLForDirectory(.DocumentDirectory,
inDomain: .UserDomainMask, appropriateForURL: nil, create: true)

let fileURL =
DocumentDirectoryURL.URLByAppendingPathComponent(fileName).URLByAppendingPathExtension("txt")
print("FilePath: \(fileURL.path)")

var toFileString = "Text to write"
do {
  // Write to file
  try toFileString.writeToURL(fileURL, atomically: true, encoding: NSUTF8StringEncoding)
} catch let error as NSError {
  print("Failed writing to URL: \(fileURL), Error:\(error.localizedDescription)")
}

// Reading
var fromFileString = ""
do {
  fromFileString = try String(contentsOfURL: fileURL)
```

```
} catch let error as NSError {
   print("从URL读取失败: \(fileURL), 错误: " + error.localizedDescription)
}
print("来自文件的文本输入: \(fromFileString)")
```



```
} catch let error as NSError {
  print("Failed reading from URL: \(fileURL), Error: " + error.localizedDescription)
}
print("Text input from file: \(fromFileString)")
```

第178章：iOS 语音合成（TTS）

了解如何在iOS设备上将文本合成为语音

第178.1节：文本转语音

Objective C

```
AVSpeechSynthesizer *synthesizer = [[AVSpeechSynthesizer alloc]init];
AVSpeechUtterance *utterance = [AVSpeechUtterance speechUtteranceWithString:@"Some text"];
[utterance setRate:0.2f];
[synthesizer speakUtterance:utterance];
```

Swift

```
let synthesizer = AVSpeechSynthesizer()
let utterance = AVSpeechUtterance(string: "Some text")
utterance.rate = 0.2
```

你也可以这样更改语音：

```
utterance.voice = AVSpeechSynthesisVoice(language: "fr-FR")
```

然后进行朗读

- 在Swift 2中：`synthesizer.speakUtterance(utterance)`
- 在Swift 3中：`synthesizer.speak(utterance)`

别忘了导入AVFoundation

有用的方法

您可以使用以下两种方法停止或暂停所有语音：

```
- (BOOL)pauseSpeakingAtBoundary:(AVSpeechBoundary)boundary;
- (BOOL)stopSpeakingAtBoundary:(AVSpeechBoundary)boundary;
```

AVSpeechBoundary 指示语音是应立即暂停或停止（AVSpeechBoundaryImmediate），还是应在当前正在朗读的单词之后暂停或停止（AVSpeechBoundaryWord）。

Chapter 178: iOS TTS

Find how to produce synthesized speech from text on an iOS device

Section 178.1: Text To Speech

Objective C

```
AVSpeechSynthesizer *synthesizer = [[AVSpeechSynthesizer alloc]init];
AVSpeechUtterance *utterance = [AVSpeechUtterance speechUtteranceWithString:@"Some text"];
[utterance setRate:0.2f];
[synthesizer speakUtterance:utterance];
```

Swift

```
let synthesizer = AVSpeechSynthesizer()
let utterance = AVSpeechUtterance(string: "Some text")
utterance.rate = 0.2
```

You can also change the voice like this :

```
utterance.voice = AVSpeechSynthesisVoice(language: "fr-FR")
```

And then speek

- In Swift 2 : `synthesizer.speakUtterance(utterance)`
- In Swift 3 : `synthesizer.speak(utterance)`

Don't forget to import AVFoundation

Helpful methods

You can Stop or Pause all speech using these two methods :

```
- (BOOL)pauseSpeakingAtBoundary:(AVSpeechBoundary)boundary;
- (BOOL)stopSpeakingAtBoundary:(AVSpeechBoundary)boundary;
```

The AVSpeechBoundary indicates if the speech should pause or stop immediately (`AVSpeechBoundaryImmediate`) or it should pause or stop after the word currently being spoken (`AVSpeechBoundaryWord`).

第179章：MPVolumeView

MPVolumeView 类是一个音量视图，向用户展示一个滑块控件用于设置系统音频输出音量，以及一个按钮用于选择音频输出路径。

第179.1节：添加 MPVolumeView

```
// 在一个容器视图中添加 MPVolumeView
let mpVolumeHolderView = UIView(frame: CGRect(x: 0, y: view.bounds.midY, width: view.bounds.width,
height: view.bounds.height))
// 设置容器视图的背景色为透明
mpVolumeHolderView.backgroundColor = .clear
let mpVolume = MPVolumeView(frame: mpVolumeHolderView.bounds)
mpVolume.showsRouteButton = true
mpVolumeHolderView.addSubview(mpVolume)
view.addSubview(mpVolumeHolderView)
// 音量视图是白色的，为了在此示例中更好地显示它，将父视图背景设置为黑色
view.backgroundColor = .black
```

!!! 一个非常重要的注意事项是，MPVolumeView 仅在真实设备上有效，模拟器上无效。

Chapter 179: MPVolumeView

The MPVolumeView class is volume view to presents the user with a slider control for setting the system audio output volume, and a button for choosing the audio output route.

Section 179.1: Adding a MPVolumeView

```
// Add MPVolumeView in a holder view
let mpVolumeHolderView = UIView(frame: CGRect(x: 0, y: view.bounds.midY, width: view.bounds.width,
height: view.bounds.height))
// Set the holder view's background color to transparent
mpVolumeHolderView.backgroundColor = .clear
let mpVolume = MPVolumeView(frame: mpVolumeHolderView.bounds)
mpVolume.showsRouteButton = true
mpVolumeHolderView.addSubview(mpVolume)
view.addSubview(mpVolumeHolderView)
// the volume view is white, set the parent background to black to show it better in this example
view.backgroundColor = .black
```

!!! A very important note is that the MPVolumeView only works on an actual device and not on a simulator.

第180章：Objective-C 关联对象

参数	详细信息
object	你想要修改的现有对象
key	这基本上可以是任何具有固定内存地址的指针，但一个好的做法是在这里使用一个计算属性（getter）
value	您想要添加的对象
policy	这个新value的内存策略，即是否应被保留/赋值、复制等，就像你声明的其他属性一样属性

首次在 iOS 3.1 中作为 Objective-C 运行时的一部分引入，关联对象提供了一种向现有类对象添加实例变量的方法（无需子类化）。

这意味着你可以将任何对象附加到任何其他对象上，而无需子类化。

第180.1节：基本关联对象示例

假设我们需要向SomeClass添加一个 NSString 对象（我们不能子类化）。

在此示例中，我们不仅创建了一个关联对象，还将其包装在类别中的计算属性中，以实现更整洁的代码

```
#import <objc/runtime.h>

@interface SomeClass (MyCategory)
// 这是包装关联对象的属性。下面我们实现了 setter 和 getter
，实际上利用了对象关联机制
@property (nonatomic, retain) NSString *associated;
@end

@implementation SomeClass (MyCategory)

- (void)setAssociated:(NSString *)object {
    objc_setAssociatedObject(self, @selector(associated), object,
                             OBJC_ASSOCIATION_RETAIN_NONATOMIC);
}

- (NSString *)associated {
    return objc_getAssociatedObject(self, @selector(associated));
}
```

现在使用该属性就这么简单

```
SomeClass *instance = [SomeClass alloc] init];
instance.associated = @"this property is an associated object under the hood";
```


Chapter 180: Objective-C Associated Objects

Param	Details
object	The existing object you want to modify
key	This can basically be any pointer that has a constant memory address, but a nice practice is to use here a computed property (getter)
value	The object you want to add
policy	The memory policy for this new `value` i.e. should it be retained / assigned, copied etc.. just like any other property you'd declare

First introduced in iOS 3.1 as part of the Objective-C runtime, associated objects provide a way to add instance variables to an existing class object (w\o subclassing.

This means you'll be able to attach any object to any other object without subclassing.

Section 180.1: Basic Associated Object Example

Assume we need to add an NSString object to `SomeClass` (we cant subclass).

In this example we not only create an associated object but also wrap it in a computed property in a category for extra neatness

```
#import <objc/runtime.h>

@interface SomeClass (MyCategory)
// This is the property wrapping the associated object. below we implement the setter and getter
which actually utilize the object association
@property (nonatomic, retain) NSString *associated;
@end

@implementation SomeClass (MyCategory)

- (void)setAssociated:(NSString *)object {
    objc_setAssociatedObject(self, @selector(associated), object,
                             OBJC_ASSOCIATION_RETAIN_NONATOMIC);
}

- (NSString *)associated {
    return objc_getAssociatedObject(self, @selector(associated));
}
```

Now it would be as easy as this to use the property

```
SomeClass *instance = [SomeClass alloc] init];
instance.associated = @"this property is an associated object under the hood";
```

第181章：视图控制器之间传递数据（基于MessageBox概念）

MessageBox是一个用于解耦实体的简单概念。

例如，实体A可以发送一条消息，实体B可以在合适的时候读取。

一个视图控制器想要与另一个视图控制器通信，但你不想创建强引用或弱引用关系。

第181.1节：简单示例用法

```
let messageBox:MessageBox = MessageBox()

// 设置
messageBox.setObject("TestObject1", forKey:"TestKey1")

// 获取
// 但不移除它，保持存储状态，以便稍后仍可检索
let someObject:String = messageBox.getObject(forKey:"TestKey1", removeIfFound:false)

// 获取
// 并移除它
let someObject:String = messageBox.getObject(forKey:"TestKey1", removeIfFound:true)
```


Chapter 181: Passing Data between View Controllers (with MessageBox-Concept)

MessageBox is a simple concept for decoupling entities.

For example entity A can place a message that entity B can read whenever suitable.

A view controller would like to talk to another view controller, but you don't want to create a strong or weak relationship.

Section 181.1: Simple Example Usage

```
let messageBox:MessageBox = MessageBox()

// set
messageBox.setObject("TestObject1", forKey:"TestKey1")

// get
// but don't remove it, keep it stored, so that it can still be retrieved later
let someObject:String = messageBox.getObject(forKey:"TestKey1", removeIfFound:false)

// get
// and remove it
let someObject:String = messageBox.getObject(forKey:"TestKey1", removeIfFound:true)
```

第182章：MVVM

第182.1节：无响应式编程的MVVM

我将先简短地解释一下什么是Model-View-ViewModel（MVVM）设计模式以及为什么在你的iOS应用中使用它。当iOS刚出现时，苹果建议使用MVC（Model-View-Controller）作为设计模式。他们在所有示例中都展示了它，所有初期开发者都很乐意使用它，因为它很好地将业务逻辑和用户界面分离开来。随着应用变得越来越大和复杂，一个新的问题出现了，恰当地称为“庞大视图控制器”（Massive View Controllers，MVC）。因为所有业务逻辑都被添加到了ViewController中，随着时间推移，它们通常变得过于庞大和复杂。为了解决MVC问题，iOS世界引入了一种新的设计模式——Model-View-ViewModel（MVVM）模式。

上图展示了MVVM的结构。你有一个标准的ViewController + View（在Storyboard、XIB或代码中），它作为MVVM的View（后文中“View”将指MVVM的View）。View持有对ViewModel的引用，ViewModel中包含我们的业务逻辑。需要注意的是，ViewModel对View一无所知，也从不持有对View的引用。ViewModel持有对Model的引用。

理论部分关于MVVM就讲到这里。更多内容可以阅读 here。

MVVM的一个主要问题是，当ViewModel没有任何引用且甚至对View一无所知时，如何通过ViewModel更新View。

本示例的主要部分是展示如何使用MVVM（更准确地说，如何绑定ViewModel和View）而不使用任何响应式编程（ReactiveCocoa、ReactiveSwift或RxSwift）。顺便提一句：如果你想使用响应式编程，那更好，因为使用它可以非常轻松地完成MVVM绑定。但本示例是关于如何在不使用响应式编程的情况下使用MVVM。

让我们创建一个简单的示例来演示如何使用MVVM。

我们的MVVMExampleViewController是一个简单的ViewController，包含一个标签和一个按钮。当按钮被按下时，标签文本应设置为“Hello”。由于决定用户交互时该做什么是业务逻辑的一部分，ViewModel将负责决定用户按下按钮时该做什么。MVVM的View不应包含任何业务逻辑。

```
class MVVMExampleViewController: UIViewController {

    @IBOutlet weak var helloLabel: UILabel!

    var viewModel: MVVMExampleViewModel?

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    @IBAction func sayHelloButtonPressed(_ sender: UIButton) {
        viewModel?.userTriggeredSayHelloButton()
    }
}
```

MVVMExampleViewModel 是一个简单的视图模型（ViewModel）。

Chapter 182: MVVM

Section 182.1: MVVM Without Reactive Programming

I'll start with a really short explanation what is and why use Model-View-ViewModel (MVVM) design pattern in your iOS apps. When iOS first appeared, Apple suggested to use MVC (Model-View-Controller) as a design pattern. They showed it in all of their examples and all first developers were happy using it because it nicely separated concerns between business logic and user interface. As applications became larger and more complex a new problem appeared appropriately called Massive View Controllers (MVC). Because all business logic was added in the ViewController, with time they usually became too large and complex. To avoid MVC issue, a new design pattern was introduced to the world of iOS - Model-View-ViewModel (MVVM) pattern.

The diagram above shows how MVVM looks like. You have a standard ViewController + View (in storyboard, XIB or Code), which acts as MVVM's View (in later text - View will reference MVVM's View). A view has a reference to a ViewModel, where our business logic is. It's important to notice that ViewModel doesn't know anything about the View and never has a reference to the view. ViewModel has a reference to a Model.
This is enough with a theoretical part of the MVVM. More about it can be read here.

One of the **main issues with MVVM** is how to update View via the ViewModel when ViewModel doesn't have any references and doesn't even know anything about the View.

The main part of this example is to show how to use MVVM (more precisely, how to bind ViewModel and View) without any reactive programming (ReactiveCocoa, ReactiveSwift or RxSwif). Just as a note: if you want to use Reactive programming, even better since MVVM bindings are done really easy using it. But this example is on how to use MVVM without Reactive programming.

Let's create a simple example to demonstrate how to use MVVM.

Our `MVVMExampleViewController` is a simple ViewController with a label and a button. When button is pressed, the label text should be set to 'Hello'. Since deciding what to do on user user interaction is part of the business logic, ViewModel will have to decide what to do when user presses the button. MVVM's View shouldn't do any business logic.

```
class MVVMExampleViewController: UIViewController {

    @IBOutlet weak var helloLabel: UILabel!

    var viewModel: MVVMExampleViewModel?

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    @IBAction func sayHelloButtonPressed(_ sender: UIButton) {
        viewModel?.userTriggeredSayHelloButton()
    }
}
```

`MVVMExampleViewModel` is a simple ViewModel.

```
class MVVMExampleViewModel {

    func userTriggeredSayHelloButton() {
        // 当没有视图的引用时，如何更新视图的标签？
    }
}
```

你可能会想，如何在视图中设置视图模型的引用。我通常在视图控制器（ViewController）初始化时或显示之前进行设置。对于这个简单的例子，我会在AppDelegate中这样做：

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        if let rootVC = window?.rootViewController as? MVVMExampleViewController {
            let viewModel = MVVMExampleViewModel()
rootVC.viewModel = viewModel
        }

        返回 true
```

现在真正的问题是：如何在不将视图的引用传给视图模型的情况下，从视图模型更新视图？（记住，我们不会使用任何iOS的响应式编程库）

你可以考虑使用KVO，但那样会让事情变得过于复杂。一些聪明的人思考过这个问题，提出了Bond库。这个库起初看起来可能有些复杂且难以理解，所以我只取其中一小部分，让我们的MVVM完全可用。

让我们介绍Dynamic类，它是我们简单但功能齐全的MVVM模式的核心。

```
class Dynamic<T> {
    typealias Listener = (T) -> Void
    var listener: Listener?

    func bind(_ listener: Listener?) {
        self.listener = listener
    }

    func bindAndFire(_ listener: Listener?) {
        self.listener = listener
listener?(value)
    }

    var value: T {
        didSet {
listener?(value)
        }
    }

    init(_ v: T) {
       value = v
    }
}
```

`Dynamic` 类使用泛型和闭包将我们的视图模型与视图绑定。我不会详细讲解这个类，我们可以在评论中讨论（以使这个示例更简短）。现在让我们更新我们的 `MVVMExampleViewController` 和 `MVVMExampleViewModel` 用于使用这些类。

我们更新的 `MVVMExampleViewController`


```
class MVVMExampleViewModel {

    func userTriggeredSayHelloButton() {
        // How to update View's label when there is no reference to the View??
    }
}
```

You might wonder, how to set the ViewModel's reference in the View. I usually do it when ViewController is being initialized or before it will be shown. For this simple example, I would do something like this in the `AppDelegate`:

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        if let rootVC = window?.rootViewController as? MVVMExampleViewController {
            let viewModel = MVVMExampleViewModel()
            rootVC.viewModel = viewModel
        }

        return true
```

The real question now is: how to update View from ViewModel without giving a reference to the View to the ViewModel? (Remember, we won't use any of the Reactive Programming iOS libraries)

You could think about using KVO, but that would just complicate things too much. Some smart people have thought about the issue and came up with the Bond library. The library might seem complicated and a little harder to understand at first, so I'll just take one small part of it and make our MVVM fully functional.

Let's introduce the `Dynamic` class which is the core of our simple yet fully functional MVVM pattern.

```
class Dynamic<T> {
    typealias Listener = (T) -> Void
    var listener: Listener?

    func bind(_ listener: Listener?) {
        self.listener = listener
    }

    func bindAndFire(_ listener: Listener?) {
        self.listener = listener
        listener?(value)
    }

    var value: T {
        didSet {
            listener?(value)
        }
    }

    init(_ v: T) {
        value = v
    }
}
```

`Dynamic` class is using Generics and Closures to bind our ViewModel with our View. I won't go into details about this class, we can do it in the comments (to make this example shorter). Let's now update our `MVVMExampleViewController` and `MVVMExampleViewModel` to use those classes.

Our updated `MVVMExampleViewController`

```
class MVVMExampleViewController: UIViewController {

    @IBOutlet weak var helloLabel: UILabel!

    var viewModel: MVVMExampleViewModel?

    override func viewDidLoad() {
        super.viewDidLoad()
bindViewModel()
    }

    func bindViewModel() {
        if let viewModel = viewModel {
            viewModel.helloText.bind({ (helloText) in
                DispatchQueue.main.async {
                    // 当 ViewModel 中 helloText 动态变量的值被设置或更改时
                    , // 这段代码将被执行

                    self.helloLabel.text = helloText
                }
            })
        }
    }

    @IBAction func sayHelloButtonPressed(_ sender: UIButton) {
        viewModel?.userTriggeredSayHelloButton()
    }
}
```

更新的 MVVMExampleViewModel：

```
    class MVVMExampleViewModel {

    // 我们必须用想要的数据类型初始化 Dynamic 变量

    var helloText = Dynamic("")

    func userTriggeredSayHelloButton() {
         // 设置 Dynamic 变量的值
         // 将触发我们在视图中定义的闭包
        helloText.value = "Hello"
    }
}
```

就是这样。你的ViewModel现在能够更新View，而无需持有对View的引用。

这是一个非常简单的示例，但我想你已经了解了它的强大之处。我不会详细讲解MVVM的好处，但一旦你从MVC切换到MVVM，就不会再回头。试试看，你会明白的。


```
class MVVMExampleViewController: UIViewController {

    @IBOutlet weak var helloLabel: UILabel!

    var viewModel: MVVMExampleViewModel?

    override func viewDidLoad() {
        super.viewDidLoad()
        bindViewModel()
    }

    func bindViewModel() {
        if let viewModel = viewModel {
            viewModel.helloText.bind({ (helloText) in
                DispatchQueue.main.async {
                    // When value of the helloText Dynamic variable
                    // is set or changed in the ViewModel, this code will
                    // be executed
                    self.helloLabel.text = helloText
                }
            })
        }
    }

    @IBAction func sayHelloButtonPressed(_ sender: UIButton) {
        viewModel?.userTriggeredSayHelloButton()
    }
}
```

Updated `MVVMExampleViewModel`:

```
    class MVVMExampleViewModel {

    // we have to initialize the Dynamic var with the
    // data type we want
    var helloText = Dynamic("")

    func userTriggeredSayHelloButton() {
        // Setting the value of the Dynamic variable
        // will trigger the closure we defined in the View
        helloText.value = "Hello"
    }
}
```

That is it. Your `ViewModel` is now able to update `View` without it having a reference to the `View`.

This is a really simple example, but I think you have an idea how powerful this can be. I won't go into details about benefits of the MVVM, but once you switch from MVC to MVVM, you won't go back. Just try it and see for yourself.

第183章：缓存在线图片

第183.1节：AlamofireImage

使用AlamofireImage缓存在线图片。它基于Swift中的Alamofire。通过 cocoapods安装AlamofireImage

```
pod 'AlamofireImage', '~> 3.1'
```

设置：

1. 导入AlamofireImage和Alamofire
2. 设置图像缓存：let imageCache = AutoPurgingImageCache(memoryCapacity: 111_111_111, preferredMemoryUsageAfterPurge: 90_000_000)
3. 发起请求并将图像添加到缓存：

```
Alamofire.request(self.nameUrl[i]).responseImage { response in
                    if response.result.value != nil {
                        let image = UIImage(data: response.data!, scale: 1.0)!
imageCache.add(image, withIdentifier: self.nameUrl[i])
                    }
    }
```

4. 从缓存中获取图像：

```
if let image = imageCache.image(withIdentifier: self.nameUrl[self.a])
        {
            self.localImageView.image = image
        }
```

更多信息请访问此链接

Chapter 183: Cache online images

Section 183.1: AlamofireImage

Caching Online Images Using `AlamofireImage`. It works on top of `Alamofire` in Swift. Install `AlamofireImage` using cocoapods

```
pod 'AlamofireImage', '~> 3.1'
```

SetUp:

1. Import `AlamofireImage` and `Alamofire`
2. SetUp the Image cache: `let imageCache = AutoPurgingImageCache( memoryCapacity: 111_111_111, preferredMemoryUsageAfterPurge: 90_000_000)`
3. Making a request and adding the Image to Cache:

```
Alamofire.request(self.nameUrl[i]).responseImage { response in
                    if response.result.value != nil {
                        let image = UIImage(data: response.data!, scale: 1.0)!
                        imageCache.add(image, withIdentifier: self.nameUrl[i])
                    }
    }
```

4. Retrieve Images From Cache:

```
if let image = imageCache.image(withIdentifier: self.nameUrl[self.a])
        {
            self.localImageView.image = image
        }
```

For more Info follow this link

第184章：队列中的链式块（使用MKBlockQueue）

MKBlockQueue允许你创建一个块链，并按顺序一个接一个地执行它们。与NSOperation相比，使用MKBlockQueue时，你可以自己决定一个块何时完成以及何时让队列继续。你还可以将数据从一个块传递到下一个块。

https://github.com/MKGitHub/MKBlockQueue

第184.1节：示例代码

```
// 创建将传递给块的字典
var myDictionary:Dictionary<String, Any> = Dictionary<String, Any>()
myDictionary["InitialKey"] = "InitialValue"

// 创建块队列
let myBlockQueue:MKBlockQueue = MKBlockQueue()

// 块1
let b1:MKBlockQueueBlockType =
{
    (blockQueueObserver:MKBlockQueueObserver, dictionary:inout Dictionary<String, Any>) in

    print("块1开始，字典内容：\(dictionary)")
dictionary["Block1Key"] = "Block1Value"

    // 告诉该块已完成
   blockQueueObserver.blockCompleted(with:&dictionary)

}

// 块 2
let b2:MKBlockQueueBlockType =
{
    (blockQueueObserver:MKBlockQueueObserver, dictionary:inout Dictionary<String, Any>) in

    var copyOfDictionary:Dictionary<String, Any> = dictionary

    // 测试在主线程异步调用，带延迟
   DispatchQueue.main.asyncAfter(deadline:(.now() + .seconds(1)), execute:
    {
        print("块 2 开始，字典内容：\(copyOfDictionary)")

copyOfDictionary["Block2Key"] = "Block2Value"

        // 告诉该块已完成
      blockQueueObserver.blockCompleted(with:&copyOfDictionary)
    })
}
```


Chapter 184: Chain Blocks in a Queue (with MKBlockQueue)

MKBlockQueue allows you to create a chain of blocks and execute them one after the other in a queue. Compared with NSOperation, with MKBlockQueue you decide yourself when a block is complete and when you want the queue to continue. You can also pass data from one block to the next.

https://github.com/MKGitHub/MKBlockQueue

Section 184.1: Example Code

```
// create the dictionary that will be sent to the blocks
var myDictionary:Dictionary<String, Any> = Dictionary<String, Any>()
myDictionary["InitialKey"] = "InitialValue"

// create block queue
let myBlockQueue:MKBlockQueue = MKBlockQueue()

// block 1
let b1:MKBlockQueueBlockType =
{
    (blockQueueObserver:MKBlockQueueObserver, dictionary:inout Dictionary<String, Any>) in

    print("Block 1 started with dictionary: \(dictionary)")
    dictionary["Block1Key"] = "Block1Value"

    // tell this block is now completed
    blockQueueObserver.blockCompleted(with:&dictionary)

}

// block 2
let b2:MKBlockQueueBlockType =
{
    (blockQueueObserver:MKBlockQueueObserver, dictionary:inout Dictionary<String, Any>) in

    var copyOfDictionary:Dictionary<String, Any> = dictionary

    // test calling on main thread, async, with delay
    DispatchQueue.main.asyncAfter(deadline:(.now() + .seconds(1)), execute:
    {
        print("Block 2 started with dictionary: \(copyOfDictionary)")

        copyOfDictionary["Block2Key"] = "Block2Value"

        // tell this block is now completed
        blockQueueObserver.blockCompleted(with:&copyOfDictionary)
    })
}
```

```
// 块 3
let b3:MKBlockQueueBlockType =
{
    (blockQueueObserver:MKBlockQueueObserver, dictionary:inout Dictionary<String, Any>) in

    var copyOfDictionary:Dictionary<String, Any> = dictionary

    // 测试在全局后台队列上异步调用，带延迟
    DispatchQueue.global(qos:.background).asyncAfter(deadline:(.now() + .seconds(1)), execute:
    {
        print("Block 3 started with dictionary: \(copyOfDictionary)")

copyOfDictionary["Block3Key"] = "Block3Value"

        // 告诉该块已完成
       blockQueueObserver.blockCompleted(with:&copyOfDictionary)
    })
}

// 向队列添加块
myBlockQueue.addBlock(b1)
myBlockQueue.addBlock(b2)
myBlockQueue.addBlock(b3)

// 为队列添加完成块
myBlockQueue.queueCompletedBlock(
{
    (dictionary:Dictionary<String, Any>) in
    print("Queue completed with dictionary: \(dictionary)")
})

// 运行队列
print("Queue starting with dictionary: \(myDictionary)")
myBlockQueue.run(with:&myDictionary)
```



```
// block 3
let b3:MKBlockQueueBlockType =
{
    (blockQueueObserver:MKBlockQueueObserver, dictionary:inout Dictionary<String, Any>) in

    var copyOfDictionary:Dictionary<String, Any> = dictionary

    // test calling on global background queue, async, with delay
    DispatchQueue.global(qos:.background).asyncAfter(deadline:(.now() + .seconds(1)), execute:
    {
        print("Block 3 started with dictionary: \(copyOfDictionary)")

        copyOfDictionary["Block3Key"] = "Block3Value"

        // tell this block is now completed
        blockQueueObserver.blockCompleted(with:&copyOfDictionary)
    })
}

// add blocks to the queue
myBlockQueue.addBlock(b1)
myBlockQueue.addBlock(b2)
myBlockQueue.addBlock(b3)

// add queue completion block for the queue
myBlockQueue.queueCompletedBlock(
{
    (dictionary:Dictionary<String, Any>) in
    print("Queue completed with dictionary: \(dictionary)")
})

