KOTEI

武汉光庭信息技术股份有限公司
WUHAN KOTEI INFORMATICS CO., LTD.

iOS 地图 SDK

开发者指南

V1.00

iOS 地图 SDK 简介

1 概述

iOS 地图 SDK 是一套基于 iOS 7.1 及以上版本的地图应用程序开发接口，SDK 适配了 armv7、arm64 架构。通过 iOS 地图 SDK，您可使用服务端地图数据和服务轻松构建功能丰富、交互性强的地图应用。

- 丰富的地图数据
  基础地图数据一共 17 个级别，包含建筑物、道路、医院、学校等信息。同时还支持使用 KITileOverlay 对基础地图数据附加额外的特性，制作自定义特色地图。

- 多元的覆盖物显示
  通过在地图上添加覆盖物可丰富地图显示，优化地图体验，iOS 地图 SDK 提供覆盖物的种类包括：标注、折线、多边形（凹凸多边形）、圆、图片。

- 多样的手势操作
  支持双指缩放、双击放大、旋转等手势操作，且有相应的接口来控制手势操作。

1.1 文档范围

iOS 地图 SDK 使用人员相关参考。

1.2 版本兼容性

此 SDK 适合 7.1 版本以上的 iOS 系统。

2 SDK 使用工程配置

2.1 新建工程

如新建一个 Single View Application 工程，如下图所示：

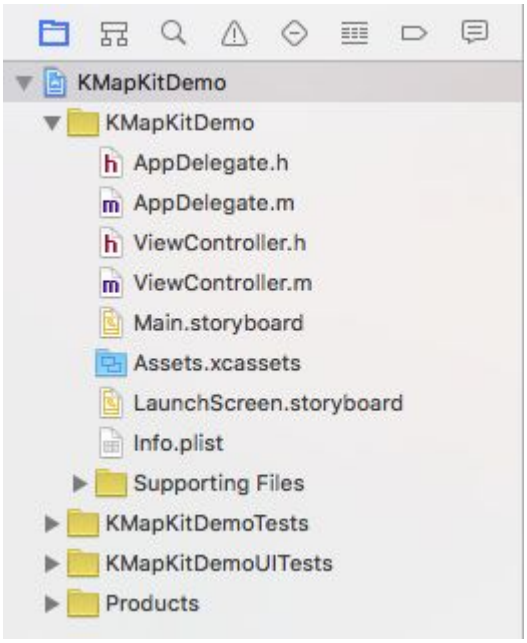

图 1

2.2 导入 SDK 工程配置

左侧目录中选中工程名，在 TARGETS->Build Phases-> Link Binary With Libaries 中点击“+”按钮，在弹出的窗口中点击“Add Other”按钮，选择解压后的 KIMapKit.framework 文件添加到工程中。

