

原文及示例代码来自raywenderlich store中的iOS Apprentice 系列3教程，经过翻译和改编。

版权归原作者所有，本系列教程仅供学习参考使用，感兴趣的朋友建议购买原英文教程教程(The iOS Apprentice Second Edition: Learn iPhone and iPad Programming via Tutorials!)

购买链接：

<http://www.raywenderlich.com/store>

欢迎回来，在火炉边找个位置坐下来吧。

在上一课的内容中，我们接触了所谓的reverse geocoding这种神秘而又高大上的东西。

在实际测试的时候，可能在Simulator上会出一些状况，在iPod touch和非3G版的iPad上面也可能会有问题。因为iPod touch和非3G版的iPad没有GPS，只能靠Wi-Fi来定位。不过Wi-Fi的精度不够，个人试过在+/-100米左右。

到现在为止，我们只能在位置信息精度进入desiredAccuracy的范围内才能停止位置信息的更新，而遗憾的是在iPod 上很难实现这一点。因此，在开发基于地理位置的应用时，我们必须在各种设备上进行测试。

为了应对上面所说的情况，我们需要改善下didUpdateLocations方法。

在Xcode中切换到CLLocationViewControllor.m，更改didUpdateLocation方法如下：

```
-(void)locationManager:(CLLocationManager *)manager didUpdateLocations:(NSArray *)locations{

    CLLocation *newLocation = [locations lastObject];
    NSLog(@"已更新坐标，当前位置：%@",newLocation);

    if([newLocation.timestamp timeIntervalSinceNow] < -5.0){
        return;
    }

    if(newLocation.horizontalAccuracy < 0){
        return;
    }

    CLLocationDistance distance = MAXFLOAT;

    if(_location != nil){
        distance = [newLocation distanceFromLocation:_location];
    }

    if(_location == nil || _location.horizontalAccuracy > newLocation.horizontalAccuracy){

        _lastLocationError = nil;
        _location = newLocation;
        [self updateLabels];

        if(newLocation.horizontalAccuracy <= _locationManager.desiredAccuracy){
            NSLog(@"***目标诺森德！ 成功完成定位");
            [self stopLocationManager];
            [self configureGetButton];
        }
        if(distance > 0){
```

```

        _performingReverseGeocoding = NO;
    }

    if(!_performingReverseGeocoding){
        NSLog(@"*** Going to geocode");
        _performingReverseGeocoding = YES;

        [_geocoder reverseGeocodeLocation:_location completionHandler:^(NSArray
*placemarks, NSError *error){

            NSLog(@"*** Found placemarks: %@, error: %@",placemarks,error);

            _lastGeocodingError = error;
            if(error == nil && [placemarks count] >0){
                _placemark = [placemarks lastObject];
            }else{
                _placemark = nil;
            }

            _performingReverseGeocoding = NO;
            [self updateLabels];
        }];
    }

}

}else if (distance <1.0){
    NSTimeInterval timeInterval = [newLocation.timestamp
timeIntervalSinceDate:_location.timestamp];
    if(timeInterval >10){
        NSLog(@"*** 强制完成! ");
        [self stopLocationManager];
        [self updateLabels];
        [self configureGetButton];
    }
}
}
}

```

现在这个方法的代码行有点多的吓人啊~

不过还是那句话，Don't Panic, Hold on

虽然代码很多，不过只有黄色高亮的部分才是新添加的。首先看第一处：

```

CLLocationDistance distance = MAXFLOAT;

if (_location != nil) {
    distance = [newLocation distanceFromLocation:_location];
}

```

以上代码的作用是测量最新读数和前一个读数间的距离差（假如有的话）。如果不存在前一个读数，那么默认的距离就是MAXFLOAT。对于这种看起来莫名其妙的东西肿么办？很简单，按option键，然后点击它，就可以看到它是在math.h这个头文件中定义的，再点击，可以看到：

```

#define MAXFLOAT 0x1.ffffep+127f
#endif /* __DARWIN_C_LEVEL >= 199506L */