// run queue
print("Queue starting with dictionary: \(myDictionary)")
myBlockQueue.run(with:&myDictionary)
```

第185章：模拟器

iOS、watchOS 和 tvOS 模拟器是测试应用程序而无需使用真实设备的绝佳方式。这里我们将讨论如何使用模拟器。

第185.1节：启动模拟器

你应该进入Xcode -> 打开开发者工具：

> 使用“Simulator”，你可以运行iOS和tvOS，但“Simulator (Watch)”只能运行watchOS。

第185.2节：3D / 力触摸模拟

进入硬件 -> 触摸压力：

Chapter 185: Simulator

iOS, watchOS and tvOS Simulators are great ways to test your apps without using an actual device. Here we will talk about working with Simulators.

Section 185.1: Launching Simulator

You should go to Xcode -> Open Developer Tool:

> Using "Simulator", you could run iOS and tvOS, but "Simulator (Watch)" will only run watchOS.

Section 185.2: 3D / Force Touch simulation

Go to Hardware -> Touch Pressure:

模拟普通按压，使用Shift-Command-1，模拟深度按压，使用Shift-Command-2。

> 如果你的MacBook配备了Force Touch触控板，你可以使用触控板的压力来模拟3D / 力触摸。

第185.3节：更改设备型号

进入硬件 -> 设备：

To simulate a normal press, use Shift-Command-1 and a deep one, Shift-Command-2.

> If your MacBook has a Force Touch trackpad, you could use your trackpad force to simulate 3D / Force Touch.

Section 185.3: Change Device Model

Go to Hardware -> Device:

第185.4节：导航模拟器

主页按钮

关闭应用时应使用Shift-Command-H以显示动画效果。

锁定

要锁定设备，请使用Command-L。

旋转

使用Command键配合方向键。

Section 185.4: Navigating Simulator

Home button

You should use Shift-Command-H to have an animation when closing an app.

Lock

To lock the device, use Command-L.

Rotation

Use Command with arrow keys.

第186章：后台模式

响应迅速是每个应用程序的需求。用户希望在打开应用时内容已经准备好，因此开发者应使用后台模式使他们的应用更加用户友好。

第186.1节：开启后台模式功能

1. 打开Xcode并打开你的项目。
2. 在你的应用目标中，导航到“功能”标签页。
3. 开启后台模式。

第186.2节：后台获取

后台获取是一种新模式，允许你的应用始终保持最新信息，同时最大限度减少对电池的影响。你可以利用此功能在固定时间间隔内下载数据。

开始操作：

1- 在Xcode的功能界面中勾选“后台获取”。

2- 在AppDelegate中的application(_:didFinishLaunchingWithOptions:)方法中，添加：

Swift

```
UIApplication.shared.setMinimumBackgroundFetchInterval(UIApplicationBackgroundFetchIntervalMinimum)
```

Objective-C

```
[[UIApplication shared]
setMinimumBackgroundFetchInterval:UIApplicationBackgroundFetchIntervalMinimum]
```

> 你可以使用任何CGFloat值来设置
>
> 然而对于，用于取UIApplicationBackgroundFetchIntervalMinimum。

3- 你必须实现application(_:performFetchWithCompletionHandler:)。将其添加到你的AppDelegate中：

Chapter 186: Background Modes

Being responsive is a need for every app. Users want to have apps which have their content ready when they open it, so developers should use Background Modes to make their apps more user friendly.

Section 186.1: Turning on the Background Modes capability

1. Go to Xcode and open your project.
2. In your app target, navigate to Capabilities tab.
3. Turn on Background Modes.

Section 186.2: Background Fetch

Background fetch is a new mode that lets your app appear always up-to-date with the latest information while minimizing the impact on battery. You could download feeds within fixed time intervals with this capability.

To get started:

1- Check Background Fetch in capabilities screen in Xcode.

2- In `application(_:didFinishLaunchingWithOptions:)` method in `AppDelegate`, add:

Swift

```
UIApplication.shared.setMinimumBackgroundFetchInterval(UIApplicationBackgroundFetchIntervalMinimum)
```

Objective-C

```
[[UIApplication shared]
setMinimumBackgroundFetchInterval:UIApplicationBackgroundFetchIntervalMinimum]
```

> Instead of `UIApplicationBackgroundFetchIntervalMinimum`, you could use any `CGFloat` value to set fetch intervals.

3- You must implement `application(_:performFetchWithCompletionHandler:)`. Add that to your `AppDelegate`:

Swift

```
func application(_ application: UIApplication, performFetchWithCompletionHandler completionHandler:
@escaping (UIBackgroundFetchResult) -> Void) {
    // 你的代码写在这里
}
```

第186.3节：测试后台获取

1- 在真机上运行应用并连接Xcode调试器。

2- 从调试菜单中选择模拟后台获取：

Swift

```
func application(_ application: UIApplication, performFetchWithCompletionHandler completionHandler:
@escaping (UIBackgroundFetchResult) -> Void) {
    // your code here
}
```

Section 186.3: Testing background fetch

1- Run the app on a real device and attach it to Xcode debugger.

2- From Debug menu, select **Simulate Background Fetch**:

Debug
Source Control
Window
Help
Pause ^⌘Y
Continue To Current Line ^⌘C
Step Over F6
Step Into F7
Step Out F8
Step Over Instruction ^F6
Step Over Thread ^⇧F6
Step Into Instruction ^F7
Step Into Thread ^⇧F7
Capture GPU Frame
GPU Overrides
Simulate Location
Simulate Background Fetch
Simulate UI Snapshot
iCloud
View Debugging
Deactivate Breakpoints ⌘Y
Breakpoints
Debug Workflow
Attach to Process by PID or Name...
Attach to Process
Detach

Debug
Source Control
Window
Help
Pause ^⌘Y
Continue To Current Line ^⌘C
Step Over F6
Step Into F7
Step Out F8
Step Over Instruction ^F6
Step Over Thread ^⇧F6
Step Into Instruction ^F7
Step Into Thread ^⇧F7
Capture GPU Frame
GPU Overrides
Simulate Location
Simulate Background Fetch
Simulate UI Snapshot
iCloud
View Debugging
Deactivate Breakpoints ⌘Y
Breakpoints
Debug Workflow
Attach to Process by PID or Name...
Attach to Process
Detach

3- 现在 Xcode 会用 SIGSTOP 信号暂停应用。只需点击继续按钮，让应用执行后台获取。

现在你会看到数据已被获取并准备好供你使用。

第186.4节：后台音频

默认情况下，当你在播放音频时，退出应用会停止播放，但你可以通过在 Xcode 的后台功能页面中勾选第一个复选框来防止这种情况发生。

iOS 会自动为你处理这个问题，你无需编写任何代码！

3- Now Xcode will pause the app with SIGSTOP signal. Just tap the continue button to let the app do the background fetch.

Now you will see that data is fetched and ready for you.

Section 186.4: Background Audio

By default, when you are streaming an audio, by exiting the app it will stop, but you can prevent this by turning on the first check box in Background capability page in Xcode.

iOS will automatically handle this for you, and you don't need to write any code!

第187章：OpenGL

OpenGL ES 是 iOS 用于进行3D渲染的图形库。

第187.1节：示例项目

一个示例项目（Git 仓库），可用作进行一些3D渲染的起点。设置OpenGL和着色器的代码相当长且繁琐，因此不适合放在此示例格式中。后续部分可以在单独的示例中展示，详细说明每段代码的具体作用，但现在这里是Xcode项目。

Chapter 187: OpenGL

OpenGL ES is a graphics library that iOS uses to do 3D rendering.

Section 187.1: Example Project

An example project (Git repo) that can be used as a starting point for doing some 3D rendering. The code for setting up OpenGL and the shaders is quite long and tedious so it won't fit well under this example format. Later pieces of it can be shown in separate examples detailing what exactly is going on with each piece of code, but for now here is the Xcode project.

第188章：MVP架构

组件：

- 模型 是负责领域数据的接口（用于在GUI中显示或其他操作）
- 视图 负责表现层（GUI）
- 演示者 是模型和视图之间的“中间人”。它响应用户在视图上执行的操作，从模型中检索数据，并将其格式化以在视图中显示

组件职责：

模型	视图	演讲者
与数据库层通信，呈现数据		对模型执行查询
触发适当的事件	接收事件	格式化模型数据
	非常基础的验证逻辑，发送	格式化数据到视图
		复杂的验证逻辑

MVC和MVP的区别：

- MVC中的视图与控制器紧密耦合，MVP的视图部分由UIViews和UIViewController组成
- MVP视图尽可能简单，几乎不包含逻辑（类似MVVM），MVC视图包含一些业务逻辑并且可以查询模型
- MVP视图处理用户手势并将交互委托给Presenter，MVC中控制器处理手势并命令模型
- MVP模式高度支持单元测试，MVC支持有限
- MVC控制器依赖大量UIKit，而MVP的Presenter没有依赖

优点：

- MVP使UIViewController成为视图组件的一部分，它是简单的、被动的且.....不那么庞大 ;]由于视图简
- 单，大部分业务逻辑被封装，这带来了极佳的可测试性。可以引入模拟对象来测试领域部分。

- 分离的实体更容易理解，职责划分清晰。

缺点

- 你需要编写更多代码。
- 对于没有经验的开发者或尚未使用该模式的人来说是一个障碍。

Chapter 188: MVP Architecture

Components:

- **Model** is an interfaceresponsible for the domain data (to be displayed or otherwise acted upon in the GUI)
- **View** is responsible for the presentation layer (GUI)
- **Presenter** is the "middle-man" between Model and View. It reacts to the user's actions performed on the View, retrieves data from the Model, and formats it for display in the View

Component duties:

Model	View	Presenter
Communicates with DB layer	Renders data	Performs queries to the Model
Raising appropriate events	Receives events	Formats data from Model
	Very basic validation logic	Sends formatted data to the View
		Complex validation logic

Differences between MVC and MVP:

- View in MVC is tightly coupled with the Controller, the View part of the MVP consists of both UIViews and UIViewController
- MVP View is as dumb as possible and contains almost no logic (like in MVVM), MVC View has some business logic and can query the Model
- MVP View handles user gestures and delegates interaction to the Presenter, in MVC the Controller handles gestures and commands Model
- MVP pattern highly supports Unit Testing, MVC has limited support
- MVC Controller has lots of UIKit dependencies, MVP Presenter has none

Pros:

- MVP makes UIViewController a part of the View component it's dumb, passive and...less massive ;]
- Most of the business logic is incapsulated due to the dumb Views, this gives an excellent testability. Mock objects can be introduced to test the domain part.
- Separated entities are easier to keep in head, responsibilities are clearly divided.

Cons

- You will write more code.
- Barrier for unexperienced developers or for those who don't yet work with the pattern.

MVP是一种架构模式，是模型-视图-控制器（Model–View–Controller）的衍生。它由三个独立的组件组成：模型、视图和Presenter。它的设计目的是便于自动化单元测试并改善表现逻辑中的关注点分离。

示例中你会找到一个以MVP模式构建的简单项目。

第188.1节：Dog.swift

```
import Foundation

enum Breed: String {
    case 牛头犬  = "Bulldog"
    case 杜宾犬 = "Doberman"
    case 拉布拉多 = "Labrador"
}

struct Dog {
    let name:  String
    let breed: String
    let age:   Int
}
```

第188.2节：DoggyService.swift

```
import Foundation

typealias Result = ([Dog]) -> Void

class DoggyService {

    func deliverDoggies(_ result: @escaping Result) {

        let firstDoggy  = Dog(name: "阿尔弗雷德", breed: Breed.拉布拉多.rawValue, age: 1)
        let secondDoggy = Dog(name: "维尼",  breed: Breed.杜宾犬.rawValue, age: 5)
        let thirdDoggy  = Dog(name: "幸运",  breed: Breed.拉布拉多.rawValue, age: 3)

        let delay = DispatchTime.now() + Double(Int64(Double(NSEC_PER_SEC)*2)) /
Double(NSEC_PER_SEC)

DispatchQueue.main.asyncAfter(deadline: delay) {
            result([firstDoggy,
secondDoggy,
                    thirdDoggy])
        }
    }
}
```

第188.3节：DoggyPresenter.swift

```
import Foundation

class DoggyPresenter {

    // 标记: - 私有
fileprivate let dogService: DoggyService
    weak fileprivate var dogView: DoggyView?

    init(dogService: DoggyService){
```


MVP is an architectural pattern, a derivation of the Model–View–Controller. It's represented by three distinct components: Model, View and the Presenter. It was engineered to facilitate automated unit testing and improve the separation of concerns in presentation logic.

In examples you'll find a simple project built with MVP pattern in mind.

Section 188.1: Dog.swift

```
import Foundation

enum Breed: String {
    case bulldog  = "Bulldog"
    case doberman = "Doberman"
    case labrador = "Labrador"
}

struct Dog {
    let name:  String
    let breed: String
    let age:   Int
}
```

Section 188.2: DoggyService.swift

```
import Foundation

typealias Result = ([Dog]) -> Void

class DoggyService {

    func deliverDoggies(_ result: @escaping Result) {

        let firstDoggy  = Dog(name: "Alfred", breed: Breed.labrador.rawValue, age: 1)
        let secondDoggy = Dog(name: "Vinny",  breed: Breed.doberman.rawValue, age: 5)
        let thirdDoggy  = Dog(name: "Lucky",  breed: Breed.labrador.rawValue, age: 3)

        let delay = DispatchTime.now() + Double(Int64(Double(NSEC_PER_SEC)*2)) /
Double(NSEC_PER_SEC)

        DispatchQueue.main.asyncAfter(deadline: delay) {
            result([firstDoggy,
                    secondDoggy,
                    thirdDoggy])
        }
    }
}
```

Section 188.3: DoggyPresenter.swift

```
import Foundation

class DoggyPresenter {

    // MARK: - Private
    fileprivate let dogService: DoggyService
    weak fileprivate var dogView: DoggyView?

    init(dogService: DoggyService){
```

```
        self.dogService = dogService
    }

    func attachView(_ attach: Bool, view: DoggyView?) {
        if attach {
dogView = nil
        } else {
            if let view = view { dogView = view }
        }
    }

    func getDogs(){
        self.dogView?.startLoading()

dogService.deliverDoggies { [weak self] doggies in
            self?.dogView?.finishLoading()

        if doggies.count == 0 {
            self?.dogView?.setEmpty()
        } else {
            self?.dogView?.setDoggies(doggies.map {
                return DoggyViewData(name: "\($0.name) \($0.breed)",
                                     age: "\($0.age)")
                })
            }
        }
    }
}

struct DoggyViewData {
    let name: String
    let age:  String
}
```

第188.4节：DoggyView.swift

```
import Foundation

协议 DoggyView: NSObjectProtocol {
    func startLoading()
    func finishLoading()
    func setDoggies(_ doggies: [DoggyViewData])
    func setEmpty()
}
```

第188.5节：DoggyListViewController.swift

```
import UIKit

类 DoggyListViewController: UIViewController, UITableViewDataSource {

    @IBOutlet 弱引用 var emptyView: UIView?
    @IBOutlet 弱引用 var tableView: UITableView?
    @IBOutlet 弱引用 var spinner:   UIActivityIndicatorView?

fileprivate let dogPresenter = DoggyPresenter(dogService: DoggyService())
    fileprivate var dogsToDisplay = [DoggyViewData]()

    override func viewDidLoad() {
        super.viewDidLoad()
```



```
        self.dogService = dogService
    }

    func attachView(_ attach: Bool, view: DoggyView?) {
        if attach {
            dogView = nil
        } else {
            if let view = view { dogView = view }
        }
    }

    func getDogs(){
        self.dogView?.startLoading()

        dogService.deliverDoggies { [weak self] doggies in
            self?.dogView?.finishLoading()

        if doggies.count == 0 {
            self?.dogView?.setEmpty()
        } else {
            self?.dogView?.setDoggies(doggies.map {
                return DoggyViewData(name: "\($0.name) \($0.breed)",
                                     age: "\($0.age)")
                })
            }
        }
    }
}

struct DoggyViewData {
    let name: String
    let age:  String
}
```

Section 188.4: DoggyView.swift

```
import Foundation

protocol DoggyView: NSObjectProtocol {
    func startLoading()
    func finishLoading()
    func setDoggies(_ doggies: [DoggyViewData])
    func setEmpty()
}
```

Section 188.5: DoggyListViewController.swift

```
import UIKit

class DoggyListViewController: UIViewController, UITableViewDataSource {

    @IBOutlet weak var emptyView: UIView?
    @IBOutlet weak var tableView: UITableView?
    @IBOutlet weak var spinner:   UIActivityIndicatorView?

    fileprivate let dogPresenter = DoggyPresenter(dogService: DoggyService())
    fileprivate var dogsToDisplay = [DoggyViewData]()

    override func viewDidLoad() {
        super.viewDidLoad()
```

```
tableView?.dataSource = self
        spinner?.hidesWhenStopped = true
        dogPresenter.attachView(true, view: self)
        dogPresenter.getDogs()
    }

    // 标记: 数据源
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return dogsToDisplay.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell
{
        let cell = UITableViewCell(style: .subtitle, reuseIdentifier: "Cell")
        let userViewData = dogsToDisplay[indexPath.row]
cell.textLabel?.text = userViewData.name
        cell.detailTextLabel?.text = userViewData.age
        return cell
    }
}

extension DoggyListViewController: DoggyView {

    func startLoading() {
        spinner?.startAnimating()
    }

    func finishLoading() {
        spinner?.stopAnimating()
    }

    func setDoggies(_ doggies: [DoggyViewData]) {
       dogsToDisplay = doggies
        tableView?.isHidden = false
        emptyView?.isHidden = true;
        tableView?.reloadData()
    }

    func setEmpty() {
tableView?.isHidden = true
        emptyView?.isHidden = false;
    }
}
```



```
        tableView?.dataSource = self
        spinner?.hidesWhenStopped = true
        dogPresenter.attachView(true, view: self)
        dogPresenter.getDogs()
    }

    // MARK: DataSource
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return dogsToDisplay.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell
{
        let cell = UITableViewCell(style: .subtitle, reuseIdentifier: "Cell")
        let userViewData = dogsToDisplay[indexPath.row]
        cell.textLabel?.text = userViewData.name
        cell.detailTextLabel?.text = userViewData.age
        return cell
    }
}

extension DoggyListViewController: DoggyView {

    func startLoading() {
        spinner?.startAnimating()
    }

    func finishLoading() {
        spinner?.stopAnimating()
    }

    func setDoggies(_ doggies: [DoggyViewData]) {
        dogsToDisplay = doggies
        tableView?.isHidden = false
        emptyView?.isHidden = true;
        tableView?.reloadData()
    }

    func setEmpty() {
        tableView?.isHidden = true
        emptyView?.isHidden = false;
    }
}
```

第189章：使用 CoreBluetooth配置信标

如何读取和写入蓝牙低功耗设备的数据。

第189.1节：显示所有蓝牙低功耗（BLE）的名称

- 在此示例中，我有一个受控房间，启用了单个BLE设备。
- 您的类应继承CBCentralManagerDelegate。
- 实现方法：centralManagerDidUpdateState(_ central: CBCentralManager)。
- 使用全局队列以避免在搜索设备时屏幕冻结。
- 实例化CBCentralManager并等待回调centralManagerDidUpdateState响应。

```
class BLEController: CBCentralManagerDelegate{

var cb_manager: CBCentralManager!
var bles : [CBPeripheral] = []

    override func viewDidLoad() {
        super.viewDidLoad()
cb_manager = CBCentralManager(delegate: self, queue: DispatchQueue.global())
    }

    func centralManagerDidUpdateState(_ central: CBCentralManager) {
        print("更新状态 - \(central)")
    }
}
```

回调 centralManagerDidUpdateState 表示 CoreBluetooth 已准备好，因此您现在可以搜索 BLE 设备。
更新 centralManagerDidUpdateState 代码，在准备好时搜索所有 BLE 设备。

```
func centralManagerDidUpdateState(_ central: CBCentralManager) {
     print("更新状态 - \(central)")
SearchBLE()
}

func SearchBLE(){
cb_manager.scanForPeripherals(withServices: nil, options: nil)
    StopSearchBLE()
}

func StopSearchBLE() {
     let when = DispatchTime.now() + 5 // 将 5 改为所需的秒数
    DispatchQueue.main.asyncAfter(deadline: when) {
          self.cb_manager.stopScan()
     }
}
```

- SearchBLE() 用于搜索 BLE 设备，并在 5 秒后停止搜索
- cb_manager.scanForPeripherals(withServices: nil, options: nil) 会搜索范围内的所有 BLE 设备。
- StopSearchBLE() 会在 5 秒后停止搜索。
- 每发现一个 BLE 设备，都会回调函数 centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral，advertisementData: [String : Any]，rssi RSSI: NSNumber）

```
func centralManager(_ central: CBCentralManager，didDiscover peripheral:
```


Chapter 189: Configure Beacons with CoreBluetooth

Hot to read and write data to a bluetooth low energy device.

Section 189.1: Showing names of all Bluetooth Low Energy (BLE)

- For this example I have a controlled room with a single BLE device enable.
- Your class should extend CBCentralManagerDelegate.
- Implement the method: centralManagerDidUpdateState(_ central: CBCentralManager).
- Use global queue to not freeze the screen while searching for a device.
- Instantiate CBCentralManager and wait for callback centralManagerDidUpdateState response.

```
class BLEController: CBCentralManagerDelegate{

var cb_manager: CBCentralManager!
var bles : [CBPeripheral] = []

    override func viewDidLoad() {
        super.viewDidLoad()
        cb_manager = CBCentralManager(delegate: self, queue: DispatchQueue.global())
    }

    func centralManagerDidUpdateState(_ central: CBCentralManager) {
        print("UPDATE STATE - \(central)")
    }
}
```

Callback to centralManagerDidUpdateState indicates that CoreBluetooth is ready, so you can search for BLE now.
Update centralManagerDidUpdateState code to search for all BLE device when it is ready.

```
func centralManagerDidUpdateState(_ central: CBCentralManager) {
    print("UPDATE STATE - \(central)")
    SearchBLE()
}

func SearchBLE(){
    cb_manager.scanForPeripherals(withServices: nil, options: nil)
    StopSearchBLE()
}

func StopSearchBLE() {
    let when = DispatchTime.now() + 5 // change 5 to desired number of seconds
    DispatchQueue.main.asyncAfter(deadline: when) {
        self.cb_manager.stopScan()
    }
}
```

- SearchBLE() search for BLE devices and stop searching after 5s
- cb_manager.scanForPeripherals(withServices: nil, options: nil) looks for every BLE in range with you.
- StopSearchBLE() will stop the search after 5s.
- Each BLE found will callback func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNumber)

```
func centralManager(_ central: CBCentralManager, didDiscover peripheral:
```

```
CBPeripheral，advertisementData: [String : Any]，rssi RSSI: NSNumber） {
    guard let name = peripheral.name else {
        返回
    }
    print(name)
bles.append(peripheral)
}
```

第189.2节：连接并读取major值

- 我在一个受控房间内，只有一个使用IBEACON协议的minew信标。
- BLEController需要扩展CBPeripheralDelegate
- 我将在搜索停止后使用第一个BLE进行连接。
- 修改方法StopSearchBLE()

```
class BLEController: CBCentralManagerDelegate, CBPeripheralDelegate{
//...
    func StopSearchMiniewBeacon() {
        let when = DispatchTime.now() + 5 // 将2改为所需的秒数
      DispatchQueue.main.asyncAfter(deadline: when) {
            self.cb_manager.stopScan()
            self.cb_manager.connect(bles.first)
        }
    }
/...
}
```

- 在您的BLE设备文档中，您应查找服务UUID和主UUID
  特征

```
var service_uuid =  CBUUID(string: "0000fff0-0000-1000-8000-00805f9b34fb")
var major_uuid =  CBUUID(string: "0000fff2-0000-1000-8000-00805f9b34fb")
func centralManager(_ central: CBCentralManager, didConnect peripheral:
CBPeripheral) {
peripheral.delegate = self
    peripheral.discoverServices([service_uuid])
}

func peripheral(_ peripheral: CBPeripheral, didDiscoverServices error: Error?) {print("服务: \(servic
    e) 错误: \(error)")
peripheral.discoverCharacteristics([major_uuid], for: (peripheral.services?[0])!)
}
```

- 创建变量 'service_uuid' 和 'major_uuid'，如上面的代码。'-0000-1000-8000-00805f9b34fb' 是标准的一部分。'fff0' 是我的服务 UUID，'fff2' 是我的主 UUID 特征，'0000' 是填充4字节 UUID 所需的部分。

- 调用 discoverCharacteristics([major_uuid], for: (peripheral.services?[0])!) 会从我的设备 GATT 服务器获取主特征，目前其值为 NIL。
- (peripheral.services?[0])! - 这里是 0，因为在调用 peripheral.discoverS
  ervices([service_uuid]) 后会返回单个值。

```
func peripheral(_ peripheral: CBPeripheral, didDiscoverCharacteristicsFor service: CBService,
error: Error?) {
    for characteristic in service.characteristics! {print("特征: \(c
        haracteristic) 错误: \(error)")if(characteristic.uuid.uuidString == "FFF2"){

peripheral.readValue(for: characteristic)
        }
```



```
    CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNumber) {
    guard let name = peripheral.name else {
        return
    }
    print(name)
    bles.append(peripheral)
}
```

Section 189.2: Connect and read major value

- I'm in a controlled room with a single minew beacon that use IBEACON protocol.
- BLEController needs to extend CBPeripheralDelegate
- I'll use the first BLE to connect after the search has stop.
- Modify the method StopSearchBLE()

```
class BLEController: CBCentralManagerDelegate, CBPeripheralDelegate{
//...
    func StopSearchMiniewBeacon() {
        let when = DispatchTime.now() + 5 // change 2 to desired number of seconds
        DispatchQueue.main.asyncAfter(deadline: when) {
            self.cb_manager.stopScan()
            self.cb_manager.connect(bles.first)
        }
    }
/...
}
```

- In the documention of your BLE device, you should look for the SERVICE UUID and MAJOR UUID
  CHARACTERISTIC

```
var service_uuid =  CBUUID(string: "0000fff0-0000-1000-8000-00805f9b34fb")
var major_uuid =  CBUUID(string: "0000fff2-0000-1000-8000-00805f9b34fb")
func centralManager(_ central: CBCentralManager, didConnect peripheral:
CBPeripheral) {
    peripheral.delegate = self
    peripheral.discoverServices([service_uuid])
}

func peripheral(_ peripheral: CBPeripheral, didDiscoverServices error: Error?) {
    print("Service: \(service)\n error: \(error)")
    peripheral.discoverCharacteristics([major_uuid], for: (peripheral.services?[0])!)
}
```

- Create a variable 'service_uuid' and 'major_uuid' like code above. '-0000-1000-8000-00805f9b34fb' is part of the standard. 'fff0' is my SERVICE UUID, 'fff2' is my MAJOR UUID characteristic and '0000' are required to fill the 4 bytes uuid 1º block.
- discoverCharacteristics([major_uuid], for: (peripheral.services?[0])!) will get major characteristic from my device gatt server and it will have NIL as value for now.
- (peripheral.services?[0])! - 0 beacuse will return a single value once I did
  peripheral.discoverServices([service_uuid])

```
func peripheral(_ peripheral: CBPeripheral, didDiscoverCharacteristicsFor service: CBService,
error: Error?) {
    for characteristic in service.characteristics! {
        print("Characteristic: \(characteristic)\n error: \(error)")
        if(characteristic.uuid.uuidString == "FFF2"){
            peripheral.readValue(for: characteristic)
        }
```

```
    }
}

func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic,
error: Error?) {
    print("读取特征: \(characteristic) 错误: \(error)")let major = UInt16.init(bigEndian:
     UInt16(data: characteristic.value!)!)print("major: \(major)")

}
```

- 特征值只有在调用 peripheral.readValue(for: characteristic) 后才能读取
- readValue 的结果会触发 func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic，错误：带有数据类型值的错误（Error?）。

第189.3节：写入主值

- 您需要发现服务和特征
- 写入特征之前无需读取其值。
- 将继续，例如，在读取值之后。修改函数 peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic, error: Error?)
- 添加变量 new_major 和 reset_characteristic

```
var reset_characteristic : CBCharacteristic!
func peripheral(_ peripheral: CBPeripheral, didDiscoverCharacteristicsFor service: CBService,
error: Error?) {
    for characteristic in service.characteristics! {print("特征: \(c
        haracteristic) 错误: \(error)")if(characteristic.uuid.uuidString == "FFF2"){

peripheral.readValue(for: characteristic)
        }
        if(characteristic.uuid.uuidString == "FFFF"){
          reset_characteristic = characteristic
        }
    }
}
let new_major : UInt16 = 100
func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic,
error: Error?) {
    print("读取特征: \(characteristic) 错误: \(error)")let major = UInt16.init(bigEndian:
     UInt16(data: characteristic.value!)!)print("major: \(major)")

peripheral.writeValue(new_major.data, for: characteristic, type:
CBCharacteristicWriteType.withResponse)
}
```

- iPhone 默认以小端格式发送和接收字节，但我的设备 MINEW 使用的芯片组 NRF51822 采用 ARM 架构，需要大端格式的字节，因此我必须进行字节交换。
- BLE 设备文档会说明每个特征的输入和输出类型，以及是否可以像上面那样读取（CBCharacteristicWriteType.withResponse）。

```
func peripheral(_ peripheral: CBPeripheral, didWriteValueFor characteristic: CBCharacteristic,
error: Error?) {
    print("特征写入: \(characteristic) 错误: \(error)")if(characteristic.uuid.uuidString ==
    "FFF2"){
            print("重置中")
peripheral.writeValue("minew123".data(using: String.Encoding.utf8)!, for:
reset_characteristic, type: CBCharacteristicWriteType.withResponse)
        }
    if(characteristic.uuid.uuidString == "FFFF"){
```



```
    }
}

func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic,
error: Error?) {
    print("Characteristic read: \(characteristic)\n error: \(error)")
    let major = UInt16.init(bigEndian: UInt16(data: characteristic.value!)!)
    print("major: \(major)")
}
```

- Characteristic value will only be readable after call peripheral.readValue(for: characteristic)
- readValue will result in func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic, error: Error?) with value in Data type.

Section 189.3: Write major value

- You need discover the services and characteristic
- You don't need read value from the characteristic before writing over it.
- will continue for, for this example, after read value. Modify func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic, error: Error?)
- Add a variable new_major and reset_characteristic

```
var reset_characteristic : CBCharacteristic!
func peripheral(_ peripheral: CBPeripheral, didDiscoverCharacteristicsFor service: CBService,
error: Error?) {
    for characteristic in service.characteristics! {
        print("Characteristic: \(characteristic)\n error: \(error)")
        if(characteristic.uuid.uuidString == "FFF2"){
            peripheral.readValue(for: characteristic)
        }
        if(characteristic.uuid.uuidString == "FFFF"){
            reset_characteristic = characteristic
        }
    }
}
let new_major : UInt16 = 100
func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic,
error: Error?) {
    print("Characteristic read: \(characteristic)\n error: \(error)")
    let major = UInt16.init(bigEndian: UInt16(data: characteristic.value!)!)
    print("major: \(major)")
    peripheral.writeValue(new_major.data, for: characteristic, type:
CBCharacteristicWriteType.withResponse)
}
```

- iPhone by default will send and receive bytes in Little Endian format, but my device MINEW witch chipset NRF51822 have ARM architecture and need bytes in Big Endian format, so I have to swap it.
- BLE Device documentation will say what type of input and output each characteristic will have and if you can read it like above (CBCharacteristicWriteType.withResponse).

