从零开始学iOS7开发系列3-我的地盘我做主-Cha8

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。

没词了，欢迎继续我们的学习。

这一课我们将学习将位置坐标信息放到界面中。

刚才的didUpdateLocation代理方法向应用返回了一个CLLocation对象数组，每个CLLocation对象都包含了用户的当前经度和纬度信息。这些对象其实还有其它属性信息，比如海拔高度和速度等，不过在当前应用中我们暂时还用不到。

我们将使用数组中的最后一个CLLocation对象，因为它是最新的信息，并在界面的标签中显示坐标信息。

打开Xcode，在CurrentLocationViewController.m中添加一个新的实例变量_location

```
@implementation CurrentLocationViewController{
    CLLocationManager *_locationManager;
    CLLocation *_location;
}
```

我们将把用户的当前位置信息保存在这个变量里面。

更改didUpdateLocations方法的内容如下：

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

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

    _location = newLocation;
    [self updateLabels];
}
```

这里依然要保留NSLog()这行代码，因为它对后续的debugging调试会很有帮助。
在新添加的代码中，我们将所获取的最新位置信息保存在刚刚创建的实例变量中，然后调用给一个updateLabels方法。

接下来添加updateLabels方法的实现代码：

```
-(void)updateLabels{

    if(_location !=nil){
        self.latitudeLabel.text = [NSString stringWithFormat:@"%.8f",_location.coordinate.latitude];
        self.longtitudeLabel.text = [NSString stringWithFormat:@"%.
8f",_location.coordinate.longitude];
        self.tagButton.hidden = NO;
```

```
        self.messageLabel.text = @"";
    }else{

        self.latitudeLabel.text = @"";
        self.longtitudeLabel.text = @"";
        self.adderssLabel.text = @"";
        self.tagButton.hidden = YES;
        self.messageLabel.text = @"Press the Button to Start";

    }
}
```

下面解释下刚才的方法内容：
如果_location这个实例变量的内容不是nil，也就是说存在一个最新的位置信息对象，那么就把它里

面的double类型的latitude和longtitude数值转换成strings字符串类型，然后把字符串的内容放到标签中。stringWithFormat方法的%.8f格式符和之前的%f格式符作用类似，区别在于它会保留8位小数

这也是.8的作用所在。

编译运行应用，然后在Simulator的Debug菜单中选择一个地址，点击Get My Location按钮，就可以在界面上看到经度和纬度信息了。

当应用刚启动的时候，并没有位置信息对象，因此需要显示一条Press the Button to Start信息，不过这里并没有这样。
这是因为当前只有在应用接收到坐标信息后才会调用updateLabels方法。因此我们需要在viewDidLoad中也调用updateLabels方法。

在CurrentLocationViewController.m中更改viewDidLoad方法如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    [self updateLabels];
}
```

再次编译运行应用。初始状态下界面上会提示Press the Button to Start这条信息，而经度和纬度的标签内容是空的。

处理意外错误

在使用设备获取GPS坐标信息的时候很容易出错。比如有时候你在房间里面，或是在四面高楼林立的包围之下不见天日，此时GPS信号很容易被阻断。Wi-Fi就更不用说了，在我朝除了在家里和办公

室里面，基本上很难搜到免费的WiFi信号，所以指望Wi-Fi定位来提供位置信息是很不靠谱的。苍然如果你运气不好的话，很可能手机也没信号。这时候靠所谓的三角测量定位法基本上就是水中捞月了。
当然，以上所有的一切都假定你的设备商有GPS或是蜂窝信号。如果你带着自己的iPod touch出门拍照和获取位置信息，那么即便在市中心你也会非常失望。iPhone要好得多，不过仍然不是很理想。

为什么要说这些？因为这意味着我们的LBS应用必须知道如何来处理各种意外和错误。很有可能我们无法获取位置信息纠偏，即便可以，那也需要花上几秒钟的时间。
说白了，纸上谈兵很容易，但是要处理真正世界的复杂却很难。因此我们需要在应用中为应付各种意外情况的发生添加一些代码。

