# 让不懂编程的人爱上iPhone开发(2013球iOS7版)-第7篇

欢迎回来继续我们的学习。

现在我们的to-do 清单上还有不少事情要做，接下来先处理一个比较简单的：产生一个随机数，然后让它显示在屏幕上。
在游戏开发的时候，我们经常会用到随机数。当然，大家最感兴趣的随机数可能就是体彩双色球了，可惜的是彩票里面的所谓随机数未必随机啊。。。我会告诉你连续追5年的号只中过最高5元钱吗？还好不是五毛，说多了都是泪啊。


算了还是别做梦了，来说说游戏里面的随机数吧。在此之前提一句，如果哪位哥中了头奖，别忘了留个联系方式，土豪我们做个朋友吧！实际上，我们不可能让一个计算机真正

产生完全随机不可预测的数字，而使用所谓的伪随机数生成器来产生类似的效果。这里我们将用到一个比较常用的，arc4random()函数。
你应该还记得函数和方法的区别吧？不怕麻烦再重复一遍，函数不属于某个对象，而方法则必须寄生于某个对象。
在我们这个游戏里，最适合生成随机数的地方当然就是游戏开始处。
在viewDidLoad方法中添加以下代码：

```
targetValue = 1 + (arc4random() % 100);
```

此时viewDidLoad方法的代码如下：

```
- (void)viewDidLoad
{
   [super viewDidLoad];
   currentValue = self.slider.value;
   targetValue = 1 + (arc4random() % 100);
}
```

当然，你也会看到一个红色的错误提示信息。这里我们做了什么呢？首先使用了一个新的变量targetValue。因为之前没有定义过，所以你看到了错误提示。待会儿我们会修复这个问题。如果不告诉编译器targetValue这个变量的类型是什么，它就不知道该为这个变量分配多少存储空间，也不能检查我们在其它地方是不是正确的使用了这个变量。
这里我们调用了arc4random()函数来产生一个随机整数，它的范围在0到40亿之间。显然滑动条的数值不可能有那么高，所以我们希望把这个随机数限定在1到100之间。为了实现这一点，首先用一个模操作数(%)来限制arc4random()的范围在0到99之间，然后再加上1，让它的范围变成1-100。好吧，什么是模操作数？所谓的模是Mod的音译，实际就是求余的意思。举个例子，11 % 2，得到的余数是1。任何一个整数%2的结果只能是0或者1，也就是0-1之间。所以arc4random()%100实际上得到的是0到99之间的随机数。
希望你别被上面的数学运算吓倒。虽然很多人说只有学好数学才能成为一个好的程序猿，但实际上绝大多数应用中所用到的数学都不会比上面的算术复杂。当然，在游戏和科学计算中用到的数学要稍微复杂一些。不过你的目的并不是成为一个算法高手或者数学家，所以也不必太担心。

让我们来消灭红牌吧。在实例变量声明部分添加targetValue这个变量：

```
@interface CrazyDragViewController (){
   int currentValue;
   int targetValue;
}
```

这里只是定义了一个新的int类型的变量，名为targetValue。当然，我希望你明白为什么要把targetValue定义为实例变量。这是因为我们不光在viewDidLoad方法中要用到它，在后面的showAlert方法中也需要用到它。

更改showAlert的方法如下：

```
- (IBAction)showAlert:(id)sender {

   NSString *message = [NSString stringWithFormat:@"滑动条的当前数值是:%d，我们的目标数值是：%d",currentValue,targetValue];

   [[[UIAlertView alloc]initWithTitle:@"您好，苍老师" message:message delegate:nil cancelButtonTitle:@"本老师知道了" otherButtonTitles:nil, nil]show];

}
```

在上面的代码中，我们在message字符串里面加上了一句话，其中的第一个%d实际运行的时候会用currentValue变量中的数值来替代，而第二个%d实际运行时会用targetValue变量中的数值来替代。
\n这个符号之前没接触过。实际上它在字符串中出现的时候，就意味着我们要换行了。

好了，点击Run运行游戏，试试看有什么变化。

新的回合

如果你多试几次，会发现对话框所提示的随机数目标数值几乎就没改变过。这样的话这游戏也就没啥意思了。还好意思说自己是随机数，一次都不变。这就好比双色球每期中奖号码都是重复的，还有谁愿意买？（我！！！）这是因为我们在viewDidLoad方法里面生成的随机数，而之后就没有再次生成。而这个方法只会在应用第一次打开创建视图控制器的时候才会执行，之后就没它的事了。
我们的to-do清单上写的是，”在游戏的每个回合开始的时候生成一个随机数“。好吧，这里的回合指的是什么呢？
当游戏开始的时候，玩家的初始分数是0，游戏回合数是1。我们把滑动条设置在中间的位置（50）,然后计算一个随机数。接着我们等玩家触碰求婚体按钮。当玩家触碰按钮之后，一轮游戏就结束了。这个时候我们会计算这一轮的得分，然后把它添加到总得分里面去。接着我们让游戏回合数加1，然后开启新的一轮游戏。我们再次把滑动条重置到中间