```
func peripheral(_ peripheral: CBPeripheral, didWriteValueFor characteristic: CBCharacteristic,
error: Error?) {
    print("Characteristic write: \(characteristic)\n error: \(error)")
    if(characteristic.uuid.uuidString == "FFF2"){
            print("Resetting")
            peripheral.writeValue("minew123".data(using: String.Encoding.utf8)!, for:
reset_characteristic, type: CBCharacteristicWriteType.withResponse)
        }
    if(characteristic.uuid.uuidString == "FFFF"){
```

```
        print("重启完成")
cb_manager.cancelPeripheralConnection(peripheral)
    }
}
```

- 要更新gatt服务器信息，您必须通过编程方式重启它，或者保存数据到服务器后手动关闭并重新开启。

- FFFF 是该设备的特征。
- “minew123” 是在此情况下重启以保存信息的默认密码。
- 运行你的应用并查看控制台是否有错误，希望没有，但你还看不到新值。

```
func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic,
error: Error?) {
    print("读取特征: \(characteristic) 错误: \(error)")let major = UInt16.init(bigEndian:
     UInt16(data: characteristic.value!)!)print("major: \(major)")

    //peripheral.writeValue(new_major.data, for: characteristic, type:
CBCharacteristicWriteType.withResponse)
```

}

- 最后一步是在 didUpdateValueFor 方法中注释掉最后一行代码并重新运行应用，现在你将看到新值。


```
        print("Reboot finish")
        cb_manager.cancelPeripheralConnection(peripheral)
    }
}
```

- To update a gatt server information you have to reboot it programmatically or save data to it and turn off and turn on manually.
- FFFF is characteristic that do it in this device.
- 'minew123' is the default password for reboot o save information in this case.
- run your app and watch you console for any error, I hope none, but you will not see the new value yet.

```
func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic,
error: Error?) {
    print("Characteristic read: \(characteristic)\n error: \(error)")
    let major = UInt16.init(bigEndian: UInt16(data: characteristic.value!)!)
    print("major: \(major)")
    //peripheral.writeValue(new_major.data, for: characteristic, type:
CBCharacteristicWriteType.withResponse)
```

}

- Last step is to comment last line in method didUpdateValueFor and rerun the app, now you will the new value.

第190章：核心数据

核心数据是你应用程序的模型层，涵盖了最广泛的意义。它是贯穿 iOS SDK 的模型-视图-控制器模式中的模型部分。

核心数据不是你的应用程序的数据库，也不是用于将数据持久化到数据库的 API。核心数据是一个管理对象图的框架。就是这么简单。核心数据可以通过写入磁盘来持久化该对象图，但这不是该框架的主要目标。

第190.1节：核心数据操作

获取上下文：

```
NSManagedObjectContext *context = ((AppDelegate*)[[UIApplication sharedApplication]
delegate]).persistentContainer.viewContext;
```

获取数据：

```
NSFetchRequest<EntityName *> *fetchRequest = [EntityName fetchRequest];
NSError *error ;
NSArray *resultArray= [context executeFetchRequest:fetchRequest error:&error];
```

带排序获取数据：

```
NSFetchRequest<EntityName *> *fetchRequest = [EntityName fetchRequest];
NSSortDescriptor *sortDescriptor = [NSSortDescriptor sortDescriptorWithKey:@"someKey"
ascending:YES];
fetchRequest.sortDescriptors = @[sortDescriptor];
NSError *error ;
NSArray *resultArray= [context executeFetchRequest:fetchRequest error:&error];
```

添加数据：

```
NSManagedObject *entityNameObj = [NSEntityDescription insertNewObjectForEntityForName:@"EntityName"
inManagedObjectContext:context];
[entityNameObj setValue:@"someValue" forKey:@"someKey"];
```

保存上下文：

```
[((AppDelegate*)[[UIApplication sharedApplication] delegate]) saveContext];
```


Chapter 190: Core Data

Core Data is the model layer of your application in the broadest sense possible. It's the Model in the Model-View-Controller pattern that permeates the iOS SDK.

Core Data isn't the database of your application nor is it an API for persisting data to a database. Core Data is a framework that manages an object graph. It's as simple as that. Core Data can persist that object graph by writing it to disk, but that is not the primary goal of the framework.

Section 190.1: Operations on core data

To Get context:

```
NSManagedObjectContext *context = ((AppDelegate*)[[UIApplication sharedApplication]
delegate]).persistentContainer.viewContext;
```

To fetch data:

```
NSFetchRequest<EntityName *> *fetchRequest = [EntityName fetchRequest];
NSError *error ;
NSArray *resultArray= [context executeFetchRequest:fetchRequest error:&error];
```

To fetch data with sorting:

```
NSFetchRequest<EntityName *> *fetchRequest = [EntityName fetchRequest];
NSSortDescriptor *sortDescriptor = [NSSortDescriptor sortDescriptorWithKey:@"someKey"
ascending:YES];
fetchRequest.sortDescriptors = @[sortDescriptor];
NSError *error ;
NSArray *resultArray= [context executeFetchRequest:fetchRequest error:&error];
```

To add data:

```
NSManagedObject *entityNameObj = [NSEntityDescription insertNewObjectForEntityForName:@"EntityName"
inManagedObjectContext:context];
[entityNameObj setValue:@"someValue" forKey:@"someKey"];
```

To save context:

```
[((AppDelegate*)[[UIApplication sharedApplication] delegate]) saveContext];
```

第191章：使用Instruments进行性能分析

Xcode包含一个名为Instruments的性能调优应用程序，您可以使用它通过各种不同的指标来分析您的应用程序性能。它提供了检查CPU使用率、内存使用、内存泄漏、文件/网络活动以及能耗等工具，仅举几例。从Xcode启动应用性能分析非常简单，但有时理解分析结果并不容易，这使得一些开发者无法充分利用该工具的全部潜力。

第191.1节：时间分析器

您将首先查看的工具是时间分析器。Instruments会在测量间隔内暂停程序执行，并对每个运行线程进行堆栈跟踪。可以把它看作是在Xcode调试器中按下暂停按钮。以下是时间分析器的预览：

此界面显示调用树。调用树展示了应用程序中各个方法执行所花费的时间。每一行代表程序执行路径中经过的不同方法。通过分析分析器停止在每个方法的次数，可以确定在每个方法中花费的时间。例如，如果

100 采样间隔为1毫秒，且某个特定方法被发现位于堆栈顶部时
10个样本，那么你可以推断出大约10%的总执行时间—10毫秒—花费在该方法中。这是一个相当粗略的估算，但它有效！

从Xcode的菜单栏中，选择Product\Profile，或按下⌘I。这将构建应用并启动Instruments。你将看到一个如下所示的选择窗口：

Chapter 191: Profile with Instruments

Xcode includes a performance tuning application named Instruments that you can use to profile your application using all sorts of different metrics. They have tools to inspect CPU usage, memory usage, leaks, file/network activity, and energy usage, just to name a few. It's really easy to start profiling your app from Xcode, but it's sometimes not as easy to understand what you see when it's profiling, which deters some developers from being able to use this tool to its fullest potential.

Section 191.1: Time Profiler

The first instrument you'll look at is the `Time Profiler`. At measured intervals, Instruments will halt the execution of the program and take a stack trace on each running thread. Think of it as pressing the pause button in Xcode's debugger.Here's a sneak preview of the Time Profiler :-

This screen displays the `Call Tree`. The `Call Tree` shows the amount of time spent executing in various methods within an app. Each row is a different method that the program's execution path has followed. The time spent in each method can be determined from the number of times the profiler is stopped in each method. For instance, if **100** samples are done at **1 millisecond intervals**, and a particular method is found to be at the top of the stack in 10 samples, then you can deduce that approximately **10%** of the total execution time — **10 milliseconds** — was spent in that method. It's a fairly crude approximation, but it works!

From `Xcode`'s menu bar, select `Product\Profile`, or `press ⌘I`. This will build the app and launch Instruments. You will be greeted with a selection window that looks like this:

这些都是Instruments自带的不同模板。

选择Time Profiler工具并点击选择。这将打开一个新的Instruments文档。点击左上角的红色录制按钮开始录制并启动应用。你可能会被要求输入密码以授权Instruments分析其他进程—不用担心，这里提供密码是安全的！在Instruments窗口中，你可以看到时间在计数，屏幕中央图表上方有一个小箭头从左向右移动。这表示应用正在运行。

现在，开始使用应用。搜索一些图片，并深入查看一个或多个搜索结果。你可能已经注意到，进入搜索结果非常缓慢，滚动搜索结果列表也非常令人烦躁–这是一个非常笨拙的应用！

好消息是，你很幸运，因为你即将开始修复它！不过，首先你将快速了解一下你在Instruments中看到的内容。首先，确保工具栏右侧的视图选择器中两个选项都被选中，如下所示：

这将确保所有面板都打开。现在请查看下面的截图及其下方对各部分的说明：

These are all different templates that come with Instruments.

Select the `Time Profiler` instrument and click Choose. This will open up a new Instruments document. Click the red **record button** in the top left to start recording and launch the app. You may be asked for your password to authorize **Instruments** to analyze other processes — fear not, it's safe to provide here! In the **Instruments window**, you can see the time counting up, and a little arrow moving from left to right above the **graph** in the center of the screen. This indicates that the app is running.

Now, start using the app. Search for some images, and drill down into one or more of the search results. You have probably noticed that going into a search result is tediously slow, and scrolling through a list of search results is also incredibly annoying – it's a terribly clunky app!

Well, you're in luck, for you're about to embark on fixing it! However, you're first going to get a quick run down on what you're looking at in **Instruments**. First, make sure the view selector on the right hand side of the toolbar has both options selected, like so:

That will ensure that all panels are open. Now study the screenshot below and the explanation of each section beneath it:

1. 这些是录制控制。红色的‘录制’按钮将停止和启动当前被分析的应用，当它被点击时（在录制图标和停止图标之间切换）。暂停按钮的功能正如你所期望的那样，暂停应用程序的当前执行。

2. 这是运行计时器。计时器会统计被分析的应用程序运行了多长时间，以及运行了多少次。如果你使用录制控制停止然后重新启动应用程序，那将开始一个新的运行，显示屏将显示“运行 2 的 2”。

3. 这称为轨迹。在你选择的时间分析器模板中，只有一个工具，因此只有一个轨迹。你将在教程后面了解这里显示的图表的具体内容。

4. 这是详细面板。它显示你正在使用的特定工具的主要信息。在本例中，它显示的是“最热”的方法——也就是使用了最多 CPU 时间的方法。如果你点击顶部写着“调用树”（左边那个）的栏，并选择“样本列表”，你将看到数据的另一种视图。该视图显示每一个样本。点击几个样本，你将在扩展详细信息检查器中看到捕获的堆栈跟踪。

5. 这是检查器面板。有三个检查器：录制设置、显示设置和扩展详细信息。你很快就会学习到这些选项中的一些内容。

> 深入挖掘

执行图像搜索，并深入查看结果。我个人喜欢搜索“狗”，但你可以选择任何你喜欢的——你可能是那些喜欢猫的人之一！

1. These are the **recording controls**. The red ‘record’ button will stop & start the app currently being profiled when it is clicked (it toggles between a record and stop icon). The pause button does exactly what you’d expect and pauses the current execution of the app.

2. This is the run timer. The timer counts how long the app being profiled has been running, and how many times it has been run. If you stop and then restart the app using the recording controls, that would start a new run and the display would then show Run 2 of 2.

3. This is called a track. In the case of the Time Profiler template you selected, there’s just one instrument so there’s just one track. You’ll learn more about the specifics of the graph shown here later in the tutorial.

4. This is the detail panel. It shows the main information about the particular instrument you’re using. In this case, it’s showing the methods which are “hottest” — that is, the ones that have used up the most CPU time. If you click on the bar at the top which says Call Tree (the left hand one) and select Sample List, then you are presented with a different view of the data. This view is showing every single sample. Click on a few samples, and you’ll see the captured stack trace appear in the Extended Detail inspector.

5. This is the inspectors panel. There are three inspectors: Record Settings, Display Settings, and Extended Detail. You’ll be learning more about some of these options shortly.

> Drilling Deep

Perform an image search, and drill into the results. I personally like searching for “dog”, but choose whatever you wish – you might be one of those cat people!

现在，上下滚动列表几次，这样你就在时间分析器中获得了足够的数据。你应该会注意到屏幕中间的数字在变化，**图表**也在填充；这表明**CPU周期**正在被使用。

你真的不会期望任何用户界面像这样笨拙，任何表格视图在滚动时都应该像黄油一样顺滑，才能准备好发布！为了帮助定位问题，你需要设置一些选项。

在右侧，选择显示设置检查器（或按⌘+2）。在检查器中，调用树部分，选择按线程分开，反转调用树，隐藏缺失符号和隐藏系统库。界面将显示如下：

以下是每个选项对左侧表格中显示数据的作用：

按线程分开：每个线程应单独考虑。这使你能够了解哪些线程负责最多的**CPU**使用。

反转调用树：启用此选项后，堆栈跟踪将从上到下考虑。这通常是你想要的，因为你想看到**CPU**花费时间最深的方法。

隐藏缺失符号：如果你的应用或系统框架找不到dSYM文件，那么表格中不会显示方法名（符号），而是显示对应于二进制内部地址的十六进制值。如果选择此选项，则只显示完全解析的符号，未解析的十六进制值将被隐藏。这有助于简化呈现的数据。

隐藏系统库：选择此选项时，只显示你自己应用的符号。通常选择此选项很有用，因为你通常只关心CPU在你自己代码中花费的时间–你无法控制系统库使用了多少CPU！

扁平化递归：此选项将递归函数（即调用自身的函数）视为每个堆栈跟踪中的一个条目，而不是多个。

主要功能： 启用此功能后，仪器 会将函数中花费的总时间视为该函数内直接花费的时间与该函数调用的其他函数中花费时间的总和。

所以如果函数A调用了B，那么A的时间报告为在A中花费的时间加上在B中花费的时间。这非常有用，因为它让你每次深入调用栈时都能选出最大的时间数字，从而聚焦于你最

Now, scroll up and down the list a few times so that you've got a good amount of data in the `Time Profiler`. You should notice the numbers in the middle of the screen changing and the **graph** filling in; this tells you that **CPU cycles** are being used.

You really wouldn't expect any UI to be as clunky as this no `table view` is ready to ship until it scrolls like butter! To help pinpoint the problem, you need to set some options.

On the right hand side, select the **Display Settings inspector** (`or press ⌘+2`). In the **inspector**, under the `Call Tree` section, select Separate by **Thread**, Invert `Call Tree`, Hide Missing Symbols and Hide System Libraries. It will look like this:

Here's what each option is doing to the data displayed in the table to the left:

Separate by Thread: Each thread should be considered separately. This enables you to understand which threads are responsible for the greatest amount of **CPU** use.

Invert Call Tree: With this option, the `stack trace` is considered from top to bottom. This is usually what you want, as you want to see the deepest methods where the **CPU** is spending its time.

Hide Missing Symbols: If the `dSYM` file cannot be found for your app or a `system framework`, then instead of seeing method names (symbols) in the table, you'll just see hex values corresponding to addresses inside the binary. If this option is selected, then only fully resolved symbols are displayed and the unresolved **hex** values are hidden. This helps to declutter the data presented.

Hide System Libraries: When this option is selected, only symbols from your own app are displayed. It's often useful to select this option, since usually you only care about where the **CPU** is spending time in your own code – you can't do much about how much **CPU** the `system libraries` are using!

Flatten Recursion: This option treats **recursive** functions (ones which call themselves) as one entry in each `stack trace`, rather than multiple.

Top Functions: Enabling this makes `Instruments` consider the total time spent in a function as the sum of the time directly within that function, as well as the time spent in functions called by that function.

So if function A calls B, then A's time is reported as the time spent in A PLUS the time spent in B. This can be really useful, as it lets you pick the largest time figure each time you descend into the call stack, zeroing in on your most

耗时的方法。

如果你正在运行一个Objective-C应用程序，还有一个“仅显示Obj-C”的选项：如果选中此项，则只显示Objective-C方法，而不显示任何C或C++函数。你的程序中没有这些，但如果你查看一个OpenGL应用程序，例如，它可能会包含一些C++代码。

虽然某些数值可能略有不同，但启用上述选项后，条目的顺序应与下表类似：

嗯，这看起来确实不太好。绝大多数时间都花在了对缩略图应用“调子”滤镜的方法上。这对你来说不应该太意外，因为表格加载和滚动是界面中最笨重的部分，而这正是表格单元格不断被更新的时候。

要了解该方法内部发生了什么，双击表格中的该行。这样会弹出以下视图：

这很有趣，不是吗！applyTonalFilter()是添加到UIImage的一个扩展方法，几乎100%的时间都花在应用图像滤镜后创建CGImage输出上。

这部分其实没什么太多加速空间：创建图像是一个相当密集的过程，耗时就是耗时。我们试着退一步，看看applyTonalFilter()是从哪里调用的。点击代码视图顶部面包屑导航中的“调用树”返回到上一个界面：

现在点击表格顶部applyTonalFilter行左侧的小箭头。这将展开调用树

time-consuming methods.

If you're running an `Objective-C` app, there's also an option of Show **Obj-C Only**: If this is selected, then only `Objective-C` methods are displayed, rather than any `C` or `C++` functions. There are none in your program, but if you were looking at an `OpenGL` app, it might have some `C++`, for example.

Although some values may be slightly different, the order of the entries should be similar to the table below once you have enabled the options above:

Well, that certainly doesn't look too good. The vast majority of time is spent in the method that applies the 'tonal' filter to the thumbnail photos. That shouldn't come as too much of a shock to you, as the table loading and scrolling were the clunkiest parts of the UI, and that's when the table cells are constantly being updated.

To find out more about what's going on within that method, double click its row in the table. Doing so will bring up the following view:

Well that's interesting, isn't it! `applyTonalFilter()` is a method added to `UIImage` in an extension, and almost **100**% of the time spent in it is spent creating the CGImage output after applying the image filter.

There's not really much that can be done to speed this up: creating the image is quite an intensive process, and takes as long as it takes. Let's try stepping back and seeing where `applyTonalFilter()` is called from. **Click** `Call Tree` in the breadcrumb trail at the top of the code view to get back to the previous screen:

Now click the small arrow to the left of the applyTonalFilter row at the top of the table. This will unfold the Call Tree

显示 applyTonalFilter 的调用者。你可能还需要展开下一行；在分析 Swift 时，调用树中有时会出现重复的行，前缀为 @objc。你'感兴趣的是第一个以你的应用'目标名称（InstrumentsTutorial）为前缀的行：

在这种情况下，这一行指的是结果集合视图'的 cellForItemAtIndexPath。双击该行以查看项目中相关的代码。

现在你可以看到问题所在。应用色调滤镜的方法执行时间较长，并且它'是直接从 cellForItemAtIndexPath 调用的，这会阻塞主线程（因此阻塞整个 UI），每次请求滤镜图像时都会如此。

分配

这里有关于所有被创建的对象及其所占用内存的详细信息；它还显示了每个对象的引用计数。要重新开始一个新的instruments 配置，请退出 Instruments 应用。这次，构建并运行应用，然后在导航区域打开调试导航器。接着点击内存，在主窗口显示内存使用图表：

这些图表有助于快速了解你的应用性能。但你需要更强大的功能。点击Instruments中的Profile按钮，然后选择 Transfer 将此会话导入Instruments。Allocations 仪器将自动启动。

to show the caller of applyTonalFilter. You may need to unfold the next row too; when profiling Swift, there will sometimes be duplicate rows in the Call Tree, prefixed with @objc. You're interested in the first row that's prefixed with your app's target name (InstrumentsTutorial):

In this case, this row refers to the results collection view's `cellForItemAtIndexPath`. Double click the row to see the associated code from the project.

Now you can see what the problem is. The method to apply the tonal filter takes a long time to execute, and it's called directly from cellForItemAtIndexPath, which will block the `main thread` (and therefore the entire UI) each time it's ask for a filtered image.

Allocations

There are detailed information about all the **objects** that are being created and the memory that backs them; it also shows you `retain counts` of each object. To start afresh with a new `instruments profile`, quit the Instruments app. This time, build and run the app, and open the Debug Navigator in the Navigators area. Then click on **Memory** to display graphs of memory usage in the main window:

These graphs are useful for to get a quick idea about how your app is performing. But you're going to need a bit more power. Click the `Profile in Instruments` button and then Transfer to bring this session into **Instruments**. The **Allocations instrument** will start up automatically.

这次你会注意到两个轨迹。一个叫做分配（Allocations），另一个叫做泄漏（Leaks）。分配轨迹将在后面详细讨论；泄漏轨迹通常在Objective-C中更有用，本教程不会涉及。所以你接下来要追踪的是什么错误？项目中隐藏着一些你可能不知道的东西。你可能听说过内存泄漏，但你可能不知道实际上有两种泄漏：

真正的内存泄漏是指一个对象不再被任何东西引用，但仍然被分配着——这意味着内存永远无法被重用。即使有Swift和自动引用计数（ARC）帮助管理内存，最常见的内存泄漏类型是保留环（retain cycle）或强引用环（strong reference cycle）。这是指两个对象相互持有强引用，导致彼此无法被释放。这意味着它们的内存永远不会被释放！

无限制的内存增长是指内存持续被分配，但从未有机会被释放。如果这种情况持续下去，系统的内存最终会被占满，你将面临严重的内存问题。在iOS中，这意味着应用程序会被系统强制终止。

在应用程序上运行分配（Allocations）工具时，进行五次不同的搜索，但暂时不要深入查看结果。确保搜索有一些结果！然后等待几秒钟，让应用程序稳定下来。

你应该注意到分配轨迹中的图表一直在上升。这告诉你内存正在被分配。正是这个特性将引导你发现无限制的内存增长。

你将执行的是一次代分析（generation analysis）。为此，按下名为“标记代”（Mark Generation）的按钮。你可以在显示设置检查器的顶部找到该按钮：

This time you'll notice two tracks. One is called Allocations, and one is called Leaks. The Allocations track will be discussed in detail later on; the Leaks track is generally more useful in Objective-C, and won't be covered in this tutorial. So what bug are you going to track down next? There's something hidden in the project that you probably don't know is there. You've likely heard about memory leaks. But what you may not know is that there are actually two kinds of leaks:

True memory leaks are where an object is no longer referenced by anything but still allocated – that means the memory can never be re-used. Even with Swift and `ARC` helping manage memory, the most common kind of memory leak is a `retain cycle or strong reference cycle`. This is when two objects hold strong references to one another, so that each object keeps the other one from being deallocated. This means that their memory is never released!

Unbounded memory growth is where memory continues to be allocated and is never given a chance to be **deallocated**. If this continues forever, then at some point the `system's memory` will be filled and you'll have a big memory problem on your hands. In iOS this means that the app will be killed by the system.

With the Allocations **instrument** running on the app, make five different searches in the app but do not drill down into the results yet. Make sure the searches have some results! Now let the app settle a bit by waiting a few seconds.

You should have noticed that the **graph** in the Allocations track has been rising. This is telling you that memory is being allocated. It's this feature that will guide you to finding `unbounded memory growth`.

What you're going to perform is a `generation analysis`. To do this, press the button called Mark Generation. You'll find the button at the top of the Display Settings inspector:

按下它后，你会看到轨迹中出现一个红旗，如下所示：

代分析的目的是多次执行某个操作，观察内存是否以无限制的方式增长。深入某个搜索，等待几秒钟让图片加载，然后返回主页面。然后再次标记代。对不同的搜索重复此操作。深入几次搜索后，Instruments会显示如下：

此时，你应该开始怀疑了。注意蓝色图表随着每次深入搜索而上升。这显然是不好的。但等等，内存警告（memory warnings）呢？你知道那些，对吧？
内存警告是iOS告诉应用程序内存紧张，需要清理一些内存的方式。

Press it and you will see a red flag appear in the track, like so:

The purpose of `generation analysis` is to perform an action multiple times, and see if memory is growing in an unbounded `fashion`. **Drill** into a search, wait a few seconds for the images to load, and then go back to the main page. Then mark the generation again. Do this repeatedly for different searches. After a **drilling** into a few searches, **Instruments** will look like this:

At this point, you should be getting suspicious. Notice how the blue graph is going up with each search that you **drill** into. Well, that certainly isn't good. But wait, what about `memory warnings`? You know about those, right? `Memory warnings` are iOS's way of telling an app that things are getting tight in the memory department, and you need to clear out some memory.

这种增长可能不仅仅是由于你的应用程序；也可能是UIKit深层次的某些部分占用了内存。在指责系统框架或你的应用程序之前，先给它们一个机会来清理内存。

通过在Instruments菜单栏选择“Instrument\Simulate Memory Warning”，或在模拟器菜单栏选择“Hardware\Simulate Memory Warning”来模拟内存警告。你会注意到内存使用量会稍微下降，或者可能根本没有变化，当然不会回到理想状态。所以某处仍然存在无界的内存增长。

在每次深入搜索迭代后标记一代的原因是，你可以看到每代之间分配了多少内存。查看详细面板，你会看到一堆代。

It's possible that this growth is not just due to your app; it could be something in the depths of `UIKit` that's holding onto memory. Give the system frameworks and your app a chance to clear their **memory** first before pointing a finger at either one.

Simulate a `memory warning` by selecting `Instrument\Simulate Memory Warning` in Instruments' menu bar, or `Hardware\Simulate Memory Warning` from the `simulator`'s menu bar. You'll notice that memory usage dips a little, or perhaps not at all. Certainly not back to where it should be. So there's still **unbounded memory growth** happening somewhere.

The reason for marking a generation after each iteration of drilling into a search is that you can see what **memory** has been allocated between each generation. Take a look in the detail panel and you'll see a bunch of generations.

第192章：应用评分/评论请求

从iOS 10.3开始，不再需要跳转到苹果商店进行评分/评论。苹果在StoreKit框架中引入了SKStoreReviewController类，开发者只需调用SKStoreReviewController类的requestReview()类方法，系统会为你处理整个过程。

此外，你仍然可以在应用的设置或配置界面中包含一个持久链接，深度链接到你的App Store产品页面。以自动打开

第192.1节：评分/评论iOS应用程序

只需在您希望用户对您的应用进行评分/评论的位置输入以下一行代码。

SKStoreReviewController.requestReview()

Chapter 192: Application rating/review request

Now from iOS 10.3 , there is no need to navigate application to apple store for rating/review. Apple has introduced SKStoreReviewController class in storekit framework. In which developer just need to call requestReview() class method of SKStoreReviewController class and the system handles the entire process for you.

In addition, you can continue to include a persistent link in the settings or configuration screens of your app that deep-links to your App Store product page. To automatically ope

Section 192.1: Rate/Review iOS Application

Just type below one line code from where you want user to rate/review your application.

SKStoreReviewController.requestReview()

第193章：MyLayout

MyLayout是一个简单易用的iOS视图布局Objective-C框架。MyLayout提供了一些简单的功能来构建各种复杂界面。它集成了包括iOS的Autolayout和SizeClass、Android的五种布局类、HTML/CSS的float、flex-box和bootstrap等功能。你可以访问：

Objective-C: https://github.com/youngsoft/MyLinearLayout Swift: https://github.com/youngsoft/TangramKit

第193.1节：使用MyLayout的简单示例

1. 有一个容器视图S，宽度为100，高度根据所有子视图高度自适应。里面有四个子视图A、B、C、D，从上到下排列。
2. 子视图A的左边距是S宽度的20%，右边距是S宽度的30%，高度等于A的宽度。
3. 子视图B的左边距是40，宽度填充S的剩余宽度，高度为40。子视图C的宽度填充S，高度为

4. 子视图D的右边距是20，宽度是S宽度的50%，高度为40

如下面图所示：

