CoreLocation & MapKit

作用

1. 提供相关的本地信息，找到前进的方向
2. 根据用户需求，查找地点、确定行车路线的应用
3. 帮助用户使用特殊通勤服务的应用
4. 自动记录用户的行程

获取用户的位置

使用CoreLocation可以获取设备的当前位置。需要满足以下条件：

- 只有再得到用户的许可，应用才能获取设备的当前位置
- 获取设备位置前，应用还必须确保设备启用了定位服务
- 满足这些请求条件后，应用便可启动位置请求，并对 Core Location 提供的结果进行分析和使用

需求和许可

再应用中使用CoreLocation，需要将框架 CoreLocation 加入项目目标，并且根据需要导入Core Location 头文件：

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

要在应用中使用MapKit,需要将框架MapKit加入项目目标，并在所用使用它的类中导入MapKit头文件：

```
#import <MapKit/MapKit.h>
```

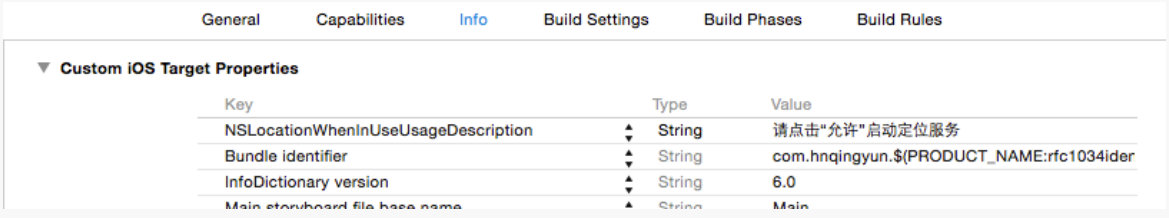
Core Location 尊重用户隐私,仅在用户许可时获取设备的当前位置。在应用“设置”的“隐私”部分，可以关闭或开启定位服务，还可以禁止或允许特定应用获取位置信息,

要请求用户允许使用定位服务，应用需要让CLLocationManager实例开始更新当前位置或将MKMapView实例的属性 ShowsUserLocation设置为YES。如果设备关闭了定位服务，Core Location将提醒用户再应用“设置”中开启定位服务；如果Core Location以前未请求用户允许获取设备的位置，它将显示一个提示框，请求用户许可。

```
    self.locationManager  = [[CLLocationManager alloc] init];
    self.locationManager.delegate = self;
    [self.locationManager startUpdatingLocation];
```

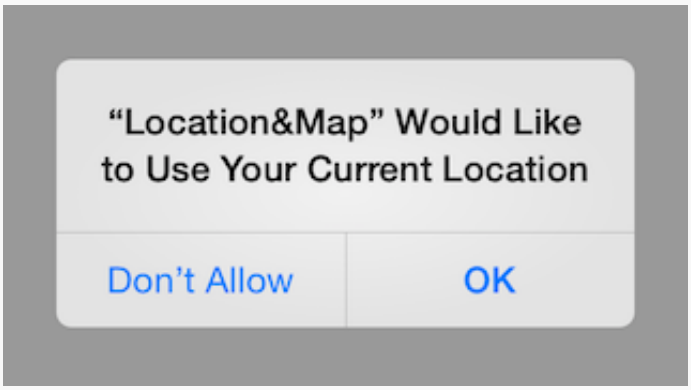
问题，为什么在这里CLLocationManager实例对象必须保存?

在ios8中需要专门作出申请，首先在info.plist中添加key `NSLocationWhenInUseUsageDescription` 或 `NSLocationAlwaysUsageDescription`,申请时给用户的提示的文字信息作为key的内容。



在authorizationStatus状态是 `kCLAuthorizationStatusNotDetermined` 时：

```
if ([CLLocationManager authorizationStatus] == kCLAuthorizationStatusNotDetermined) {
    //or [self.locationManager requestAlwaysAuthorization];
    [self.locationManager requestWhenInUseAuthorization];
}
```



