从零开始学Swift计时器App开发

“不停的跋涉、是生就是命运”。 本文大部分改编自其他文章!
本教程将从零开始教大家使用Swift来开发一款iOS计时器App。
这款App的灵感来自于我家厨房的百利达计时器，平时我主要用它来控制烹饪的时间和实践番茄工作法。

它的操作很简单：
点击复位可以对时间清零；
点击秒、1分、3分、5分可以不断增加倒计时时间；
点击 开始/停止 来启动或停止倒计时；
通过此教程，你将学习：
如何使用Xcode
如何灵活运用Swift中的语法来解决实际问题；
如何使用基本的UI控件UIButton、UILabel 来创建界面，使用NSTimer来触发定时事件，以及使用UILocalNotification来实现本地提醒。

项目源代码

请直接可从本文复制粘贴即可，运行环境为OSX yosemite10.0，xcode6.1。

创建项目

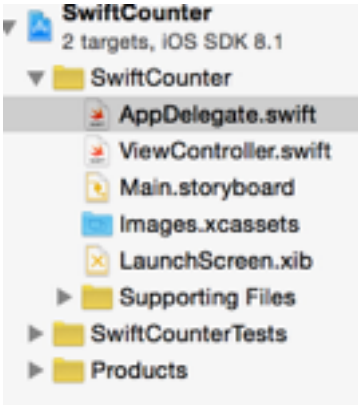
首先，打开Xcode，新建一个项目，Xcode将提示选择一个工程模板。由于我们将从零开始学习，请在左侧窗口选则iOS/Application，右侧窗口选择Single View Application，点击Next，然后在Product Name项填入SwiftCounter，Language注意选择Swift，再点击Next，选择项目保存的路径，最后点击Create即可完成项目创建。
我们可以查看AppDelegate.swift文件。并添加代理类内容，以下为最终AppDelegate.swift中内容，建议您根据提示一步一步添加，最终结果如下：

```
import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?


    func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject: AnyObject]?) -> Bool {
        // Override point for customization after application launch.
        self.window = UIWindow(frame: UIScreen.mainScreen().bounds)
        self.window!.backgroundColor = UIColor.whiteColor()
        self.window!.makeKeyAndVisible()
        self.window!.rootViewController = ViewController()
        application.registerUserNotificationSettings(UIUserNotificationSettings(forTypes:
UIUserNotificationType.Sound | UIUserNotificationType.Alert |
            UIUserNotificationType.Badge, categories: nil
            ))

        return true
    }
```

```
    func applicationWillResignActive(application: UIApplication) {
        // 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 throttle down OpenGL ES frame rates. Games should use this method to pause the game.
    }

    func applicationDidEnterBackground(application: UIApplication) {
        // 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.
    }

    func applicationWillEnterForeground(application: UIApplication) {
        // Called as part of the transition from the background to the inactive state; here you can undo many of the changes made on entering the background.
    }

    func applicationDidBecomeActive(application: UIApplication) {
        // 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.
    }

    func applicationWillTerminate(application: UIApplication) {
        // Called when the application is about to terminate. Save data if appropriate. See also applicationDidEnterBackground:.
    }


}
```