```
    MyLinearLayout *S = [MyLinearLayout linearLayoutWithOrientation:MyLayoutViewOrientation_Vert];
S.subviewSpace = 10;
    S.widthSize.equalTo(@100);

    UIView *A = UIView.new;
A.leftPos.equalTo(@0.2);
A.rightPos.equalTo(@0.3);
A.heightSize.equalTo(A.widthSize);
    [S addSubview:A];

    UIView *B = UIView.new;
B.leftPos.equalTo(@40);
B.widthSize.等于(@60);
B.heightSize.equalTo(@40);
    [S addSubview:B];

    UIView *C = UIView.new;
```


Chapter 193: MyLayout

MyLayout is a simple and easy objective-c framework for iOS view layout. MyLayout provides some simple functions to build a variety of complex interface. It integrates the functions including: Autolayout and SizeClass of iOS, five layout classes of Android, float and flex-box and bootstrap of HTML/CSS. you can visit from:

Objective-C: https://github.com/youngsoft/MyLinearLayout Swift: https://github.com/youngsoft/TangramKit

Section 193.1: A Simple Demo to use MyLayout

1. There is a container view S which width is 100 and height is wrap to all subviews height. there are four subviews A,B,C,D arranged from top to bottom.
2. Subview A's left margin is 20% width of S, right margin is 30% width of S, height is equal to width of A.
3. Subview B's left margin is 40, width is filled in to residual width of S,height is 40. Subview C's width is filled in to S, height is
4. Subview D's right margin is 20, width is 50% width of S, height is 40

like below figure:

```
 MyLinearLayout *S = [MyLinearLayout linearLayoutWithOrientation:MyLayoutViewOrientation_Vert];
 S.subviewSpace = 10;
S.widthSize.equalTo(@100);

 UIView *A = UIView.new;
 A.leftPos.equalTo(@0.2);
 A.rightPos.equalTo(@0.3);
 A.heightSize.equalTo(A.widthSize);
 [S addSubview:A];

 UIView *B = UIView.new;
 B.leftPos.equalTo(@40);
 B.widthSize.equalTo(@60);
 B.heightSize.equalTo(@40);
 [S addSubview:B];

 UIView *C = UIView.new;
```

```
C.leftPos.equalTo(@0);
C.rightPos.equalTo(@0);
C.heightSize.equalTo(@40);
    [S addSubview:C];

    UIView *D = UIView.new;
D.rightPos.equalTo(@20);
D.widthSize.equalTo(S.widthSize).multiply(0.5);
D.heightSize.equalTo(@40);
    [S addSubview:D];
```



```
    C.leftPos.equalTo(@0);
    C.rightPos.equalTo(@0);
    C.heightSize.equalTo(@40);
    [S addSubview:C];

    UIView *D = UIView.new;
    D.rightPos.equalTo(@20);
    D.widthSize.equalTo(S.widthSize).multiply(0.5);
    D.heightSize.equalTo(@40);
    [S addSubview:D];
```

第194章：模拟器构建

在哪里可以找到模拟器构建？

前往 ~/Library/Developer/CoreSimulator/Devices/

你会发现带有字母数字名称的目录

然后点击其中一个目录，进行以下选择

Data/Containers/Bundle/Application/

你会再次看到带有字母数字名称的目录，点击后你会找到

模拟器构建文件

注意：

在模拟器上安装iOS设备构建是行不通的。

iPhone模拟器使用i386架构，iPad模拟器构建使用x8架构

第194.1节：在模拟器上手动安装构建

```
xcrun simctl install booted *.app
```


Chapter 194: Simulator Builds

Where to find simulator build ?

Go to ~/Library/Developer/CoreSimulator/Devices/

You will find directories with alphanumeric names

then click on the one of the directories and make following selection

Data/Containers/Bundle/Application/

Again you will find directories with alphanumeric names if you click on that you will find

Simulator build over there

Note:

Installing iOS device build on simulator will not work out.

iPhone simulator uses i386 architecture iPad simulator builds uses x8

Section 194.1: Installing the build manually on simulator

```
xcrun simctl install booted *.app
```

第195章：使用GPX文件在iOS上模拟位置

第195.1节：您的.gpx文件：MPS_HQ.gpx

```
<gpx xmlns="http://www.topografix.com/GPX/1/1"
xmlns:gpxx = "http://www.garmin.com/xmlschemas/GpxExtensions/v3"
xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.topografix.com/GPX/1/1
http://www.topografix.com/GPX/1/1/gpx.xsd
http://www.garmin.com/xmlschemas/GpxExtensions/v3
http://www8.garmin.com/xmlschemas/GpxExtensions/v3/GpxExtensionsv3.xsd"
version="1.1"
creator="gpx-poi.com">
 <wpt lat="38.9072" lon="77.0369">38.9072/-77.0369
 <time>2015-04-16T22:20:29Z</time>
  <name>华盛顿特区</name>
  <extensions>
     <gpxx:WaypointExtension>
        <gpxx:Proximity>10</gpxx:Proximity>
        <gpxx:Address>
           <gpxx:StreetAddress>华盛顿特区</gpxx:StreetAddress>
           <gpxx:City>华盛顿</gpxx:City>
           <gpxx:State>DC</gpxx:State>
           <gpxx:Country>美国</gpxx:Country>
           <gpxx:PostalCode> 20005 </gpxx:PostalCode>
        </gpxx:Address>
     </gpxx:WaypointExtension>
  </extensions>
```

第195.2节：设置此位置：

1. 进入编辑方案。
2. 选择运行 -> 选项。
3. 勾选“允许位置模拟”。
4. 从“默认位置”下拉列表中选择 *.GPX 文件名。

Chapter 195: Simulating Location Using GPX files iOS

Section 195.1: Your .gpx file: MPS_HQ.gpx

```
<gpx xmlns="http://www.topografix.com/GPX/1/1"
xmlns:gpxx = "http://www.garmin.com/xmlschemas/GpxExtensions/v3"
xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.topografix.com/GPX/1/1
http://www.topografix.com/GPX/1/1/gpx.xsd
http://www.garmin.com/xmlschemas/GpxExtensions/v3
http://www8.garmin.com/xmlschemas/GpxExtensions/v3/GpxExtensionsv3.xsd"
version="1.1"
creator="gpx-poi.com">
 <wpt lat="38.9072" lon="77.0369">38.9072/-77.0369
 <time>2015-04-16T22:20:29Z</time>
  <name>Washington, DC</name>
  <extensions>
     <gpxx:WaypointExtension>
        <gpxx:Proximity>10</gpxx:Proximity>
        <gpxx:Address>
           <gpxx:StreetAddress>Washington DC</gpxx:StreetAddress>
           <gpxx:City>Washington</gpxx:City>
           <gpxx:State>DC</gpxx:State>
           <gpxx:Country>United States</gpxx:Country>
           <gpxx:PostalCode> 20005 </gpxx:PostalCode>
        </gpxx:Address>
     </gpxx:WaypointExtension>
  </extensions>
```

Section 195.2: To set this location:

1. Go to Edit Scheme.
2. Select Run -> Options.
3. Check "Allow Location Simulation".
4. Select the *.GPX File Name from the "Default Location" drop down list.

Build
1 target
Run
Staging
Test
Staging
Profile
Staging
Analyze
Staging
Archive
Staging
Info
Arguments
Options
Diagnostics
Core Location
Allow Location Simulation
Default Location
MPS_HQ
Application Data
None
Routing App Coverage File
None
Background Fetch
Launch due to a background fetch event
Localization Debugging
Show non-localized strings
Application Language
System Language
Application Region
System Region
XPC Services
Debug XPC services used by this application
View Debugging
Enable user interface debugging
Queue Debugging
Enable backtrace recording
Duplicate Scheme
Manage Schemes...
Shared
Close

Build
1 target
Run
Staging
Test
Staging
Profile
Staging
Analyze
Staging
Archive
Staging
Info
Arguments
Options
Diagnostics
Core Location
Allow Location Simulation
Default Location
MPS_HQ
Application Data
None
Routing App Coverage File
None
Background Fetch
Launch due to a background fetch event
Localization Debugging
Show non-localized strings
Application Language
System Language
Application Region
System Region
XPC Services
Debug XPC services used by this application
View Debugging
Enable user interface debugging
Queue Debugging
Enable backtrace recording
Duplicate Scheme
Manage Schemes...
Shared
Close

第196章：SqlCipher 集成

SQLite 已经是 iOS 应用中持久数据存储的流行 API，因此对开发来说优势显而易见。作为程序员，你使用的是一个稳定且文档完善的 API，且该 API 在 Objective-C 中有许多优秀的封装库，如 FMDB 和加密核心数据。所有安全相关的问题都与应用代码清晰分离，由底层框架管理。

第196.1节：代码集成：

使用密码集成打开数据库。

```
-(void)checkAndOpenDB{
    sqlite3 *db;
     NSString *strPassword = @"password";

        if (sqlite3_open_v2([[databaseURL path] UTF8String], &db, SQLITE_OPEN_READWRITE |
SQLITE_OPEN_CREATE, NULL) == SQLITE_OK) {
const char* key = [strPassword UTF8String];
            sqlite3_key(db, key, (int)strlen(key));
              if (sqlite3_exec(db1, (const char*) "SELECT count(*) FROM sqlite_master;", NULL, NULL,
NULL) == SQLITE_OK) {
NSLog(@"密码正确，或者已初始化新数据库");
              } else {
NSLog(@"密码错误！");
              }
sqlite3_close(db);
        }
}

- (NSURL *)databaseURL
{
    NSArray *URLs = [[NSFileManager defaultManager] URLsForDirectory:NSDocumentDirectory
inDomains:NSUserDomainMask];
    NSURL *directoryURL = [URLs firstObject];
    NSURL *databaseURL = [directoryURL URLByAppendingPathComponent:@"database.sqlite"];
    return  databaseURL;
}
```


Chapter 196: SqlCipher integration

SQLite is already a popular API for persistent data storage in iOS apps so the upside for development is obvious. As a programmer you work with a stable, well-documented API that happens to have many good wrappers available in Objective-C, such as FMDB and Encrypted Core Data. All security concerns are cleanly decoupled from application code and managed by the underlying framework.

Section 196.1: Integration of code:

Integration to open database using password.

```
-(void)checkAndOpenDB{
    sqlite3 *db;
    NSString *strPassword = @"password";

        if (sqlite3_open_v2([[databaseURL path] UTF8String], &db, SQLITE_OPEN_READWRITE |
SQLITE_OPEN_CREATE, NULL) == SQLITE_OK) {
            const char* key = [strPassword UTF8String];
            sqlite3_key(db, key, (int)strlen(key));
            if (sqlite3_exec(db1, (const char*) "SELECT count(*) FROM sqlite_master;", NULL, NULL,
NULL) == SQLITE_OK) {
                NSLog(@"Password is correct, or a new database has been initialized");
            } else {
                NSLog(@"Incorrect password!");
            }
            sqlite3_close(db);
        }
}

- (NSURL *)databaseURL
{
    NSArray *URLs = [[NSFileManager defaultManager] URLsForDirectory:NSDocumentDirectory
inDomains:NSUserDomainMask];
    NSURL *directoryURL = [URLs firstObject];
    NSURL *databaseURL = [directoryURL URLByAppendingPathComponent:@"database.sqlite"];
    return  databaseURL;
}
```

第197章：安全性

iOS中的安全性涉及数据安全、传输安全、代码安全等方面

第197.1节：保护iTunes备份中的数据安全

如果我们希望应用数据在iTunes备份中受到保护，就必须让应用数据不被iTunes备份。

每当iOS设备使用macOS上的iTunes备份时，所有应用存储的数据都会被复制到该备份中，并存储在备份电脑上。

但我们可以使用URLResourceKey.isExcludedFromBackupKey键将应用数据排除在此备份之外。
以下是我们应用的目录结构：

注意： 通常敏感数据存储在“应用支持”目录中。

例如，如果我们想排除存储在**Application Support**目录中的所有数据，则可以按如下方式使用上述提到的键：

```
    let urls = FileManager.default.urls(for: .applicationSupportDirectory, in: .userDomainMask)
    let baseURL = urls[urls.count-1];

                let bundleIdentifier = Bundle.main.object(forInfoDictionaryKey: "CFBundleIdentifier") as!
String
    let pathURL = baseURL.appendingPathComponent(bundleIdentifier)
    let persistentStoreDirectoryPath = pathURL.path
    if !FileManager.default.fileExists(atPath: persistentStoreDirectoryPath) {
        do {
try FileManager.default.createDirectory(atPath: path, withIntermediateDirectories:
true, attributes: nil)
        }catch {
            //handle error
        }
    }
    let dirURL = URL.init(fileURLWithPath: persistentStoreDirectoryPath, isDirectory: true)
    do {
try (dirURL as NSURL).setResourceValue((true), forKey: .isExcludedFromBackupKey)
    } catch {
        //处理错误
    }
```

有很多工具可以查看iTunes备份中的所有备份数据，以确认上述
方法是否有效。

Chapter 197: Security

Security in iOS is related to data security, transport security, code security, etc

Section 197.1: Securing Data in iTunes Backups

If we want our app data to be protected against iTunes backups, we have to skip our app data from being backed up in iTunes.
Whenever iOS device backed up using iTunes on macOS, all the data stored by all the apps is copied in that backup and stored on backing computer.

But we can exclude our app data from this backup using `URLResourceKey.isExcludedFromBackupKey` key.
Here is the directory structure of our app:

Note: Generally sensitive data is stored in 'Application Support' directory.

e.g. If we want to exclude all our data stored in **Application Support** directory then we can use above mentioned key as follow:

```
    let urls = FileManager.default.urls(for: .applicationSupportDirectory, in: .userDomainMask)
    let baseURL = urls[urls.count-1];

    let bundleIdentifier = Bundle.main.object(forInfoDictionaryKey: "CFBundleIdentifier") as!
String
    let pathURL = baseURL.appendingPathComponent(bundleIdentifier)
    let persistentStoreDirectoryPath = pathURL.path
    if !FileManager.default.fileExists(atPath: persistentStoreDirectoryPath) {
        do {
            try FileManager.default.createDirectory(atPath: path, withIntermediateDirectories:
true, attributes: nil)
        }catch {
            //handle error
        }
    }
    let dirURL = URL.init(fileURLWithPath: persistentStoreDirectoryPath, isDirectory: true)
    do {
        try (dirURL as NSURL).setResourceValue((true), forKey: .isExcludedFromBackupKey)
    } catch {
        //handle error
    }
```

There are lots of tools available to see iTunes backups for all the backed up data to confirm whether above approach works or not.

iExplorer 是一个很好的工具，用于浏览iTunes备份。

第197.2节：使用SSL的传输安全

iOS应用需要以一种方式编写，以保障通过网络传输的数据的安全性。
SSL是实现这一点的常用方法。

每当应用尝试调用网络服务以拉取或推送数据到服务器时，应使用基于HTTP的SSL，即HTTPS。
为此，应用必须调用 https://server.com/part 这样的网络服务，而不是 http://server.com/part。
在这种情况下，应用程序需要信任使用 SSL 证书的服务器 server.com。

以下是验证服务器信任的示例-

实现`URLSessionDelegate`如下：

```
func urlSession(_ session: URLSession, didReceive challenge: URLAuthenticationChallenge,
completionHandler: @escaping (URLSession.AuthChallengeDisposition, URLCredential?) -> Void) {

    if challenge.protectionSpace.authenticationMethod == NSURLAuthenticationMethodServerTrust {
        let serverTrust:SecTrust = challenge.protectionSpace.serverTrust!

        func acceptServerTrust() {
            let credential:URLCredential = URLCredential(trust: serverTrust)
            challenge.sender?.use(credential, for: challenge)
            completionHandler(.useCredential, URLCredential(trust:
challenge.protectionSpace.serverTrust!))
        }

        let success = SSLTrustManager.shouldTrustServerTrust(serverTrust, forCert:
"Server_Public_SSL_Cert")
        if success {
acceptServerTrust()
            return
        }
    }
    else if challenge.protectionSpace.authenticationMethod ==
NSURLAuthenticationMethodClientCertificate {
        completionHandler(.rejectProtectionSpace, nil);
        return
    }
completionHandler(.cancelAuthenticationChallenge, nil)
}
```

这是信任管理器：（未找到Swift代码）

```
@implementation SSLTrustManager
+ (BOOL)shouldTrustServerTrust:(SecTrustRef)serverTrust forCert:(NSString*)certName {
// 加载捆绑的证书。
NSString *certPath = [[NSBundle mainBundle] pathForResource:certName ofType:@"der"];
NSData *certData = [[NSData alloc] initWithContentsOfFile:certPath];
CFDataRef certDataRef = (__bridge_retained CFDataRef)certData;
SecCertificateRef cert = SecCertificateCreateWithData(NULL, certDataRef);

// 建立以我们捆绑证书为锚点的信任链。
CFArrayRef certArrayRef = CFArrayCreate(NULL, (void *)&cert, 1, NULL);
SecTrustSetAnchorCertificates(serverTrust, certArrayRef);

// 验证该信任。
SecTrustResultType trustResult;
```


iExplorer is good one to explore iTunes backups.

Section 197.2: Transport Security using SSL

iOS apps needs to be written in a way to provide security to data which is being transported over network.
SSL is the common way to do it.

Whenever app tries to call web services to pull or push data to servers, it should **use SSL over HTTP, i.e. HTTPS**.
To do this, **app must call `https://server.com/part`** such web services and not `http://server.com/part`.
In this case, app needs to trust the server `server.com` using SSL certificate.

Here is the example of validating server trust-

Implement `URLSessionDelegate` as:

```
func urlSession(_ session: URLSession, didReceive challenge: URLAuthenticationChallenge,
completionHandler: @escaping (URLSession.AuthChallengeDisposition, URLCredential?) -> Void) {

    if challenge.protectionSpace.authenticationMethod == NSURLAuthenticationMethodServerTrust {
        let serverTrust:SecTrust = challenge.protectionSpace.serverTrust!

        func acceptServerTrust() {
            let credential:URLCredential = URLCredential(trust: serverTrust)
            challenge.sender?.use(credential, for: challenge)
            completionHandler(.useCredential, URLCredential(trust:
challenge.protectionSpace.serverTrust!))
        }

        let success = SSLTrustManager.shouldTrustServerTrust(serverTrust, forCert:
"Server_Public_SSL_Cert")
        if success {
            acceptServerTrust()
            return
        }
    }
    else if challenge.protectionSpace.authenticationMethod ==
NSURLAuthenticationMethodClientCertificate {
        completionHandler(.rejectProtectionSpace, nil);
        return
    }
    completionHandler(.cancelAuthenticationChallenge, nil)
}
```

Here is trust manager: (couldn't found Swift code)

```
@implementation SSLTrustManager
+ (BOOL)shouldTrustServerTrust:(SecTrustRef)serverTrust forCert:(NSString*)certName {
// Load up the bundled certificate.
NSString *certPath = [[NSBundle mainBundle] pathForResource:certName ofType:@"der"];
NSData *certData = [[NSData alloc] initWithContentsOfFile:certPath];
CFDataRef certDataRef = (__bridge_retained CFDataRef)certData;
SecCertificateRef cert = SecCertificateCreateWithData(NULL, certDataRef);

// Establish a chain of trust anchored on our bundled certificate.
CFArrayRef certArrayRef = CFArrayCreate(NULL, (void *)&cert, 1, NULL);
SecTrustSetAnchorCertificates(serverTrust, certArrayRef);

// Verify that trust.
SecTrustResultType trustResult;
```

```
SecTrustEvaluate(serverTrust, &trustResult);

// 清理。
CFRelease(certArrayRef);
CFRelease(cert);
CFRelease(certDataRef);

// 我们自定义的信任链是否成功评估？
return trustResult == kSecTrustResultUnspecified;
}
@end
```

Server_Public_SSL_Cert.der 是服务器的公用SSL密钥。

通过这种方法，我们的应用可以确保它正在与预期的服务器通信，且没有人拦截应用与服务器之间的通信。


```
SecTrustEvaluate(serverTrust, &trustResult);

// Clean up.
CFRelease(certArrayRef);
CFRelease(cert);
CFRelease(certDataRef);

// Did our custom trust chain evaluate successfully?
return trustResult == kSecTrustResultUnspecified;
}
@end
```

Server_Public_SSL_Cert.der is servers' public SSL key.

Using this approach our app can make sure that it is communicating to the intended server and no one is intercepting the app-server communication.

第198章：应用传输安全（ATS）

参数	详细信息
NSAppTransportSecurity	配置ATS
NSAllowsArbitraryLoads	设置为YES以在所有地方禁用ATS。在iOS 10及更高版本和macOS 10.12及更高版本中，如果您的应用程序'的Info.plist文件中存在以下任意键，则此键的值将被忽略：NSAllowsArbitraryLoadsInMedia，NSAllowsArbitraryLoadsInWebContent，NSAllowsLocalNetworking
NSAllowsArbitraryLoadsInMedia	设置为YES以禁用通过AV Foundation框架API加载的媒体的ATS。（iOS 10+，macOS 10.12+）
NSAllowsArbitraryLoadsInWebContent	设置为YES以禁用应用程序'的网页视图（WKWebView、UIWebView、WebView）中的ATS，而不影响NSURLSession连接。（iOS 10+，macOS 10.12+）
NSAllowsLocalNetworking	设置为YES以禁用对未限定域名和.local域的连接的ATS。（iOS 10+，macOS 10.12+）
NSExceptionDomains	为特定域配置例外
NSIncludesSubdomains	设置为YES以将例外应用于所选域的所有子域。
NSRequiresCertificateTransparency	设置为YES以要求针对域上的服务器（X.509）证书，必须提供来自已知CT日志的有效签名证书透明度（CT）时间戳。（iOS 10+，macOS 10.12+）
NSExceptionAllowsInsecureHTTPLoads	设置为YES以允许所选域使用HTTP。
NSExceptionRequiresForwardSecrecy	默认值为YES；设置为NO以禁用前向保密并接受更多密码套件。
NSExceptionMinimumTLSVersion	默认值为TLSv1.2；可能的值有：TLSv1.0、TLSv1.1、TLSv1.2
NSThirdPartyExceptionAllowsInsecureHTTPLoads	类似于 NSExceptionAllowsInsecureHTTPLoads，但适用于您无法控制的域名
NSThirdPartyExceptionRequiresForwardSecrecy	类似于 NSExceptionRequiresForwardSecrecy，但适用于您无法控制的域名
NSThirdPartyExceptionMinimumTLSVersion	类似于 NSExceptionMinimumTLSVersion，但适用于您无法控制的域名

第198.1节：加载所有HTTP内容

苹果在iOS 9中引入了ATS，作为一项新的安全功能，以提升应用与网络服务之间的隐私和安全。ATS默认会拒绝所有非HTTPS请求。虽然这对生产环境非常有利，但在测试过程中可能会带来不便。

ATS在目标的Info.plist文件中通过NSAppTransportSecurity字典（Xcode Info.plist编辑器中的App Transport Security Settings）进行配置。要允许所有HTTP内容，添加Allow Arbitrary Loads布尔值（NSAllowsArbitraryLoads）并设置为YES。生产应用不推荐这样做，如果需要HTTP内容，建议选择性启用。

第198.2节：选择性加载HTTP内容

与启用所有HTTP内容类似，所有配置均在App Transport Security Settings下进行。向顶层ATS设置中添加Exception Domains字典（NSExceptionDomains）。

对于每个域，向"例外域"中添加一个字典项，其中键为相应的域名。设置
将 NSExceptionAllowsInsecureHTTPLoads 设置为 YES，以禁用该域的 HTTPS 要求。

Chapter 198: App Transport Security (ATS)

Parameter	Details
NSAppTransportSecurity	Configure ATS
NSAllowsArbitraryLoads	Set to YES to disable ATS everywhere. In iOS 10 and later, and macOS 10.12 and later, the value of this key is ignored if any of the following keys are present in your app's Info.plist file: NSAllowsArbitraryLoadsInMedia, NSAllowsArbitraryLoadsInWebContent, NSAllowsLocalNetworking
NSAllowsArbitraryLoadsInMedia	Set to YES to disable ATS for media loaded using APIs from the AV Foundation framework. ***(iOS 10+, macOS 10.12+)***
NSAllowsArbitraryLoadsInWebContent	Set to YES to disable ATS in your app's web views (WKWebView, UIWebView, WebView) without affecting your NSURLSession connections. ***(iOS 10+, macOS 10.12+)***
NSAllowsLocalNetworking	Set to YES to disable for connections to unqualified domains and to .local domains. ***(iOS 10+, macOS 10.12+)***
NSExceptionDomains	Configure exceptions for specific domains
NSIncludesSubdomains	Set to YES to apply the exceptions to all subdomains of the selected domain.
NSRequiresCertificateTransparency	Set to YES to require that valid, signed Certificate Transparency (CT) timestamps, from known CT logs, be presented for server (X.509) certificates on a domain. ***(iOS 10+, macOS 10.12+)***
NSExceptionAllowsInsecureHTTPLoads	Set to YES to allow HTTP on the selected domain.
NSExceptionRequiresForwardSecrecy	Defaults to YES; Set to NO to disable Forward Secrecy and accept more ciphers.
NSExceptionMinimumTLSVersion	Defaults to TLSv1.2; Possible values are: TLSv1.0, TLSv1.1, TLSv1.2
NSThirdPartyExceptionAllowsInsecureHTTPLoads	Similar to NSExceptionAllowsInsecureHTTPLoads, but for domains that you have no control over
NSThirdPartyExceptionRequiresForwardSecrecy	Similar to NSExceptionRequiresForwardSecrecy, but for domains that you have no control over
NSThirdPartyExceptionMinimumTLSVersion	Similar to NSExceptionMinimumTLSVersion, but for domains that you have no control over

Section 198.1: Load all HTTP content

Apple introduced ATS with iOS 9 as a new security feature to improve privacy and security between apps and web services. ATS by default fails all non HTTPS requests. While this can be really nice for production environments, it can be a nuisance during testing.

ATS is configured in the target's Info.plist file with the NSAppTransportSecurity dictionary (App Transport Security Settings in the Xcode Info.plist editor). To allow all HTTP content, add the Allow Arbitrary Loads boolean (NSAllowsArbitraryLoads) and set it to YES. This is not recommended for production apps, and if HTTP content is required, it is recommended that it be selectively enabled instead.

Section 198.2: Selectively load HTTP content

Similar to enabling all HTTP content, all configuration happens under the App Transport Security Settings. Add the Exception Domains dictionary (NSExceptionDomains) to the top level ATS settings.

For every domain, add a dictionary item to the Exception Domains, where the key is the domain in question. Set NSExceptionAllowsInsecureHTTPLoads to YES to disable the HTTPS requirement for that domain.

第198.3节：端点需要 SSL

自 iOS 9 起，所有端点必须遵守 HTTPS 规范。
任何未使用 SSL 的端点将在控制台日志中出现警告。对于您的应用程序来说，网络连接将显示为失败。

要配置例外，请将以下内容放入您的 Info.plist 文件：

1. 仅允许特定域（testdomain.com）：