关于这一点，可以想想近来被千万人所唾骂的12306网站。作为一个开发者，更多的用户数量就意味着更大的责任！你可以不在乎自己的薪水，不在乎别人的看法，但别忘了只要你开发的产品有自己的用户，就要为此负责。即便没有法律上的责任，起码也有道德和良知的责任。
回到CurrentLocationViewController.m，更改didFailWithError方法的内容如下：

```
-(void)locationManager:(CLLocationManager *)manager didFailWithError:(NSError *)error{

    NSLog(@"定位失败：%@",error);

    if(error.code ==kCLErrorLocationUnknown){
        return;
    }

    [self stopLocationManager];
    _lastLocationError = error;

    [self updateLabels];
}
```

location manager可能会提供不同类型的错误，我们需要根据NSError对象的code属性来判断错误的具体类型。

这里大概列举几种Core Location 错误信息：
1.kCLErrorLocationUnknown

这种错误是指位置信息当前是未知的，不过Core Location一直在尝试获取。

2.kCLErrorDenied

用户拒绝应用使用位置服务。

3.kCLErrorNetwork

出现网络相关的错误

当然错误信息远远不止这3种，有些和compass罗盘和geocoding地理位置编码有关，。

第一次看到这种kCLErrorXXX的命名方式时，可能你会觉得有点可怕。这都是些什么东西啊？实际上这些错误编码只是简单的int类型整数。不过如果直接用0，1，2之类的，可能你更加不懂错误的 原

因了，所以Core Location勉为其难的帮你提供了这种看起来有意义的名称。这就使得代码更加清晰易读，开发者也不会在没有意义的数字上纠结。
其实k是iOS开发中常用的前缀，通常用k开头的名称代表一个constant常量。为什么要用k而不是c？我猜第一个发明这个前缀的人e文不怎么样-konstant？还是纯粹恶搞？不过CL就很直观了，就是Core Location的缩写。

预知更多的错误信息类型，还是直接查询苹果的官方帮助文档吧。至于查找的方法，已经教过你了。

在刚才的didFailWithError方法中，有这样一段语句：

```
if(error.code ==kCLErrorLocationUnknown){
    return;
}
```

这是神马意思呢？kCLErrorLocationUnknown这个错误编码意味着location manager目前无法获取一个位置信息，不过并不代表后面就完全没机会了。它可能还需要花1秒或者更多的时间来获取到GPS卫星的通讯链接。同时它还会告诉你目前无法获得任何位置信息。当我们看到该错误时，可以继续尝试，直到成功获取一个位置信息，或者是收到更严重的错误信息。

如果出现了更加严重的错误，就需要执行下面的几行语句：

```
[self stopLocationManager];
_lastLocationError = error;

[self updateLabels];
```

之前我们已经见过updateLabels这个方法。很快我们会扩展该方法，让它向用户显示错误信息。因为我们不想让用户被抛弃在黑暗的深渊中不见天日。到最后，我们会把错误对象保存到一个新的实例变量_lastLocationError中。这样一来，我们就可以在后面根据需要查找要处理的错误类型了。

stopLocationManager方法是个新的方法。为了省电和省流量（你可以在关闭WIFI的情况下开一晚导航试试，据说第二天早上你的房子就归运营商所有了！），应用应该在不需要使用iPhone的通讯信号的情况下立即将其关闭。如果当前用户无法获取位置信息，不如直接告诉lcoation manager停止工作。

如果你的用户因为这个破产了，我敢保证你会被千千万万人BS和唾骂的。当然，有些无良开发商会说，赚到钱就行了，我管它用户的死活。。。好吧，如果你学习编程就是为了加入这类人的团体，那还有什么好说的。这是你的自由选择，我也阻止不了，只有以后千万别说是看了这里的教程入门的。

只有在一种情况下才需要让location manager持续工作，神马情况？刚才说过了，如果你的应用需要实时导航，那么即便是出现网络连接错误的情况下也要让它保持工作。因为谁知道走两步以后信号会不会变好了？对我们这款应用来说，用户只需要在需要的时候再次触碰Get My Location就好了。