的位置，然后计算一个新的随机数。就这样，只要你愿意，直到地老天荒，直到宇宙的终结（实际上直到你的手机没电为止~）。
每当你发现自己在想，”在新一轮的游戏开始后我们要做这个，做那个。。。“，这个时候就该创建一个新的方法了。这个方法会创建自己的独有功能。

记住这一点，让我们继续前进吧 。
回到Xcode,在CrazyDragViewController.m中找到viewDidLoad方法。在这个方法的前面添加一个新的方法：

```
-(void)startNewRound{

   targetValue = 1+(arc4random()%100);
   currentValue = 50;
   self.slider.value = currentValue;

}
```

然后更改viewDidLoad方法的代码如下：

```
- (void)viewDidLoad
{
   [super viewDidLoad];
      // Do any additional setup after loading the view, typically from a nib.

   [self startNewRound];
}
```

这里和我们之前所做的事情没有太大区别，只不过我们把开始新一轮游戏的逻辑机制放到了一个独立的方法startNewRound里面去。这样做的好处是，当玩家触碰按钮显示提示对话框后，可以在showAlert方法里面调用这个方法来开启新的一轮游戏。

让我们更改showAlert方法里面的代码：

```
- (IBAction)showAlert:(id)sender {

   NSString *message = [NSString stringWithFormat:@"滑动条的当前数值是:%d，我们的目标数值是：%d",currentValue,targetValue];

   [[[UIAlertView alloc]initWithTitle:@"您好，苍老师" message:message delegate:nil cancelButtonTitle:@"本老师知道了" otherButtonTitles:nil, nil]show];
```

```
    [self startNewRound];
}
```

这里添加的唯一一行新代码就是最后的[self startNewRound]; 之前我们曾经用self 来表示视图控制器的属性，比如self.slider.，而这里我们用self来调用视图控制器的某个方法。

在此之前，我们所接触到的视图控制器里面的方法都是在某个事件发生的时候调用：比如当应用加载视图控制器的时候调用viewDidLoad方法，当玩家触碰按钮的时候调用showAlert方法，当玩家拖动滑动条的时候调用sliderMoved方法。

以上的方法调用都是自动的，而这里开启新的一轮游戏则是手动调用方法的例子。通常我们用[object methodName]的形式来手动调用方法。其实我们对NSString字符串对象已经使用过这样的形式，比如[NSString stringWithFormat]，对UIAlertView我们用过[alertView show]。在Objective-C语言编写的代码里面，只要你看到[]方括号，就应该知道这是一个方法调用。
当我们用self 关键词的时候，实际上我们是从对象的一个方法给该对象的另一个方法来发送消息。
比如在这里，视图控制器给自己发送一条startNewRound的消息以开启新的一轮游戏。
秘密：在iOS开发里面，发送消息和调用方法其实是一个意思~
然后iPhone就会切换到新的方法，并从头到尾执行里面的语句。当方法里面的语句执行完毕后，就会返回之前的方法，继续其中的代码。

这里要说明一点，把开启新一轮游戏的逻辑机制放到一个独立的方法里面是很好的编码习惯。如果我们不这样做，你就会看到下面的代码：

```
- (void)viewDidLoad
{
    [super viewDidLoad];

    targetValue = 1 + (arc4random() % 100);
    currentValue = 50;
    self.slider.value = currentValue;
}

- (IBAction)showAlert:(id)sender
{
    . . .
```

```
    [alertView show];

    targetValue = 1 + (arc4random() % 100);
    currentValue = 50;
    self.slider.value = currentValue;
}
```

这样做有什么不好呢？首先，我们在两个不同的地方重新写了相同的代码。好吧，这里只是3行代码，如果是300行怎么办？如果我们打算更改这部分逻辑又该怎么办？如果我们要在10个不同的地方用这部分代码怎么办？

还有，你刚开始写这段代码的时候非常清楚它是干吗用的。但1周以后呢？1个月以后呢？半年以后呢？你还记得这样的代码究竟是干嘛的吗？假如几个月后你发现要修改其中的一两行，但又忘了其它地方的类似代码，你就完蛋了。代码重复是产品中出现bug的主要来源之一。