```

可以看到，这个MAXFLOAT实际上是一个宏定义。额，不清楚宏定义是啥意思？待会儿再说。通过这个宏定义，MAXFLOAT这个常量就代表了最大的浮点数,而这个浮点数是用16进制来表示的。额，十六进制又是啥意思？莫急莫急，先把这段代码解完了再说。

通过这个小小的技巧，如果当前读数是第一个读数，那么距离就是一个巨大无比的数值。这样即便我们无法获取一个真正的距离也不会影响下面的计算。

接着看下一个黄色高亮部分，这里用一个if判断语句来判断是否已经达到了所需要的精度。当我们停止了location manager时才可能发生：

```
if (distance > 0) {  
    _performingReverseGeocoding = NO;  
}
```

通过上面的代码即便应用已经在执行另一个地理编码请求了，我们还是会强制进行一个反向地理编码。当然，如果distance是0的话，就表示当前位置信息和前一个读数中的位置信息是相同的，也就没必要进行反向地理编码了。

之所以这样做，是因为我们希望获得最终位置的地址信息，因为那才是最精确的位置。但如果之前有一些位置正在进行反向地理编码，这一步骤就会被忽略。通过将_performingReverseGeocoding设置为NO，会强制应用对最后一个位置坐标进行反向地理编码。

真正的改善来自下一个黄色高亮部分代码：

```
} else if (distance < 1.0) {  
    NSTimeInterval timeInterval = [newLocation.timestamp  
timeIntervalSinceDate:_location.timestamp];  
    if (timeInterval > 10) {  
        NSLog(@"*** Force done!"); [self stopLocationManager]; [self  
updateLabels];  
        [self configureGetButton];  
    } }
```

如果当前读数的坐标和前一个读数没有明显差异，而且从收到前一个读数到现在已经超过10秒了，那么是时候停手了。此时我们可以假定已经不可能获得更精确的位置信息，因此停止继续获取位置信息的努力。

通过这一段代码，应用在我的iPod touch上才会停止定位。虽然我没法得到一个超过+/-100米的精度，但它会重复同一个读数。之所以挑选10秒钟的时限，是因为10秒钟已经足够获得好的结果了。

注意我们没有简单的用下面这样的代码：

```
} else if (distance = 0) {
```

因为不同读数之间的差距永远不可能等于0.很可能它会是0.0017632,或者0.1234567。因此只需要检查距离是否小于某个特定值就好了，这里设置为1米。

编译运行应用，看看是否可以正常工作。你可以在iPod touch或室内的iPhone上测试，然后看看NSLog会输出什么信息。

术语党科普- 宏定义

宏定义是C系语言比较常用的利器。所谓的宏定义又称为宏替换，或者可以理解成日常生活中用的外号、别称、昵称。

举个简单的例子：

```
#define WQNMLGB “我去年买了个表”
```

```
#define TH “SB王晓明”
```

那么下次对王小明的某些行为表示愤慨的时候，就不必打一大堆汉字了，可以直接用TH，WQNMLGB代替，可以骂人于无形之中，决胜于千里之外~

详细的不多说了，这里笔者偷个懒吧，直接给你链接参考下：

<http://baike.baidu.com/view/2076445.htm>

[http://en.wikipedia.org/wiki/Macro_\(computer_science\)](http://en.wikipedia.org/wiki/Macro_(computer_science))

术语党充电- 十六进制

我们都知道，普通人类有两只手，十个手指，因此日常生活中最常用的是十进制。

一加一等于一，一加二等于三...逢十进一。只要学过小学数学的都懂。

上上上上章（记不清了）的内容中有提到过数字电路和二进制，简单来说，在计算机的世界里面，所有的生物只有0和1两种状态，在或者不在，土豪或者穷D，没有第三种状态。

神奇的是，早在18世纪，NB轰轰的德国哲学大师和数学大师（不是大湿，是真正的大师!!!）莱布尼兹从他的传教士朋友鲍威特那里搞到了一本拉丁文版的《易经》，从中读到了我天朝文化的精髓-八卦，并惊奇的发现易经中的阴爻和阳爻的进位制就是二进制。