好吧，现在就来实现stopLocationManager这个方法：

```
-(void)stopLocationManager{

    if(_updatingLocation){
        [_locationManager stopUpdatingLocation];
        _locationManager.delegate = nil;
        _updatingLocation = NO;
    }

}
```

在上面的方法中有一个if判断语句，它的作用是检查布尔变量_updatingLocation是YES还是NO。如果是NO，那么就说明location manager当前并不处于活跃状态，因此就美必要将其关闭了。之所以要设置这个_updatingLocation变量，是因为当应用尝试获取位置纠偏的时候，我们需要更改Get My Location按钮和状态消息标签的内容，这样用户才知道应用正在努力工作。

恩，为了让应用可以正常运行，现在显然还有几个错误需要弥补。

首先在@implementation中添加量过实例变量声明：

```
    BOOL _updatingLocation;
    NSError *_lastLocationError;
```

接下来让我们在updateLabels方法中加入一些代码:

```
-(void)updateLabels{

    if(_location !=nil){
        self.latitudeLabel.text = [NSString stringWithFormat:@"%.8f",_location.coordinate.latitude];
        self.longtitudeLabel.text = [NSString stringWithFormat:@"%.
8f",_location.coordinate.longitude];
```

```
        self.tagButton.hidden = NO;
        self.messageLabel.text = @"";
    }else{

        self.latitudeLabel.text = @"";
        self.longtitudeLabel.text = @"";
        self.adderssLabel.text = @"";
        self.tagButton.hidden = YES;

        NSString *statusMessage;
        if(_lastLocationError != nil){

            if([_lastLocationError.domain isEqualToString:kCLErrorDomain] &&_lastLocationError.code
== kCLErrorDenied)
            {
                statusMessage = @"对不起，用户禁用了定位功能";
            }else{
                statusMessage =@"对不起，获取位置信息错误";
            }
        }else if(![CLLocationManager locationServicesEnabled]){
            statusMessage =@"对不起，用户禁用了定位功能";

        }else if(_updatingLocation){
            statusMessage = @"定位中...";
        }else{
            statusMessage = @"请触碰按钮开始定位";
        }
        self.messageLabel.text = statusMessage;

    }


}
```

以上这段新的代码主要是根据不同的情况在界面上显示不同的信息。这里用到了一系列的if判断语句。

首先，当location manager获取了一个错误信息时，会在标签中显示一个错误消息。
首先要检查的错误类型是kCLErrorDenied（错误的domain是kCLErrorDomain，也就是和Core Location相关的错误信息）。在这种情况下用户拒绝向应用提供定位功能。如果不是这种错误，那么就直接显示“对不起，获取位置信息错误”，也就是说没办法获取定位信息。

好吧，即便没有错误信息，但应用仍然有可能无法获取位置信息。这里的原因是用户很可能完全关闭了设备上的位置服务功能（不仅仅是针对当前应用）。因此，我们需要根据CLLocationManager的locationServicesEnabled方法来进行判断。

假定没有出现任何错误，一切工作正常，那么在获取第一个有效的位置对象前状态标签上会显示“Searching...”（再提醒下，这些标签上的文字内容可以随便改，只要你喜欢）。
如果你的设备可以很快获取一个位置信息那当然很好，但现实的情况是往往需要花费一点时间来获取第一个位置信息。因此我们有必要让用户了解到应用当前正在查找其位置信息。这就是我们使用_updatingLocation这个布尔变量的原因。