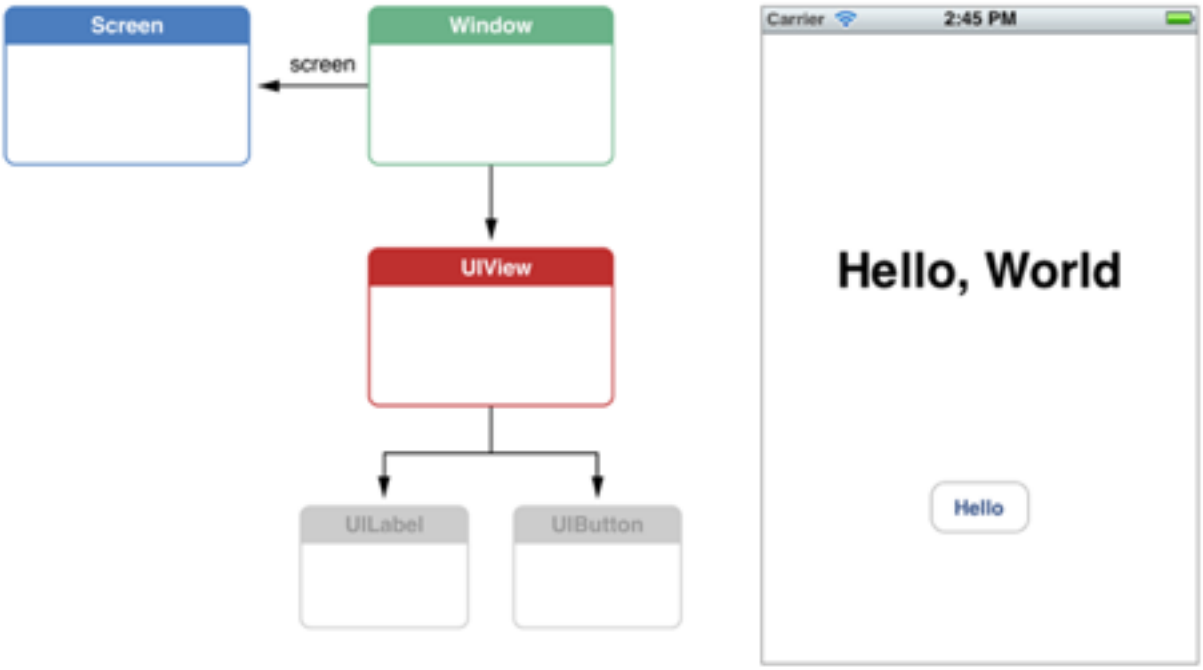
应用代理类（AppDelegate）

AppDelegate类中定义了app进入不同生命周期（包括app启动动、闲置、进入后台、进入前台、激活、完全退出）时的回调方法。实际上在app启动时，app会自动执行一个叫main的入口函数，它通过调用UIApplicationMain函数来创建出AppDelegate类实例，并委托其实现app在不同生命周期的定制行为。
项目缺省建立了如下文件：



屏幕（Screen）、窗口（Window）和视图（View）

在app启动完成的回调方法application:didFinishLaunchingWithOptions中，首先创建一个UIWindow对象。在此，我先简单介绍一下iOS开发中基本UI元素：



UIScreen 代表一块物理屏幕；
UIWindow 代表一个窗口，在iPhone上每个app一般只有一个窗口，而在Mac上一个app经常有多个窗口；
UIView 代表窗口里某一块矩形显示区域，用来展示用户界面和响应用户操作；
UILabel和UIButton，继承自UIView的特定UI控件，实现了特定的样式和行为。
继续看application:didFinishLaunchingWithOptions中的默认实现：

```
self.window = UIWindow(frame: UIScreen.mainScreen().bounds)
self.window!.backgroundColor = UIColor.whiteColor()
self.window!.makeKeyAndVisible()
```

首先，它通过获取主屏幕的尺寸，创建了一个跟屏幕一样大小的窗口；然后将其背景色为白色；并调用makeKeyAndVisible()方法将此窗口显示在屏幕上。

视图控制器（ViewController）

在iOS开发中，主要使用ViewController来管理与之关联的View、响应界面横竖屏变化以及协调处理事务的逻辑。每个ViewController都有一个view对象，定制的UI对象都将添加到此view上。
为了给计时器创建页面并实现功能，我们需要新建一个试图控制器，命名为CounterViewController：点击文件，新建文件，类型选择Swift，然后输入类名CounterViewController，确定。
我们先为其添加基础的结构代码：

```
import Foundation
import UIKit

class CounterViewController : UIViewController {

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

其中重载的方法viewDidLoad非常重要，它在控制器对应的view装载入内存后调用，主要用来创建和初始化UI。
在介绍如何定制计时器UI之前，我们需要先将ViewController的view跟app中唯一的窗口Window关联起来。完成此操作只需在application:didFinishLaunchingWithOptions中加一行代码：

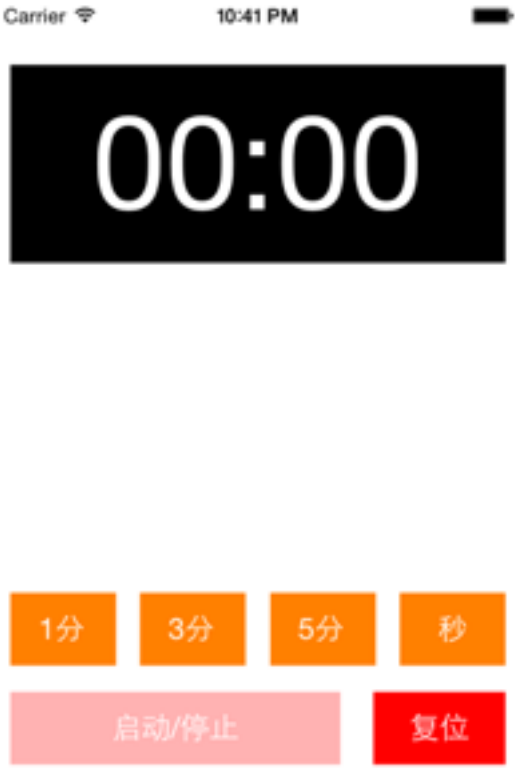
```
self.window!.rootViewController = ViewController()
```

这样ViewController的view会自动添加到Window上，用户启动app后直接看到的将是ViewController中view的内容。
注：在创建Single View Application时，类已经设好，注意不要随意修改类名，一般类名为：

