

说明:

本系列文章的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>

因为时间和精力有限（最近更换了base的城市），系列3全部完成预计要到3月底，系列4全部完成预计要到4月底。此外有很多童鞋发了邮件问了很多iOS开发的具体问题，我在考虑弄一个入门讨论论坛，然后请志愿者或者高手朋友来帮忙解答。

欢迎继续我们的学习。

在之前的学习中，我们已经把Tag Location界面的基本布局搞定了。
接下来我们将会把地理位置相关信息放到界面中。

首先在LocationDetailsViewController.h中添加两个属性变量的声明：

```
@property(nonatomic,assign) CLLocationCoordinate2D coordinate;  
@property(nonatomic,strong) CLPlacemark *placemark;
```

当然，你很快就看到Xcode给你两个红色提示，这个问题稍后解决。

之前我们曾看到过CLPlacemark对象。它包含了地址位置信息- 街道名称，城市名称等等。这些我们通过反向地理编码都曾经获取过。

CLLocationCoordinate2D对象对我们来说是个新东西。它包含了从location manager那里接收到的CLLocation对象的经度和纬度信息。我们只需要这两个部分的信息，因此没有必要发送整个CLLocation对象。

注意到这里我们在.h文件中添加了属性变量，而不是在.m文件中。这是因为这两个属性需要被其它类对象访问，因此它们是public（公开的），而不是private(私有的)。通过Current Location界面到Tag Location界面的segue，我们将给这两个属性赋值，然后Tag Location界面就可以把各自的数值放到对应的标签中。

理论知识充电：

之前提到过CLLocationCoordinate2D是个对象，其实这个说法并不严谨。因为如果它是一个对象的话，应该在名称前面有个星号。CLLocationCoordinate2D实际上不是一个对象，而是一个struct（结构体）。

Struct（结构体）是继承自C语言的遗产。通过使用结构体，可以将多个变量放到一个单一的数据结构中。在ObjectiveC中我们通常用类来实现这一目的，不过C语言中没有类，只有结构体。和类class不同的时，结构体没有方法，只有数据。因此它实际上是一组变量的列表。

比如CLLocationCoordinate2D的定义如下：

```
typedef struct{  
    CLLocationCoordinateDegrees latitude;  
    CLLocationCoordinateDegrees longitude;  
}CLLocationCoordinate2D;
```

该结构体有两个成员，latitude和longitude。但这两个字段都属于CLLocationDegrees数据类型，它是double的同义词：

```
typedef double CLLocationCoordinateDegrees;
```

double数据类型是语言中内置的基本数据类型。它和浮点数float类型比较类似，只是精度更高。好吧，不要被这种看起来很卡帕的同义词吓到，CLLocationCoordinate2D实际上可以这样来看：

```
typedef struct{  
    double latitude;  
    double longitude;  
} CLLocationCoordinate2D;
```

苹果Core Location的设计者之所以用CLLocationDegrees而不是double,是因为当你看到” CL Location Degrees”时，自然就知道该数据类型的作用：存储一个地理位置度。虽然它本质上是一个double类型，但是作为Core Location的使用者，我们只需要知道可以用CLLocationDegrees类型来保存经度和纬度信息。这种命名方式可以让一切一目了然。

因此CLLocationCoordinate2D实际上是有两个字段的结构体。和基本数据类型int或BOOL类似，我们可以直接使用结构体变量。同时我们无需使用alloc或init来初始化一个结构体（实际上你也不能这么做，因为它们不是对象）。

比如可以这样来：

```
CLLocationCoordinate2D myCoordinate;  
myCoordinate.latitude = 12.345;  
myCoordinate.longitude = 67.890;
```

注意我们可以使用点表示法来访问结构体的字段。虽然这和访问属性的形式有点类似，但本质上不是一回事。

在UIKit和其它iOS框架中，我们会经常碰到结构体，比如最常用的CGPoint和CGRect。结构体比对象更轻量级。如果我们只需要传递一组数值，那么就可以把它们放到一个结构体中，然后传递这个结构体，而这正是在Core Location框架中传递地理位置信息的方法。