虽然二进制可以有效表示计算机中的数据，但问题在于二进制数太长了。

为了让人类和计算机生物进行沟通，代码诸神决定引入八进制和十六进制，因为2，8，16分别是2的1次方，3次方和4次方，这一点使得三种进制可以非常直接的互相转换。

关于各种进制间如何转换这里就不多说了，大家可以去GOOGLE或度娘找，如今很少还需要自己手动来转换的。虽然我朝人心算能力普遍爆表，但有这时间还不如想想如何解决一个新问题，或者发明一个新算法。

目前各种编程语言和框架里面现在都提供了相关的函数或宏定义，对于如今的程序猿来说，最重要的是看到0x开头的数值就表示是十六进制。

详细参考：

<http://www.baike.com/wiki/二进制>

<http://baike.baidu.com/view/230306.htm>

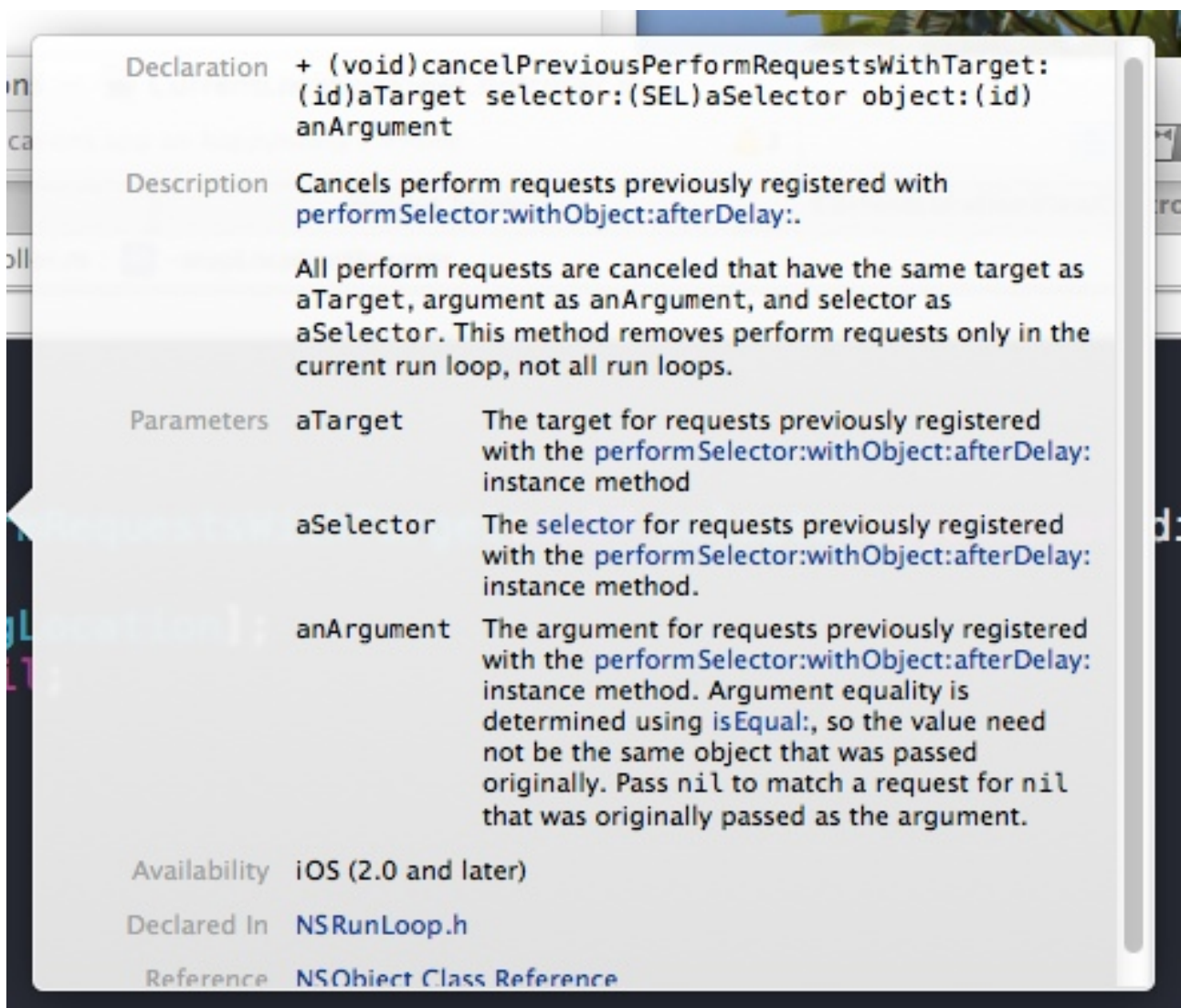
好了，继续我们的学习。

当然，还有一处地方需要优化的。为了让整个逻辑更加严密，我们需要设置整个定位过程的时限。只需要让iOS从当前时间起1分钟后执行一个方法，如果届时应用仍然没有找到位置信息，那么就需要停止location manager，并显示一条错误信息。

仍然在CLLocationViewController.m中更改startLocationManager的方法如下：

```
-(void)startLocationManager{  
    if([CLLocationManager locationServicesEnabled]){  
        _locationManager.delegate = self;  
        _locationManager.desiredAccuracy = kCLLocationAccuracyNearestTenMeters;  
        [_locationManager startUpdatingLocation];  
        _updatingLocation = YES;  
        [self performSelector:@selector(didTimeout) withObject:nil afterDelay:60];  
    }  
}
```

上面黄色高亮的这行代码会让ios系统在1分钟（60秒）后发送didTimeout:消息给self。didTimeout:是个自定义的方法，后面需要我们手动来实现。这里用到了selector的概念，网上把它翻译成选择器，个人觉得还是直接看e文更顺眼。selector（缩写SEL）在Objective-C语言中就代表某个方法的名称。而使用@selector()就表示自己打算引用圆括号后的方法。



The screenshot shows the Xcode documentation for the `cancelPreviousPerformRequestsWithTarget:` method. The documentation is presented in a table-like format with sections for Declaration, Description, Parameters, Availability, Declared In, and Reference.