```
class ViewController: UIViewController {

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

创建计时器UI界面

在Xcode中创建UI的方法有很多种，包括使用Nib或Storyboard文件的可视化方式，以及使用纯代码的方式。由于代码创建UI是可视化创建UI的基础，本篇教程只介绍使用代码创建UI的方式。
先看一下我们最终的UI效果图：



在view上方，我们将定义一个标签UILabel来显示剩余时间；
在view下方，我们定义了一排橘黄色的按钮UIButton，接受用户点击操作；
在view最下方，我们定义了启动/停止按钮和复位按钮，接受用户点击操作。
为了能保存和引用这些UI控件，我们要为它们创建属性：

```
    ///UI Controls
    var timeLabel: UILabel? //显示剩余时间
    var timeButtons: UIButton[]? //设置时间的按钮数组
    var timeButtons:[UIButton]? //设置时间的按钮数字这么书写
    var startStopButton: UIButton? //启动/停止按钮
    var clearButton: UIButton? //复位按钮
```

注意，所有的UI变量类型后面都带了?号，表示它们是Optional类型，其变量值可以为nil。Optional类型实际上是一个枚举enum，里面包含None和Some两种类型。nil其实是Optional.None，非nil是Option.Some，它通过Some(T)来包装（wrap）原始值，详细解释请参考Swift之?和!。
Swift中非Optional类型（任何不带?号的类型）必须通过设置默认值或在构造函数init中完成初始化。前面有提到，ViewController中的UI控件主要是在viewDidLoad方法中进行创建和初始化的，所以我们必须将UI控件设置为Optional类型。
其次，考虑到时间按钮的样式和功能基本相同，而且以后可能会增加或删减类似按钮，我们使用一个数组timeButtons来保存所有的时间按钮。同时，因为每个按钮的显示标题跟点击后增加的时间不同，我们还需要定义一个数组timeButtonInfos来保存不同按钮的信息：

```
let timeButtonInfos = [("1分", 60), ("3分", 180), ("5分", 300), ("秒", 1)]
```

timeButtonInfos在初始化之后不会再有改变，所以我们使用let将其定义为常量。它是一个数组，其中每个元素是一个元组（tuple），包含了对应按钮的标题和点击后增加的秒数。
由于UI创建的代码比较多，全部写到viewDidLoad中会很乱。所以接下来，我们定义了3个方法，分别用来完成3部分UI控件的创建：

创建倒计时剩余时间的标签

```
          ///UI Helpers
     func setupTimeLabel() {
        timeLabel = UILabel()
        timeLabel!.text = "00:00"
        timeLabel!.textColor = UIColor.whiteColor()
        timeLabel!.font=UIFont(name:"Arial",size:80) //必须输入一个字体名称。
        //timeLabel!.font = UIFont(_, size: 80)  //不行，估计版本兼容问题。
        timeLabel!.backgroundColor = UIColor.blackColor()
        timeLabel!.textAlignment = NSTextAlignment.Center

        self.view.addSubview(timeLabel!)
    }
```

这个方法首先使用默认构造函数UILabel()创建了一个UILabel实例并赋值给timeLabel属性。然后将timeLabel文本颜色设置为白色、字体设置为80号大小的默认字体、背景色设为黑色，标签中的文本设置为居中对齐。具体UILabel的使用方法，请参考UILabel。
需要注意的是，在赋值时，每个timeLabel后都带上了一个!号，这是因为timeLabel实际上是Optional类型，它像一个黑盒子一样包装了（wrap）原始值，所以在使用它的原始值时必须先用!操作符来拆包（unwrap），详细解释请参考Swift之?和!。
最后，我们将timeLabel添加到了控制器对应的view上。

创建一组时间按钮

```
          func setuptimeButtons() {

        var buttons = [UIButton]()

        for (index, (title, _)) in enumerate(timeButtonInfos) {

            let button: UIButton = UIButton()
            button.tag = index //保存按钮的index
            button.setTitle("\(title)", forState: UIControlState.Normal)

            button.backgroundColor = UIColor.orangeColor()
            button.setTitleColor(UIColor.whiteColor(), forState: UIControlState.Normal)
            button.setTitleColor(UIColor.blackColor(), forState: UIControlState.Highlighted)

            button.addTarget(self, action: "timeButtonTapped:", forControlEvents:
UIControlEvents.TouchUpInside)
            buttons.append(button)

            self.view.addSubview(button)

        }
        timeButtons = buttons
}
```