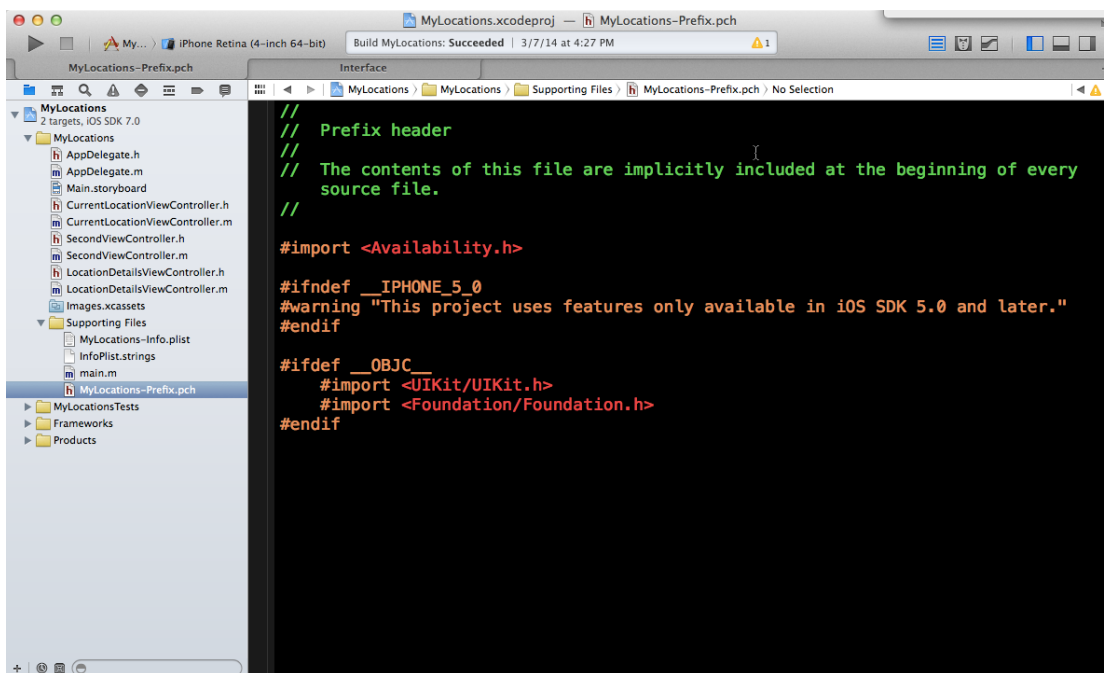
理论知识充电结束，继续我们的教程。

当前Xcode会报错”Unknown type name ‘CLLocationCoordinate2D’”。这是因为该类型是在Core Location框架中定义的，但Xcode在编译当前源代码时对Core Location框架一无所知。

当然，我们可以像在CurrentLocationViewController.h中那样在顶部使用#import来添加Core Location的头文件，
#import <CoreLocation/CoreLocation.h>

不过这里有个更简单的方法：使用预编译头文件（Prefix文件，precompiled header file）。

每个Xcode项目都有一个Prefix.pch文件，我们可以在项目结构面板的Supporting Files中找到该文件：



为了加速编译时间，我们可以使用预编译头文件。任何放到该文件中的内容都会在编译时自动引入到所有的源文件中。这里我们可以看到默认的pch(pre-compiled header file)中已有包含了UIKit和Foundation，这两个框架也是任何一个iOS应用都需要的基本框架。这就意味着我们在.h和.m文件中无需再次导入这两个头文件，因为它们已经在预编译头文件中自动导入了。

在MyLocations-Prefix.pch文件中导入CoreLocation头文件如下：

```
#ifdef __OBJC__  
    #import <UIKit/UIKit.h>  
    #import <Foundation/Foundation.h>  
    #import <CoreLocation/CoreLocation.h>  
#endif
```

