

欢迎继续回来一起学习iPhone开发。

**热身结束，让我们真正来做游戏吧！**

到目前为止我们已经完成了基本的用户界面，而且也学习了如何确定滑动条的位置，这样我们的to-do清单上已经解决了一大部分内容。

剩下的主要事情就是生成目标随机数，然后计算玩家的得分了。

不过在此之前让我们先对滑动条做一些改进。

## Outlets

这货不是奥特莱斯这种土豪长逛的购物场所，更不是插座，而是某种接口。

在之前的学习中，我们已经学会了如何把滑动条的数值保持在变量中，然后将其显示在提示对话框中。这个算是不小的进步了，不过我们还可以做的更好。

想想看，如果你想把storyboard里面的滑动条初始数值设置为1或者100，又该怎么办呢？currentValue这个时候的数值就是错的，因为应用始终假定它的初始值是50。

如果你的记忆力足够好，最简单的方法似乎是给currentValue赋予一个新的初始值。这样似乎是理所当然的。

不过当你的项目越来越大时，里面会有几个，十几个甚至更多的视图控制器，你不可能一个个去手动修改。更可怕的是，你很可能忘了还有这个事情要去处理。

所以我们最好用下面的方法来处理：

回到Xcode，点击ViewController.swift，找到viewDidLoad()方法，其中的代码如下：

```
override func viewDidLoad() {  
    super.viewDidLoad()  
    // Do any additional setup after loading the view, typically from a nib.  
}
```

当我们使用Xcode模板来创建新的项目时，Xcode就会自动把viewDidLoad()方法放进源代码。这里我们需要添加点新的代码。

当视图控制器从storyboard文件中加载用户界面时，UIKit会立即发送viewDidLoad()消息。此时视图控制器仍然不可见，因此在这里设置变量的初始值会非常的合适。

更改viewDidLoad()的方法如下：

```
override func viewDidLoad() {  
    super.viewDidLoad()  
    currentValue = lroundf(slider.value)  
}
```

为什么这样来处理呢？因为这里我们会直接获取storyboard中滑动条的初始值（不管是50，1还是100），然后把它作为currentValue的初始值。

作为一个强迫症重度患者，你可能已经看到红色的错误提示了，Use of undeclared identifier 'slider'



肿么会这样？因为viewDidLoad()方法对slider这个东西一无所知。

且慢，之前在sliderMoved()方法里面似乎用到过吧？不信你来看：

```
@IBAction func sliderMoved(slider: UISlider){  
  
//    println("滑动条的当前数值是： \ \(slider.value)")  
    currentValue = lroundf(slider.value)  
  
}
```

这里我们同样用到了slider.value，我们对slider.value四舍五入，然后保存到currentValue。那么为毛在sliderMoved()方法中 useful，但是在viewDidLoad()方法中没用呢？

区别在于这里的slider指的是sliderMoved()这个方法的参数。

```
@IBAction func sliderMoved(slider: UISlider) {
```

所谓的参数就是方法名称后面圆括号中的东西。这里只有一个名为slider的参数，它代表的是用于发送这个动作消息的UISlider对象。

动作方法通常有一个参数，代表触发该方法的UI控件。当我们需要在方法中使用该对象的时候，这种做法就会非常有用。

在我们这个例子里，当用户拖动滑动条时，UISlider滑动条对象会说，“视图控制器你好~我是一个滑动条对象，刚才我被人动了。顺便说一下，这是我的电话号码，你可以通过它来联系我。”

至于视图控制器这个大哥会不会来给滑动条报仇，就不是我们要讨论的事情了~

此时的slider变量包含了“电话号码”，可惜的是它只在这个特定的方法中存活。

换句话说，它是一个本地变量(local)。一旦方法执行完毕，它的生命也就终结了。

### 科普时间：本地变量

之前我们介绍变量的时候，曾经提到过每一个变量都有特定的生命周期，又被成为scope（我朝NB人士喜欢叫作用域，也有叫范围的）。一个变量的作用范围取决于我们定义该变量的位置。