首先我们创建了一个空数组，用来临时保存生成的按钮。接下来，考虑到在timeButtons中指定位置index的button对应的是timeButtonInfos中相同位置的信息，我们需要获得这个index。
通过使用 enumerate 全局函数我们可以为timeButtonInfos创建一个包含index以及数组中元素（也是元组）的元组(index, (title, _))。由于暂时用不到timeButtonInfos中元组的第二个参数（点击增加的时间），我们使用_替代命名，表示不生成对应的变量。
接着在每次循环开始，我们创建了一个UIButton实例。每个继承自UIView的类（包括UIButton）都继承了属性tag，它主要用一个整数来标记某个view。此处，我们将按钮信息所在的index赋值给button.tag，用来标记button对应的信息所处的位置。
接着我们设置了按钮的标题、背景色、不同点击状态下的标题颜色等，具体UIButton的使用方法请参考UIButton。
除了显示作用，按钮还可以响应用户的点击操作。我们通过addTarget:action:forControlEvents:方法给button添加了可以响应按下按钮并抬起操作的回调方法：timeButtonTapped:。
最后我们将这个临时按钮加入buttons数组，并将此按钮添加到视图上。
当所有按钮创建完毕，我们将这个临时按钮数组赋值给timeButtons，以方便日后引用。

创建2个操作按钮

```
            func setupActionButtons() {

                //create start/stop button
                startStopButton = UIButton()
                startStopButton!.backgroundColor = UIColor.redColor()
                startStopButton!.setTitleColor(UIColor.whiteColor(), forState: UIControlState.Normal)
                startStopButton!.setTitleColor(UIColor.blackColor(), forState:
UIControlState.Highlighted)
                startStopButton!.setTitle("启动/停止", forState: UIControlState.Normal)
                startStopButton!.addTarget(self, action: "startStopButtonTapped:", forControlEvents:
UIControlEvents.TouchUpInside)

                self.view.addSubview(startStopButton)

                clearButton = UIButton()
                clearButton!.backgroundColor = UIColor.redColor()
                clearButton!.setTitleColor(UIColor.whiteColor(), forState: UIControlState.Normal)
                clearButton!.setTitleColor(UIColor.blackColor(), forState: UIControlState.Highlighted)
                clearButton!.setTitle("复位", forState: UIControlState.Normal)
                clearButton!.addTarget(self, action: "clearButtonTapped:", forControlEvents:
UIControlEvents.TouchUpInside)

                self.view.addSubview(clearButton)

        }
```

上面方法中分别为startStopButton设置了点击后的回调方法startStopButtonTapped:；为clearButton设置了点击后的回调方法clearButtonTapped:。
接着，我们简单定义了这几个所需的按钮按下回调方法：

```
     ///Actions & Callbacks
    func startStopButtonTapped(sender: UIButton) {
    }

    func clearButtonTapped(sender: UIButton) {
    }

    func timeButtonTapped(sender: UIButton) {
    }

    func updateTimer(timer: NSTimer) {
    }
```

然后在CounterViewController的viewDidLoad方法中，我们通过调用上面定义好的UI创建方法来创建主页面：

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

        self.view.backgroundColor = UIColor.whiteColor()
        setupTimeLabel()
        setuptimeButtons()
        setupActionButtons()

    }
```

在Xcode中使用快捷键CMD+R运行app，发现预想中的界面并没有出现，这是因为我们还没有设置每个UI控件的位置和大小。
实际上，所有继承自UIView的UI控件类都可以使用init:frame:构造函数来创建指定位置、大小的UI控件。
如果你的app只支持一种方向的屏幕（比如说竖屏），这样做是没问题的；但如果你的app需要同时支持竖屏和横屏，那么最好重载ViewController中的viewWillLayoutSubviews方法。这个方法会在ViewController中的视图view大小改变时自动调用（横竖屏切换会改变视图控制器中view的大小），也提供了最好的时机来设置UI控件的位置和大小。

所以我们在CounterViewController中重载此方法，并为每个UI控件设置了合适的位置和大小：

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

        timeLabel!.frame = CGRectMake(10, 40, self.view.bounds.size.width-20, 120)

        let gap = ( self.view.bounds.size.width - 10*2 - (CGFloat(timeButtons!.count) * 64) ) /
CGFloat(timeButtons!.count - 1)
        for (index, button) in enumerate(timeButtons!) {
            let buttonLeft = 10 + (64 + gap) * CGFloat(index)
            button.frame = CGRectMake(buttonLeft, self.view.bounds.size.height-120, 64, 44)
        }

        startStopButton!.frame = CGRectMake(10, self.view.bounds.size.height-60,
self.view.bounds.size.width-20-100, 44)
        clearButton!.frame = CGRectMake(10+self.view.bounds.size.width-20-100+20,
self.view.bounds.size.height-60, 80, 44)

    }
```

在iOS设备中，视图坐标系是以左上角(0,0)为原点，使用CGRect结构体来表示一个矩形位置，使用CGRectMake全局函数来创建矩形结构体的实例。每个继承自UIView的UI类都有一个类型为CGRect的frame属性，用来保存其在父view中的位置。
我们首先设置了timeLabel的frame，其左上角为(10,40)，宽度为整个CounterViewController中view的宽度-20（为右边也留出10的边），高度为120；
其次我通过循环整个时间按钮数组，为每个按钮设置了合适的frame。这里为了让时间按钮的排列能够自动适应不同屏幕的宽度，我们先计算中每个按钮之间的间距gap，然后根据gap、按钮的宽度64来确定每个按钮的左边距buttonLeft，并最终得到每个按钮的frame。
最后，我们为startStopButton按钮和clearButton也设置了合适的frame。
竖屏模式iPhone4S下，各控件通过计算后得出的frame值如下：