如果用户轻按OK按钮，说明得到了用户的许可，位置管理器将获取当前位置。如果用户点击Don't Allow 禁止获取当前位置，将调用CLLocationManager的委托中响应授权状态变化的方法。

```
-(void)locationManager:(CLLocationManager *)manager didChangeAuthorizationStatus:(CLAuthorizationStatus)status{
    if (status == kCLAuthorizationStatusDenied) {
        //定位服务不可用。
        NSLog(@">>>>用户不允许使用GPS");
    } else if (status == kCLAuthorizationStatusRestricted){
        NSLog(@">>>>GPS暂时无法使用");
    }
}
```

检查定位服务是否已开启

要直接检查设备是否开启了定位服务，可使用CLLocationManager类方法 locationServivesEnabled:

```
    if ([CLLocationManager locationServicesEnabled]) {
        //当前定位服务可用
    }else{
        NSLog(@"定位服务不可用");
    }
```

开始为之请求

获得使用定位服务的许可后，便可使用CLLocationManager实例来获取当前位置了。需要根据定位需求对其进行定制：

```
    self.locationManager  = [[CLLocationManager alloc] init];

    //设置GPS的精度
    [self.locationManager setDesiredAccuracy:kCLLocationAccuracyBest];
    //移动多少距离触发新的位置事件
    [self.locationManager setDistanceFilter:100.0f];
    //委托处理位置事件，和授权状态变化
    self.locationManager.delegate = self;

    if ([CLLocationManager locationServicesEnabled]) {
        //当前定位服务可用
        [self.locationManager startUpdatingLocation];
    }else{
        NSLog(@"定位服务不可用");
    }
```

可设置CLLocationManager的多个属性，以指定它如何管理当前的位置。通过设置精度属性：desiredAccuracy,应用可告诉CLLocationManager，该以缩短电池续航时间为代价尽可能提高精度，还是为延长电池续航时间而使用用较低的精度。使用较低的精度，还可以缩短获取当前位置所需要的时间。通过设置属性distanceFilter，可告诉CLLocationManager，移动多长距离后才触发新的位置事件；这对微调基于位置变化的功能很有帮助。最后，给CLLocationManager指定了委托，这使得可以独特的方式相应位置事件和授权状态变化。为获取位置做好准备后，让位置管理器开始更新位置。`[appLocationManager.locationManager startUpdatingLocation];`

CLLocationManager将根据指定的参数在需要时利用GPS和/或Wi-Fi确定当前的位置。应实现两个委托方法，它们分别处理如下情形:位置管理器更新了当前位置或无法更新当前位置。获取位置后，将调用`locationManager:didUpdateLocations:`;无法更新当前位置将调用`-(void)locationManager:didFailWithError:`

位置管理器可能通过数组locations提供多个位置，其中最后一个是最新的的位置。位置管理器还可能在没有开始获取位置时，就快速返回GPS检测到的最后位置；这种情况下，如果GPS已关闭且设备发生了移动，返回的位置将很不准确。所以要检查位置的精确度：为负值或者精确度过差,放弃这些点。如果返回的位置相当精确，就可以使用该位置点。在逐步确定准确位置期间，位置管理器可能调用这个方法多次，编写这个方法时必须考虑到这一点。

```
-(void)locationManager:(CLLocationManager *)manager didUpdateLocations:(N
SArray *)locations{
    CLLocation *lastLocation = [locations lastObject];
    if(lastLocation.horizontalAccuracy < 0 || lastLocation.horizontalAccu
racy > 1000)
    {
        return;
    }
    NSLog(@">>>>%@",locations.lastObject);
}
```

如果位置管理器未能获取位置，它将调用 locationManager:didFailWithError: 导致错误的原因可能是未得到用户的许可，也可能是由于GPS或Wi-Fi不可用。发生错误时，通常要停止位置管理器停止更新当前位置并处理错误。

```
- (void)locationManager:(CLLocationManager *)manager
       didFailWithError:(NSError *)error
{
    [self.locationManager stopUpdatingLocation];
}
```

位置管理器委托可监控航向的变化，这很有用。可使用这些信息在地图上标出用户的前进路线相对于正确路线的偏差。要获取航向信息，需要让位置管理器对其进行监视。还可设置一个可选的筛选器，这样航向变化小于指定的度数，就不会获取更新。

```
CLLocationDegrees degreesFilter = 2.0;
if([CLLocationManager headingAvailable])
{
    [self.locationManager setHeadingFilter:degreesFilter];
    [self.locationManager startUpdatingHeading];
}
```