在Swift语言中有三种作用范围水平：

1.Global scope 全局范围（或者全局域，叫神马不重要）。

此类变量可以在应用的整个生命周期里面存活，而且可以从任何一个地方来访问。

2.Instance scope 实例范围

此类变量可以在对象的生命周期里面存活，比如currentValue这个变量。这些变量也活的比较久，只要拥有它们的对象还活着，它就不会挂。

3.Local scope 本地范围

比如sliderMoved()方法中的slider参数，它只能在方法体内存活。只要程序执行离开当前方法，这些本地变量就挂了。

让我们看看showAlert()方法中的代码：

```

@IBAction func showAlert(){

    let message = "滑动条的当前数值是： \(currentValue)"

    let alert = UIAlertController(title:"Hello Messi",
                                message:message,
                                preferredStyle: .alert)
    let action = UIAlertAction(title:"ok",style: .default,handler: nil)
    alert.addAction(action)

    present(alert, animated: true, completion: nil)
}

```

因为message,alert和action对象是在方法体内创建的，因此它们都属于本地变量。它们只能在showAlert()动作执行期间存活。

一旦showAlert()方法执行完毕，计算机就会销毁message,alert和action对象。它们的存储空间也会被释放出来。

currentValue这个变量则是不老仙翁。事实上它几乎是永生的，只要它赖以生存的ViewController小宇宙还在，它就会一直存在。这种类型的变量又被称之为实例变量，因为它的作用范围和它的主人是一样的。

换句话说，我们可以用实例变量来保存一个数值，而且在不同的动作方法中都可以使用。

因此，真正有效的方法是将滑动条的引用保存到一个实例变量中，正如我们对currentValue所做的那样。

只不过这一次的变量数据类型不是int（整数），而是UISlider(滑动条对象)。而且我们不是用的常规实例变量，而是一个特殊的实例变量，其类型是outlet。

回到Xcode，在ViewController.swift中添加如下的一行代码：

```
@IBOutlet weak var slider:UISlider!
```

那么这行代码应该添加在哪里呢？

其实并不重要，只要你确保这行代码在class ViewController的花括号中，当然方便起见最好是紧接着class ViewController: UIViewController { 这行代码的下面。

修改后的代码如下：

```

//
// ViewController.swift
// BullsEye
//
// Created by eseedo on 10/11/17.
// Copyright © 2017 eseedo. All rights reserved.
//

import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var slider:UISlider!

    var currentValue: Int = 50

    override func viewDidLoad() {
        super.viewDidLoad()
        currentValue = lroundf(slider.value)
    }

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

    @IBAction func showAlert(){

        let message = "滑动条的当前数值是： \(currentValue)"

        let alert = UIAlertController(title:"Hello Messi",
                                     message:message,
                                     preferredStyle: .alert)
        let action = UIAlertAction(title:"ok",style: .default,handler: nil)
        alert.addAction(action)

        present(alert, animated: true, completion: nil)
    }

    @IBAction func sliderMoved(_ slider: UISlider){

        print("滑动条的当前数值是： \(slider.value)")
        currentValue = lroundf(slider.value)
    }
}