使用command+b快捷键编译，会发现之前的错误提示消失了。
从此刻开始，该项目中的任何一个源文件都无需手动导入Core Location框架的头文件了~

如果你是个完美主义者或者有洁癖的人，可以从CurrentLocationViewController.h中删除下面对头文件的导入：

```
#import <UIKit/UIKit.h>  
#import <CoreLocation/CoreLocation.h>
```

删除后再次编译，一切都ok。

个人习惯在Prefix.pch中预先导入所有在项目中可能用到的系统框架，这样就会省掉不少麻烦，也不会因为这个犯低级错误。

好了，让我们回到刚才添加到LocationDetailsViewController中的新属性变量。当用户触碰Tag Location按钮的时候，我们需要设置这些属性变量的值。

首先在CurrentLocationViewController.m的顶部添加一行代码：

```
#import "LocationDetailsViewController.h"
```

在getLocation方法的后面添加prepareForSegue方法：

```
-(void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender{  
    if([segue.identifier isEqualToString:@"TagLocation"]){  
        UINavigationController *navigationController = segue.destinationViewController;  
        LocationDetailsViewController *controller =  
        (LocationDetailsViewController*)navigationController.topViewController;  
  
        controller.coordinate = _location.coordinate;  
        controller.placemark = _placemark;  
    }  
}
```

我们之前看到过类似的代码。这里我们首先获取destination view controller，然后设置其相关属性值。现在当segue被触发时，坐标和地址信息就会被发送到Tag Location界面。可以在新界面的viewDidLoad方法中显示这些信息。

在Xcode中切换到LocationDetailsViewController.m，然后添加一个viewDidLoad方法：

```
-(void)viewDidLoad{  
    [super viewDidLoad];  
  
    self.descriptionTextView.text = @"";  
    self.categoryLabel.text = @"";  
  
    self.latitudeLabel.text = [NSString stringWithFormat:@"%f",self.coordinate.latitude];  
    self.longitudeLabel.text = [NSString stringWithFormat:@"%f",self.coordinate.longitude];  
  
    if(self.placemark !=nil){  
        self.addressLabel.text = [self stringFromPlacemark:self.placemark];  
    }else{  
        self.addressLabel.text = @"No Address Found";  
    }  
    self.dateLabel.text = [self formatDate:[NSDate date]];  
}
```

以上代码的作用是把相关信息放到对应的标签中。这里用到了两个辅助方法，stringFromPlacemark，其作用是将CLPlacemark对象格式化为一个字符串，而另一个方法formatDate方法的作用则是格式化NSDate对象。

在viewDidLoad:方法下面添加一个stringFromPlacemark:方法

```
-(NSString*)stringFromPlacemark:(CLPlacemark *)placemark{

    return [NSString stringWithFormat:@"%@@ %@, %@, %@ %@,%@",
        placemark.subThoroughfare,placemark.thoroughfare,
        placemark.locality,placemark.administrativeArea,
        placemark.postalCode,placemark.country
    ];
}
```

这个方法的作用很简单，它的作用是将各种信息格式化到同一个字符串中。

接下来添加一个formatDate:方法：

```
-(NSString*)formatDate:(NSDate *)theDate{

    static NSDateFormatter *formatter = nil;

    if(formatter == nil){

        formatter = [[NSDateFormatter alloc] init];
        [formatter setDateStyle:NSDateFormatterMediumStyle];
        [formatter setTimeStyle:NSDateFormatterShortStyle];

    }

    return [formatter stringFromDate:theDate];
}
```

我们在上一个系列的教程中曾经接触过NSDateFormatter类。我们可以使用这个类将日期和时间转换成人类可读的字符串（使用用户所在地区的语言）。

不过这里还是有一些新东西，比如：

```
static NSDateFormatter *formatter = nil;
if(formatter == nil){

    ...

}
```