Declaration	<code>+ (void)cancelPreviousPerformRequestsWithTarget:(id)aTarget selector:(SEL)aSelector object:(id)anArgument</code>	
Description	Cancels perform requests previously registered with <code>performSelector:withObject:afterDelay:</code> . All perform requests are canceled that have the same target as <code>aTarget</code> , argument as <code>anArgument</code> , and selector as <code>aSelector</code> . This method removes perform requests only in the current run loop, not all run loops.	
Parameters	<code>aTarget</code>	The target for requests previously registered with the <code>performSelector:withObject:afterDelay:</code> instance method
	<code>aSelector</code>	The selector for requests previously registered with the <code>performSelector:withObject:afterDelay:</code> instance method.
	<code>anArgument</code>	The argument for requests previously registered with the <code>performSelector:withObject:afterDelay:</code> instance method. Argument equality is determined using <code>isEqual:</code> , so the value need not be the same object that was passed originally. Pass <code>nil</code> to match a request for <code>nil</code> that was originally passed as the argument.
Availability	iOS (2.0 and later)	
Declared In	NSRunLoop.h	
Reference	NSObject Class Reference	

关于selector的深入内容这里就不详细说了，感兴趣的童鞋最好看看苹果官方文档Objective-C Runtime Programming Guide (<https://developer.apple.com/library/ios/documentation/cocoa/conceptual/ObjCRuntimeGuide/Articles/ocrtHowMessagingWorks.html>)