```
<key>NSAppTransportSecurity</key>
<dict>
<key>NSExceptionDomains</key>
<dict>
    <key>testdomain.com</key>
    <dict>
        <key>NSIncludesSubdomains</key>
        <true/>
        <key>NSExceptionAllowsInsecureHTTPLoads</key>
        <true/>
    </dict>
</dict>
```

允许此类行为的键是 NSExceptionAllowsInsecureHTTPLoads。在此情况下，应用程序将仅允许与指定域（testdomain.com）的 HTTP 连接，并阻止所有其他 HTTP 连接。

关键字NSIncludesSubdomains指定提到的域名（testdomain.com）的任何及所有子域名也应被允许。

2. 允许任何域名：

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSAllowsArbitraryLoads</key>
    <true/>
```

在这种情况下，应用程序将允许对任何域名的HTTP连接。自2017年1月1日起，使用此标志将导致App Store的严格审核，应用开发者必须解释他们为何首先需要使用此例外。可能的解释包括：

- 加载不包含个性化信息的加密媒体内容的应用程序。
- 连接到无法升级为使用安全连接的设备。
- 连接到由其他实体管理且不支持安全连接的服务器。

Section 198.3: Endpoints require SSL

Introduced in iOS 9, all endpoints must adhere to the HTTPS specification.
Any endpoints not using SSL will fail with a warning in the console log. To your application it will appear that the internet connection failed.

To configure exceptions: Place the following in your Info.plist file:

1. Allow particular domain (testdomain.com) **only**:

```
<key>NSAppTransportSecurity</key>
<dict>
<key>NSExceptionDomains</key>
<dict>
    <key>testdomain.com</key>
    <dict>
        <key>NSIncludesSubdomains</key>
        <true/>
        <key>NSExceptionAllowsInsecureHTTPLoads</key>
        <true/>
    </dict>
</dict>
```

The key that permits such behavior is `NSExceptionAllowsInsecureHTTPLoads`. In this case, app will allow HTTP connection to mentioned domain (testdomain.com) only and block all other HTTP connections.

The key `NSIncludesSubdomains` specifies that any and all **subdomains** of the mentioned domain (testdomain.com) should also be allowed.

2. Allow any domain:

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSAllowsArbitraryLoads</key>
    <true/>
</dict>
```

In this case, app will allow HTTP connection to **any** domain. As of January 1st 2017, using this flag will cause thorough App Store review and the app developers will have to explain why they need to use this exception in the first place. Possible explanations include:

- An application that loads encrypted media content that contains no personalized information.
- Connections to devices that cannot be upgraded to use secure connections.
- Connection to a server that is managed by another entity and does not support secure connections.

第199章：选择最佳iOS架构模式的指南

揭开MVC、MVP、MVVM和VIPER或其他设计模式的神秘面纱，选择构建应用程序的最佳方法

第199.1节：MVP模式

```
import UIKit

结构体 Person { // 模型
    让 firstName: 字符串
    让 lastName: 字符串
}

协议 GreetingView: 类 {
    函数 setGreeting(greeting: 字符串)
}

协议 GreetingViewPresenter {
    初始化(view: GreetingView, person: Person)
    函数 showGreeting()
}

类 GreetingPresenter : GreetingViewPresenter {
    无主引用 让 view: GreetingView
    让 person: Person
    必需初始化(view: GreetingView, person: Person) {
        self.view = view
        self.person = person
    }
    函数 showGreeting() {
        让 greeting = "Hello" + " " + self.person.firstName + " " + self.person.lastName
        self.view.setGreeting(greeting)
    }
}

class GreetingViewController : UIViewController, GreetingView {
    var presenter: GreetingViewPresenter!
    let showGreetingButton = UIButton()
    let greetingLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
                self.showGreetingButton.addTarget(self, action: "didTapButton:", forControlEvents:
.TouchUpInside)
    }

    func didTapButton(button: UIButton) {
        self.presenter.showGreeting()
    }

    func setGreeting(greeting: String) {
        self.greetingLabel.text = greeting
    }

    // 布局代码写在这里
}
// MVP的组装
```


Chapter 199: Guideline to choose best iOS Architecture Patterns

Demystifying MVC, MVP, MVVM and VIPER or any other design patterns to choose the best approach to building an app

Section 199.1: MVP Patterns

```
import UIKit

struct Person { // Model
    let firstName: String
    let lastName: String
}

protocol GreetingView: class {
    func setGreeting(greeting: String)
}

protocol GreetingViewPresenter {
    init(view: GreetingView, person: Person)
    func showGreeting()
}

class GreetingPresenter : GreetingViewPresenter {
    unowned let view: GreetingView
    let person: Person
    required init(view: GreetingView, person: Person) {
        self.view = view
        self.person = person
    }
    func showGreeting() {
        let greeting = "Hello" + " " + self.person.firstName + " " + self.person.lastName
        self.view.setGreeting(greeting)
    }
}

class GreetingViewController : UIViewController, GreetingView {
    var presenter: GreetingViewPresenter!
    let showGreetingButton = UIButton()
    let greetingLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
        self.showGreetingButton.addTarget(self, action: "didTapButton:", forControlEvents:
.TouchUpInside)
    }

    func didTapButton(button: UIButton) {
        self.presenter.showGreeting()
    }

    func setGreeting(greeting: String) {
        self.greetingLabel.text = greeting
    }

    // layout code goes here
}
// Assembling of MVP
```

```
let model = Person(firstName: "大卫", lastName: "布莱恩")
let view = GreetingViewController()
let presenter = GreetingPresenter(view: view, person: model)
view.presenter = presenter
```

第199.2节：MVVM模式

```
import UIKit

结构体 Person { // 模型
    让 firstName: 字符串
    让 lastName: 字符串
}

协议 GreetingViewModelProtocol: 类 {
    变量 greeting: 字符串? { 获取 }
    变量 greetingDidChange: ((GreetingViewModelProtocol) -> ())? { 获取 设置 } // 当greeting改变时调用的函数

    初始化(person: Person)
    函数 showGreeting()
}

类 GreetingViewModel : GreetingViewModelProtocol {
    常量 person: Person
    变量 greeting: 字符串? {
        属性观察器 {
            self.greetingDidChange?(self)
        }
    }
    变量 greetingDidChange: ((GreetingViewModelProtocol) -> ())?
    必需初始化(person: Person) {
        self.person = person
    }
    函数 showGreeting() {
        self.greeting = "Hello" + " " + self.person.firstName + " " + self.person.lastName
    }
}

class GreetingViewController : UIViewController {
    var viewModel: GreetingViewModelProtocol! {
        didSet {
            self.viewModel.greetingDidChange = { [unowned self] viewModel in
                self.greetingLabel.text = viewModel.greeting
            }
        }
    }
    let showGreetingButton = UIButton()
    let greetingLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
        self.showGreetingButton.addTarget(self.viewModel, action: "showGreeting", forControlEvents:
.TouchUpInside)
    }
    // 布局代码写在这里
}
// 组装MVVM
let model = Person(firstName: "David", lastName: "Blaine")
let viewModel = GreetingViewModel(person: model)
let view = GreetingViewController()
```



```
let model = Person(firstName: "David", lastName: "Blaine")
let view = GreetingViewController()
let presenter = GreetingPresenter(view: view, person: model)
view.presenter = presenter
```

Section 199.2: MVVM Pattern

```
import UIKit

struct Person { // Model
    let firstName: String
    let lastName: String
}

protocol GreetingViewModelProtocol: class {
    var greeting: String? { get }
    var greetingDidChange: ((GreetingViewModelProtocol) -> ())? { get set } // function to call
when greeting did change
    init(person: Person)
    func showGreeting()
}

class GreetingViewModel : GreetingViewModelProtocol {
    let person: Person
    var greeting: String? {
        didSet {
            self.greetingDidChange?(self)
        }
    }
    var greetingDidChange: ((GreetingViewModelProtocol) -> ())?
    required init(person: Person) {
        self.person = person
    }
    func showGreeting() {
        self.greeting = "Hello" + " " + self.person.firstName + " " + self.person.lastName
    }
}

class GreetingViewController : UIViewController {
    var viewModel: GreetingViewModelProtocol! {
        didSet {
            self.viewModel.greetingDidChange = { [unowned self] viewModel in
                self.greetingLabel.text = viewModel.greeting
            }
        }
    }
    let showGreetingButton = UIButton()
    let greetingLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
        self.showGreetingButton.addTarget(self.viewModel, action: "showGreeting", forControlEvents:
.TouchUpInside)
    }
    // layout code goes here
}
// Assembling of MVVM
let model = Person(firstName: "David", lastName: "Blaine")
let viewModel = GreetingViewModel(person: model)
let view = GreetingViewController()
```

```
view.viewModel = viewModel
```

第199.3节：VIPER模式

```
import UIKit

结构体 Person { // 实体（通常更复杂，例如 NSManagedObject）
    let firstName: String
    让 lastName: 字符串
}

struct GreetingData { // 传输数据结构（非实体）
    let greeting: String
    let subject: String
}

protocol GreetingProvider {
    func provideGreetingData()
}

protocol GreetingOutput: class {
    func receiveGreetingData(greetingData: GreetingData)
}

class GreetingInteractor : GreetingProvider {
    weak var output: GreetingOutput!

    func provideGreetingData() {
        let person = Person(firstName: "David", lastName: "Blaine") // 通常来自数据访问层

        let subject = person.firstName + " " + person.lastName
        let greeting = GreetingData(greeting: "Hello", subject: subject)
        self.output.receiveGreetingData(greeting)
    }
}

protocol GreetingViewEventHandler {
    func didTapShowGreetingButton()
}

protocol GreetingView: class {
    func setGreeting(greeting: String)
}

class GreetingPresenter : GreetingOutput, GreetingViewEventHandler {
    weak var view: GreetingView!
    var greetingProvider: GreetingProvider!

    func didTapShowGreetingButton() {
        self.greetingProvider.provideGreetingData()
    }

    func receiveGreetingData(greetingData: GreetingData) {
        let greeting = greetingData.greeting + " " + greetingData.subject
        self.view.setGreeting(greeting)
    }
}

class GreetingViewController : UIViewController, GreetingView {
    var eventHandler: GreetingViewEventHandler!
    let showGreetingButton = UIButton()
```



```
view.viewModel = viewModel
```

Section 199.3: VIPER Pattern

```
import UIKit

struct Person { // Entity (usually more complex e.g. NSManagedObject)
    let firstName: String
    let lastName: String
}

struct GreetingData { // Transport data structure (not Entity)
    let greeting: String
    let subject: String
}

protocol GreetingProvider {
    func provideGreetingData()
}

protocol GreetingOutput: class {
    func receiveGreetingData(greetingData: GreetingData)
}

class GreetingInteractor : GreetingProvider {
    weak var output: GreetingOutput!

    func provideGreetingData() {
        let person = Person(firstName: "David", lastName: "Blaine") // usually comes from data
access layer
        let subject = person.firstName + " " + person.lastName
        let greeting = GreetingData(greeting: "Hello", subject: subject)
        self.output.receiveGreetingData(greeting)
    }
}

protocol GreetingViewEventHandler {
    func didTapShowGreetingButton()
}

protocol GreetingView: class {
    func setGreeting(greeting: String)
}

class GreetingPresenter : GreetingOutput, GreetingViewEventHandler {
    weak var view: GreetingView!
    var greetingProvider: GreetingProvider!

    func didTapShowGreetingButton() {
        self.greetingProvider.provideGreetingData()
    }

    func receiveGreetingData(greetingData: GreetingData) {
        let greeting = greetingData.greeting + " " + greetingData.subject
        self.view.setGreeting(greeting)
    }
}

class GreetingViewController : UIViewController, GreetingView {
    var eventHandler: GreetingViewEventHandler!
    let showGreetingButton = UIButton()
```

```
    let greetingLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
                self.showGreetingButton.addTarget(self, action: "didTapButton:", forControlEvents:
.TouchUpInside)
    }

    func didTapButton(button: UIButton) {
        self.eventHandler.didTapShowGreetingButton()
    }

    func setGreeting(greeting: String) {
        self.greetingLabel.text = greeting
    }

    // 布局代码写在这里
}
// 组装 VIPER 模块，无 Router
let view = GreetingViewController()
let presenter = GreetingPresenter()
let interactor = GreetingInteractor()
view.eventHandler = presenter
presenter.view = view
presenter.greetingProvider = interactor
interactor.output = presenter
```

第199.4节：MVC 模式

```
import UIKit

struct Person { // 模型
    let firstName: String
    让 lastName: 字符串
}

class GreetingViewController : UIViewController { // 视图 + 控制器
    var person: Person!
    let showGreetingButton = UIButton()
    let greetingLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
                self.showGreetingButton.addTarget(self, action: "didTapButton:", forControlEvents:
.TouchUpInside)
    }

    func didTapButton(button: UIButton) {
        let greeting = "Hello" + " " + self.person.firstName + " " + self.person.lastName
        self.greetingLabel.text = greeting

    }
    // 布局代码写在这里
}
// MVC的组装
let model = Person(firstName: "大卫", lastName: "布莱恩")
let view = GreetingViewController()
view.person = model
```



```
    let greetingLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
        self.showGreetingButton.addTarget(self, action: "didTapButton:", forControlEvents:
.TouchUpInside)
    }

    func didTapButton(button: UIButton) {
        self.eventHandler.didTapShowGreetingButton()
    }

    func setGreeting(greeting: String) {
        self.greetingLabel.text = greeting
    }

    // layout code goes here
}
// Assembling of VIPER module, without Router
let view = GreetingViewController()
let presenter = GreetingPresenter()
let interactor = GreetingInteractor()
view.eventHandler = presenter
presenter.view = view
presenter.greetingProvider = interactor
interactor.output = presenter
```

Section 199.4: MVC pattern

```
import UIKit

struct Person { // Model
    let firstName: String
    let lastName: String
}

class GreetingViewController : UIViewController { // View + Controller
    var person: Person!
    let showGreetingButton = UIButton()
    let greetingLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
        self.showGreetingButton.addTarget(self, action: "didTapButton:", forControlEvents:
.TouchUpInside)
    }

    func didTapButton(button: UIButton) {
        let greeting = "Hello" + " " + self.person.firstName + " " + self.person.lastName
        self.greetingLabel.text = greeting

    }
    // layout code goes here
}
// Assembling of MVC
let model = Person(firstName: "David", lastName: "Blaine")
let view = GreetingViewController()
view.person = model
```

第200章：多播委托

为现有iOS控件添加多播功能的模式。添加多播可以提高清晰度
和代码重用性。

第200.1节：适用于任何控件的多播委托

通过委托将消息从一个对象转发给另一个对象，将这些消息多播给多个观察者。

步骤1： 创建 NSObject 类 RRMulticastDelegate

步骤2： 在 `RRMulticastDelegate.h` 文件中实现以下代码

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

@interface RRMulticastDelegate : NSObject

{
    //处理委托的多个观察者
    NSMutableArray* _delegates;
}

// 将委托方法的实现转发给观察者列表
- (void)addDelegate:(id)delegate;
- (void)removeDelegate:(id)delegate;

// 获取多个代理
-(NSArray *)delegatesObjects;

@end
```

步骤3： 在 RRMulticastDelegate. m 文件中实现以下代码

```
#import "RRMulticastDelegate.h"

@implementation RRMulticastDelegate

- (id)init
{
    if (self = [super init])
    {
_delegates = [NSMutableArray array];
    }
    return self;
}

-(NSArray *)delegatesObjects
{
    return _delegates;
}

- (void)removeDelegate:(id)delegate
{
    if ([_delegates containsObject:delegate])
        [_delegates removeObject:delegate];
}

- (void)addDelegate:(id)delegate
```


Chapter 200: Multicast Delegates

Pattern for adding multicasting capabilities to existing iOS controls. Adding multicasting allows for improved clarity and code re-use.

Section 200.1: Multicast delegates for any controls

Forward messages one object to another by delegates, multicasting these messages to multiple observers..

Step 1: Create `NSObject` **class** of `RRMulticastDelegate`

Step 2: Following code implement in `RRMulticastDelegate.h` file

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

@interface RRMulticastDelegate : NSObject

{
    //Handle multiple observers of delegate
    NSMutableArray* _delegates;
}

// Delegate method implementation to the list of observers
- (void)addDelegate:(id)delegate;
- (void)removeDelegate:(id)delegate;

// Get multiple delegates
-(NSArray *)delegatesObjects;

@end
```

Step 3: Following code implement in `RRMulticastDelegate.m` file

```
#import "RRMulticastDelegate.h"

@implementation RRMulticastDelegate

- (id)init
{
    if (self = [super init])
    {
        _delegates = [NSMutableArray array];
    }
    return self;
}

-(NSArray *)delegatesObjects
{
    return _delegates;
}

- (void)removeDelegate:(id)delegate
{
    if ([_delegates containsObject:delegate])
        [_delegates removeObject:delegate];
}

- (void)addDelegate:(id)delegate
```

```
{
    if (![_delegates containsObject:delegate])
        [_delegates addObject:delegate];
}

- (BOOL)respondsToSelector:(SEL)aSelector
{
    if ([super respondsToSelector:aSelector])
        return YES;

    // 如果任何代理响应此选择器，则返回YES
    for(id delegate in _delegates)
    {
        if (!delegate)
            continue;

        if ([delegate respondsToSelector:aSelector])
        {
            return YES;
        }
    }
    return NO;
}

- (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector
{
    // 这个类能创建签名吗？
    NSMethodSignature* signature = [super methodSignatureForSelector:aSelector];

    // 如果不能，尝试我们的代理
    if (!signature)
    {
        for(id delegate in _delegates)
        {
            if (!delegate)
                continue;

            if ([delegate respondsToSelector:aSelector])
            {
                return [delegate methodSignatureForSelector:aSelector];
            }
        }
    }
    return signature;
}

- (void)forwardInvocation:(NSInvocation *)anInvocation
{
    // 将调用转发给每个代理
    for(id delegate in _delegates)
    {
        if (!delegate)
            continue;

        if ([delegate respondsToSelector:[anInvocation selector]])
        {
            [anInvocation invokeWithTarget:delegate];
        }
    }
}

@end
```



```
{
    if (![_delegates containsObject:delegate])
        [_delegates addObject:delegate];
}

- (BOOL)respondsToSelector:(SEL)aSelector
{
    if ([super respondsToSelector:aSelector])
        return YES;

    // if any of the delegates respond to this selector, return YES
    for(id delegate in _delegates)
    {
        if (!delegate)
            continue;

        if ([delegate respondsToSelector:aSelector])
        {
            return YES;
        }
    }
    return NO;
}

- (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector
{
    // can this class create the sinature?
    NSMethodSignature* signature = [super methodSignatureForSelector:aSelector];

    // if not, try our delegates
    if (!signature)
    {
        for(id delegate in _delegates)
        {
            if (!delegate)
                continue;

            if ([delegate respondsToSelector:aSelector])
            {
                return [delegate methodSignatureForSelector:aSelector];
            }
        }
    }
    return signature;
}

- (void)forwardInvocation:(NSInvocation *)anInvocation
{
    // forward the invocation to every delegate
    for(id delegate in _delegates)
    {
        if (!delegate)
            continue;

        if ([delegate respondsToSelector:[anInvocation selector]])
        {
            [anInvocation invokeWithTarget:delegate];
        }
    }
}

@end
```

步骤4： 创建 NSObject 类别类 RRProperty

步骤5： 在 `NSObject+RRProperty.h` 文件中实现以下代码

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

#import "RRMulticastDelegate.h"

@interface NSObject (RRProperty)<UITextFieldDelegate,UITableViewDataSource>

-(void)setObject:(id)block forKey:(NSString *)key;
-(id)objectForKey:(NSString *)key;

#pragma mark - 多播代理

- (RRMulticastDelegate *)multicastDelegate;
- (RRMulticastDelegate *)multicastDatasource;

-(void)addDelegate:(id)delegate;
-(void)addDataSource:(id)datasource;

@end
```

步骤6： 以下代码实现于NSObject+RRProperty.m文件中

```
#import "NSObject+RRProperty.h"

#import <objc/message.h>
#import <objc/runtime.h>

#pragma GCC diagnostic ignored "-Wprotocol"

static NSString *const MULTICASTDELEGATE = @"MULTICASTDELEGATE";
static NSString *const MULTICASTDATASOURCE = @"MULTICASTDATASOURCE";

@implementation NSObject (RRProperty)

-(void)setObject:(id)block forKey:(NSString *)key
{
objc_setAssociatedObject(self, (__bridge const void *)(key), block, OBJC_ASSOCIATION_RETAIN);
}

-(id)objectForKey:(NSString *)key
{
    return objc_getAssociatedObject(self, (__bridge const void *)(key));
}

#pragma mark - 多播代理

- (RRMulticastDelegate *)multicastDelegate
{
id multicastDelegate = [self objectForKey:MULTICASTDELEGATE];
    if (multicastDelegate == nil) {
multicastDelegate = [[RRMulticastDelegate alloc] init];

        [self setObject:multicastDelegate forKey:MULTICASTDELEGATE];
    }

    return multicastDelegate;
}
```


Step 4: Create `NSObject` **category class** of `RRProperty`

Step 5: Following code implement in `NSObject+RRProperty.h` file

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

#import "RRMulticastDelegate.h"

@interface NSObject (RRProperty)<UITextFieldDelegate,UITableViewDataSource>

-(void)setObject:(id)block forKey:(NSString *)key;
-(id)objectForKey:(NSString *)key;

#pragma mark - Multicast Delegate

- (RRMulticastDelegate *)multicastDelegate;
- (RRMulticastDelegate *)multicastDatasource;

-(void)addDelegate:(id)delegate;
-(void)addDataSource:(id)datasource;

@end
```

Step 6: Following code implement in `NSObject+RRProperty.m` file

```
#import "NSObject+RRProperty.h"

#import <objc/message.h>
#import <objc/runtime.h>

#pragma GCC diagnostic ignored "-Wprotocol"

static NSString *const MULTICASTDELEGATE = @"MULTICASTDELEGATE";
static NSString *const MULTICASTDATASOURCE = @"MULTICASTDATASOURCE";

@implementation NSObject (RRProperty)

-(void)setObject:(id)block forKey:(NSString *)key
{
    objc_setAssociatedObject(self, (__bridge const void *)(key), block, OBJC_ASSOCIATION_RETAIN);
}

-(id)objectForKey:(NSString *)key
{
    return objc_getAssociatedObject(self, (__bridge const void *)(key));
}

#pragma mark - Multicast Delegate

- (RRMulticastDelegate *)multicastDelegate
{
    id multicastDelegate = [self objectForKey:MULTICASTDELEGATE];
    if (multicastDelegate == nil) {
        multicastDelegate = [[RRMulticastDelegate alloc] init];

        [self setObject:multicastDelegate forKey:MULTICASTDELEGATE];
    }

    return multicastDelegate;
}
```

```
- (RRMulticastDelegate *)multicastDatasource
{
id multicastDatasource = [self objectForKey:MULTICASTDATASOURCE];
    if (multicastDatasource == nil) {
multicastDatasource = [[RRMulticastDelegate alloc] init];

        [self setObject:multicastDatasource forKey:MULTICASTDATASOURCE];
    }

    return multicastDatasource;
}

-(void)addDelegate:(id)delegate
{
    [self.multicastDelegate addDelegate:delegate];

    UITextField *text = (UITextField *) self;
    text.delegate = self.multicastDelegate;
}

-(void)addDataSource:(id)datasource
{
    [self.multicastDatasource addDelegate:datasource];
    UITableView *text = (UITableView *) self;
text.dataSource = self.multicastDatasource;
}

@end
```

最后你可以对任何控件使用多播代理...

例如...

在NSObject+RRProperty.h文件中导入你的视图控制器类，以访问其方法来设置多播代理/数据源。

```
UITextView *txtView = [[UITextView alloc]initWithFrame:txtframe];
[txtView addDelegate:self];

UITableView *tblView = [[UITableView alloc]initWithFrame:tblframe];
[tblView addDelegate:self];
[tblView addDataSource:self];
```



```
- (RRMulticastDelegate *)multicastDatasource
{
    id multicastDatasource = [self objectForKey:MULTICASTDATASOURCE];
    if (multicastDatasource == nil) {
        multicastDatasource = [[RRMulticastDelegate alloc] init];

        [self setObject:multicastDatasource forKey:MULTICASTDATASOURCE];
    }

    return multicastDatasource;
}

-(void)addDelegate:(id)delegate
{
    [self.multicastDelegate addDelegate:delegate];

    UITextField *text = (UITextField *) self;
    text.delegate = self.multicastDelegate;
}

-(void)addDataSource:(id)datasource
{
    [self.multicastDatasource addDelegate:datasource];
    UITableView *text = (UITableView *) self;
    text.dataSource = self.multicastDatasource;
}

@end
```

Finally you ca use multicast dalegate for any controls...

For ex...

Import your viewcontroller class in `NSObject+RRProperty.h` file to access its methods to **set multicast delegate/datasource**.