在上一系列的教程中，每次我们需要将一个NSDate类型的对象格式化成字符串时，都需要创建一个NSDateFormatter的新实例对象。不过NSDateFormatter是一个相对比较昂贵的对象。神马？对象还是贵贱之分？！！！额，在计算机的世界中，如果我们说某个对象很昂贵，通常是指它占用的内存或CPU时钟超出常人。而在这里，我们的意思其实是说系统需要花比较长的时间来初始化这种类型的对象。因此，如果我们需要多次初始化该类型的对象，应用的速度就会被拖慢（当然同时也会耗尽你电池中的电量）。

因此，最好是只创建NSDateFormatter对象以此，然后可以重复使用相同的对象。这里将用到一个技巧，那就是直到应用真正需要的时候才创建NSDateFormatter对象。这种做法被术语党称为lazy loading，在iOS开发中是一种非常重要的模式。

当我们在本地变量声明的前面放上关键词**static**时，就创建了一个特殊类型的便利，所谓的**static local**变量（静态本地变量）。这种类型的变量会一直保存其中的数值，即便方法结束也会如此。当下次调用该方法的时候，不会创建一个新的该类型变量，而是直接使用已有的变量。当然，我们无法在方法之外使用该变量（它是**local**本地的），但它的生命却很长，一旦创建就会永垂不朽？

当创建**formatter**变量的时候，初始情况下包含了数值**nil**。仅在第一次的时候我们才会在**if**判断语句中创建**NSDateFormatter**的实例变量。在此之后，**formatter**就不再是**nil**，而是包含了一个有效的**NSDateFormatter**对象。随后调用**formatDate:**方法时会重用已有的**NSDateFormatter**对象，而不会创建一个新的对象。

小练习：

想想看如何用另外的方式实现**lazy loading**

答案：

我们可以让**NSDateFormatter**成为一个实例变量而非静态本地变量。当然还有其它解决方法，不过我更偏爱静态本地变量这种方式。后续我们可能还会碰到**static**关键词，特别是在阅读其他人代码的时候，因此适当了解一下是有好处的。

编译运行应用，从**Simulator**的**Debug**菜单中选择**Apple**。耐心等待一会儿，当街道地址可见的时候触碰**Tag Location**按钮即可。

如果足够幸运的话（没有出现任何意外），可以看到地址，日期等信息，但地址信息只能看到第一部分（比如街道编号）。

之前我们曾对该标签进行设置，让它自动适应多行文本，但问题在于表视图并不知道这一点。

在Xcode中切换到**LocationDetailsViewController.m**，并添加以下方法：

#pragma mark - **UITableViewDelegate**

```
-(CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath{
    if(indexPath.section == 0 && indexPath.row == 0){
        return 88;
    }else if(indexPath.section == 2 && indexPath.row == 2){

        CGRect rect = CGRectMake(100, 10, 205, 10000);
        self.addressLabel.frame = rect;
        [self.addressLabel sizeToFit];

        rect.size.height = self.addressLabel.frame.size.height;
        self.addressLabel.frame = rect;

        return self.addressLabel.frame.size.height + 20;
    }else{
        return 44;
    }
}
```

当表视图加载**cell**单元格的时候会调用该代理方法。我们使用该方法来通知表视图每个**cell**的高度。通常情况下所有的**cell**单元格有相同高度，如果我们需要同时修改所有**cell**的高度，可以设置表视图的属性（**tableView.rowHeight**或是**storyboard**中**Row Height**属性）。

不过这个表视图有三种不同的cell单元格高度：

- 1.顶部Description描述单元格的高度。我们已经通过storyboard将其高度设置为88points
- 2.Address cell。这个cell的高度是可变的，因为里面可能是单行文本，也可能是多行文本，取决于街道地址字符串的长度。
- 3.其它cell。这些cell的高度是标准的44 points

heightForRowAtIndexPath方法中的三个if判断语句分别对应三种不同的情况。让我们来看看其中设置Address标签大小的相关代码：

```
//1
CGRect rect = CGRectMake(100, 10, 205, 10000);

//2
self.addressLabel.frame = rect;

//3
[self.addressLabel sizeToFit];

//4

rect.size.height = self.addressLabel.frame.size.height;
self.addressLabel.frame = rect;

//5

return self.addressLabel.frame.size.height +20;
```