还有一篇中文的文章可以参考下 (<http://www.cnblogs.com/yaski/archive/2009/04/05/1429735.html>)

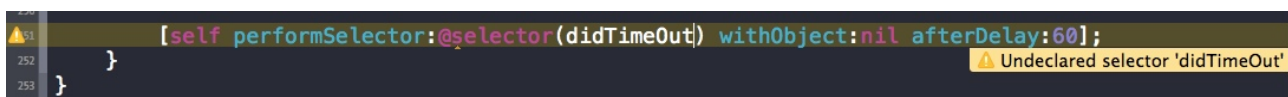
不过建议大家在入门学习的时候不要贪多，一点点慢慢来。不要一上来就限于局部的泥潭，要先建立编程的思维方式，以及某个语言或框架的整体认识，然后再进入细节。否则，你就等着陷入代码的汪洋大海吧。

接下来更改stopLocationManager方法如下：

```
-(void)stopLocationManager{  
    if(_updatingLocation){  
        [NSObject cancelPreviousPerformRequestsWithTarget:self selector:@selector(didTimeout:) object:nil];  
        [_locationManager stopUpdatingLocation];  
        _locationManager.delegate = nil;  
        _updatingLocation = NO;  
    }  
}
```

黄色高亮部分是新增的代码，之前我们在startLocationManager方法中设置了1分钟后给自己发送didTimeout:消息，而在stopLocationManager中我们需要对应的取消，以免location manager在时限到达前就被停止。

当然，这里用到了一个新方法，看不懂怎么办？按住option键然后点击方法名称，可以看到下面的提示：



对于上面这个方法的作用我就不解释了，直接看官方文档可以一清二楚。

再次强调下，我们这个系列教程的主要目的还不是教你iOS开发，而是教你编程和创造性思维，以及培养你自学的能力，以后接触新的语言、开发工具、框架和技能也可以轻松上手。因此，在这个过程中不要被动的接受，而是主动探索未知。Don't panic, just try it!

好了，接下来在stopLocationManager方法下面添加didTimeout:方法：

```
-(void)didTimeout:(id)obj{  
    NSLog(@"*** oops,超时了");  
    if(_location == nil){  
        [self stopLocationManager];  
        _lastLocationError = [NSError errorWithDomain:@"MyLocationsErrorDomain" code:1 userInfo:nil];  
        [self updateLabels];  
    }  
}
```