发生航向变化事件时，将调用委托方法locationManager:didUpdateHeading:

```
-(void)locationManager:(CLLocationManager *)manager didUpdateHeading:(CLH
eading *)newHeading{
    NSLog(@"new heading, magnetic%@",newHeading.magneticHeading);
}
```

参数newHeading提供了多项重要信息，其中包括相对于磁北的航向和相对于真北的航向，单位都是度。它还包括精度，者指出了磁北航向偏离了多少度。这个值越少的时候，航向就越准确；如果为负数，就说明航向无效，这可能是因为存在磁场干扰。时间戳支出了什么时候获取的，应通过检查它来避免使用过时的航向。

分析理解位置数据

位置管理器返回的位置是用CLLocation实例表示的。CLLocation包含多项有关位置的重要信息，首先是CLLocationCoordinate2D表示经度和纬度。
`LLocationCoordinate2D coord = lastLocation.coordinate; NSLog(@"Location lat/long: %f,%f",coord.latitude, coord.longitude);`

纬线和经线一样是人类为度量方便而假设出来的辅助线，定义为地球表面某点随地球自转所形成的轨迹。任何一根纬线都是圆形而且两两平行。纬线的长度是赤道的周长乘以纬线的纬度的余弦，所以赤道最长，离赤道越远的纬线，周长越短，到了两极就缩为0。纬线指示东西方向。从赤道向北和向南，各分90°，称为北纬和南纬，分别用“N”和“S”表示。

经线也称子午线，和纬线一样是人类为度量方便而假设出来的辅助线，定义为地球表面连接南北两极的大圆线上的半圆弧。任两根经线的长度相等，相交于南北两极点。每一根经线都有其相对应的数值，称为经度。经线指示南北方向。东经正数，西经为负数。经度是地球上一个地点离一根被称为本初子午线的南北方向走线以东或以西的度数。本初子午线的经度是0°，地球上其它地点的经度是向东到180°或向西到180°。

作为经度和纬度的补充，还有水平精度，它用CLLocationDistance或米数表示。水平精度指的是实际位置与返回的坐标之间的距离在指定米数之内。

```
    CLLocationAccuracy horizontalAccuracy = lastLocation.horizontalAccura
cy;
    NSLog(@"Horizontal accuracy: %f meters",horizontalAccuracy);
```

CLLocation 还提供了当前位置的海拔和垂直精度（单位为米）；如果设备没有GPS,返回的海拔将为零，而垂直精度为-1。

```
    CLLocationDistance altitude = lastLocation.altitude;
    NSLog(@"Location altitude: %f meters",altitude);

    CLLocationAccuracy verticalAccuracy = lastLocation.verticalAccuracy;
    NSLog(@"Vertical accuracy: %f meters",verticalAccuracy);
```

重大位置变化通知

Apple强烈建议应用在获取位置后停止位置更新，以延长电池的续航时间。如果应用不要求位置非常精确，可监视重大位置变化，这是一种高校的方式，既让应用获悉设备的位置发生了重大变化，尤可避免GPS和Wi-Fi不断监视当前位置，从而极大的节省电量。 ``` //开启 [self.locationManager startMonitoringSignificantLocationChanges];

```
//停止
[self.locationManager stopMonitoringSignificantLocationChanges];
```

通常，在设备位置变化超过500米或更换了连接的基站时，将发出通知。另外仅当最后一次通知是在5分钟之前时，才会发送新的通知。位置更新时间被交给委托方法：locationManager:didUpdateLocations:进行处理。

###使用GPX文件进行位置测试
测试基于位置的应用令人望而却步，需要对不方便的位置进行测试时尤其如此。所幸Xcode使用GPX文件提供了强大的位置测试支持。GPX文件是GPS交换格式文档，它使用XML格式，可用户在设备和GPS之间交换信息。在调试模式下，Xcode可使用GPX文件定义的“航点”来设置iOS模拟器或设备的当前位置。并且每个文件可以添加多个。

<?xml version="1.0"?> ```

显示地图

MapKit 框架为ios提供了地图用户界面功能，其中的基本类型是MKMapView，它显示地图、处理用户与地图的交互以及管理标注（像大头针）和覆盖层（如路线图或突出区域）。要更深入地了解iOS中地图的工作原理，必须明白坐标系。