```

```
}  
}
```

以上代码中仅黄色高亮部分是新添加的，其它没有任何变化。

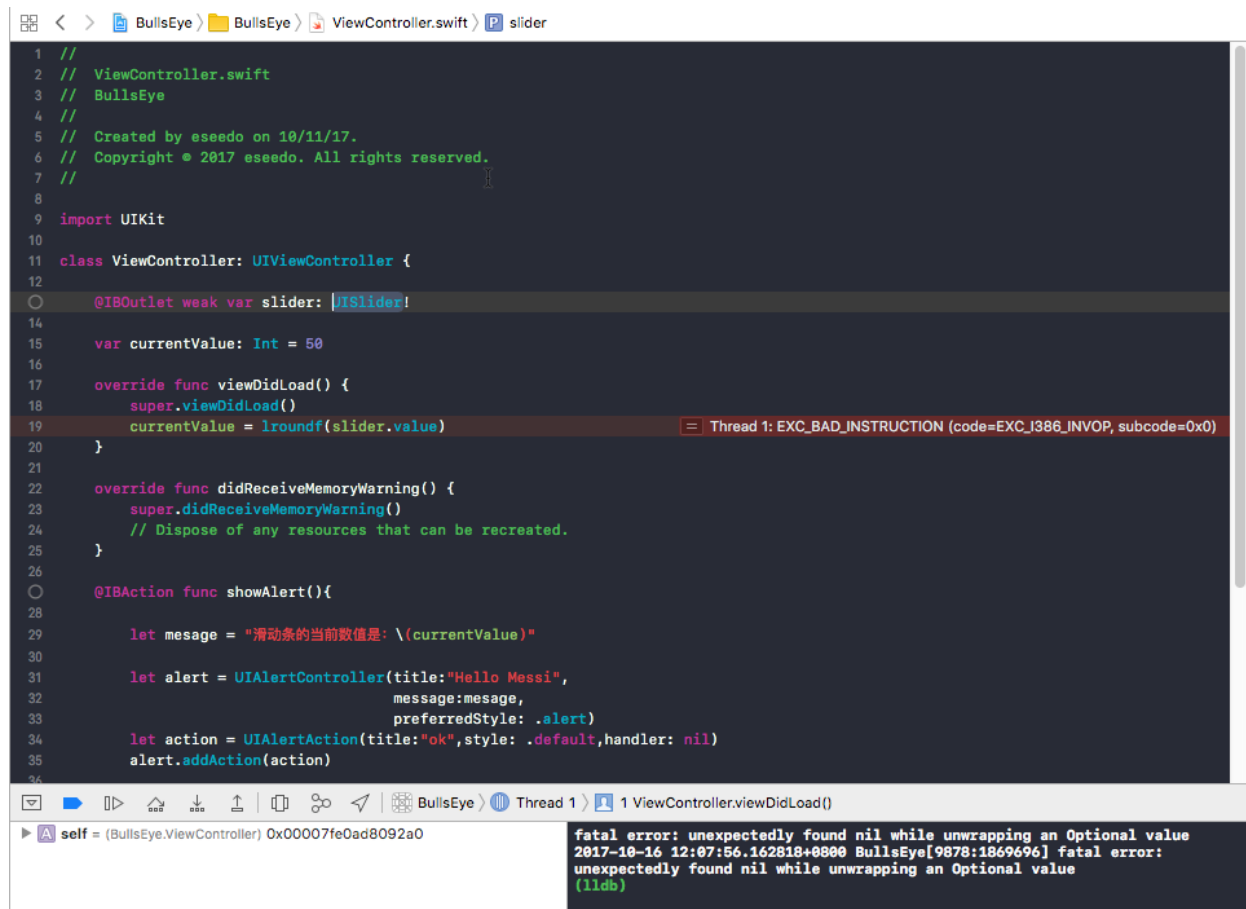
这行代码告诉Interface Builder，现在有一个名为slider的变量，它可以和某个UISlider对象关联在一起。

正如Interface Builder喜欢将方法称之为actions动作，而对这些变量习惯称之为outlet。Interface Builder会根据变量前面的@IBOutlet来判断。

至于这行代码中的weak先不必紧张。在后面的教程中会向大家解释为什么要加这个奇怪的关键词。现在只需要记住当我们创建一个outlet类型变量时，应该使用@IBOutlet weak var的形式。（好吧，最后还有个惊叹号比较难懂，有时你也会看到一个问号，这些东西会在后面来介绍。）

那么这样问题就解决了吗？显然不是。

点击工具栏上的编译运行按钮，会看到下面的错误提示：



```
1 //  
2 // ViewController.swift  
3 // BullsEye  
4 //  
5 // Created by eseedo on 10/11/17.  
6 // Copyright © 2017 eseedo. All rights reserved.  
7 //  
8  
9 import UIKit  
10  
11 class ViewController: UIViewController {  
12  
13     @IBOutlet weak var slider: UISlider!  
14  
15     var currentValue: Int = 50  
16  
17     override func viewDidLoad() {  
18         super.viewDidLoad()  
19         currentValue = lroundf(slider.value)  
20     }  
21  
22     override func didReceiveMemoryWarning() {  
23         super.didReceiveMemoryWarning()  
24         // Dispose of any resources that can be recreated.  
25     }  
26  
27     @IBAction func showAlert(){  
28  
29         let message = "滑动条的当前数值是: \(currentValue)"  
30  
31         let alert = UIAlertController(title:"Hello Messi",  
32                                     message:message,  
33                                     preferredStyle: .alert)  
34         let action = UIAlertAction(title:"ok",style: .default,handler: nil)  
35         alert.addAction(action)  
36     }  
37 }  
38  
39
```

fatal error: unexpectedly found nil while unwrapping an Optional value  
2017-10-16 12:07:56.162818+0800 BullsEye[9878:1869696] fatal error:  
unexpectedly found nil while unwrapping an Optional value  
(lldb)