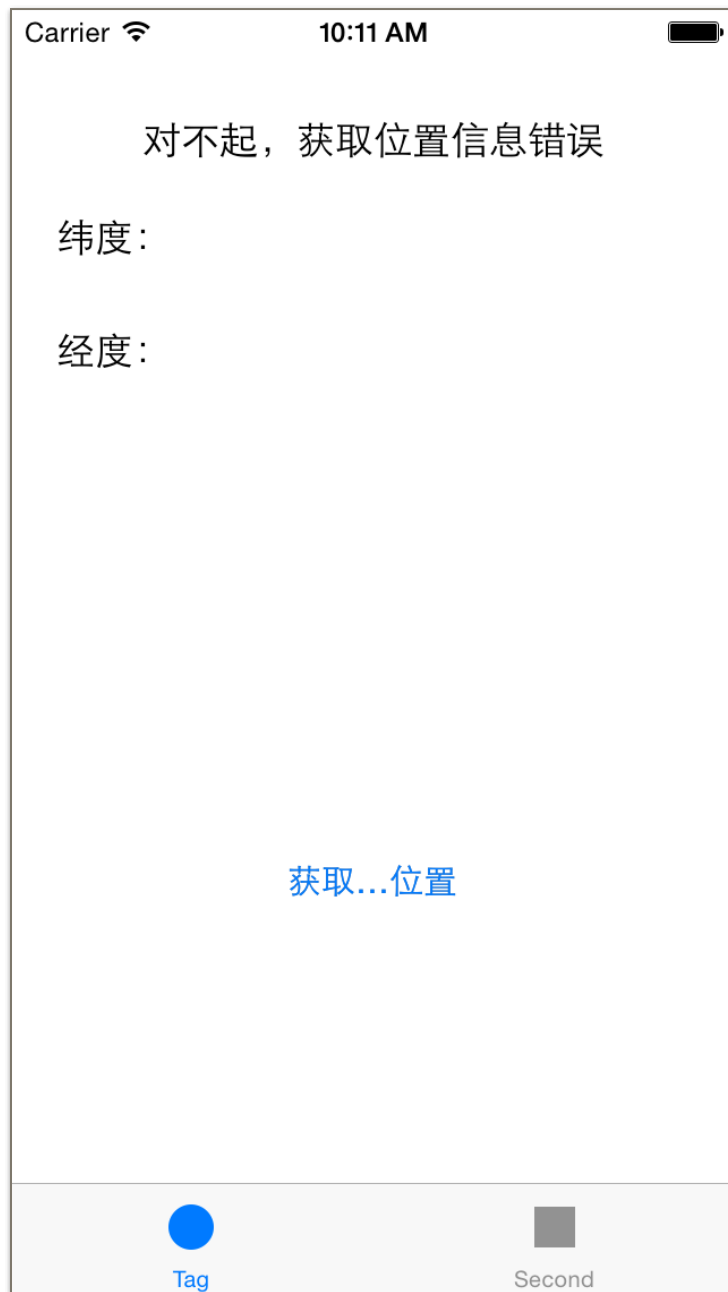


```
2014-02-09 10:09:09.796 MyLocations[1733:70b] 定位失败: Error Domain=kCLErrorDomain Code=0 "The operation couldn't be completed. (kCLErrorDomain error 0.)"
2014-02-09 10:10:09.767 MyLocations[1733:70b] *** oops,超时了
```

All Output ↕



```
    [self configureGetButton];
}
}
```



根据之前的设置，系统时间1分钟后就会调用`didTimeout:`方法，不管你是否获得了一个有效的位置信息（除非`stopLocationManager`先取消了计时器）。如果1分钟后仍然没有有效位置信息，就需要停止location manager，创建错误代码，然后更新界面。

这里我们创建了自己的`NSError`对象，然后把它保存到`_lastLocationError`这个实例变量中。不过我们还需要确保错误的domain不是`kCLErrorDomain`，因为这个错误对象不是由Core Location框架提供的，而是应用自身提供的。一个domain其实就是个字符串，因此