理解坐标系

再MapKit中，有两个坐标系：地图坐标系和视图坐标系。地图使用魔卡托投影(将地球投影成平面)，将3D世界地图，投影到2D坐标系。坐标可使用经度和纬度指定。地图视图表示显示再屏幕上的地图部分，它使用标准的UIKit视图坐标，并负责决定在什么地方显示地图坐标指定的点。

配置和定制MKMapKit

MKMapkit控件在地图上显示用户的位置，并且允许用户滚动和放缩；并且可以调整地图的类型

```
//通过UISegmentedControl切换地图的区域
- (IBAction)mapTypeSelectionChanged:(id)sender
{
    UISegmentedControl *mapSelection =
    (UISegmentedControl *)sender;

    switch (mapSelection.selectedSegmentIndex) {
        case 0:
            [self.mapView setMapType:MKMapTypeStandard];
            break;
        case 1:
            [self.mapView setMapType:MKMapTypeSatellite];
            break;
        case 2:
            [self.mapView setMapType:MKMapTypeHybrid];
            break;

        default:
            break;
    }
}
```

除了设置地图类型外，另一种常见的（必须）定制是设置地图显示的区域。

```
    CLLocationCoordinate2D centerCoordinate;
    centerCoordinate.longitude = 113.675523f;
    centerCoordinate.latitude = 34.785306f;

    MKCoordinateSpan span;
    span.longitudeDelta = 0.005;
    span.latitudeDelta = 0.005;

    MKCoordinateRegion newRegion =
    MKCoordinateRegionMake(centerCoordinate, span);

    [self.mapView setRegion:newRegion
                   animated:YES];
```

响应用户互交

可给MkMapView指定委托，以便对用户与地图互交做出响应。用户与地图的交互包括平移和放缩、拖拽注释（annotation）以及用户轻按标注（callout）时进行响应。

用户平移或放缩地图时，将调用委托方法mapView:regionWillChangeAnimated:和mapView:regionDidChangeAnimated:,如果地图上显示了大量的信息或显示的信息随放缩的等

级而异，就可以使用这些方法删除不可见的注释，以及添加新的注释。下面例子演示了如何如何获取新的地图区域：

```
- (void)mapView:(MKMapView *)mapView
regionDidChangeAnimated:(BOOL)animated
{
    MKCoordinateRegion newRegion = [mapView region];
    CLLocationCoordinate2D center = newRegion.center;
    MKCoordinateSpan span = newRegion.span;

    NSLog(@"New map region center: <%f/%f>, span: <%f/%f>",
    center.latitude,center.longitude,span.latitudeDelta,
    span.longitudeDelta);
}
```

地图注释和覆盖层

地图视图（MKMapView）是一种可滚动的视图，行为独特；以标准方式在其中添加子视图时，子视图不会随地图视图滚动，而是静止的，其相对于地图视图框架的位置始终不变。要在地图上标出感兴趣点和区域，可使用地图注释和覆盖层。地图滚动或放缩时，注释和地图覆盖层在地图上的位置保持不变。地图注释是地图上的单个坐标点定义的，而地图覆盖层可以是线段、多边形或复杂形状。MapKit将注释（覆盖层）同其关联的视图区分开来。注释和覆盖层是数据，指定了想关联的视图应显示在什么地方，这些数据被直接添加到地图视图中。在需要显示注释或覆盖层时，地图视图将请求相关联的视图，就像表视图根据数据需要为索引路径请求单元格一样。

添加注释

任何对象都可作为地图视图的注释，前提条件是实现了MkAnnotation协议。Apple建议注释对象应该是轻量级的，因为对于添加的每个注释，地图视图都将包含一个指向它的引用；另外，注释太多可能影响地图的滚动和放 缩性能。如果注释非常简单，可使用MKPointAnnotation类(UIKit提供的已经实现了MkAnnotation协议)。

```
-(void)locationManager:(CLLocationManager *)manager didUpdateLocations:(N
SArray *)locations{
    CLLocation *location = locations.lastObject;
    MKPointAnnotation *anno = [[MKPointAnnotation alloc] init];
    anno.coordinate = location.coordinate;
    anno.title = @"这里";
    anno.subtitle = @"我在这里";
    [self.mapView addAnnotation:anno];
}
```