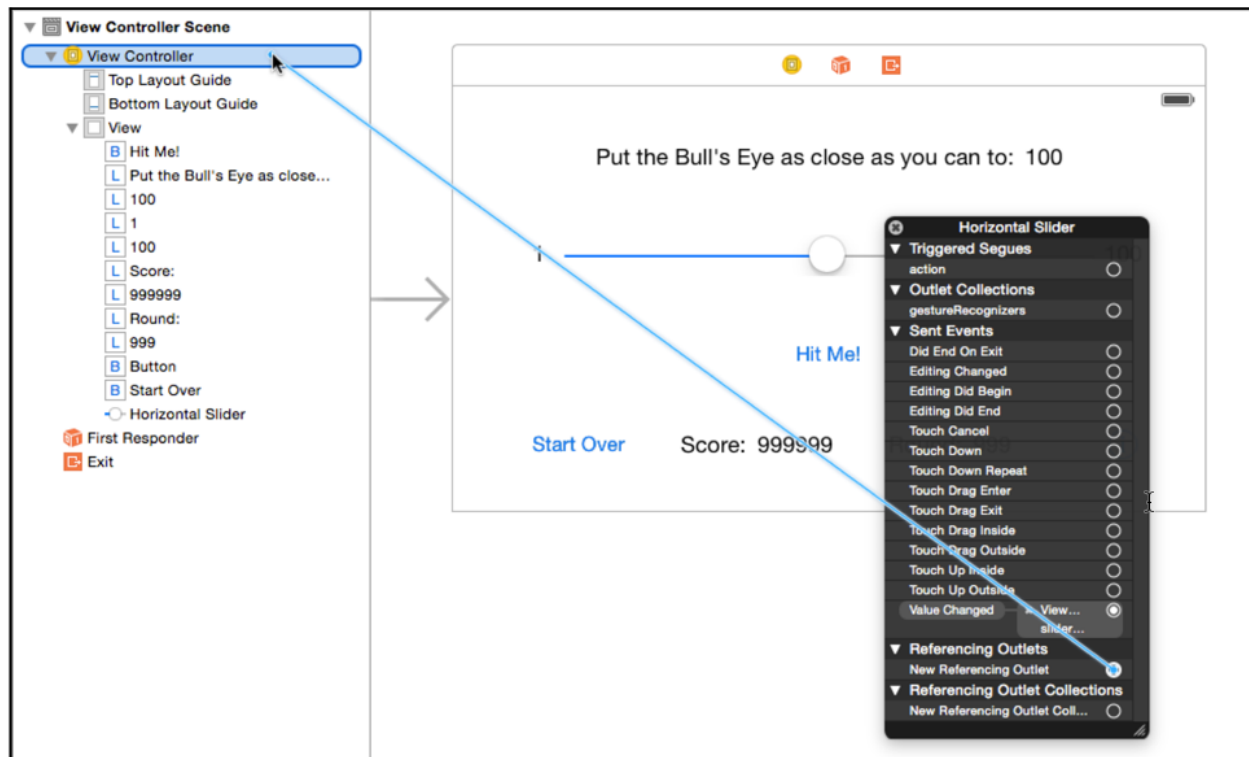
那么为什么还是会报错呢？

记住：一个outlet变量必须跟storyboard中的某个对象关联在一起。我们刚刚定义了outlet变量，但是忘了将其和storyboard中的对象关联，所以程序就崩溃了~

打开storyboard，右键点击slider，会看到一个弹出菜单，显示了slider的所有关联。

这个弹出菜单的作用和Connections inspector完全相同。我只是希望告诉你还有这种替代方式。

点击New Referencing Outlet旁边的空心圆，然后拖出一条线到ViewController:



在显示的弹出式菜单中选择slider。

这个slider就是我们刚刚在代码中所添加的对象。这里我们已经成功的将storyboard中的slider对象和视图控制器中的slider outlet关联在一起。

ok了，点击Run按钮运行应用，直接触碰按钮，它会告诉你滑动条的当前数值是50。

关闭应用，切换到Interface Builder，更改滑动条的初始值，改成你喜欢的某个数值。再次运行应用，触碰按钮，你会看到提示对话框中的数值就是你更改后的初始值。

当然，玩过了之后，最好还是把滑动条的初始值改为50，这样方便后面操作~

## 科普：关于错误和警告

在开发iOS应用时，Xcode有时候会给你发黄牌警告，严重的则会直接红牌驱逐出场。黄色的警告信息出现后，你还是可以运行自己的应用，只不过在未来的某个时候会发现自己的应用莫名崩溃。当你看到红色的错误提示时，就表示你犯的错是致命的，应用根本就没办法运行。

通常情况下，作为程序猿，我们要确保代码中没有一行错误。但很多程序猿会选中忽略代码中的黄色警告，因为它们看起来不会让程序直接崩溃。不过我个人的原则是，尽可能消灭一切的黄色警告。当然，这个也不是绝对的，在某些情况下我们也需要容忍黄色的警告。具体来说，当代码中用到了第三方的类库时，因为第三方类库未必兼容最新版本的iOS，就会导致大量的警告信息。最典型的如cocos2d，每次iOS版本更新后，cocos2d的旧版本并不能保证第一时间支持最新的iOS类库。

此外，当我们使用github，google code上的一些开源项目时，也会出现类似的情况。对于此类的警告，我们有两种处理方式。一种是选择使用较低版本的iOS，以消除警告；一种是等待最新版本的第三方类库，以消除警告。还有第三种方式就是自己手动更改，但如果是cocos2d这样的规模较大的第三方类库，就必须慎重了。

但即便是此类情况所造成的警告，程序猿也应该在出现警告的地方说明原因，以及未来可能的解决方法，以及可能造成的bug，以便测试的时候来确认和改进。

如果你是产品经理，在程序猿提交的源代码中发现了大量的黄色警告，而且没有在出现警告的地方注释说明是什么原因造成的，就一定要打回去让他们仔细再看看。不要被一大堆专业术语忽悠，不行就是不行，没有什么废话。

0错误，0警告，是程序猿所必须追求的事情。即便当前版本不能实现，也一定要在后续的版本中纠正，而不能放任不管，最终酿成大错。

## 附赠小技巧

在结束之前，再附上一个使用Xcode的小技巧，相信对新老朋友都会有所帮助。这个小技巧是在WWDC2013上由苹果的某个美女程序媛分享的。



假定你使用的是Macbook Air等屏幕相对较小的Mac电脑，那么会经常感觉自己的屏幕空间太小，特别是有多个视图控制器的时候。

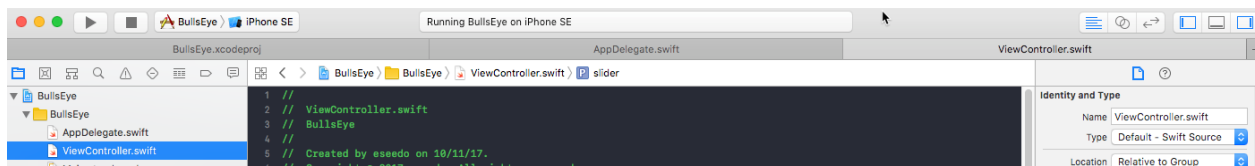
那么Xcode提供了一个类似Safari 浏览器Tab的概念。也就是可以在一个界面中打开多个子界面，然后可以轻松跳转而互不影响。