此时，再次CMD+R运行app，就能看到预期的界面了。

添加逻辑功能

界面已开发完毕，现在我们考虑为计时器app添加以下功能：

设置时间
启动和停止倒计时
在计时完成后进行提醒

---

1. 设置时间

在使用倒计时器时，我们发现每次点击时间按钮，当前倒计时的时间会累加；而当开始倒计时时，倒计时的时间又会递减。这些操作都牵扯到一个重要的状态：当前倒计时的时间，而且这个状态是不断变化的。
所以我们考虑为这个状态定义一个变量，表示当前倒计时剩余的秒数：

```
var remainingSeconds: Int = 0
```

我们期望当用户点击时间按钮时，app内部会增加remainingSeconds的值；当点击复位按钮时，会设置remainingSeconds的值为0；当计时开始时，会逐秒减少remainingSeconds的值。并且当remainingSeconds发生变化时，能够及时更新UI，在timeLabel上显示正确的剩余时间。
为实现这些功能，首先我们在时间按钮和复位按钮点击的回调方法中对remainingSeconds值作调整：

```
func clearButtonTapped(sender: UIButton) {
    remainingSeconds = 0
}

func timeButtonTapped(sender: UIButton) {
    let (_, seconds) = timeButtonInfos[sender.tag]
    remainingSeconds += seconds
}
```

按钮回调方法中的唯一参数sender，代表触发此回调方法的控件。在timeButtonTapped:方法中，我们通过控件的tag来找到对应的按钮的信息。按钮信息是一个元组，其中第二个参数存储着每次点击按钮需要增加的秒数，我们将此秒数增加到remainingSeconds上。
现在在设置或复位时间时，remainingSeconds的值可以正常更新了，我们需要考虑如何让UI界面也能及时显示剩余时间。通常的做法，是在按钮的回调方法中，除了设置remainingSeconds的值，也同时通过设置timeLabel的text属性来更新UI。这种做法可以解决问题，但并不最佳方案，因为我们除了需要在timeButtonTapped:中设置UI；也需要在clearButtonTapped:中设置UI；还需要在计时器启动后，在适当的回调中逐秒递减时设置UI。这样会造成很多重复的代码，且难于管理。
其实我们可以使用更Swift的方式来解决状态跟UI的同步问题：使用属性的willSet和/或didSet方法，请参考Property Observers。

```
var remainingSeconds: Int = 0 {
willSet(newSeconds) {
    let mins = newSeconds/60
    let seconds = newSeconds%60
    self.timeLabel!.text = NSString(format:"%02d:%02d", mins, seconds)
}
}
```

在此，我们给remainingSeconds属性添加了一个willSet方法，这个方法会在remainingSeconds的值将要变化的时候调用，并传入变化后的新值作为唯一参数。
在这个方法里，我们先通过整除/和取余%的方式得到倒计时秒数对应的分钟，和除分钟数外的秒数。假如新值为80（秒），那么计算后，mins值为1，second值为20。
然后，我们通过使用Objective-C中定义的字符串类型NSString来格式化这两个数值，让其显示为分钟:秒钟的形式：比如新值为80，那么格式化后的字符串为01:20。这里多提一句，Swift中提供了自带的String类，它能跟Obejctive-C定义的NSString互相兼容。在使用Swift编程时，我们主要使用String来处理字符串，但由于String类目前还没有提供格式字符串相关的方法，我们只能求助于NSString类型。

2.启动和停止倒计时

通过点击启动/停止按钮，我们可以启动或停止倒计时。这种操作能让计时器呈现2种不同的状态：正在计时 和 没有计时。为了实现此功能，我们定义了一个布尔类型变量isCounting：

```
var isCounting: Bool = false
```

同时，在启动计时器后，我们需要每间隔1秒钟就更新一次UI界面的剩余时间。为实现这种定时触发的功能，我们需要用到Foundation库中定义的NSTimer。
为此我们定义了一个NSTimer类型变量timer：

```
var timer: NSTimer?
```

接着，在用户点击启动/停止按钮时触发的回调方法startStopButtonTapped:中，我们切换了isCounting的状态：

```
func startStopButtonTapped(sender: UIButton) {
    isCounting = !isCounting
}
```

同样，为了实现界面同步，我们为isCounting属性添加了willSet方法：

```
var isCounting: Bool = false {
willSet(newValue) {
    if newValue {
        timer = NSTimer.scheduledTimerWithTimeInterval(1, target: self, selector:
"updateTimer:", userInfo: nil, repeats: true)
    } else {
        timer?.invalidate()
        timer = nil
    }
    setSettingButtonsEnabled(!newValue)
}
}
```