```
UITextView *txtView = [[UITextView alloc]initWithFrame:txtframe];
[txtView addDelegate:self];

UITableView *tblView = [[UITableView alloc]initWithFrame:tblframe];
[tblView addDelegate:self];
[tblView addDataSource:self];
```

第201章：使用图像资源

图像资源用于在我们的iOS应用中通过Xcode管理和组织不同类型的图像资源。

这些资源可以是**应用图标、启动图像、应用中使用的图像、全尺寸图像、随机尺寸图像**等。

第201.1节：使用图像资源的启动图像

启动屏幕是在启动应用时出现的屏幕，持续到应用的第一个屏幕出现为止。
在这里了解更多关于启动屏幕及其指南的信息。

类似于应用图标，我们必须在项目设置中说明使用图像资源作为启动屏幕图像。
默认项目设置如下：

我们必须改成这样：

一旦我们更改这些设置，Xcode 会自动提示我们迁移到资产并在资产中创建 LaunchImage 文件，如下所示：

Chapter 201: Using Image Aseets

Image assets are used to manage and organize different types of image assets in our iOS app using Xcode.

These assets can be **App Icons, Launch Images, images used throughout the app, full size images, random sized images** etc.

Section 201.1: LaunchImage using Image Assets

Launch screen is a screen which appears while launching app and lasts till first screen of app appears.
Learn more about Launch Screen and guidelines here.

Similar to AppIcons we have to mention in project settings about using image assets for launch screen image.
By default project settings are like:

We have to change to like this:

Once we change these settings, Xcode will asks us to migrate to assets and create LaunchImage file in assets automatically as:

如果未创建，我们可以通过点击底部的+按钮手动创建，如下所示：

If not created, we can manually create one by clicking **+** button at the bottom as:

之后，根据我们的需求，可以通过属性检查器勾选/取消勾选框来将空白框更改为我们支持的设备。

我为屏幕尺寸从4英寸到5.5英寸的iPhone以及所有iPad填充了这些图片，如下所示：

After this, according to our requirement we can change the empty boxes to devices which we support using attributes inspector by checking/unchecking boxes.

I filled these images for iPhones of 4" screen to 5.5" and for all iPads as:

以下是所有启动图片的尺寸：

```
Retina HD5.5英寸 iPhone 竖屏 - iPhone（6、6S、7）Plus - 1242x2208像素
Retina HD 4.7" iPhone 竖屏 - iPhone 6、6S、7 - 750x1334像素
Retina HD 5.5 英寸 iPhone 横屏 - iPhone (6, 6S, 7) Plus - 2208x1242像素
2倍 iPhone 竖屏 - (3.5") iPhone 4S - 640x960像素
Retina 4 英寸 iPhone 竖屏 - (4") iPhone 5, 5S, 5C, iPod Touch, SE - 640x1136像素
2倍 iPad 竖屏 - 所有 Retina iPad - 1536x2048像素
2倍 iPad 横屏 - 所有 Retina iPad - 2048x1536像素
```

注意：

1 非 Retina iPad: 我留空了 1倍 iPad 竖屏和横屏 因为非 Retina iPad 会通过缩放使用 2倍 启动画面图片

2 12.9英寸 iPad Pro: 这个 iPad 没有对应的方形图标，因为该 iPad 也会通过缩放使用 2倍 iPad 图片

3 Retina HD 5.5英寸: iPad 应该使用 1920x1080像素 竖屏和 1080x1920像素 横屏，但 Xcode 会给出警告，且启动画面不会在这些设备上显示

4 **分屏视图:** 由于我们使用的是 `LaunchImage Asset` 而非 `LaunchScreen XIB`，我们的应用将不支持 iPad 和 5.5英寸横屏 iPhone 的 分屏视图功能

5 **重新安装:** 如果我们的应用已经安装在设备上，并尝试使用这些新添加的启动画面资源运行应用，有时设备启动应用时不会显示启动画面。此时只需从设备删除应用，清理+构建项目并运行，即可显示新的启动画面

第201.2节：使用图像资源的应用图标

每当我们在 Xcode 中为新应用创建新项目时，它会提供各种内置类、目标、测试、plist 文件等。同样，它还会提供一个 `Assets.xcassets` 文件，用于管理项目中的所有图像资源。

这是该文件在文件导航器中的显示方式：

Here are sizes of all launch images:

```
Retina HD 5.5" iPhone Portrait - iPhone (6, 6S, 7)Plus - 1242x2208px
Retina HD 4.7" iPhone Portrait - iPhone 6, 6S, 7 - 750x1334px
Retina HD 5.5" iPhone Landscape - iPhone (6, 6S, 7)Plus - 2208x1242px
2x iPhone Portrait - (3.5") iPhone 4S - 640x960px
Retina 4 iPhone Portrait - (4") iPhone 5, 5S, 5C, iPod Touch, SE - 640x1136px
2x iPad Portrait - All Retina iPads - 1536x2048px
2x iPad Landscape - All Retina iPads - 2048x1536px
```

Notes:

1 **non-retina iPads:** I left blank `1x iPad Portrait and Landscape` because non-retina iPads will use `2x` launch images by scaling

2 **12.9" iPad Pro**: there is no square for this iPad because this iPad will also use `2x iPad` images by scaling them

3 **Retina HD 5.5":** iPads should have `1920x1080px` for portrait and `1080x1920px` for landscape but Xcode will give waring and launch image will not be shown on those devices

4 **SplitView:** as we are using `LaunchImage Asset` instead of `LaunchScreen XIB`, our app will not support `SplitView` on iPads and landscape 5.5" iPhones

5 **Reinstall:** if our app is already installed on device and we try to run with these newly added launch image assets, then sometimes device will not show launch images while launching app. In this case just delete app from device, clean+build project and run it, it'll show new launch images

Section 201.2: App Icon using Image Assets

Whenever we create a new project in Xcode for our new app, it gives us various in-built classes, targets, tests, plist file, etc. Similarly it also gives us as `Assets.xcassets` file, which manages all the image assets in our project.

This is how this file looks like in file navigator:

如果我们点击它，它会是这样的：

正如我所说，AppIcon 资源已经为我们创建好了。
我们只需将相应的图片拖放到每个空白的方块中。每个黑色方块会告诉我们图片的尺寸，尺寸写在方块正下方。

将所有图片拖放到所有方块后，它会是这样的：

If we click it, it'll look like this:

As I said, `AppIcon` asset is already created for us.
We just have to **drag and drop** respective image on each empty square block. Each black will tell us what size that image should be, it's written just below it.
After dragging and dropping all the images in all the squares, it'll look like this:

我们也可以在实用工具 -> 属性检查器中更改设备设置，针对图标资源，设置如下：

We can change the devices setting also for icon assets in Utilities -> Attributes Inspector as:

完成这些后，只需运行应用程序，我们就会有一个漂亮的应用图标，如下所示：

它默认存在，但如果没有，请确保在目标 -> 常规设置中此设置如下：

Once we finished this, just run an app and we'll be having nice icon to app as this:

It is there by default, but if it's not then make sure this settings is as in Target->General settings:

App Icons and Launch Images

App Icons Source AppIcon

App Icons and Launch Images

App Icons Source AppIcon

第202章：Objective-C中的运行时

第202.1节：使用关联对象

当你想为现有类添加需要保存状态的功能时，关联对象非常有用。

例如，给每个UIView添加一个活动指示器：

Objective-C实现

```
#import <objc/runtime.h>

static char ActivityIndicatorKey;

@implementation UIView (ActivityIndicator)

- (UIActivityIndicatorView *)activityIndicator {
    return (UIActivityIndicatorView *)objc_getAssociatedObject(self, &ActivityIndicatorKey);
}

- (void)setActivityIndicator: (UIActivityIndicatorView *)activityIndicator {
    objc_setAssociatedObject(self, &ActivityIndicatorKey, activityIndicator,
OBJC_ASSOCIATION_RETAIN_NONATOMIC);
}

- (void)showActivityIndicator {
    UIActivityIndicatorView *activityIndicator = [[UIActivityIndicatorView alloc]
initWithActivityIndicatorStyle: UIActivityIndicatorViewStyleGray];

    [self setActivityIndicator:activityIndicator];

activityIndicator.center = self.center;
    activityIndicator.autoresizingMask = UIViewAutoresizingFlexibleTopMargin |
UIViewAutoresizingFlexibleLeftMargin | UIViewAutoresizingFlexibleRightMargin |
UIViewAutoresizingFlexibleBottomMargin;

    [activityIndicator startAnimating];

    [self addSubview: activityIndicator];
}

- (void)hideActivityIndicator {
    UIActivityIndicatorView * activityIndicator = [self activityIndicator];

    if (activityIndicator != nil) {
        [[self activityIndicator] removeFromSuperview];
    }
}

@end
```

你也可以通过 Swift 访问 Objective-C 运行时：

Swift 代码

```
extension UIView {
    private struct AssociatedKeys {
        static var activityIndicator = "UIView.ActivityIndicatorView"
    }
```


Chapter 202: Runtime in Objective-C

Section 202.1: Using Associated Objects

Associated objects are useful when you want to add functionality to existing classes which requires holding state.

For example, adding a activity indicator to every UIView:

Objective-C Implementation

```
#import <objc/runtime.h>

static char ActivityIndicatorKey;

@implementation UIView (ActivityIndicator)

- (UIActivityIndicatorView *)activityIndicator {
    return (UIActivityIndicatorView *)objc_getAssociatedObject(self, &ActivityIndicatorKey);
}

- (void)setActivityIndicator: (UIActivityIndicatorView *)activityIndicator {
    objc_setAssociatedObject(self, &ActivityIndicatorKey, activityIndicator,
OBJC_ASSOCIATION_RETAIN_NONATOMIC);
}

- (void)showActivityIndicator {
    UIActivityIndicatorView *activityIndicator = [[UIActivityIndicatorView alloc]
initWithActivityIndicatorStyle: UIActivityIndicatorViewStyleGray];

    [self setActivityIndicator:activityIndicator];

    activityIndicator.center = self.center;
    activityIndicator.autoresizingMask = UIViewAutoresizingFlexibleTopMargin |
UIViewAutoresizingFlexibleLeftMargin | UIViewAutoresizingFlexibleRightMargin |
UIViewAutoresizingFlexibleBottomMargin;

    [activityIndicator startAnimating];

    [self addSubview: activityIndicator];
}

- (void)hideActivityIndicator {
    UIActivityIndicatorView * activityIndicator = [self activityIndicator];

    if (activityIndicator != nil) {
        [[self activityIndicator] removeFromSuperview];
    }
}

@end
```

You can also access the Objective-C runtime through Swift:

Swift Code

```
extension UIView {
    private struct AssociatedKeys {
        static var activityIndicator = "UIView.ActivityIndicatorView"
    }
```

```
    private var activityIndicatorView: UIActivityIndicatorView? {
        get {
                        return objc_getAssociatedObject(self, &AssociatedKeys.activityIndicator) as?
UIActivityIndicatorView
        }
        set (activityIndicatorView) {
objc_setAssociatedObject(self, &AssociatedKeys.activityIndicator,
activityIndicatorView, .OBJC_ASSOCIATION_RETAIN_NONATOMIC)
        }
    }

      func showActivityIndicator() {
activityIndicatorView = UIActivityIndicatorView(activityIndicatorStyle: .gray)
        activityIndicatorView.center = center
activityIndicatorView.autoresizingMask = [.flexibleLeftMargin, .flexibleRightMargin,
.flexibleTopMargin, .flexibleBottomMargin]

activityIndicatorView.startAnimating()

        addSubview(activityIndicatorView)
    }

    func hideActivityIndicator() {
        activityIndicatorView.removeFromSuperview()
    }
}
```



```
    private var activityIndicatorView: UIActivityIndicatorView? {
        get {
            return objc_getAssociatedObject(self, &AssociatedKeys.activityIndicator) as?
UIActivityIndicatorView
        }
        set (activityIndicatorView) {
            objc_setAssociatedObject(self, &AssociatedKeys.activityIndicator,
activityIndicatorView, .OBJC_ASSOCIATION_RETAIN_NONATOMIC)
        }
    }

     func showActivityIndicator() {
        activityIndicatorView = UIActivityIndicatorView(activityIndicatorStyle: .gray)
        activityIndicatorView.center = center
        activityIndicatorView.autoresizingMask = [.flexibleLeftMargin, .flexibleRightMargin,
.flexibleTopMargin, .flexibleBottomMargin]

        activityIndicatorView.startAnimating()

        addSubview(activityIndicatorView)
    }

    func hideActivityIndicator() {
        activityIndicatorView.removeFromSuperview()
    }
}
```

第203章：模型展示样式

模态展示样式用于从一个视图控制器过渡到另一个视图控制器。有两种实现此自定义的方法。一种是通过代码，另一种是通过界面构建器（使用segue）。
此效果是通过将modalPresentationStyle变量设置为UIModalPresentationStyle枚举的一个实例来实现的。modalPresentationStyle属性是UIViewController的一个类变量，用于指定视图控制器在屏幕上的展示方式。

第203.1节：使用界面构建器探索ModalPresentationStyle

这是一个非常基础的应用程序，用于演示iOS中不同的ModalPresentationStyle。根据此处找到的文档，UIModalPresentationStyle有9种不同的取值，具体如下，

1.全屏（fullScreen）
2.页面表（pageSheet）
3.表单表（formSheet）
4.当前上下文（currentContext）
5.自定义（custom）
6.覆盖全屏（overFullScreen）
7.覆盖当前上下文（overCurrentContext）
8.弹出框（popover）
9.无

要设置一个项目，只需创建一个普通的 iOS 项目并添加两个ViewControllers。在你的初始ViewController中放置一个UIButton，并通过Target -> Action机制将其连接到第二个ViewController。为了区分两个`ViewControllers`，设置ViewController中UIView的背景属性为其他颜色。如果一切顺利，你的Interface Builder 应该看起来像这样，

Chapter 203: ModelPresentationStyles

Modal Presentation styles are used when you are transitioning from one view controller to another. There are 2 ways of achieving this customization. One is through code and another through Interface Builder(using segues). This effect is achieved by setting`modalPresentationStyle` variable to an instance of `UIModalPresentationStyle` enum. `modalPresentationStyle` property is a class variable of `UIViewController` and is used to specify how a `ViewController` is presented on screen.

Section 203.1: Exploring ModalPresentationStyle using Interface Builder

This will be a very basic app which will illustrate different `ModalpresentationStyle` in iOS. According to documentation found here, There are 9 different values for `UIModalPresentationStyle` which are as follows,

1. `fullScreen`
2. `pageSheet`
3. `formSheet`
4. `currentContext`
5. `custom`
6. `overFullScreen`
7. `overCurrentContext`
8. `popover`
9. `none`

To setup a project, just create a normal iOS project and add 2 `ViewControllers`. Put a `UIButton` in you initial `ViewController` and connect it to 2nd `ViewController` via a `Target -> Action` mechanism. To distinguish both `ViewControllers`, set background property of `UIView` in `ViewController` some other color. If all goes well, your Interface Builder should look something this,

确保你构建此项目并在iPad上运行（关于为什么选择 iPad 的详细信息，请参阅备注部分）。设置好项目后，选择 segue 并进入attributes inspector。你应该能看到

Make sure you build this project and run it on **iPad** (For details on why iPad, refer to Remarks section). Once you are done setting up your project, select the segue and go to the `attributes inspector`. You should be able to see

类似这样的内容，

将 kind 属性设置为Present Modally。

现在，在这个示例中我们看不到所有效果，因为其中一些需要少量代码。

让我们从fullscreen开始。当你在Kind标签中选择Present Modally时，这个效果默认被选中。构建并运行后，第二个ViewController将占据你 iPad 的全屏。

something like this,

Set the kind property to `Present Modally`.

Now, we won't see all of the effects in this example as some of them requires little bit of code.

Let's start with `fullscreen`. This effect is selected by default when you select `Present Modally` in `Kind` tab. When you build and run, the 2nd `ViewController` would occupy the full screen of you iPad.

接下来是pageSheet。您可以从Presentation选项卡中选择此选项。在此选项中，当设备处于纵向时

Next is `pageSheet`. You can select this option from `Presentation` tab. In this option, when device is in portrait

模式下，第二个ViewController类似于全屏，但为横屏模式，第二个ViewController的宽度远小于设备宽度。未被第二个ViewController覆盖的内容将被遮暗。

对于formSheet样式，第二个ViewController位于设备中心，大小小于设备尺寸。
当设备处于横屏模式且键盘可见时，视图位置会向上调整以显示ViewController。

mode, the 2nd `ViewController` is similar to full screen but in landscape mode, 2nd `ViewController` is much narrow the device width. Also, any content not covered by 2nd `ViewController` will be dimmed.

For `formSheet` style, the 2nd `ViewController` is placed in center of device and the size is smaller to that of device. Also when device is in landscape mode and keyboard is visible position of view is adjusted upwards to show the `ViewController`.

我们将尝试的最后一种样式是popover。要选择此样式，请在Kind标签中选择Present as Popover。第二个

Last style which we are going to try is `popover`. To select this style, select `Present as Popover` in `Kind` tab. The 2nd

ViewController以小型弹出视图形式呈现（大小可设置）。背景内容会被遮暗。点击弹出视图外部任意位置将关闭弹出视图。你的Attributes Inspector应类似如下，

Anchor是你希望弹出视图箭头指向的UI元素。Directions是允许弹出视图Anchor指向的方向。

`ViewController` is presented as a small popover(size can be set). The background content is dimmed. Any tap outside the popover would dismiss the popover. Your `Attributes Inspector` should look something like this,

`Anchor` is the UI element to which you want your popover arrow to point. `Directions` are the directions you allow your popover `Anchor` to point in.

除了这些基本的模态展示样式外，还有更多样式，但它们实现起来比较复杂，需要一些代码。

There more then these basic Modal Presentation Styles but they are little complicated to achieve and require some

更多细节可以在苹果文档中找到。

code. More details can be found in the Apple Documentation.

第204章：CydiaSubstrate插件

学习如何为越狱的iPhone创建CydiaSubstrate插件。

这些插件将使你能够修改操作系统的行为，使其按照你希望的方式运行。

第204.1节：使用Theos创建新插件

使用nic创建一个新项目

在终端中输入此命令

```
$THEOS/bin/nic.pl
```

```
NIC 2.0 - 新实例创建器
------------------------------
[1.] iphone/activator_event
[2.] iphone/application_modern
[3.] iphone/cydget
[4.] iphone/flipswitch_switch
[5.] iphone/framework
[6.] iphone/ios7_notification_center_widget
[7.] iphone/library
[8.] iphone/notification_center_widget
[9.] iphone/preference_bundle_modern
[10.] iphone/tool
[11.] iphone/tweak
[12.] iphone/xpc_service
请选择一个模板（必填）：
```

选择模板 `[11.] iphone/tweak`

填写详细信息后，将创建以下文件：

```
-rw-r--r--@  1 gkpln3  staff   214B 6月 12 15:09 Makefile
-rw-r--r--@  1 gkpln3  staff    89B 6月 11 22:58 TorchonFocus.plist
-rw-r--r--   1 gkpln3  staff   2.7K 6月 12 16:10 Tweak.xm
-rw-r--r--   1 gkpln3  staff   224B 6月 11 16:17 control
drwxr-xr-x   3 gkpln3  staff   102B 6月 11 16:18 obj
drwxr-xr-x  16 gkpln3  staff   544B 6月 12 16:12 packages
```

重写 iOS 截图保存方法

使用你喜欢的代码编辑器打开`Tweak.xm`文件。

钩住操作系统中的某个方法。

```
%hook SBScreenShotter
- (void)saveScreenshot:(BOOL)screenshot
{
    %orig;
NSLog(@"saveScreenshot: 被调用");
}
%end
```

注意你可以选择是否调用原始函数，例如：

Chapter 204: CydiaSubstrate tweak

Learn how to create cydia substrate tweaks for jailbroken iPhones.

Those tweaks will enable you to modify the operating system's behavior to act the way you would like it to.

Section 204.1: Create new tweak using Theos

Use nic to create a new project

Enter this command in your terminal

```
$THEOS/bin/nic.pl
```

```
NIC 2.0 - New Instance Creator
------------------------------
[1.] iphone/activator_event
[2.] iphone/application_modern
[3.] iphone/cydget
[4.] iphone/flipswitch_switch
[5.] iphone/framework
[6.] iphone/ios7_notification_center_widget
[7.] iphone/library
[8.] iphone/notification_center_widget
[9.] iphone/preference_bundle_modern
[10.] iphone/tool
[11.] iphone/tweak
[12.] iphone/xpc_service
Choose a Template (required):
```

Choose template `[11.] iphone/tweak`

Fill in the details and you will get the following files created:

```
-rw-r--r--@  1 gkpln3  staff   214B Jun 12 15:09 Makefile
-rw-r--r--@  1 gkpln3  staff    89B Jun 11 22:58 TorchonFocus.plist
-rw-r--r--   1 gkpln3  staff   2.7K Jun 12 16:10 Tweak.xm
-rw-r--r--   1 gkpln3  staff   224B Jun 11 16:17 control
drwxr-xr-x   3 gkpln3  staff   102B Jun 11 16:18 obj
drwxr-xr-x  16 gkpln3  staff   544B Jun 12 16:12 packages
```

Override iOS screenshots saving method

open the `Tweak.xm` file using your favorite code editor.

hook to a certain method from the operating system.

```
%hook SBScreenShotter
- (void)saveScreenshot:(BOOL)screenshot
{
    %orig;
    NSLog(@"saveScreenshot: is called");
}
%end
```

Note you can choose wether or not the original function should be called, for example:

```
%hook SBScreenShotter
- (void)saveScreenshot:(BOOL)screenshot
{
NSLog(@"saveScreenshot: 被调用");
}
%end
```

将覆盖该函数而不调用原始函数，因此导致截图无法保存。


```
%hook SBScreenShotter
- (void)saveScreenshot:(BOOL)screenshot
{
    NSLog(@"saveScreenshot: is called");
}
%end
```

will override the function without calling the original one, thus casing screenshots not being saved.

第205章：从图像创建视频

使用AVFoundation从图像创建视频

第205.1节：从UIImage创建视频

首先你需要创建AVAssetWriter

```
NSError *error = nil;
NSURL *outputURL = <#表示你想保存视频的URL的NSURL对象#>;
AVAssetWriter *assetWriter = [AVAssetWriter assetWriterWithURL:outputURL
fileType:AVFileTypeQuickTimeMovie error:&error];
if (!assetWriter) {
    // 处理错误
}
```

AVAssetWriter至少需要一个资产写入输入。

```
 NSDictionary *writerInputParams = [NSDictionary dictionaryWithObjectsAndKeys:
AVVideoCodecH264, AVVideoCodecKey,
                                                [NSNumber numberWithInt:renderSize.width],
AVVideoWidthKey,
                                                [NSNumber numberWithInt:renderSize.height],
AVVideoHeightKey,
AVVideoScalingModeResizeAspectFill,
AVVideoScalingModeKey,
                                                 nil];

AVAssetWriterInput *assetWriterInput = [AVAssetWriterInput
assetWriterInputWithMediaType:AVMediaTypeVideo outputSettings:writerInputParams];
 if ([assetWriter canAddInput:assetWriterInput]) {
     [assetWriter addInput:assetWriterInput];
 } else {
     // 显示错误信息
 }
```

要向 AVAssetWriterInput 添加 CVPixelBufferRef，需要创建 `AVAssetWriterInputPixelBufferAdaptor`

```
NSDictionary *attributes = [NSDictionary dictionaryWithObjectsAndKeys:
                            [NSNumber numberWithUnsignedInt:kCVPixelFormatType_32ARGB],
(NSString*)kCVPixelBufferPixelFormatTypeKey,
                            [NSNumber numberWithBool:YES], (NSString
*)kCVPixelBufferCGImageCompatibilityKey,
                            [NSNumber numberWithBool:YES], (NSString
*)kCVPixelBufferCGBitmapContextCompatibilityKey,
                            nil];
AVAssetWriterInputPixelBufferAdaptor *writerAdaptor = [AVAssetWriterInputPixelBufferAdaptor
assetWriterInputPixelBufferAdaptorWithAssetWriterInput:assetWriterInput
sourcePixelBufferAttributes:attributes];
```

现在我们可以开始写入了

```
[assetWriter startWriting];
[assetWriter startSessionAtSourceTime:kCMTimeZero];
[assetWriterInput requestMediaDataWhenReadyOnQueue:exportingQueue usingBlock:^{
    for (int i = 0; i < images.count; ++i) {
        while (![assetWriterInput isReadyForMoreMediaData]) {
```


Chapter 205: Create a video from images

Create a video from images using AVFoundation

Section 205.1: Create Video from UIImages

First of all you need to create `AVAssetWriter`

```
NSError *error = nil;
NSURL *outputURL = <#NSURL object representing the URL where you want to save the video#>;
AVAssetWriter *assetWriter = [AVAssetWriter assetWriterWithURL:outputURL
fileType:AVFileTypeQuickTimeMovie error:&error];
if (!assetWriter) {
    // handle error
}
```

`AVAssetWriter` needs at least one asset writer input.