具体是怎样的呢？请看下图。

首先，我会让整个Xcode界面全屏显示，从而focus在开发之中，而不受其它程序的影响。

其次，我使用Tab建立了多个子界面，分别用于项目导航，界面编辑和代码编辑。

创建Tab的方式是在Xcode中使用Command +T的快捷键，然后双击修改相关子界面的名称即可。

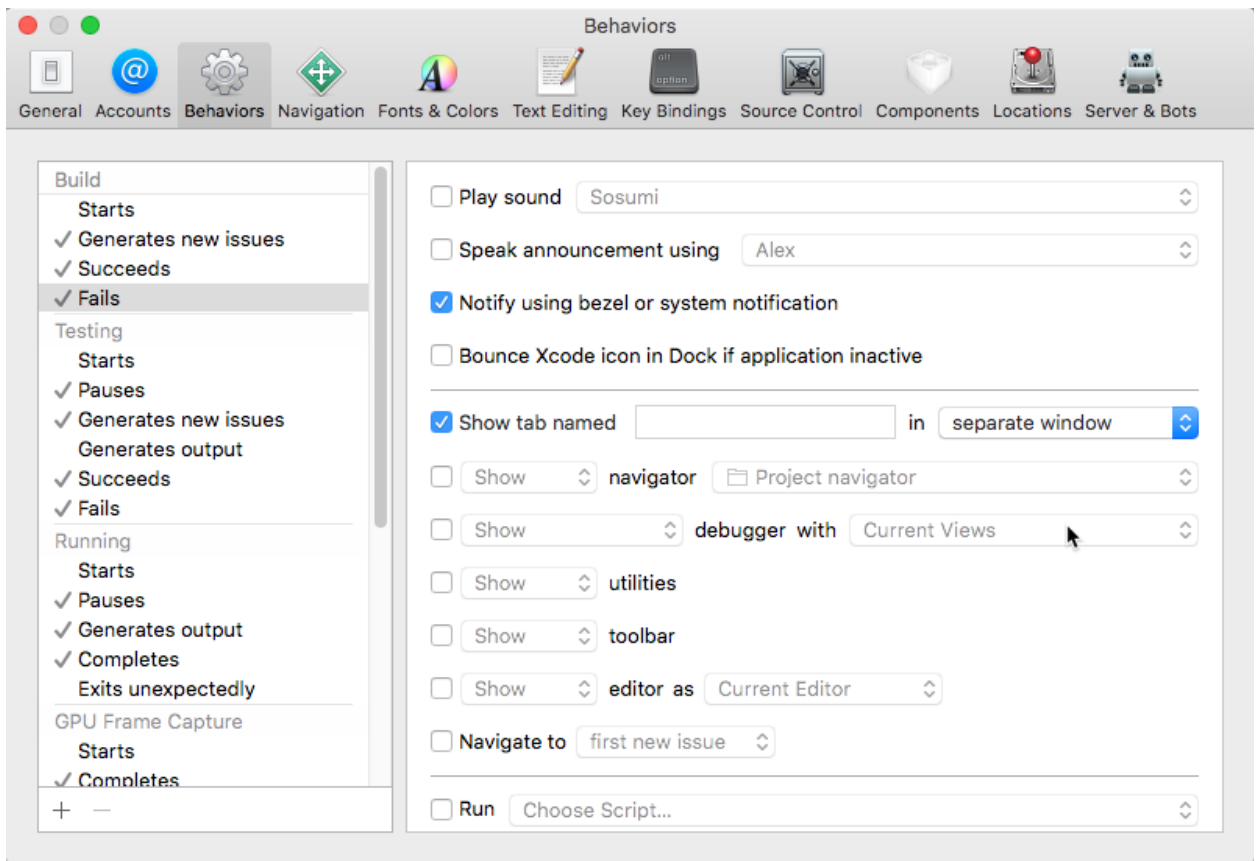


注意到这里的Tab就和浏览器打开的不同网页一样可以随意跳转而不受影响，而且名字是可以自己随便取的。如果不再需要其中的某个子界面了，直接点x关掉就好了。

这样一来，其实也会给编程增加了小小的乐趣。

3.通过在Xcode的Preferences—>Behaviors里面设置还可以设置动态的Tab，比如在进行编译的时候单独打开一个Debug的子界面。

工欲善其事必先利其器，使用Tab来尽可能利用屏幕空间，在代码注释区写小说，就和之前设置代码区的背景和字体一样，看似对编程没有任何实际帮助，但实际上这些不起眼的



举动至少会让你的编程工作没那么枯燥，甚至多一些意想不到的乐趣。当然，在代码注释区写小说神马的还是算了，毕竟会影响到别人。不过如果是你自娱自乐或者学习的话，完全可以这么做。

ok,在一大堆概念之后，又到了发送福利的时间。