当isCounting的新值newValue为true时，我们将通过调用NSTimer的类方法scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:创建并启动一个每1秒钟调用1次updateTimer:方法的timer，并将返回的实例保存到timer中。
同时我们定义了updateTimer:方法来更新当前的倒计时时间：

```
func updateTimer(timer: NSTimer) {
    remainingSeconds -= 1
}
```

当isCounting的新值newValue为false时，我们将暂停timer并将timer设置为nil。此处对timer使用了?修饰符，意思是只有timer是非nil时才做拆包，并调用后面的方法，否则什么也不做。?的用处很多，善用它能写出更安全的代码。
此外，由于时间按钮和复位按钮只在计时器停止时起作用，在计时器启动时无效，我们还提供了辅助方法 setSettingButtonsEnabled: 用来设置这些按钮在不同isCounting状态下的样式（settingButtons是指在设置时间时，也就是计时器停止时可以操作的按钮）：

```
func setSettingButtonsEnabled(enabled: Bool) {
    for button in self.timeButtons! {
        button.enabled = enabled
        button.alpha = enabled ? 1.0 : 0.3
    }
    clearButton!.enabled = enabled
    clearButton!.alpha = enabled ? 1.0 : 0.3
}
```

3.在计时完成后进行提醒

当倒计时自然结束时（不是人为点击启动/停止按钮来停止），如果当前app还处于激活状态（用户没有按Home键退出），那么我们此时将弹出一个警告窗口(UIAlertView)，来提示倒计时已完成：

```
func updateTimer(timer: NSTimer) {
    remainingSeconds -= 1

    if remainingSeconds <= 0 {
        let alert = UIAlertView()
        alert.title = "计时完成！"
        alert.message = ""
        alert.addButtonWithTitle("OK")
        alert.show()
    }

}
```

我们使用通用构造函数创建了一个UIAlertView实例，设置好标题和按钮，最后调用show()方法将其显示出来。效果参照下图：

很多情况下，用户在app计时未结束时就离开了计时器app（计时器处于未激活状态），那么当计时完成时，我们如何来通知用户呢？对这种情况，我们可以使用系统的本地通知UILocalNotification。我们先定义一个辅助方法createAndFireLocalNotificationAfterSeconds:来创建和注册一个N秒钟后的本地提醒事件：

```
    func createAndFireLocalNotificationAfterSeconds(seconds: Int) {

        UIApplication.sharedApplication().cancelAllLocalNotifications()
        let notification = UILocalNotification()

        let timeIntervalSinceNow =  NSNumber(integer: seconds).doubleValue
        notification.fireDate = NSDate(timeIntervalSinceNow:timeIntervalSinceNow);

        notification.timeZone = NSTimeZone.systemTimeZone();
        notification.alertBody = "计时完成！";

        UIApplication.sharedApplication().scheduleLocalNotification(notification);

    }
```

在方法实现中，我们先调用cancelAllLocalNotifications取消了所有当前app已注册的本地消息。之后创建了一个新的本地消息对象notification。

接下来我们要为notifcation设置消息的激活时间。我们通过NSDate(timeIntervalSinceNow: double)构造器创建了从当前时间往后推N秒的一个时间。由于方法接受的参数timeIntervalSinceNow是double类型，我们先将Int类型seconds通过NSNumber方法转换成兼容的NSNumber对象，再调用其doubleValue方法获得对应的值。

如同String之于NSString，Swift中的Int、Float类都能跟Objective-C中的NSNumber类互相兼容。由于Swift中没有提供将Int转换为double类型的方法，我们也不得不求助于NSNumber，通过将Int对象转换成对应的NSNumber对象。

之后我们将本地消息的时区设置为系统时区，提示消息为“计时完成！”。并最终完成此消息的注册。

接下来，我们更新了启动/停止按钮响应的startStopButtonTapped:回调方法：

```
    func startStopButtonTapped(sender: UIButton) {
        isCounting = !isCounting

        if isCounting {
            createAndFireLocalNotificationAfterSeconds(remainingSeconds)
```

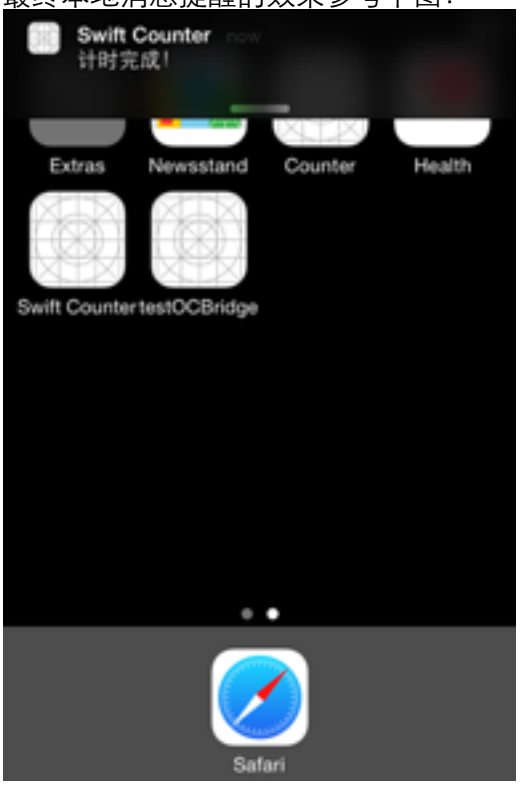
```
        } else {
            UIApplication.sharedApplication().cancelAllLocalNotifications()
        }

    }
```