只要你的代码需要在两个不同的地方完成相同的事情，你就需要考虑创建一个新的方法。相信我，别偷懒。偷懒一时爽，代码死光光。

还有，方法的名称也有助于我们了解它的确切功能。如果你没看过前面的教程，我直接给你下面的三行代码：

```
targetValue = 1 + (arc4random() % 100);
currentValue = 50;
self.slider.value = currentValue;
```

你能知道这几行代码究竟是干吗用的吗？当然，你可能了解这里用随机数创建了一个数值，然后设置了一个初始值，再把初始值赋予了滑动条。仅此而已，你能知道这其实是在开启新一轮的游戏吗？不管你能不能，我反正是不知道。有些程序猿会加上一些注释代码，但最好的方法还是学学哥：

```
[self startNewRound];
```

清清楚楚明明白白真真切切，比注释行还要清晰明了。虽然注释行也很有用，但让代码具有自解释性更重要。

而且在后面你可以随时查看和修改startNewRound方法，更不用说在Xcode里面搜索它也变得简单了。

点击Run运行游戏，确保每次触碰都会产生新的1到100之间的随机数。

你或许注意到了，每个新的回合开启时，滑动条都会重置到中间的位置。这是因为在startNewRound方法中我们重置currentValue的数值到50，然后让滑动条的结点位置更改为这一点。这个和我们最初的做法不同，之前我们是读取滑动条的位置，然后把它的数值赋予currentValue。哥这样做的目的是让每一轮新的游戏都从相同的位置开始，感觉要靠谱点。

继续前进！

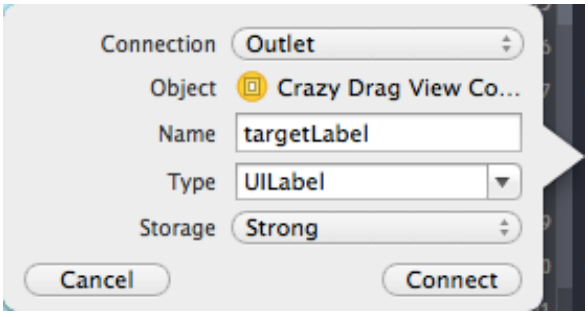
把我们的目标数值放到标签里

好吧，我们已经知道如何生成随机数，然后把它保存在一个名为targetValue的变量里。现在我们需要把这个数值放到屏幕上，不然玩家就是瞎猫去抓活老鼠，要多难有多难。

之前我们设计storyboard界面的时候，已经添加了一个用来放置目标值的标签。现在要做的事情就是读取targetValue变量的数值，然后把它放到标签里。为了做到这一点，我们需要完成两件事情：
1.创建一个到标签对象的引用，这样我们才好向它发送消息
2.让标签显示新的文本内容

其实之前对滑动条已经做过类似的事情了。还记得我们用@property来获取到滑动条对象的引用吗？我们使用的是self.slider.value。对于标签对象，我们会用到类似的方法。

如果你之前在Xcode中设置了不同的Tab，可以直接切换到界面编辑所在的Tab。 如果没有，那么需要在Xcode中点击Main.storyboard，同时打开Assistant Editor。
接下来选中目标数值所在的标签。按住control键不放，按住鼠标左键拖出一条线到辅助编辑器区域，记住线的终点在@interface和@end之间，在之前添加的@property变量代码之后（或之前）。



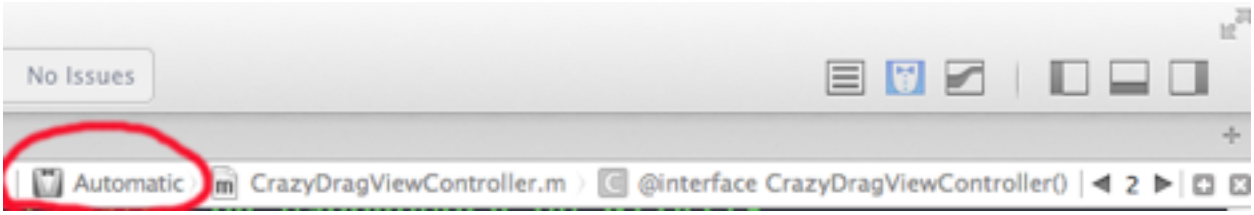
在弹出的提示框中，Connection部分选择默认的Outlet，Name那里填上targetLabel，其它不管，然后点击Connect，就会在辅助编辑区的CrazyDragViewController.m代码中自动添加一行代码：