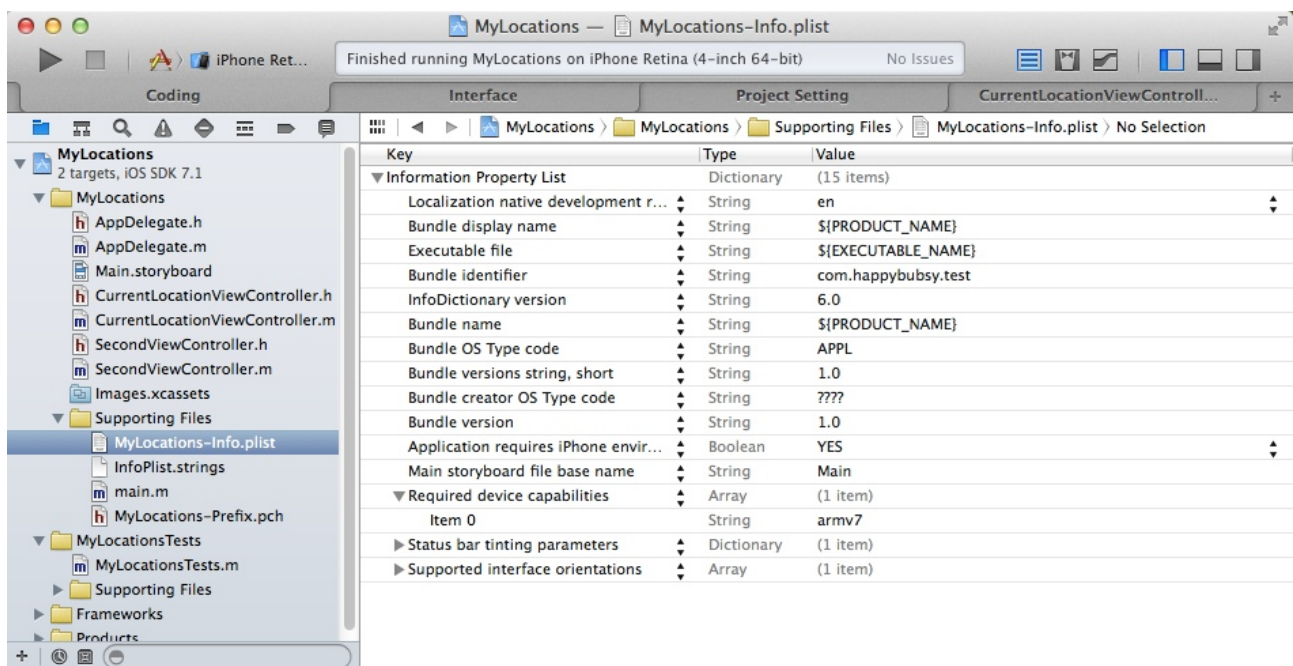
@“MyLocationsErrorDomain”是个不错的名称。而错误代码则选择了1.这里错误代码的数值不重要，因为我们只有一个错误，不过考虑到将来应用的规模可能变大，因此可能会存在多个错误代码当然，我们不是非得用NSError对象不可，还有其它方法让代码的其它部分了解到有错误发生。不过考虑到这里的updateLabels本身就在使用NSError，因此提供一个定制的错误对象是可以理解的。如果对NSError这个类很感兴趣，可以在Xcode的Documents中查看相关定义。

好了，编译运行应用。

且慢，为何有一个黄色警告啊？

Undeclared selector ‘didTimeout’

不可能啊？didTimeout这个方法已经实现了啊？？？



好吧，其实这是iOS初学者最常犯的错误之一。在@selector()圆括号中的方法名称必须是全称，这里的方法名称应该是didTimeout:

这才是这个方法的真正名称，didTimeout:和didTimeout是两个完全不同的方法！！！！

把这行代码改成：

```
[self performSelector:@selector(didTimeout:) withObject:nil afterDelay:60];
```

警告消失了。

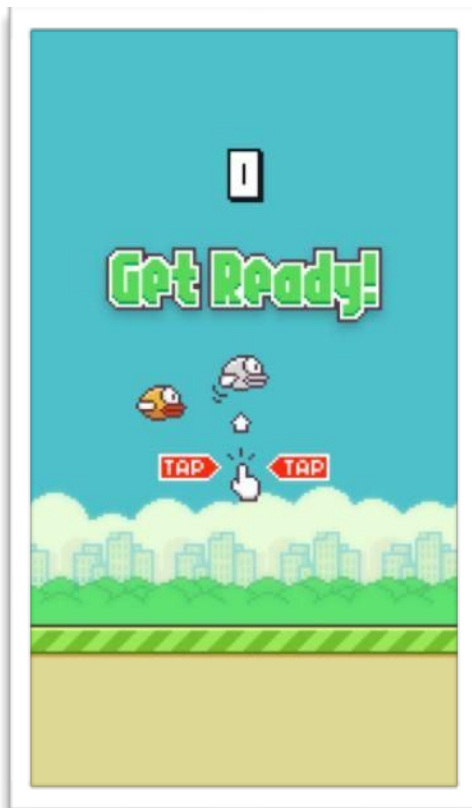
虽然Objective-C的这种方方法名称让人有些不爽，不过它也避免了实际运行中可能出现的方法混淆。