这里用到了一些小技巧让UILabel自动根据文本内容来调整大小（自动换行），然后使用新计算的标签高度来判断cell的高度。这样说可能不太明白，我们还是一步步看吧：

1.CGRect是一个用来描述矩形信息的结构体。一个矩形有原点（包含X,Y坐标）和大小（宽度和高度）。CGRectMake()函数使用四个数值作为参数（x,y, width和height），然后放到一个新的CGRect结构体中。宽度值是205 points，而高度值是10000points。之所以设置这么大的值是为了让矩形足够高，以便容纳足够多的文本。

2.一旦我们有了初始化的矩形，就可以重新设置标签的大小。frame是CGRect的一个属性，用来描述视图的位置和大小。所有的UIView对象（及其子类，比如UILabel）都有一个frame矩形框。通过使用代码来调整这个frame,就可以调整界面上视图的位置。对多行的UILabel设置frame还有一个好处：现在我们可以让文本自动换行，以适应205points的宽度。因为在viewDidLoad中我们已经设置了标签上的文本内容，这一点得以实现。

3.现在标签已经让其内容自动换行了，因此我们需要调整标签的高度，因为我们不需要cell真的有10000points这么高。还记得storyboard里面Size to Fit Content这个选项吗？在代码中我们可以通过sizeToFit方法来实现这一效果。

4.通过调用sizeToFit发放，可以删除标签中右侧和底部的多余空间。但我们只需要调整标签的高度而非宽度，因此我们将新计算出的高度放回之前的rect。此时我们就获得了一个原点在x:100,y:10，宽度为205，高度适应文本内容的矩形。

5.现在我们已经知道标签的高度了，可以添加一个边界值（顶部10points,底部10points），从而计算出整个cell的高度。

如果你看了上面的方法吓尿了，哥绝不会笑你，因为这样的做法实现是反人类的。当然我们可以使用Auto Layout来帮忙，只是Auto Layout也有自己的一系列问题。

所以，在写上面的代码之前，最重要的是思考，否则你真不知道该怎么来实现。

好了，现在编译运行应用，现在反转后的地理位置信息会自动填充整个Address cell。你还可以试试其它的位置。

接下来让我们处理描述信息用的text view文本视图。

对text view文本视图的处理方式和上一系列教程中处理text field的方式比较类似。

首先在Xcode中切换到LocationDetailsViewController.m，然后添加一个新的实例变量。

该变量将用来保存描述用的文本信息。

@implementation LocationDetailsViewController

```
{  
    NSString *_descriptionText;  
}
```

然后在顶部添加一个initWithCoder:方法如下：

```
-(id)initWithCoder:(NSCoder *)aDecoder{  
    if((self = [super initWithCoder:aDecoder])){  
        _descriptionText = @"";  
    }  
    return self;  
}
```

这个方法的作用仅仅是为实例变量赋予一个初始值。

至于为什么用initWithCoder:方法，好好回忆一下几个init方法的区别吧，不知道就google,stack overflow，或者查官方文档，或者查之前的教程。这里就不废话了。

在viewDidLoad方法中将变量的内容放到text view中（之前这里放的是空字符串变量），只看黄色高亮部分：

```
-(void)viewDidLoad{  
    [super viewDidLoad];  
    self.descriptionTextView.text = _descriptionText;  
    self.categoryLabel.text = @"";  
  
    self.latitudeLabel.text = [NSString stringWithFormat:@"%0.8f",self.coordinate.latitude];  
    self.longitudeLabel.text = [NSString stringWithFormat:@"%0.8f",self.coordinate.longitude];  
  
    if(self.placemark != nil){
```



```

        self.addressLabel.text = [self stringFromPlacemark:self.placemark];
    }else{
        self.addressLabel.text = @"No Address Found";
    }
    self.dateLabel.text = [self formatDate:[NSDate date]];
}