```
 NSDictionary *writerInputParams = [NSDictionary dictionaryWithObjectsAndKeys:
                                                AVVideoCodecH264, AVVideoCodecKey,
                                                [NSNumber numberWithInt:renderSize.width],
AVVideoWidthKey,
                                                [NSNumber numberWithInt:renderSize.height],
AVVideoHeightKey,
                                                AVVideoScalingModeResizeAspectFill,
AVVideoScalingModeKey,
                                                nil];

 AVAssetWriterInput *assetWriterInput = [AVAssetWriterInput
assetWriterInputWithMediaType:AVMediaTypeVideo outputSettings:writerInputParams];
 if ([assetWriter canAddInput:assetWriterInput]) {
     [assetWriter addInput:assetWriterInput];
 } else {
     // show error message
 }
```

To append `CVPixelBufferRef`'s to `AVAssetWriterInput` we need to create `AVAssetWriterInputPixelBufferAdaptor`

```
NSDictionary *attributes = [NSDictionary dictionaryWithObjectsAndKeys:
                            [NSNumber numberWithUnsignedInt:kCVPixelFormatType_32ARGB],
(NSString*)kCVPixelBufferPixelFormatTypeKey,
                            [NSNumber numberWithBool:YES], (NSString
*)kCVPixelBufferCGImageCompatibilityKey,
                            [NSNumber numberWithBool:YES], (NSString
*)kCVPixelBufferCGBitmapContextCompatibilityKey,
                            nil];
AVAssetWriterInputPixelBufferAdaptor *writerAdaptor = [AVAssetWriterInputPixelBufferAdaptor
assetWriterInputPixelBufferAdaptorWithAssetWriterInput:assetWriterInput
sourcePixelBufferAttributes:attributes];
```

Now we can start writing

```
[assetWriter startWriting];
[assetWriter startSessionAtSourceTime:kCMTimeZero];
[assetWriterInput requestMediaDataWhenReadyOnQueue:exportingQueue usingBlock:^{
    for (int i = 0; i < images.count; ++i) {
        while (![assetWriterInput isReadyForMoreMediaData]) {
```

```
            [NSThread sleepForTimeInterval:0.01];
            // 可以检查是否尝试避免创建无限循环
        }

        UIImage *uIImage = images[i];

CVPixelBufferRef buffer = NULL;
        CVReturn err = PixelBufferCreateFromImage(uIImage.CGImage, &buffer);
        if (err) {
            // 处理错误
        }

        // 帧持续时间是单张图像的持续时间，单位为秒
       CMTime presentationTime = CMTimeMakeWithSeconds(i * frameDuration, 0000000);

        [writerAdaptor appendPixelBuffer:buffer withPresentationTime:presentationTime];

CVPixelBufferRelease(buffer);
    }

    [assetWriterInput markAsFinished];
    [assetWriter finishWritingWithCompletionHandler:^{
        if (assetWriter.error) {
            // 显示错误信息
        } else {
            // outputURL
        }
    }];
}];
```

这是一个从CGImageRef获取CVPixelBufferRef的函数

```
CVReturn PixelBufferCreateFromImage(CGImageRef imageRef, CVPixelBufferRef *outBuffer) {
    CIContext *context = [CIContext context];
    CIImage *ciImage = [CIImage imageWithCGImage:imageRef];

    NSDictionary *attributes = [NSDictionary dictionaryWithObjectsAndKeys:
                                [NSNumber numberWithBool:YES], (NSString
*)kCVPixelBufferCGBitmapContextCompatibilityKey,
                                [NSNumber numberWithBool:YES], (NSString
*)kCVPixelBufferCGImageCompatibilityKey
                                ,nil];

CVReturn err = CVPixelBufferCreate(kCFAllocatorDefault, CGImageGetWidth(imageRef),
CGImageGetHeight(imageRef), kCVPixelFormatType_32ARGB, (__bridge CFDictionaryRef
_Nullable)(attributes), outBuffer);
    if (err) {
        return err;
    }

    if (outBuffer) {
        [context render:ciImage toCVPixelBuffer:*outBuffer];
    }

    return kCVReturnSuccess;
}
```



```
            [NSThread sleepForTimeInterval:0.01];
            // can check for attempts not to create an infinite loop
        }

        UIImage *uIImage = images[i];

        CVPixelBufferRef buffer = NULL;
        CVReturn err = PixelBufferCreateFromImage(uIImage.CGImage, &buffer);
        if (err) {
            // handle error
        }

        // frame duration is duration of single image in seconds
        CMTime presentationTime = CMTimeMakeWithSeconds(i * frameDuration, 1000000);

        [writerAdaptor appendPixelBuffer:buffer withPresentationTime:presentationTime];

        CVPixelBufferRelease(buffer);
    }

    [assetWriterInput markAsFinished];
    [assetWriter finishWritingWithCompletionHandler:^{
        if (assetWriter.error) {
            // show error message
        } else {
            // outputURL
        }
    }];
}];
```

Here is a function to get `CVPixelBufferRef` from `CGImageRef`

```
CVReturn PixelBufferCreateFromImage(CGImageRef imageRef, CVPixelBufferRef *outBuffer) {
    CIContext *context = [CIContext context];
    CIImage *ciImage = [CIImage imageWithCGImage:imageRef];

    NSDictionary *attributes = [NSDictionary dictionaryWithObjectsAndKeys:
                                [NSNumber numberWithBool:YES], (NSString
*)kCVPixelBufferCGBitmapContextCompatibilityKey,
                                [NSNumber numberWithBool:YES], (NSString
*)kCVPixelBufferCGImageCompatibilityKey
                                ,nil];

    CVReturn err = CVPixelBufferCreate(kCFAllocatorDefault, CGImageGetWidth(imageRef),
CGImageGetHeight(imageRef), kCVPixelFormatType_32ARGB, (__bridge CFDictionaryRef
_Nullable)(attributes), outBuffer);
    if (err) {
        return err;
    }

    if (outBuffer) {
        [context render:ciImage toCVPixelBuffer:*outBuffer];
    }

    return kCVReturnSuccess;
}
```

第206章：Codable

Codable 是在Xcode 9、iOS 11和Swift 4中新增的。Codable用于使你的数据类型可编码和可解码，以便与JSON等外部表示兼容。

Codable用于支持编码和解码，声明遵循Codable协议，该协议结合了Encodable和Decodable协议。这个过程称为使你的类型可编码。

第206.1节：在Swift 4中使用Codable与JSONEncoder和JSONDecoder

让我们以电影结构体为例，这里我们将结构体定义为Codable。因此，我们可以轻松地对其进行编码和解码。

```
struct Movie: Codable {
    enum MovieGenere: String, Codable {
        case horror, skifi, comedy, adventure, animation
    }

    var name : String
    var moviesGenere : [MovieGenere]
    var rating : Int
}
```

我们可以像下面这样创建一个电影对象：

```
let upMovie = Movie(name: "Up", moviesGenere: [.comedy , .adventure, .animation], rating : 4)
```

upMovie 包含名称“Up”，其电影类型为喜剧、冒险和动画，评分为4（满分5分）。

编码

JSONEncoder 是一个将数据类型实例编码为 JSON 对象的对象。JSONEncoder 支持 Codable 对象。

```
// 编码数据
let jsonEncoder = JSONEncoder()
do {
    let jsonData = try jsonEncoder.encode(upMovie)
    let jsonString = String(data: jsonData, encoding: .utf8)
    print("JSON String : " + jsonString!)
}
catch {
}
```

JSONEncoder 会给我们 JSON 数据，用于获取 JSON 字符串。

输出字符串将会是：

```
{
  "name": "Up",
  "moviesGenere": [
    "comedy",
    "adventure",
    "animation"
```


Chapter 206: Codable

Codable is added with Xcode 9, iOS 11 and Swift 4. Codable is used to make your data types encodable and decodable for compatibility with external representations such as JSON.

Codable use to support both encoding and decoding, declare conformance to Codable, which combines the Encodable and Decodable protocols. This process is known as making your types codable.

Section 206.1: Use of Codable with JSONEncoder and JSONDecoder in Swift 4

Let's Take an Example with Structure of Movie, here we have defined the structure as Codable. So, We can encode and decode it easily.

```
struct Movie: Codable {
    enum MovieGenere: String, Codable {
        case horror, skifi, comedy, adventure, animation
    }

    var name : String
    var moviesGenere : [MovieGenere]
    var rating : Int
}
```

We can create a object from movie like as:

```
let upMovie = Movie(name: "Up", moviesGenere: [.comedy , .adventure, .animation], rating : 4)
```

The upMovie contains the name “Up” and it's movieGenere is comedy, adventure and animation witch contains 4 rating out of 5.

Encode

JSONEncoder is an object that encodes instances of a data type as JSON objects. JSONEncoder supports the Codable object.

```
// Encode data
let jsonEncoder = JSONEncoder()
do {
    let jsonData = try jsonEncoder.encode(upMovie)
    let jsonString = String(data: jsonData, encoding: .utf8)
    print("JSON String : " + jsonString!)
}
catch {
}
```

JSONEncoder will give us the JSON data which is used to retrieve JSON string.

Output string will be like :

```
{
  "name": "Up",
  "moviesGenere": [
    "comedy",
    "adventure",
    "animation"
```

```
  ],
  "rating": 4
}
```

解码

JSONDecoder 是一个用于从 JSON 对象解码数据类型实例的对象。我们可以从 JSON 字符串中获取该对象。

```
do {
    // 将数据解码为对象

    let jsonDecoder = JSONDecoder()
    let upMovie = try jsonDecoder.decode(Movie.self, from: jsonData)
    print("评分 : \(upMovie.name)")
    print("评分 : \(upMovie.rating)")
}
catch {
}
```

通过解码 JSONData，我们将收到 Movie 对象。因此我们可以获取保存在该

对象中的所有值。

输出将会是：

```
名称 : Up
评分 : 4
```



```
  ],
  "rating": 4
}
```

Decode

JSONDecoder is an object that decodes instances of a data type from JSON objects. We can get the object back from the JSON string.

```
do {
    // Decode data to object

    let jsonDecoder = JSONDecoder()
    let upMovie = try jsonDecoder.decode(Movie.self, from: jsonData)
    print("Rating : \(upMovie.name)")
    print("Rating : \(upMovie.rating)")
}
catch {
}
```

By decoding the JSONData we will receive the Movie object back. So we can get all the values which is saved in that object.

Output will be like:

```
Name : Up
Rating : 4
```

第207章：异步加载图片

第207.1节：最简单的方法

创建这个最简单的方法是使用Alamofire及其UIImageViewExtension。我们需要的是一个带有 imageView 的表格视图单元格，称之为imageView。

在 tableView 的 cellForRowAt: 函数中，我们将以以下方式下载图片并设置：

```
let url = URL(string: "https://httpbin.org/image/png")!
let placeholderImage = UIImage(named: "placeholder")!

imageView.af_setImage(withURL: url, placeholderImage: placeholderImage)
```

url 应指向你想下载的图片，placeHolder 图片应为已存储的图片。
然后我们调用 af_setImage 方法在 imageView 上，该方法会下载指定 url 的图片，在下载过程中会显示占位图片。一旦图片下载完成，请求的图片就会显示出来。

第 207.2 节：下载后检查单元格是否仍然可见

有时下载时间比单元格显示时间长。在这种情况下，可能会出现下载的图片显示在错误的单元格中。为了解决这个问题，我们不能使用 UIImageView Extension。

我们仍然会使用 Alamofire，但会使用完成处理器来显示图片。

在这种情况下，我们仍然需要一个带有 imageView 的 tableView 单元格。在 cellForRowAt: 方法中，我们会用以下代码下载图片：

```
let placeholderImage = UIImage(named: "placeholder")!
imageView.image = placeholderImage

let url = URL(string: "https://httpbin.org/image/png")!

Alamofire.request(url!, 方法: .get).responseImage { response in
    guard let image = response.result.value else { return }

    if let updateCell = tableView.cellForRow(at: indexPath) {
        updateCell.imageView.image = image
    }
}
```

在这个例子中，我们首先将图像设置为占位符图像。随后，我们使用Alamofire的request方法下载图像。我们将URL作为第一个参数传入，由于我们只想获取图像，因此使用.get HTTP方法。因为我们下载的是图像，所以希望响应为图像，因此使用.responseImage方法。

图像下载完成后，闭包被调用，首先确认下载的图像确实存在。然后通过检查cellForRow(at: indexPath)不返回nil，确保该单元格仍然可见。如果返回nil，则不做任何操作；如果不为nil，则将刚下载的图像赋值给该单元格。

最后这个if语句确保单元格仍然可见，如果用户已经滚动过该单元格，updateCell将为nil，if语句返回nil。这有助于防止在单元格中显示错误的图像。

Chapter 207: Load images async

Section 207.1: Easiest way

The most simple way to create this is to use Alamofire and its UIImageViewExtension. What we need is a tableview with a cell that has an imageView in it and lets call it `imageView`.

In the cellForRowAt: function of the tableView we would download the image and set in the following way:

```
let url = URL(string: "https://httpbin.org/image/png")!
let placeholderImage = UIImage(named: "placeholder")!

imageView.af_setImage(withURL: url, placeholderImage: placeholderImage)
```

The url should point to the image that you want to download and the placeHolder image should be a stored image. We then call the `af_setImage` method on the `imageView` which downloads the image at the given url and during the download the placeholder image will be shown. As soon as the image is downloaded the requested image is displayed

Section 207.2: Check that the cell is still visible after download

Sometimes the download takes longer than the cell is being displayed. In this case it can happen, that the downloaded image is shown in the wrong cell. To fix this we can not use the UIImageView Extension.

We still will be using Alamofire however we will use the completion handler to display the image.

In this scenario we still need a tableView with a cell which has a imageView in it. In the cellForRowAt: method we would download the image with the following code:

```
let placeholderImage = UIImage(named: "placeholder")!
imageView.image = placeholderImage

let url = URL(string: "https://httpbin.org/image/png")!

Alamofire.request(url!, method: .get).responseImage { response in
    guard let image = response.result.value else { return }

    if let updateCell = tableView.cellForRow(at: indexPath) {
        updateCell.imageView.image = image
    }
}
```

In this example we first set the image to the placeholder image. Afterwards we download the image with the `request` method of Alamofire. We pass the url as the first argument and since we just want to get the image we will use the `.get` HTTP method. Since we are downloading an image we want the response to be an image therefore we use the `.responseImage` method.

After the image has been downloaded the closure gets called and first of all we make sure that the downloaded image actually exists. Then we make sure that the cell is still visible by checking that the cellForRow(at: indexPath) doesn't return nil. If it does nothing happens, if it doesn't we assign the recently downloaded image.

This last if statement ensures that the cell is still visible if the user already scrolled over the cell the updateCell will be nil and the if statement returns nil. This helps us prevent displaying the wrong image in a cell.

第208章：添加Swift桥接头文件

第208.1节：如何手动创建Swift桥接头文件

- 在Xcode中添加新文件（文件 > 新建 > 文件），然后选择“源文件”并点击“头文件”。
- 将文件命名为“YourProjectName-Bridging-Header.h”。例如：在我的应用Station中，文件名为“Station-Bridging-Header”。
- 创建文件。
- 导航到你的项目构建设置，找到“Swift 编译器–代码生成”部分。你可以在搜索框中输入“Swift 编译器”来更快地缩小搜索结果。注意：如果你没有“Swift 编译器–代码生成”部分，这意味着你可能还没有向项目中添加任何 Swift 类。添加一个 Swift 文件，然后再试一次。
- 在“Objective-C 桥接头文件”旁边，你需要添加头文件的名称/路径。如果你的文件位于项目的根文件夹，只需填写头文件的名称。示例：“ProjectName/ProjectName-Bridging-Header.h”或简单地“ProjectName-Bridging-Header.h”。
- 打开你新创建的桥接头文件，使用#import语句导入你的 Objective-C 类。该文件中列出的任何类都可以从你的 Swift 类中访问。

第208.2节：Xcode 自动创建

向你的 Xcode 项目添加一个新的 Swift 文件。随意命名，系统会弹出一个提示框，询问你是否想创建桥接头文件。注意：如果你没有收到添加桥接头文件的提示，可能是你之前拒绝过此消息，你需要手动添加头文件（见下文）。

Chapter 208: Adding a Swift Bridging Header

Section 208.1: How to create a Swift Bridging Header Manually

- Add a new file to Xcode (File > New > File), then select “Source” and click “Header File“.
- Name your file “YourProjectName-Bridging-Header.h”. Example: In my app Station, the file is named “Station-Bridging-Header”.
- Create the file.
- Navigate to your project build settings and find the “Swift Compiler – Code Generation” section. You may find it faster to type in “Swift Compiler” into the search box to narrow down the results. Note: If you don’t have a “Swift Compiler – Code Generation” section, this means you probably don’t have any Swift classes added to your project yet. Add a Swift file, then try again.
- Next to “Objective-C Bridging Header” you will need to add the name/path of your header file. If your file resides in your project’s root folder simply put the name of the header file there. Examples: “ProjectName/ProjectName-Bridging-Header.h” or simply “ProjectName-Bridging-Header.h”.
- Open up your newly created bridging header and import your Objective-C classes using #import statements. Any class listed in this file will be able to be accessed from your swift classes.

Section 208.2: Xcode create automatically

Add a new Swift file to your Xcode project. Name it as you please and you should get an alert box asking if you would like to create a bridging header. Note: If you don’t receive a prompt to add a bridging header, you probably declined this message once before and you will have to add the header manually (see below)

LearnSwift: Ready | Today at 9:08 AM
Choose a template fo
iOS
Source
User Interface
Core Data
Resource
Other
OS X
Source
User Interface
Core Data
Resource
Other
Would you like to configure an Objective-C bridging header?
Adding this file to LearnSwift will create a mixed Swift and Objective-C target. Would you like Xcode to automatically configure a bridging header to enable classes to be accessed by both languages?
Cancel
No
Yes
Swift File
Objective-C File
Header File
C File
C++ File
Swift File
An empty Swift file.
Cancel
Previous
Finish

LearnSwift: Ready | Today at 9:08 AM
Choose a template fo
iOS
Source
User Interface
Core Data
Resource
Other
OS X
Source
User Interface
Core Data
Resource
Other
Would you like to configure an Objective-C bridging header?
Adding this file to LearnSwift will create a mixed Swift and Objective-C target. Would you like Xcode to automatically configure a bridging header to enable classes to be accessed by both languages?
Cancel
No
Yes
Swift File
Objective-C File
Header File
C File
C++ File
Swift File
An empty Swift file.
Cancel
Previous
Finish

第209章：创建应用程序ID

第209.1节：创建应用内购买产品

- 在应用内提供IAP时，您必须先在iTunes Connect中为每个单独的购买添加一个条目。如果您曾经在商店中上架过应用程序，流程类似，包括选择购买的定价等级。当用户进行购买时，App Store会处理向用户的iTunes账户收费的复杂过程。您可以添加多种不同类型的IAP：
    - **消耗型：这些可以多次购买并且会被消耗掉。比如额外生命、游戏内货币、临时增强道具等。**
    - **非消耗型：一次购买，期望永久拥有的内容，如额外关卡和可解锁内容。**
    - **非自动续订订阅：在固定时间内可用的内容。**
    - **自动续订订阅：重复订阅，例如每月的raywenderlich.com订阅。**

您只能为数字商品提供应用内购买，不能用于实物商品或服务。有关详细信息，请查看Apple关于创建应用内购买产品的完整文档。现在，在查看您应用在iTunes Connect中的条目时，点击“功能”标签，然后选择“应用内购买”。要添加新的IAP产品，点击“应用内购买”右侧的“+”号。

您将看到以下对话框出现：

Chapter 209: Creating an App ID

Section 209.1: Creating In-App Purchase Products

- When offering IAP within an an app, you must first add an entry for each individual purchase within iTunes Connect. If you've ever listed an app for sale in the store, it's a similar process and includes things like choosing a pricing tier for the purchase. When the user makes a purchase, the App Store handles the complex process of charging the user's iTunes account. There are a whole bunch of different types of IAP you can add:
    - **Consumable**: These can be bought more than once and can be used up. These are things such as extra lives, in-game currency, temporary power-ups, and the like.
    - **Non-Consumable**: Something that you buy once, and expect to have permanently such as extra levels and unlockable content.
    - **Non-Renewing Subscription**: Content that's available for a fixed period of time.
    - **Auto-Renewing Subscription**: A repeating subscription such as a monthly raywenderlich.com subscription.

You can only offer In-App Purchases for digital items, and not for physical goods or services. For more information about all of this, check out Apple's full documentation on Creating In-App Purchase Products. Now, while viewing your app's entry in iTunes Connect, click on the Features tab and then select In-App Purchases. To add a new IAP product, click the + to the right of In-App Purchases.

You will see the following dialog appear:

当用户在你的应用中购买一个rage漫画时，你会希望他们始终能够访问它，所以选择非消耗型（Non-Consumable），然后点击创建。接下来，按如下填写IAP的详细信息：

- 参考名称：在iTunes Connect中识别IAP的昵称。此名称不会出现在应用的任何地方。你将通过此次购买解锁的漫画标题是“鼓手的女朋友”，所以在这里输入该名称。
- 产品ID：这是识别IAP的唯一字符串。通常最好以Bundle ID开头，然后附加一个特定于该可购买项目的唯一名称。对于本教程，请确保附加“GirlfriendOfDrummerRage”，因为稍后应用中会用它来查找要解锁的漫画。例如：com.theNameYouPickedEarlier.Rage.GirlFriendOfDrummerRage。
- 允许销售：启用或禁用IAP的销售。你需要启用它！
- 价格等级：IAP的价格。选择等级1。

现在向下滚动到本地化部分，注意默认有一个英语（美国）的条目。将“鼓手的女朋友”输入到显示名称和描述中。点击保存。太好了！你已经创建了第一个IAP产品。

When a user purchases a rage comic in your app, you'll want them to always have access to it, so select Non-Consumable, and click Create. Next, fill out the details for the IAP as follows:

- **Reference Name**: A nickname identifying the IAP within iTunes Connect. This name does not appear anywhere in the app. The title of the comic you'll be unlocking with this purchase is “**Girlfriend of Drummer**“, so enter that here.
- **Product ID**: This is a unique string identifying the IAP. Usually it's best to start with the Bundle ID and then append a unique name specific to this purchasable item. For this tutorial, make sure you append “GirlfriendOfDrummerRage“, as this will be used later within the app to look up the comic to unlock. So, for example: com.theNameYouPickedEarlier.Rage.GirlFriendOfDrummerRage.
- **Cleared for Sale**: Enables or disables the sale of the IAP. You want to enable it!
- **Price Tier**: The cost of the IAP. Choose Tier 1.

Now scroll down to the Localizations section and note that there is a default entry for English (U.S.). Enter “Girlfriend of Drummer” for both the Display Name and the Description. Click Save. Great! You've created your first IAP product.

在深入编写代码之前，还有一步需要完成。当在应用的开发版本中测试应用内购买时，苹果提供了一个测试环境，允许你'购买'你的IAP产品而无需进行真实的财务交易。

第209.2节：创建沙盒用户

在 iTunes Connect 中，点击窗口左上角的 iTunes Connect 返回主菜单。选择用户与角色，然后点击沙盒测试人员标签。点击"测试人员"标题旁的 + 号。

填写信息并在完成后点击保存。您可以为测试用户编造一个名字和姓氏，但所选的电子邮件地址必须是真实的，因为苹果会向该地址发送验证邮件。
收到邮件后，请务必点击其中的链接以验证您的地址。您输入的电子邮件地址也不应已关联任何 Apple ID 账户。提示：如果您有 Gmail 账户，可以使用地址别名，而无需创建全新的账户。

There's one more step required before you can delve into some code. When testing in-app purchases in a development build of an app, Apple provides a test environment which allows you to 'purchase' your IAP products without creating financial transactions.

Section 209.2: Creating a Sandbox User

In iTunes Connect, click iTunes Connect in the top left corner of the window to get back to the main menu. Select Users and Roles, then click the Sandbox Testers tab. Click + next to the "Tester" title.

Fill out the information and click Save when you're done. You can make up a first and last name for your test user, but the email address chosen must be a real email address as a verification will be sent to the address by Apple. Once you receive that email, be sure to click the link in it to verify your address. The email address you enter should also NOT already be associated with an Apple ID account. Hint: if you have a gmail account, you can simply use an address alias instead of having to create a brand new account

第210章：Swift：在 AppDelegate 中更改 rootViewController 以呈现主界面或登录/引导流程

向新用户展示首次使用体验通常很有用。这可能出于多种原因，例如提示他们登录（如果您的情况需要）、解释如何使用应用，或仅仅是告知他们更新中的新功能（如 iOS11 中的备忘录、照片和音乐）。

第210.1节：选项1：切换根视图控制器（良好）

切换根视图控制器有其好处，尽管过渡选项仅限于 UIViewAnimationOptions 支持的那些，因此根据您希望在流程间如何过渡，可能需要实现自定义过渡——这可能比较繁琐。

您可以通过简单设置 UIApplication.shared.keyWindow.rootViewController 来显示引导流程。解除显示则通过使用 UIView.transition(with:) 并传入过渡样式作为 UIViewAnimationOptions，这里使用的是交叉淡入淡出（Cross Dissolve）。（翻转和卷曲效果也支持）。

在切换回主视图之前，您还必须设置主视图的 frame，因为这是您第一次实例化它。

```
// MARK: - 引导流程

extension AppDelegate {

    func 显示引导流程() {
        if let window = UIApplication.shared.keyWindow, let onboardingViewController =
UIStoryboard(name: "Onboarding", bundle: nil).instantiateInitialViewController() as?
OnboardingViewController {
onboardingViewController.delegate = self
            window.rootViewController = onboardingViewController
        }
    }

    func 隐藏引导流程() {
        if let window = UIApplication.shared.keyWindow, let mainViewController = UIStoryboard(name:
"Main", bundle: nil).instantiateInitialViewController() {
            mainViewController.view.frame = window.bounds
            UIView.transition(with: window, duration: 0.5, options: .transitionCrossDissolve,
animations: {
window.rootViewController = mainViewController
            }, completion: nil)
        }
    }
}
```

第210.2节：选项2：以模态方式呈现替代流程（更好）

在最简单的实现中，入职流程可以直接以模态窗口的形式呈现，
因为从语义上讲，用户处于单一的流程中。

Chapter 210: Swift: Changing the rootViewController in AppDelegate to present main or login/onboarding flow

It is often useful to present a first-run experience to new users of your App. This could be for any number of reasons, such as prompting them to sign in (if required for your situation), explaining how to use the App, or simply informing them of new features in an update (as Notes, Photos and Music do in iOS11).

Section 210.1: Option 1: Swap the Root View Controller (Good)

There are benefits to switching the root view controller, although the transition options are limited to those supported by `UIViewAnimationOptions`, so depending on how you wish to transition between flows might mean you have to implement a custom transition - which can be cumbersome.

You can show the Onboarding flow by simply setting the `UIApplication.shared.keyWindow.rootViewController`

Dismissal is handled by utilizing `UIView.transition(with:)` and passing the transition style as a `UIViewAnimationOptions`, in this case Cross Dissolve. (Flips and Curls are also supported).

You also have to set the frame of the Main view before you transition back to it, as you're instantiating it for the first time.