再次编译运行应用，在Simulator中将location设置为None，然后触碰界面上获取位置按钮。1分钟后，debug区会显示oops,超时了

而界面上的停止按钮会恢复到获取位置按钮。

在结束这一部分内容前，我们再来点小知识。

关于Info.plist文件中的Required device Capabilities



地球人都知道Info.plist文件里面保存了很多对应用至关重要的设置。神马，你没听说过这东西？那好吧，建议你回过头看系列1的教程先。

在Info.plist中有一个名为Require device capabilities的字段，其中列出了当前应用正常运行所需的硬件。App Store也会使用它来判断某个用户能否在其设备商安装你的应用。

当前的默认值是armv7，也就是iPhone 3GS和之后版本设备所用的CPU架构。

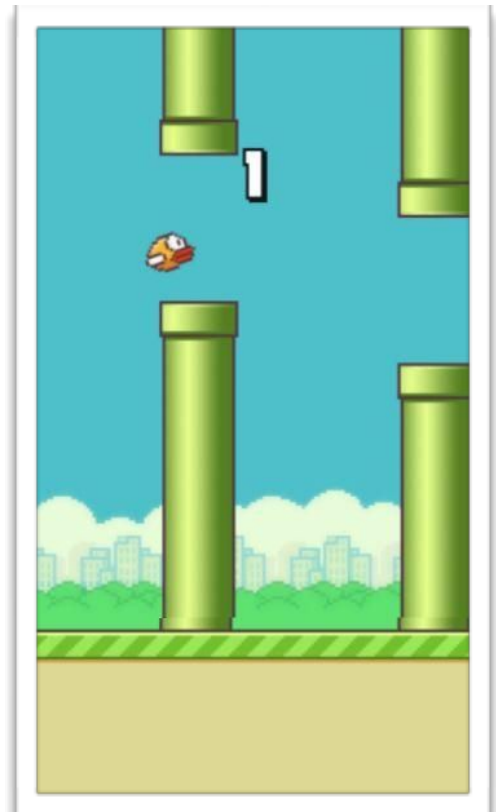
关于armv和CPU架构就不敢展开来说了，对于开发者来说，需要知道的是：

第一代水果移动设备用的是ARM11处理器，也就是armv6，而自iPhone 3GS之后的水果移动设备用的是Cortex A8处理器，也就是armv7，苹果的A6芯片也是armv7，iPhone5对应的是armv7s，当然仍然属于armv7的架构，armv8对应苹果的A7芯片(iPhone5S)，其中armv8是64位的，而armv7和之前的都是32位的CPU。

假如你对此很好奇，可以参考这里：

http://en.wikipedia.org/wiki/ARM_architecture

<http://wanderingcoder.net/2010/07/19/ought-arm/>



如果你在编译应用的时候碰到了和armv相关的问题，不妨回过头看看链接里面的文章，会有帮助的
~



回到Info.plist，如果我们的应用需要添加其它特性，比如Core Location，那么最好在这里列出来。

在Xcode中，在MyLocations-Info.plist的Required device capabilities部分添加一个新的键值location-services。

当然，如果我们的应用需要用到GPS接收器的话，还可以添加一个新的键值gps。
不过如果添加了这个键值，用户就没法在iPod touch和非3G版的iPad上用了。

在水果官方的iOS App Programming Guide中列出了所有的Required device capabilities的键值
https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/App-RelatedResources/App-RelatedResources.html#//apple_ref/doc/uid/TP40007072-CH6-SW10

小提示：

在将应用提交到App Store前，别忘了注释那些NSLog()语句。

好了，今天的课程到此结束。

据说任天堂终于要在手机上推出自家游戏的模拟器了，虽然不是最新作品，但聊胜于无啊，不然一年前的盈利承诺咋办？如何对得起九泉之下的山内溥大人？

最近有个Flappy Bird的奇葩游戏超级火爆，我下了，玩了，删了，貌似我随便做的游戏也不亚于这个水准啊。哎，为毛人家就能成为最火免费应用呢？这就是高富帅和穷D的差距吧，缺RP啊！！

算了，为了求RP，还是多发送福利吧