```

接下来在文件底部提那家text view的delegate方法：

```

#pragma mark - UITextViewDelegate

-(BOOL)textView:(UITextView*)textView shouldChangeTextInRange:(NSRange)range
replacementText:(NSString *)text{
    _descriptionText = [textView.text stringByReplacingCharactersInRange:range
withString:text];
    return YES;
}

-(void)textViewDidEndEditing:(UITextView*)textView{
    _descriptionText = textView.text;
}

```

这些方法的作用是当用户在文本视图中输入内容的时候更新_descriptionText变量的内容。当然，如果你不告诉text view它有个delegate代理，这些方法就会毫无用处。

因此我们需要更改视图控制器的类声明部分为：

```
@interface LocationDetailsViewController () <UITextViewDelegate>
```

注意：这个修改是在.m，而非.h中完成的。之前提到过，我们可以在类扩展中声明outlet属性变量，从而让它们保持私有性，而对于delegate代理同样可以这样做。应用中还有什么对象会在乎LocationDetailsViewController是一个text view的代理呢？实际上这是内部事务，也就是所谓的implementation细节。

最后，让我们切换到storyboard，按住ctrl键，从text view拖一条线到view controller，并连接delegate这个outlet。

为了测试_descriptionText变量是否能够获取用户输入到text view中的信息，可以在done动作方法中添加一行NSLog()语句：

```

-(IBAction)done:(id)sender{

    NSLog(@"Description '%@'",_descriptionText);

    [self closeScreen];
}


```

编译运行应用，标记一个location，然后在描述文本区域输入一些文字。触碰Done按钮，然后看NSLog()中是否会显示你所输入的文本。

好了，今天的学习就到此结束吧。

话说，最新版本的Xcode(5.1)和iOS(7.1)下载了没有呢？

Downloads




Xcode 5.1

This is the complete Xcode developer toolset for Mac, iPhone, and iPad. It includes the Xcode IDE, iOS Simulator, and all required tools and frameworks for building OS X and iOS apps.

[Looking for an older version of Xcode? ▶](#)


[Download Xcode 5](#)

Posted Date: Mar 10, 2014
Build: 5B130a
Included iOS SDK: iOS 7.1
Included Mac SDK: OS X 10.9.2



Updating to iOS 7

- To determine the proper download for your iPhone 5s, iPhone 5c, or iPhone 5, click [here](#). To determine the proper download for your iPad Air, iPad mini, or iPad (4th generation), click [here](#).
- If you have questions about updating from an expired version of iOS 7 beta to the GM version, you can find detailed instructions [here](#).












iOS 7.1

This is the release version of iOS 7.1 for iPad, iPhone, and iPod touch. Devices updated to iOS 7.1 can not be restored to earlier versions of iOS.

Builds: 11D167
Posted Date: Mar 10, 2014

Downloads

iPad:

-  iPad Air (Model A1474)
-  iPad Air (Model A1475)
-  iPad mini (Model A1489)
-  iPad mini (Model A1490)
-  iPad (4th generation Model A1458)
-  iPad (4th generation Model A1459)
-  iPad (4th generation Model A1460)
-  iPad mini (Model A1432)
-  iPad mini (Model A1454)

[Single App binary supports 64-bit and 32-bit Apps](#)
January 10, 2014

[Getting Help with App Review Results](#)
January 10, 2014

[Make Your Apps Work Seamlessly with iOS 7](#)
December 17, 2013

[iTunes Connect Holiday Shutdown](#)
December 14, 2013

[iAd Workbench Expands to New Countries](#)
November 12, 2013

[Manage Game Center Leaderboard Scores](#)
October 28, 2013

[Announcing iOS 7 Tech Talks](#)
September 25, 2013

[Managing Availability of Your Apps' Previous Versions](#)
September 18, 2013

最近昆明恐怖袭击，马航失联等等事件让我们感叹，平平安安就是福。这两天就不送福利了，只送祝福和祈福。