要实现MKAnnotation,类必须实现属性coordinate，地图视图将使用这个属性确定注释放在地图的什么地方。属性coordinate获取方法返回一个CLLocationCoordinate2D,同时要存贮其相应的经纬度。

```
//其隐性产生了一个get/set方法
@property (nonatomic)CLLocationCoordinate2D coordinate;
```

协议MkAnnotation还定义了另外两个可选属性：title和subtitle。在用户轻按注释视图时，地图视图可使用这两个属性来显示标注（callout）。标注的上面的那行为属性title，而下面那行为属性subtitle。

```
@property (nonatomic, copy)NSString *title;
@property (nonatomic, copy)NSString *subtitle;
```

在更新地图视图的注释的时候，如果地图上的注释不多，可以将地图上的视图都删除掉，让后再添加现在的注释。这种方法在注释不多的时候，完全可行，但注释很多时，必须设计更智能的方式，以高效地删除不需要的注释并添加新的注释。

```
[self.mapView removeAnnotations:self.mapView.annotations];
```

显示标准和自定义的注释视图

注释视图在地图上表示注释。MapKit提供了两种标准注释视图：大头针（表示搜索到的位置）和蓝点（表示当前位置）。可使用静态图像定制注释视图，还可通过创建MKAnnotationView或其子类来全面定制注释视图。

为让地图视图显示注释视图，地图视图委托需要实现方法 mapView:viewForAnnotation: ,如果Annotation使我们需要自定义的注释：

实现MKAnnotation协议

```
typedef enum{
    AnnotationGo,
    AnnotationSuspend,
    AnnotationEnd,
    AnnotationNow
} AnnotationType;

@interface QYAnnotation : NSObject<MKAnnotation>

@property (nonatomic, assign)CLLocationCoordinate2D coordinate;

@property (nonatomic, copy)NSString *title;

@property (nonatomic, copy)NSString *subtitle;

@property (nonatomic, assign)AnnotationType type;

@end
```

添加注释：

```
-(void)locationManager:(CLLocationManager *)manager didUpdateLocations:(N
SArray *)locations{
    CLLocation *location = locations.lastObject;
    QYAnnotation *annotation = [[QYAnnotation alloc] init];
    annotation.coordinate  = location.coordinate;

    if (self.nowAnnotation) {
        // 将现有的点移除;
        [self.mapView removeAnnotation:self.nowAnnotation];
        annotation.type = AnnotationNow;
        [self.mapView addAnnotation:annotation];
        self.nowAnnotation = annotation;
    }else {
        //设置region
        CLLocationCoordinate2D coordinate =location.coordinate;
        MKCoordinateRegion region = MKCoordinateRegionMake(coordinate, MK
CoordinateSpanMake(0.05f, 0.05));
        self.mapView.region = region;

        annotation.type = AnnotationGo;
        [self.mapView addAnnotation:annotation];
        [self.typeAnnotations addObject:annotation];
    }
}
```

添加注释视图

```
#pragma mark - map delegate

-(MKAnnotationView *)mapView:(MKMapView *)mapView viewForAnnotation:(id<M
KAnnotation>)annotation{
    if ([annotation isKindOfClass:[QYAnnotation class]]) {

        QYAnnotation *annotation_ = (QYAnnotation *)annotation;
        static NSString *Identifier = @"QYAnnotation";
        MKAnnotationView *annotationView = [mapView dequeueReusableAnnota
tionViewWithIdentifier:Identifier];
        if (!annotationView) {
            annotationView = [[MKAnnotationView alloc] initWithAnnotation
:annotation reuseIdentifier:Identifier];
        }

        switch (annotation_.type) {
            case AnnotationGo:
            {
```

```
                [annotationView setCenterOffset:CGPointMake(0, -12)];
                [annotationView setImage:[UIImage imageNamed:@"map_start_
icon"]];
                [annotationView setDraggable:YES];
            }
                break;
            case AnnotationNow:
            {
                [annotationView setImage:[UIImage imageNamed:@"currentloc
ation"]];
            }
                break;
            case AnnotationSuspend:{
                [annotationView setCenterOffset:CGPointMake(0, -12)];
                [annotationView setImage:[UIImage imageNamed:@"map_susoen
d_icon"]];
            }
                break;
            case AnnotationEnd:{
                [annotationView setCenterOffset:CGPointMake(0, -12)];
                [annotationView setImage:[UIImage imageNamed:@"map_stop_i
con"]];
            }
                break;
            default:
                break;
        }

        return annotationView;
    }
    return nil;
}
```