@property (strong, nonatomic) IBOutlet UILabel *targetLabel;

这部分的操作不是第一次了。其实我们有几种不同的方法来完成这个关联操作：按住Ctrl键从对象拖出一条线，Ctrl键然后鼠标点击对象弹出一个菜单，右键单击对象弹出一个菜单，或者从Connections Inspector弹出菜单。随便你喜欢用哪种方法，你喜欢就好~

小提示：

需要记住一点的是，这种通过拖曳来自动创建关联和代码的方法很方便，但有时候会发现创建关联后切换到真正的代码编辑区，却发现没有出现相关代码。此时你要检查Xcode右上，如下图中，应该显示Automatic，如果是manual或者其它，那么一定要手动切换到Automatic。




最后一步，在Xcode中切换到CrazyDragViewController.m，在上一条@synthesize代码的下面添加一行代码：

@synthesize targetLabel;

现在到了实质性的东西了。在startNewRound方法前添加下面的新方法：

```
-(void)updateLabels{

    self.targetLabel.text = [NSString stringWithFormat:@"%d",targetValue];
}
```

我们把这个逻辑放到一个单独的方法里，这是因为在其它地方我们还会用到。方法的名称已经很清晰的说明了它的作用：使用它来更新标签的文本内容。当然，现在我们只是用它来更新单个标签的文本内容，后面还会添加代码来更新其它标签的文本内容（总得分，游戏回合数）。
updateLabels这个方法里面的代码现在应该可以看懂了，我们把一个字符串的内容赋予目标标签的文本属性。

不过你可能要问，为什么我们不能更简单一点，用下面的代码呢：

self.targetLabel.text = targetValue;

原因很简单，我们不能把整数类型的数值直接赋予字符串类型的变量。方井盖没法放进圆孔。
self.targetLabel获取了视图控制器的targetLabel属性，它是一个UILabel标签对象。UILabel标签对象有自己的text属性，它是一个NSString字符串对象。我们只能把字符串类型的数据放到NSString类型的变量里面。但是刚才的代码试图把targetValue,一个int类型的整数直接放到字符串类型的变量里。所以我们需要先把整数类型的变量转换为一个字符串，这就是stringWithFormat这个方法的作用。其实之前我们也做过类似的事情，现在你应该知道为什么要这么样做了吧。
updateLabels是个常规的方法，它没有作为一个动作方法关联到任何界面控件上，因此除非我们手动调用它，它就难有出头之日。一个比较合适的地方是在每次调用startNewRound方法后就调用它。因为在那里我们将要获取新的目标值。

现在我们通过viewDidLoad和showAlert方法来发送startNewRound消息，所以这里需要修改下这些方法的内容：

```
- (void)viewDidLoad
{
   [super viewDidLoad];
      // Do any additional setup after loading the view, typically from a nib.

   [self startNewRound];
   [self updateLabels];
}
```

```
- (IBAction)showAlert:(id)sender {

    NSString *message = [NSString stringWithFormat:@"滑动条的当前数值是:%d，我们的目标数值是：%d",currentValue,targetValue];

    [[[UIAlertView alloc]initWithTitle:@"您好，苍老师" message:message delegate:nil cancelButtonTitle:@"本老师知道了" otherButtonTitles:nil, nil]show];
    [self startNewRound];
    [self updateLabels];
}
```

点击Xcode上的Run按钮来运行游戏，现在我们可以在屏幕上看到随机生成的目标值了。在结束之前，我们再来点理论知识恶补一下吧。

科普：动作方法vs一般方法

好吧，我们现在已经用到了两种方法，分别是和xib界面中控件关联起来的动作方法，以及有着独立意识的一般方法。那么它们之间有什么区别呢？其实动作方法和一般方法没什么区别。唯一不同的就是(IBAction)这个标示。当我们在方法前面加了这个东西后，Interface Builder就会知道我们可以把它和界面中的按钮，滑动条或者其它控件关联起来。
至于那些没有IBAction标识的方法就不能和界面中的控件建立关联了。

```
// 基本动作方法
- (IBAction)doSomething;

// 动作方法，不过添加了到触发这个动作的对象的引用
- (IBAction)buttonTapped:(UIButton *)button;

// 不能和Interface Builder中的控件建立关联
- (void)someOtherMethod;
```

好了，今天的学习到此结束，上福利了。


那等在冬节里的，容

美女不够吸引力？那就来大奖吧。最近新鲜出炉的双色球大奖，加菲猫和财神爷一起来了，祝你好运！