```
// MARK: - Onboarding

extension AppDelegate {

    func showOnboarding() {
        if let window = UIApplication.shared.keyWindow, let onboardingViewController =
UIStoryboard(name: "Onboarding", bundle: nil).instantiateInitialViewController() as?
OnboardingViewController {
            onboardingViewController.delegate = self
            window.rootViewController = onboardingViewController
        }
    }

    func hideOnboarding() {
        if let window = UIApplication.shared.keyWindow, let mainViewController = UIStoryboard(name:
"Main", bundle: nil).instantiateInitialViewController() {
            mainViewController.view.frame = window.bounds
            UIView.transition(with: window, duration: 0.5, options: .transitionCrossDissolve,
animations: {
                window.rootViewController = mainViewController
            }, completion: nil)
        }
    }
}
```

Section 210.2: Option 2: Present Alternative Flow Modally (Better)

In the most straightforward implementation, the Onboarding flow can simply be presented in a modal context, since semantically the User is on a single journey.

> [Apple 人机界面指南 – 模态][1]:
>
> 仅在必须引起用户注意、任务必须完成或放弃才能继续使用应用程序，或需要保存重要数据时，才考虑创建模态上下文。

以模态方式呈现允许在流程结束时简单地关闭，避免了切换控制器的繁琐操作。

标准方式也支持自定义过渡，因为这使用了ViewController.present() API：

```
// MARK: - 引导流程

extension AppDelegate {

    func 显示引导流程() {
        if let window = window, let onboardingViewController = UIStoryboard(name: "Onboarding",
bundle: nil).instantiateInitialViewController() as? OnboardingViewController {
            onboardingViewController.delegate = self
window.makeKeyAndVisible()
window.rootViewController?.present(onboardingViewController, animated: false,
completion: nil)
        }
    }

    func 隐藏引导流程() {
        if let window = UIApplication.shared.keyWindow {
            window.rootViewController?.dismiss(animated: true, completion: nil)
        }
    }
}
```


> [Apple Human Interface Guidelines – Modality][1]:
>
> Consider creating a modal context only when it's critical to get someone's attention, when a task must be completed or abandoned to continue using the app, or to save important data.

Presenting modally allows the simple option of dismissal at the end of the journey, with little of the cruft of swapping controllers.

Custom transitions are also supported in the standard way, since this uses the `ViewController.present()` API:

```
// MARK: - Onboarding

extension AppDelegate {

    func showOnboarding() {
        if let window = window, let onboardingViewController = UIStoryboard(name: "Onboarding",
bundle: nil).instantiateInitialViewController() as? OnboardingViewController {
            onboardingViewController.delegate = self
            window.makeKeyAndVisible()
            window.rootViewController?.present(onboardingViewController, animated: false,
completion: nil)
        }
    }

    func hideOnboarding() {
        if let window = UIApplication.shared.keyWindow {
            window.rootViewController?.dismiss(animated: true, completion: nil)
        }
    }
}
```

致谢

非常感谢所有来自Stack Overflow Documentation的人员帮助提供这些内容，
更多更改可以发送至web@petercv.com以发布或更新新内容

4oby	第2章
阿比吉特	第192章
abjurato	第149章
亚当·埃伯巴赫	第11章和第38章
亚当·普雷布尔	第27章
阿德南·阿夫塔布	第79章
阿德里安娜·卡雷利	第46章和第151章
艾哈迈德·阿卜杜拉	第41章、第89章和第107章
艾哈迈德·哈拉夫	第28章
AJ9	第14章
ajmccall	第65章
Ajumal	第15章
阿基兰·阿拉苏	第2章
alaphao	第20章和第67章
亚历克斯·卡拉姆	第74章
亚历克斯·科希	第2、14、34、38、74和95章
亚历克斯·劳斯	第61章
亚历山大·特卡琴科	第64章
阿列克西	第40和92章
阿里·阿巴斯	第178章
阿里·比德尔	第1章
阿里·埃尔索卡里	第34章
阿利斯特拉	第89章
艾尔文·阿比亚	第122章
阿曼普里特	第107章
阿马尔	第67、96和97章
阿南德·尼姆杰	第1、38、40、74、76和117章
阿纳托利	第1章
安德烈亚斯	第25、45和151章
安德烈斯·卡内拉	第2章
安德烈斯·基耶夫斯基	第14章
安德里·切尔年科	第2章
安·范	第11和20章
阿尼什 웃	第20章、第72章和第97章
安基特·乔汉	第190章
ApolloSoftware	第79章
Arefly	第5章和第96章
阿希什·卡卡德	第107章、第113章和第206章
阿舒托什·戴夫	第1章
askielboe	第142章
AstroCB	第2章
阿西莫夫	第47、56和119章
反斜杠	第52章
巴达尔·沙阿	第30、67和144章
巴拉古鲁巴兰	第77章
豆	第53章和第67章
本塞·帕托加托	第74章

Credits

Thank you greatly to all the people from Stack Overflow Documentation who helped provide this content,
more changes can be sent to web@petercv.com for new content to be published or updated

4oby	Chapter 2
Abhijit	Chapter 192
abjurato	Chapter 149
Adam Eberbach	Chapters 11 and 38
Adam Preble	Chapter 27
Adnan Aftab	Chapter 79
Adriana Carelli	Chapters 46 and 151
Ahmed Abdallah	Chapters 41, 89 and 107
Ahmed Khalaf	Chapter 28
AJ9	Chapter 14
ajmccall	Chapter 65
Ajumal	Chapter 15
Akilan Arasu	Chapter 2
alaphao	Chapters 20 and 67
Alex Kallam	Chapter 74
Alex Koshy	Chapters 2, 14, 34, 38, 74 and 95
Alex Rouse	Chapter 61
Alexander Tkachenko	Chapter 64
Alexi	Chapters 40 and 92
Ali Abbas	Chapter 178
Ali Beadle	Chapter 1
Ali Elsokary	Chapter 34
Alistra	Chapter 89
Alvin Abia	Chapter 122
Amanpreet	Chapter 107
amar	Chapters 67, 96 and 97
Anand Nimje	Chapters 1, 38, 40, 74, 76 and 117
Anatoliy	Chapter 1
Andreas	Chapters 25, 45 and 151
Andres Canella	Chapter 2
Andres Kievsky	Chapter 14
Andrii Chernenko	Chapter 2
Anh Pham	Chapters 11 and 20
Anish 웃	Chapters 20, 72 and 97
Ankit chauhan	Chapter 190
ApolloSoftware	Chapter 79
Arefly	Chapters 5 and 96
Ashish Kakkad	Chapters 107, 113 and 206
Ashutosh Dave	Chapter 1
askielboe	Chapter 142
AstroCB	Chapter 2
azimov	Chapters 47, 56 and 119
backslash	Chapter 52
Badal Shah	Chapters 30, 67 and 144
balagurubaran	Chapter 77
Bean	Chapters 53 and 67
Bence Pattogato	Chapter 74

BennX	第131章
本特福德	第2章
贝托·卡尔达斯	第189章
beyowulf	第55、57和59章
巴德雷什·卡蒂里亚	第107章和第151章
巴文·拉马尼	第50章
布米特·梅赫塔	第2章、第7章、第38章、第67章和第93章
布莱克德沃罗	第67章
蓝翼	第171章
bluey31	第75章
bmike	第1章
Boletrone	第24章
Bonnie	第73、107和134章
Brandon	第33、43、116和124章
breakingobstacles	第65章和第198章
布伦登·罗伯托	第14章和第83章
布雷特·庞德	第23章
布赖恩	第2、7、10、14、68、104和119章
光明的未来	第47章
bryanjclark	第135章
作者：Jeevan	第2章和第78章
凯勒布·克莱维特	第2章、第11章和第20章
查尔斯	第73章
查图兰加·席尔瓦	第2章
奇拉格·德赛	第134章
克里斯·布兰兹马	第2章
Cin316	第2章
cleverbit	第210章
代码战士	第2章和第52章
代码更改者	第174章
科里·威尔海特	第20章
克里斯	第24章
克里斯蒂娜	第40章
西里尔·伊瓦尔·加西亚	第101、148和154章
达塔特拉亚	第7、20、35、133、145、197和201章
丹尼尔·奥尔梅诺	第137章
dannyzlo	第27章
danshevluk	第2和61章
达尔希特·沙阿	第21章
dasdom	第2、5、23、26、36、75和133章
ddb	第2、5、20和38章
DeyaEldeen	第2、11、20和23章
deyanm	第1章
dgatwood	第75章
Dima Deplov	第14章、第63章和第67章
迪内什·拉贾	第67章
迪彭·潘查萨拉	第8章
迪尚特·卡帕迪亚	第203章
争议	第63章
文档	第86章
道格拉斯·斯塔恩斯	第20章
杜利·金斯基	第34章、第75章和第97章
敦雅·拉利奇	第8章、第24章、第27章、第36章、第73章、第119章和第135章

BennX	Chapter 131
bentford	Chapter 2
Beto Caldas	Chapter 189
beyowulf	Chapters 55, 57 and 59
Bhadresh Kathiriya	Chapters 107 and 151
Bhavin Ramani	Chapter 50
Bhumit Mehta	Chapters 2, 7, 38, 67 and 93
BlackDeveraux	Chapter 67
Bluewings	Chapter 171
bluey31	Chapter 75
bmike	Chapter 1
Boletrone	Chapter 24
Bonnie	Chapters 73, 107 and 134
Brandon	Chapters 33, 43, 116 and 124
breakingobstacles	Chapters 65 and 198
Brendon Roberto	Chapters 14 and 83
Brett Ponder	Chapter 23
Brian	Chapters 2, 7, 10, 14, 68, 104 and 119
Bright Future	Chapter 47
bryanjclark	Chapter 135
byJeevan	Chapters 2 and 78
Caleb Kleveter	Chapters 2, 11 and 20
Charles	Chapter 73
Chathuranga Silva	Chapter 2
Chirag Desai	Chapter 134
Chris Brandsma	Chapter 2
Cin316	Chapter 2
cleverbit	Chapter 210
Code.Warrior	Chapters 2 and 52
CodeChanger	Chapter 174
Cory Wilhite	Chapter 20
Cris	Chapter 24
Cristina	Chapter 40
Cyril Ivar Garcia	Chapters 101, 148 and 154
D4ttatraya	Chapters 7, 20, 35, 133, 145, 197 and 201
Daniel Ormeño	Chapter 137
dannyzlo	Chapter 27
danshevluk	Chapters 2 and 61
Darshit Shah	Chapter 21
dasdom	Chapters 2, 5, 23, 26, 36, 75 and 133
ddb	Chapters 2, 5, 20 and 38
DeyaEldeen	Chapters 2, 11, 20 and 23
deyanm	Chapter 1
dgatwood	Chapter 75
Dima Deplov	Chapters 14, 63 and 67
Dinesh Raja	Chapter 67
Dipen Panchasara	Chapter 8
Dishant Kapadiya	Chapter 203
dispute	Chapter 63
Doc	Chapter 86
Douglas Starnes	Chapter 20
Duly Kinsky	Chapters 34, 75 and 97
Dunja Lalic	Chapters 8, 24, 27, 36, 73, 119 and 135

杜赖·阿穆坦.H	第194章
埃弗拉伊姆·韦斯	第171章
埃隆·陈	第7章
Emptyless	第76章
Eonil	第24章
ERbittuu	第73章
埃里克	第2章
埃里克·戈达德	第1章和第14章
欧文	第2章
esthepiking	第2章和第198章
法比奥	第127章
法比奥·贝尔格（Fabio Berger）	第34章
法希姆·帕尔卡尔（Fahim Parkar）	第2章
法兰·加尼（Faran Ghani）	第118章
费利克斯	第2章
FelixSFD	第1、2、5、8、11、22、34、37、54、68、117和198章
福尼克斯（Fonix）	第125和187章
gadu	第2章和第24章
乔治·李	第159章
ggrana	第2章
gkpln3	第204章
格洛芬戴尔	第14章
格雷格	第64章
gvuksic	第2章
H. M. 马德罗内	第122章
HaemEternal	第143章
halil_g	第202章
hankide	第2章
HariKrishnan.P	第115章
Harshal Bhavsar	第23章和第76章
Heberti Almeida	第8章
hgwhittle	第7章、第20章和第28章
霍萨姆·加里布	第107章
侯赛因·贝布迪·拉德	第76章
鱼人马	第74章
伊丹	第177章
idobn	第2章
idocode	第52章
ignotusverum	第89章
伊戈尔·比迪纽克	第53章
邪恶的火腿博士	第11章
伊马努·佩蒂	第2章
iphonic	第20章和第23章
伊尔凡	第121章
J F	第107章和第114章
J.帕拉维奇尼	第207章
杰克·奈	第1章
雅各班克斯	第2章
雅科波·彭佐	第74章
杰克·伦泽	第63章
[illegible]	第23章
詹姆斯	第1章和第5章

Durai Amuthan.H	Chapter 194
Efraim Weiss	Chapter 171
ElonChan	Chapter 7
Emptyless	Chapter 76
Eonil	Chapter 24
ERbittuu	Chapter 73
Eric	Chapter 2
Erik Godard	Chapters 1 and 14
Erwin	Chapter 2
esthepiking	Chapters 2 and 198
Fabio	Chapter 127
Fabio Berger	Chapter 34
Fahim Parkar	Chapter 2
Faran Ghani	Chapter 118
Felix	Chapter 2
FelixSFD	Chapters 1, 2, 5, 8, 11, 22, 34, 37, 54, 68, 117 and 198
Fonix	Chapters 125 and 187
gadu	Chapters 2 and 24
George Lee	Chapter 159
ggrana	Chapter 2
gkpln3	Chapter 204
Glorfindel	Chapter 14
Greg	Chapter 64
gvuksic	Chapter 2
H. M. Madrone	Chapter 122
HaemEternal	Chapter 143
halil_g	Chapter 202
hankide	Chapter 2
HariKrishnan.P	Chapter 115
Harshal Bhavsar	Chapters 23 and 76
Heberti Almeida	Chapter 8
hgwhittle	Chapters 7, 20 and 28
Hossam Ghareeb	Chapter 107
Husein Behboodi Rad	Chapter 76
Ichthyocentaurs	Chapter 74
Idan	Chapter 177
idobn	Chapter 2
idocode	Chapter 52
ignotusverum	Chapter 89
Igor Bidiniuc	Chapter 53
il Malvagio Dottor Prosciutto	Chapter 11
Imanou Petit	Chapter 2
iphonic	Chapters 20 and 23
Irfan	Chapter 121
J F	Chapters 107 and 114
J.Paravicini	Chapter 207
Jack Ngai	Chapter 1
Jacobanks	Chapter 2
Jacopo Penzo	Chapter 74
Jake Runzer	Chapter 63
JAL	Chapter 23
James	Chapters 1 and 5

詹姆斯·P	第5、74、76、85、98、119和122章
简·阿塔克	第133章
雅诺	第67和111章
吉米·詹姆斯	第8章
李金环	第70章
约翰·莱昂纳多	第27章
johnpenning	第2章
Jojodmo	第2章、第5章和第20章
乔什·布朗	第2章和第133章
乔什·卡斯韦尔	第76章
约书亚	第2章、第5章、第34章、第83章、第94章、第98章和第152章
约书亚·J·麦金农	第2章
JPetric	第182章
jrf	第1章
juanjo	第2章和第67章
Julian135	第41章
Justin	第66章
卡米尔·哈拉西莫维奇	第18章、第19章、第62章和第89章
Kampai	第1章
kamwysoc	第7章
keithbhunter	第67章
肯德尔·利斯特	第63章
凯文·迪特拉格利亚	第38章和第86章
基利安·科尔茨施	第88章
基雷因	第2章
基里特·莫迪	第22章、第70章、第74章和第76章
基里特·瓦格赫拉	第112章
Kof	第30章和第34章
近田·亚达夫	第134章
小川康介	第68章
考希克	第44章、第48章和第109章
克鲁纳尔	第69章
林纳斯·格法斯	第29章
洛西奥瓦蒂	第1章
迷失海洋的约书亚	第73章和第179章
卢卡·达尔贝尔蒂	第2章、第14章、第20章、第23章和第63章
路易斯	第14章
路易斯·恩里克·吉马良斯	第34章、第70章和第89章
卢克·帕特森	第2章
卢米亚尔克	第2章
玛迪ヅヅ	第34章、第72章和第92章
马赫什	第68章
马哈茂德·亚当	第2章、第8章和第116章
马尼	第106章
曼西·潘查尔	第70章和第100章
马克	第38章
mattblessed	第11章
马修·考利	第1章
马修·西曼	第149章
maxkonovalov	第8章、第23章、第29章、第73章和第80章
马尤里·R·塔拉维亚	第47章和第103章
MCMatan	第54章

James P	Chapters 5, 74, 76, 85, 98, 119 and 122
Jan ATAC	Chapter 133
Jano	Chapters 67 and 111
Jimmy James	Chapter 8
Jinhuan Li	Chapter 70
John Leonardo	Chapter 27
johnpenning	Chapter 2
Jojodmo	Chapters 2, 5 and 20
Josh Brown	Chapters 2 and 133
Josh Caswell	Chapter 76
Joshua	Chapters 2, 5, 34, 83, 94, 98 and 152
Joshua J. McKinnon	Chapter 2
JPetric	Chapter 182
jrf	Chapter 1
juanjo	Chapters 2 and 67
Julian135	Chapter 41
Justin	Chapter 66
Kamil Harasimowicz	Chapters 18, 19, 62 and 89
Kampai	Chapter 1
kamwysoc	Chapter 7
keithbhunter	Chapter 67
Kendall Lister	Chapter 63
Kevin DiTraglia	Chapters 38 and 86
Kilian Koeltzsch	Chapter 88
Kireyin	Chapter 2
Kirit Modi	Chapters 22, 70, 74 and 76
Kirit Vaghela	Chapter 112
Kof	Chapters 30 and 34
Konda Yadav	Chapter 134
Kosuke Ogawa	Chapter 68
Koushik	Chapters 44, 48 and 109
Krunal	Chapter 69
LinusGeffarth	Chapter 29
Losiowaty	Chapter 1
lostAtSeaJoshua	Chapters 73 and 179
Luca D'Alberti	Chapters 2, 14, 20, 23 and 63
Luis	Chapter 14
Luiz Henrique Guimaraes	Chapters 34, 70 and 89
Luke Patterson	Chapter 2
Lumialxk	Chapter 2
Maddyヅヅ	Chapters 34, 72 and 92
Mahesh	Chapter 68
Mahmoud Adam	Chapters 2, 8 and 116
MANI	Chapter 106
Mansi Panchal	Chapters 70 and 100
Mark	Chapter 38
mattblessed	Chapter 11
Matthew Cawley	Chapter 1
Matthew Seaman	Chapter 149
maxkonovalov	Chapters 8, 23, 29, 73 and 80
Mayuri R Talaviya	Chapters 47 and 103
MCMatan	Chapter 54

穆罕默德·易卜拉欣·哈桑	第2、3、8、9、11、13、16、20、21、24、25、27、37、41、45、50、51、54、59、71、81、88、125、140、146、147、151、155、157、158、160、163、164、165、166、167、168、169、170和183章
梅胡尔·楚阿汉	第39、40、50、83、103、104、110、117、149、152、153、155和198章
梅胡尔·塔卡尔	第68和73章
梅尔特·布兰	第67章
米敦·MP	第93章
绿	第25和156章
穆罕默德·拉纳	第16章
莫谢	第2章
穆罗德里戈	第32章
Xcoder先生	第46、49、61、64和85章
mtso	第23、39和103章
穆阿祖德	第20和63章
穆罕默德·齐山	第8章
穆罕默德·佐海布·埃赫桑	第22章和第110章
纳伦德拉·潘迪	第20章、第22章、第75章、第85章、第100章、第102章和第115章
内特·李	第5章
内森	第198章
内森·莱维特	第2章
Nef10	第50章
内尔敏·塞希奇	第175章
尼克莱什·巴格迪亚	第176章
尼基塔·库尔廷	第22章
尼拉夫	第196章
尼拉夫·巴特	第52章
恩朱里	第2、11、24、91、92和98章
诺亚姆	第180章
NobodyNada	第129章
NSNoob	第5、20、22、23、67和73章
Nykholas	第73章
OhadM	第63、73、104、105、106和128章
Oleh Zayats	第26、108、174和188章
Ollie	第2章
奥努尔·图纳	第147章
奥特温·根茨	第41章
奥兹古尔	第67章
帕尔塞克	第110章
P. Pawluś	第5、34和60章
帕布莱罗斯	第20、50和65章
帕维尔·加蒂洛夫	第6章
帕维尔·古罗夫	第107章
pckill	第20章
彼得·德维斯	第20章
法尼·赛	第199章
pkc456	第173章
quant24	第103章
Quantm	第2章和第11章
布朗·拉达加斯特	第2章
拉胡尔	第12章、第17章、第31章、第85章和第200章
拉胡尔·维亚斯	第20章和第117章
拉姆库马尔·钦塔拉	第41章和第50章
regetskcob	第2章

Md. Ibrahim Hassan	Chapters 2, 3, 8, 9, 11, 13, 16, 20, 21, 24, 25, 27, 37, 41, 45, 50, 51, 54, 59, 71, 81, 88, 125, 140, 146, 147, 151, 155, 157, 158, 160, 163, 164, 165, 166, 167, 168, 169, 170 and 183
Mehul Chuahan	Chapters 39, 40, 50, 83, 103, 104, 110, 117, 149, 152, 153, 155 and 198
Mehul Thakkar	Chapters 68 and 73
Mert Buran	Chapter 67
Midhun MP	Chapter 93
midori	Chapters 25 and 156
Mohammad Rana	Chapter 16
Moshe	Chapter 2
mourodrigo	Chapter 32
Mr. Xcoder	Chapters 46, 49, 61, 64 and 85
mtso	Chapters 23, 39 and 103
muazhud	Chapters 20 and 63
Muhammad Zeeshan	Chapter 8
Muhammad Zohaib Ehsan	Chapters 22 and 110
Narendra Pandey	Chapters 20, 22, 75, 85, 100, 102 and 115
Nate Lee	Chapter 5
nathan	Chapter 198
Nathan Levitt	Chapter 2
Nef10	Chapter 50
Nermin Sehic	Chapter 175
Nikhlesh Bagdiya	Chapter 176
Nikita Kurtin	Chapter 22
Nirav	Chapter 196
Nirav Bhatt	Chapter 52
njuri	Chapters 2, 11, 24, 91, 92 and 98
Noam	Chapter 180
NobodyNada	Chapter 129
NSNoob	Chapters 5, 20, 22, 23, 67 and 73
Nykholas	Chapter 73
OhadM	Chapters 63, 73, 104, 105, 106 and 128
Oleh Zayats	Chapters 26, 108, 174 and 188
Ollie	Chapter 2
Onur Tuna	Chapter 147
Ortwin Gentz	Chapter 41
ozgur	Chapter 67
Pärserk	Chapter 110
P. Pawluś	Chapters 5, 34 and 60
pableiros	Chapters 20, 50 and 65
Pavel Gatilov	Chapter 6
Pavel Gurov	Chapter 107
pckill	Chapter 20
Peter DeWeese	Chapter 20
Phani Sai	Chapter 199
pkc456	Chapter 173
quant24	Chapter 103
Quantm	Chapters 2 and 11
Radagast the Brown	Chapter 2
Rahul	Chapters 12, 17, 31, 85 and 200
Rahul Vyas	Chapters 20 and 117
Ramkumar chintala	Chapters 41 and 50
regetskcob	Chapter 2

雷尼尔·梅利安	第11章、第23章、第34章、第68章和第123章
雷克斯	第23章和第29章
理查德·阿什	第64章
rigdonmr	第101章
罗布	第11章和第85章
rob180	第8章
罗德里戈·德·圣地亚哥	第38章
rohit90	第141章
罗兰·基索姆	第5章、第24章和第76章
罗纳克·查尼亚拉	第8章
Ruby	第25章
赛义德	第161章和第162章
sage444	第8章和第75章
萨赫布·罗伊	第90章
萨莉	第23章、第67章和第73章
萨姆·费舍尔	第87章
萨默·穆拉德	第23章
塞缪尔·斯宾塞	第7章、第11章、第14章和第82章
塞缪尔·特费拉	第72、73和98章
samwize	第34章
桑迪	第8、23、73和98章
sanman	第29、107、110和135章
sasquatch	第20、30和34章
satheeshwaran	第99章
萨米尔·沙阿	第174章
塞斯林	第14章
赛义德·帕尔萨·内沙伊	第1、2、8、22、73、76、80、107、117、130、131、133、136、138、139、140、185和186章
沙哈布丁·万西瓦拉	第2章
沙杜尔	第132章
鲨鱼诱饵胡哈哈	第127章
shim	第22、34和61章
施里坎特·K	第33章
悉达特·苏尼尔	第97章
simple_code	第14章和第118章
skyline75489	第125章
slxl	第64章
SM18	第114章
SNarula	第172章
solidcell	第2章
SpaceDog	第21章
Sravan	第2章
Srinija	第40章
StackUnderflow	第58、181和184章
史蒂夫·莫泽（Steve Moser）	第14、61、63和64章
subv3rsion	第120章
苏贾尼亚	第8、14、27、29和97章
苏贾伊	第37章
苏尼尔·夏尔马	第2、8、11、29、41和136章
苏拉格奇	第1、2、11、14、22、24、26、27、28、29、34、36、38、53、63、64、67和89章
苏山特·贾格塔普	第2、24、34、65、103和117章
塔马罗斯	第14和152章
塔伦·西拉	第74章

Reinier Melian	Chapters 11, 23, 34, 68 and 123
Rex	Chapters 23 and 29
Richard Ash	Chapter 64
rigdonmr	Chapter 101
Rob	Chapters 11 and 85
rob180	Chapter 8
Rodrigo de Santiago	Chapter 38
rohit90	Chapter 141
Roland Keesom	Chapters 5, 24 and 76
Ronak Chaniyara	Chapter 8
Ruby	Chapter 25
Saeed	Chapters 161 and 162
sage444	Chapters 8 and 75
Saheb Roy	Chapter 90
Sally	Chapters 23, 67 and 73
Sam Fischer	Chapter 87
Samer Murad	Chapter 23
Samuel Spencer	Chapters 7, 11, 14 and 82
Samuel Teferra	Chapters 72, 73 and 98
samwize	Chapter 34
Sandy	Chapters 8, 23, 73 and 98
sanman	Chapters 29, 107, 110 and 135
sasquatch	Chapters 20, 30 and 34
satheeshwaran	Chapter 99
Saumil Shah	Chapter 174
Seslyn	Chapter 14
Seyyed Parsa Neshaei	Chapters 1, 2, 8, 22, 73, 76, 80, 107, 117, 130, 131, 133, 136, 138, 139, 140, 185 and 186
Shahabuddin Vansiwala	Chapter 2
Shardul	Chapter 132
SharkbaitWhohaha	Chapter 127
shim	Chapters 22, 34 and 61
Shrikant K	Chapter 33
Siddharth Sunil	Chapter 97
simple_code	Chapters 14 and 118
skyline75489	Chapter 125
slxl	Chapter 64
SM18	Chapter 114
SNarula	Chapter 172
solidcell	Chapter 2
SpaceDog	Chapter 21
Sravan	Chapter 2
Srinija	Chapter 40
StackUnderflow	Chapters 58, 181 and 184
Steve Moser	Chapters 14, 61, 63 and 64
subv3rsion	Chapter 120
Sujania	Chapters 8, 14, 27, 29 and 97
Sujay	Chapter 37
Sunil Sharma	Chapters 2, 8, 11, 29, 41 and 136
Suragch	Chapters 1, 2, 11, 14, 22, 24, 26, 27, 28, 29, 34, 36, 38, 53, 63, 64, 67 and 89
sushant jagtap	Chapters 2, 24, 34, 65, 103 and 117
Tamarous	Chapters 14 and 152
Tarun Seera	Chapter 74

塔沃·梅塞普	第39章
塔西纳里	第36章
特贾·南达穆里	第53章
tfrank377	第72章
tharkay	第34、73、114和123章
The_Curry_Man	第8章
田（Tien）	第86和150章
Tiko	第205章
tilo	第107章
蒂姆	第70章和第126章
蒂姆·埃本泽尔	第70章
timbroder	第63章
tktsubota	第7章
tobeiosdev	第73章和第147章
托米·C。	第23章和第92章
乌玛	第195章
撤销	第2、72、85和146章
user1374	第151和152章
user3480295	第2、8、22和67章
user3760892	第76章
维克多·M	第47章
维格南	第1章
维克托·辛科	第5章、第20章、第23章和第89章
维尼特·乔德哈里	第99章
维诺德·库马尔	第191章
维韦克·莫尔卡尔	第42章
vp2698	第4章
wdywayne	第84章
william205	第2、5、7、10、11、20、24、26、28、30、34、36、64、72、76、89和103章
WMios	第2、5、23、24、26、34、61、94和101章
金刚狼	第7章
氙	第7章
亚格内什·多巴里亚	第2、23和63章
叶夫亨·杜比宁	第116章
约格什·瓦德瓦	第95、208和209章
东安	第67章
欧阳大哥	第193章

Tarvo Mäesepp	Chapter 39
tassinari	Chapter 36
Teja Nandamuri	Chapter 53
tfrank377	Chapter 72
tharkay	Chapters 34, 73, 114 and 123
The_Curry_Man	Chapter 8
Tien	Chapters 86 and 150
Tiko	Chapter 205
tilo	Chapter 107
Tim	Chapters 70 and 126
Tim Ebenezer	Chapter 70
timbroder	Chapter 63
tktsubota	Chapter 7
tobeiosdev	Chapters 73 and 147
Tommie C.	Chapters 23 and 92
Uma	Chapter 195
Undo	Chapters 2, 72, 85 and 146
user1374	Chapters 151 and 152
user3480295	Chapters 2, 8, 22 and 67
user3760892	Chapter 76
Victor M	Chapter 47
Vignan	Chapter 1
Viktor Simkó	Chapters 5, 20, 23 and 89
Vineet Choudhary	Chapter 99
Vinod Kumar	Chapter 191
Vivek Molkar	Chapter 42
vp2698	Chapter 4
wdywayne	Chapter 84
william205	Chapters 2, 5, 7, 10, 11, 20, 24, 26, 28, 30, 34, 36, 64, 72, 76, 89 and 103
WMios	Chapters 2, 5, 23, 24, 26, 34, 61, 94 and 101
Wolverine	Chapter 7
Xenon	Chapter 7
Yagnesh Dobariya	Chapters 2, 23 and 63
Yevhen Dubinin	Chapter 116
yogesh wadhwa	Chapters 95, 208 and 209
Đông An	Chapter 67
欧阳大哥	Chapter 193

你可能也喜欢

You may also like