在启动计时器时创建并注册计时完成时的本地提醒；当计时器停止时，取消当前app所注册的所有本地提醒。

值得注意的是，在iOS8中使用本地消息也需要先获得用户的许可，否则无法成功注册本地消息。因此，我们将询问用户许可的代码片段也添加到了app启动后的入口方法中（AppDelegate中的didFinishLaunchingWithOptions）：

```
        //register notification
        application.registerUserNotificationSettings(UIUserNotificationSettings(forTypes:
UIUserNotificationType.Sound | UIUserNotificationType.Alert |
            UIUserNotificationType.Badge, categories: nil
            ))
```

最终本地消息提醒的效果参考下图：



至此，一个完整的倒计时app已开发完毕。

后记

1.在倒计时结束后，要加一行代码self.isCounting = false，用以停止重复alert().

```
    func updateTimer(timer: NSTimer) {
        remainingSeconds -= 1

        if remainingSeconds <= 0 {
            let alert = UIAlertView()
            alert.title = "计时完成！"
            alert.message = ""
            alert.addButtonWithTitle("OK")
            alert.show()
            self.isCounting = false     //增加此行
        }

    }
```

2.在刚打开“倒计时器”后，建议对timeLabel进行初始化，self.timeLabel!.text = “00:00″; 然后，点击启动/停止按钮后，进行判断提示用户先设置倒计时间
建议自己先处理，再看下面代码。

```
   func prewaring(){
        let alert=UIAlertView()
        alert.title="请先设置时间"
        alert.message = ""
        alert.addButtonWithTitle("OK")
        alert.show()
    }
```

在viewDidLoad中调用即可。

```
override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        self.view.backgroundColor = UIColor.whiteColor()
        setupTimeLabel()
        setuptimeButtons()
        setupActionButtons()
        prewaring()  //增加此行

    }
```

通过完成此教程，我对Swift语言的理解也更进了一步。Swift是一门全新的语言，作为开发者，我们需要不断加深对这门语言的理解，并灵活使用语言提供的特性来编程。虽然在开发中还需要大量使用Cocoa(Touch)中提供的Objective-C类库，但编程的方式已经完全改变了，不仅仅是将Objective-C代码翻译成Swift代码，而需要在代码层面进行重新思考。
本文根据以下内容修撰而成
1：Swiftist社区 - 从零开始学Swift计时器App开发 原作者：Jason Li @lifedim
2: Let's Swif实战 – 从零开始学Swift计时器App开发 BY 史薇芙特 转载。