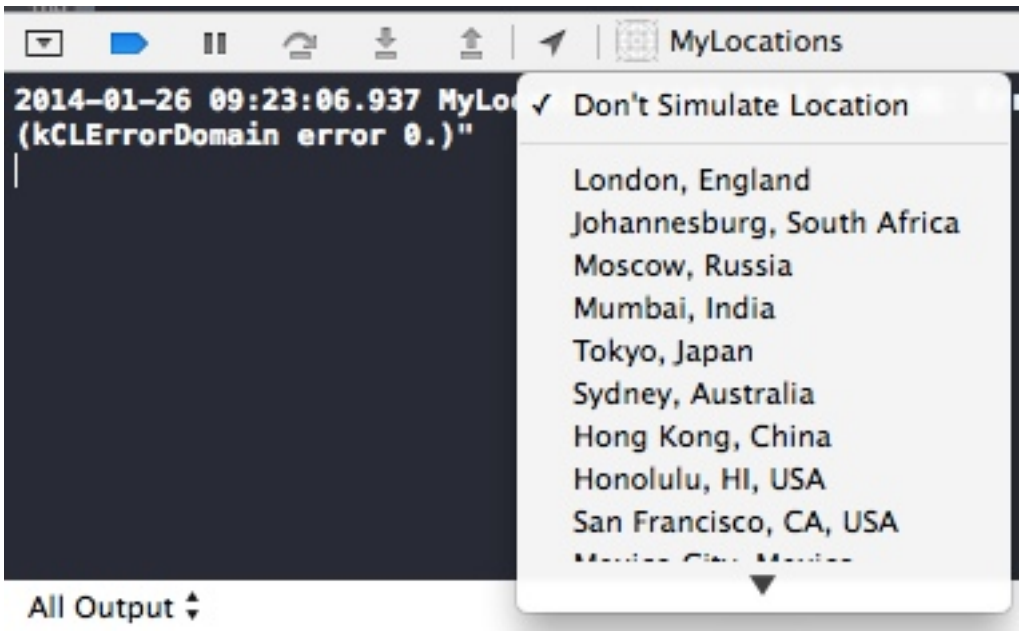
我们把所有相关逻辑放在一个单独的方法里面，这样就可以在情况发生变化时更改界面显示内容。获取了一个位置信息？只需要调用updateLabels方法来更新界面内容。收到一个错误信息？同样使用该方法向用户提示。

既然有stopLocationManager这样的方法，与之对应的也应该有startLocationManager方法。

在CurrentLocationViewController.m中添加一个新的startLocationManager方法如下：

```
-(void)startLocationManager{

    if([CLLocationManager locationServicesEnabled]){
        _locationManager.delegate = self;
        _locationManager.desiredAccuracy = kCLLocationAccuracyNearestTenMeters;
        [_locationManager startUpdatingLocation];
        _updatingLocation = YES;
    }
```



}

通常我们会考虑在getLocation这个动作方法中启动location manager。因为我们已经有了一个stopLocationManager方法，那么最好把启用location manager的相关代码放到一个单独的方法中，也就是startLocationManager。两者 区别在于startLocationManager方法中会检查用户设备商是否

已启用了位置服务。此外我们还将把_updatingLocation这个布尔变量设置为YES，表示正在定位。

接下来更改getLocation这个动作方法的代码：

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

    [self startLocationManager];
    [self updateLabels];
```

}

好了，还有一个地方的代码需要做一点小调整。假设有这样的一种情况，刚开始定位时出现错误，而且无法获得位置信息。不过当用户走两步以后，又可以获取有效的位置信息了。对于这种情况，，最好可以去掉之前的错误信息。

更改didUpdateLocations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);

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

这里的唯一变化就是将_lastLocationError设置为nil，来清除之前的错误状态。当我们成功获取到一个有效的位置信息时，无论之前出现了何种错误，都不会影响当前的应用。

编译运行应用。当应用在等待有效坐标时，顶部的表标签会显示“Searching...
，直到获取一个有效的位置坐标，或是出现致命的错误。

现在可以尝试在Simulator中更改location设置，注意当我们把Location设置为none时，仍然会返回一个kCLErrorLocationUnknown错误，不过在界面中不会有神马变化，因为它不是一个致命的错误。

除了更改Simulator里面debug的location信息，我们还可以直接在Xcode中来模拟。当应用使用Core Location框架时，会在debug区域的顶部显示一个箭头图标。点击那里可以更改所模拟的位置。

当然，仅仅在simulator或者Xcode里面更改设置都不够，最好的方法还是在设备上来直接测试。

好了，送上今日福利（2014年1月26日）