可拖拽的注释视图

可拖拽的注释视图很有用，它让用户能够再地图上指定地点。通过设置属性draggable设置为YES, 让注释视图是可拖拽的。通过地图视图的委托方法，mapView:annotationView:didChangeDragState:fromOldState:，可以更详细的了解用户是如何拖拽注释视图的。

使用地图覆盖层

地图覆盖层类似于地图注释，可以是任何实现了协议MKOverlay的对象，而地图视图委托将负责提供与地图覆盖层相关联的视图。地图覆盖层不同于注释的地方在于，它们不仅可以表示点，还可以表示线段和形状。因此非常适合用于再地图上表示路线或兴趣的区域。

添加覆盖层

```
#pragma mark - location delegate

-(void)locationManager:(CLLocationManager *)manager didUpdateLocations:(N
SArray *)locations{
    CLLocation *location = locations.lastObject;
    [self.locations addObject:location];

    QYAnnotation *annotation = [[QYAnnotation alloc] init];
    annotation.coordinate  = location.coordinate;

    if (!self.nowAnnotation) {
        annotation.type = AnnotationGo;
        [self.mapView addAnnotation:annotation];
        [self.typeAnnotations addObject:annotation];

    }


    if (self.nowAnnotation) {
        // 将现有的点移除;
        if (self.nowAnnotation) {
            [self.mapView removeAnnotation:self.nowAnnotation];
        }
        annotation.type = AnnotationNow;
        [self.mapView addAnnotation:annotation];
        self.nowAnnotation = annotation;
    }else {
        //设置region的中心
        //设置region
        CLLocationCoordinate2D coordinate =location.coordinate;
        MKCoordinateRegion region = MKCoordinateRegionMake(coordinate, MK
CoordinateSpanMake(0.05f, 0.05));
        self.mapView.region = region;
    }

    MKMapPoint *pointArray = malloc(sizeof(MKMapPoint)*self.locations.cou
nt);
    for (int i = 0; i < self.locations.count; i++) {
        CLLocation *loca = [self.locations objectAtIndex:i];
        pointArray[i] = MKMapPointForCoordinate([loca coordinate]);
    }
```

```
    MKPolyline *route = [MKPolyline polylineWithPoints:pointArray count:s
elf.locations.count];
    free(pointArray);

    [self.mapView addOverlay:route];
    self.nowLocation = location;
}
```

当地图需要显示地图覆盖层的时候，地图视图将调用委托方法mapView:viewForOverlay:,这个方法创建一个覆盖层视图，供地图进行显示。有三种形式：圆圈、多边形、和折线；也可自定义形状和覆盖层。

```
- (MKOverlayRenderer *)mapView:(MKMapView *)mapView
            viewForOverlay:(id < MKOverlay >)overlay
{
    MKOverlayRenderer *returnView = nil;

    if ([overlay isKindOfClass:[ICFFavoritePlace class]]) {
        ICFFavoritePlace *place = (ICFFavoritePlace *)overlay;

        CLLocationDistance radius =
        [[place valueForKeyPath:@"displayRadius"] floatValue];

        MKCircle *circle =
        [MKCircle circleWithCenterCoordinate:[overlay coordinate]
                                      radius:radius];

        MKCircleRenderer *circleView =
        [[MKCircleRenderer alloc] initWithCircle:circle];

        circleView.fillColor =
        [[UIColor redColor] colorWithAlphaComponent:0.2];

        circleView.strokeColor =
        [[UIColor redColor] colorWithAlphaComponent:0.7];

        circleView.lineWidth = 3;

        returnView = circleView;
    }
    if ([overlay isKindOfClass:[MKPolyline class]]) {
        MKPolyline *line = (MKPolyline *)overlay;

        MKPolylineRenderer *polylineRenderer =
        [[MKPolylineRenderer alloc] initWithPolyline:line];

        [polylineRenderer setLineWidth:3.0];
        [polylineRenderer setFillColor:[UIColor blueColor]];
        [polylineRenderer setStrokeColor:[UIColor blueColor]];
        returnView = polylineRenderer;
    }

    return returnView;
}
```