3:修撰后部分源码如下，因为自己在学习中非常费劲，所以希望这篇文章能带给你快乐。

```
//
//  ViewController.swift
//  SwiftCounter
//

import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        self.view.backgroundColor = UIColor.whiteColor()
        setupTimeLabel()
        setuptimeButtons()
        setupActionButtons()
        prewaring()

    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
    var timeLabel: UILabel? //显示剩余时间
    var timeButtons: [UIButton]? //设置时间的按钮数组
    var startStopButton: UIButton? //启动/停止按钮
    var clearButton: UIButton? //复位按钮
    let timeButtonInfos = [("1分", 60), ("3分", 180), ("5分", 300), ("秒", 1)]
    ///UI Helpers
    func setupTimeLabel() {
        timeLabel = UILabel()
        timeLabel!.text = "00:00"
        timeLabel!.textColor = UIColor.whiteColor()
        timeLabel!.font=UIFont(name:"Arial",size:80) //必须输入一个字体名称。
        //timeLabel!.font = UIFont(_, size: 80)  //不行，估计版本兼容问题。
        timeLabel!.backgroundColor = UIColor.blackColor()
        timeLabel!.textAlignment = NSTextAlignment.Center

        self.view.addSubview(timeLabel!)
    }
    func setuptimeButtons() {
```

```
        var buttons = [UIButton]()

        for (index, (title, _)) in enumerate(timeButtonInfos) {

            let button: UIButton = UIButton()
            button.tag = index //保存按钮的index
            button.setTitle("\(title)", forState: UIControlState.Normal)

            button.backgroundColor = UIColor.orangeColor()
            button.setTitleColor(UIColor.whiteColor(), forState: UIControlState.Normal)
            button.setTitleColor(UIColor.blackColor(), forState: UIControlState.Highlighted)

            button.addTarget(self, action: "timeButtonTapped:", forControlEvents:
UIControlEvents.TouchUpInside)
            buttons.append(button)

            self.view.addSubview(button)

        }
        timeButtons = buttons

    }
    func setupActionButtons() {

        //create start/stop button
        startStopButton = UIButton()
        startStopButton!.backgroundColor = UIColor.redColor()
        startStopButton!.setTitleColor(UIColor.whiteColor(), forState: UIControlState.Normal)
        startStopButton!.setTitleColor(UIColor.blackColor(), forState: UIControlState.Highlighted)
        startStopButton!.setTitle("启动/停止", forState: UIControlState.Normal)
        startStopButton!.addTarget(self, action: "startStopButtonTapped:", forControlEvents:
UIControlEvents.TouchUpInside)

        self.view.addSubview(startStopButton!)

        clearButton = UIButton()
        clearButton!.backgroundColor = UIColor.redColor()
        clearButton!.setTitleColor(UIColor.whiteColor(), forState: UIControlState.Normal)
        clearButton!.setTitleColor(UIColor.blackColor(), forState: UIControlState.Highlighted)
        clearButton!.setTitle("复位", forState: UIControlState.Normal)
        clearButton!.addTarget(self, action: "clearButtonTapped:", forControlEvents:
UIControlEvents.TouchUpInside)

        self.view.addSubview(clearButton!)

    }

    //var remainingSeconds: Int = 0
    var remainingSeconds: Int = 0 {
        willSet(newSeconds) {
            let mins = newSeconds/60
            let seconds = newSeconds%60
            self.timeLabel!.text = NSString(format:"%02d:%02d", mins, seconds)
        }
    }
    //var isCounting: Bool = false
    var isCounting: Bool = false {
        willSet(newValue) {
            if newValue {
                timer = NSTimer.scheduledTimerWithTimeInterval(1, target: self, selector:
"updateTimer:", userInfo: nil, repeats: true)
            } else {
                timer?.invalidate()
                timer = nil
            }
            setSettingButtonsEnabled(!newValue)
        }
    }
    var timer: NSTimer?


    func startStopButtonTapped(sender: UIButton) {
        isCounting = !isCounting
        if isCounting {
            createAndFireLocalNotificationAfterSeconds(remainingSeconds)
        } else {
            UIApplication.sharedApplication().cancelAllLocalNotifications()
        }

    }
```

```
    func clearButtonTapped(sender: UIButton) {
        remainingSeconds = 0

    }

    func timeButtonTapped(sender: UIButton) {
        let (_, seconds) = timeButtonInfos[sender.tag]
        remainingSeconds += seconds

    }

    override func viewWillLayoutSubviews() {
        super.viewWillLayoutSubviews()

        timeLabel!.frame = CGRectMake(10, 40, self.view.bounds.size.width-20, 120)

        let gap = ( self.view.bounds.size.width - 10*2 - (CGFloat(timeButtons!.count) * 64) ) /
CGFloat(timeButtons!.count - 1)
        for (index, button) in enumerate(timeButtons!) {
            let buttonLeft = 10 + (64 + gap) * CGFloat(index)
            button.frame = CGRectMake(buttonLeft, self.view.bounds.size.height-120, 64, 44)
        }

        startStopButton!.frame = CGRectMake(10, self.view.bounds.size.height-60,
self.view.bounds.size.width-20-100, 44)
        clearButton!.frame = CGRectMake(10+self.view.bounds.size.width-20-100+20,
self.view.bounds.size.height-60, 80, 44)

    }

    func updateTimer(timer: NSTimer) {
        remainingSeconds -= 1

        if remainingSeconds <= 0 {
            let alert = UIAlertView()
            alert.title = "计时完成！"
            alert.message = ""
            alert.addButtonWithTitle("OK")
            alert.show()
            self.isCounting = false
        }

    }

    func setSettingButtonsEnabled(enabled: Bool) {
        for button in self.timeButtons! {
            button.enabled = enabled
            button.alpha = enabled ? 1.0 : 0.3
        }
        clearButton!.enabled = enabled
        clearButton!.alpha = enabled ? 1.0 : 0.3
    }

    func createAndFireLocalNotificationAfterSeconds(seconds: Int) {

        UIApplication.sharedApplication().cancelAllLocalNotifications()
        let notification = UILocalNotification()

        let timeIntervalSinceNow =  NSNumber(integer: seconds).doubleValue
        notification.fireDate = NSDate(timeIntervalSinceNow:timeIntervalSinceNow);

        notification.timeZone = NSTimeZone.systemTimeZone();
        notification.alertBody = "计时完成！";

        UIApplication.sharedApplication().scheduleLocalNotification(notification);

    }
    func prewaring(){
        //var value = self.timeLabel!.text;
        //if(value == “00:00”){
        let alert=UIAlertView()
        alert.title="请先设置时间"
        alert.message = ""
        alert.addButtonWithTitle("OK")
        alert.show()
    }

} //end
```