地理编码和反向地理编码

地理编码指的是找出人类能够看懂的地址对应的经度和纬度坐标；反向地理编码指的是根据坐标做出人类能看得懂的地址。

对地址进行地理编码

将用户提供的地址信息合成一个字符串，以便提供给地理编码器。

```
-(void)editTextEnd:(UITextField *)textField{
    NSLog(@">>>>>%@", textField.text);

    CLGeocoder *geocodr = [[CLGeocoder alloc] init];
    [geocodr geocodeAddressString:textField.text completionHandler:^(NSAr
ray *placemarks, NSError *error) {
        NSLog(@"%@", placemarks.firstObject);

        CLPlacemark *placemark = placemarks.firstObject;
        CLLocation *location = [placemark location];
        MKCoordinateRegion region = MKCoordinateRegionMake(location.coord
inate, MKCoordinateSpanMake(0.005, 0.005));
        [self.mapView setRegion:region animated:YES];
    }];

}
```

对位置进行反向地理编码

根据得到的经纬度， 反向的得到位置信息。

```
-(void)longPress:(UIGestureRecognizer *)press{
    CGPoint point = [press locationInView:self .mapView];
    NSLog(@"%@", NSStringFromCGPoint(point));
    CLLocationCoordinate2D location = [self.mapView convertPoint:point to
CoordinateFromView:self.mapView];
    CLLocation *cl = [[CLLocation alloc] initWithLatitude:location.latitu
de longitude:location.longitude];
    CLGeocoder *geocodr = [[CLGeocoder alloc] init];
    [geocodr reverseGeocodeLocation:cl completionHandler:^(NSArray *place
marks, NSError *error) {
        NSLog(@"%@", placemarks.firstObject);

        MKPointAnnotation *annotation = [[MKPointAnnotation alloc] init];
        MKPlacemark *placemark = [placemarks objectAtIndex:0];
        CLLocation *location = placemark.location;
        annotation.coordinate = location.coordinate;
        annotation.subtitle = placemark.name;
        annotation.title = placemark.country;
        [self.mapView addAnnotation:annotation];
    }];

}
```

地理围栏

地理围栏（Geogencing）也叫区域监视（regional monitoring），指的是能够知道设备已进入或离开指定的地图区域。iOS在siri中充分利用了这项功能，使其能够完成类似于下面的任务“在我离开办公室时提醒我带上面包”；“在我回到家时提醒我将烤肉放进烤箱”。iOS还在Passbook中使用了地址围栏功能，让用户能够再主屏幕上看到相关的凭证。

定义边界

可使用 Core Location 位置管理器（CLLocationManager）存贮一组应用要监视的区域。

```
    NSSet *monitoredRegions = [locManager monitoredRegions];

    for (CLRegion *region in monitoredRegions) {
        [locManager stopMonitoringForRegion:region];
    }
```

对于每个设置了地理围栏的地点，这个方法都像前一节描述的那样给它添加覆盖层视图，再让位置管理器开始监视该区域。要监视的区域必须有中心坐标、半径、以及供应用对区域进行跟

踪的标示符。当前只能监视圆形区域。

```
        CLRegion *region = [[CLCircularRegion alloc] initWithCenter:coord
inate radius:100 identifier:@"region"];
        [self.locationManager startMonitoringForRegion:region];
```

设备进入或者离开监视区域后，位置管理器将这一点告诉其委托：调用方法:locationManager:didEnterRegion: 或 locationManager:didEnterRegion:通过获取监视区域的标示符。这个标识符是在让位置管理器对区域监视时指定。

```
-(void)locationManager:(CLLocationManager *)manager didEnterRegion:(CLReg
ion *)region{
    NSLog(@"====%@", region.identifier);
}

-(void)locationManager:(CLLocationManager *)manager didExitRegion:(CLRegi
on *)region{
    NSLog(@">>>>%@", region.identifier);
}
```