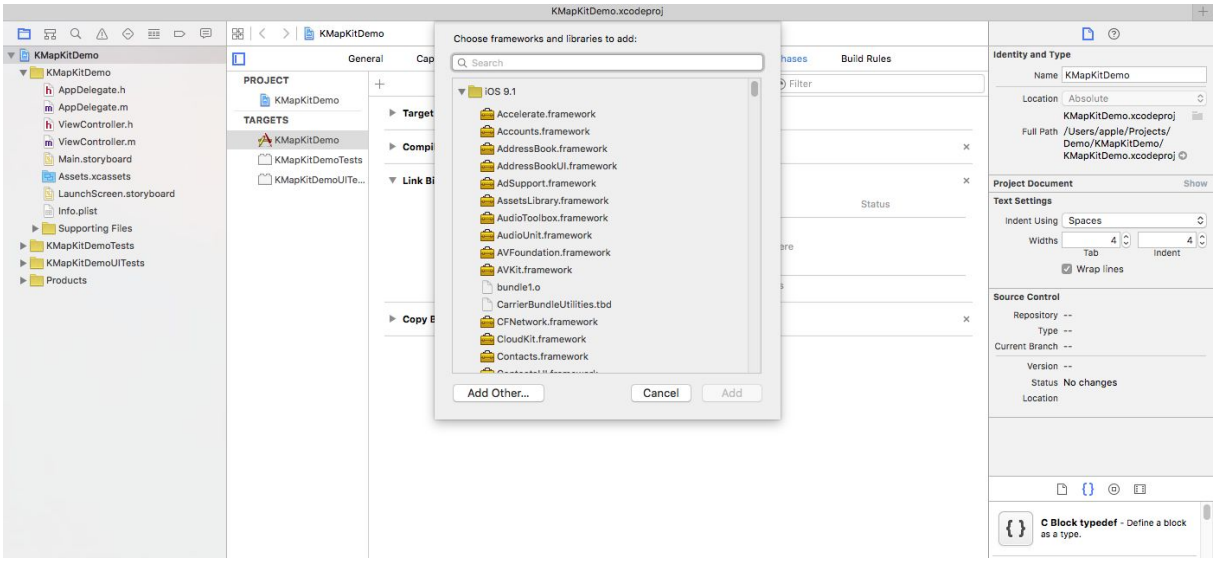

图 2

2.3 导入 libKIMapKit.bundle 资源文件

libKIMapKit.bundle 资源文件中存储了定位、默认大头针标注视图等图片， 可利用这些

资源图片进行开发。左侧目录中选中工程名，在右键菜单中选择 Add Files to“工程名”…，从 KIMapKit.framework->Resources 文件中选择 libKIMapKit.bundle 文件，并勾选“Copy items if needed”复选框，单击“Add”按钮，将资源文件添加到工程中。

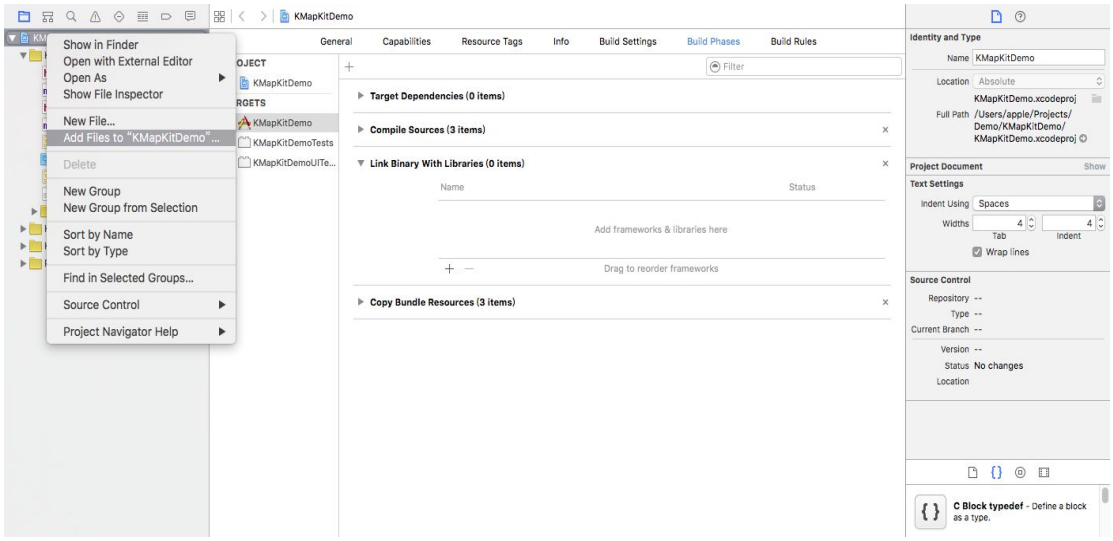

图 3

2.4 导入系统库

左侧目录中选中工程名，在 TARGETS->Build Settings-> Link Binary With Libaries 中点击“+”按钮，在弹出的窗口中查找并选择所需的库（见下表），单击“Add”按钮，将库文件添加到工程中。

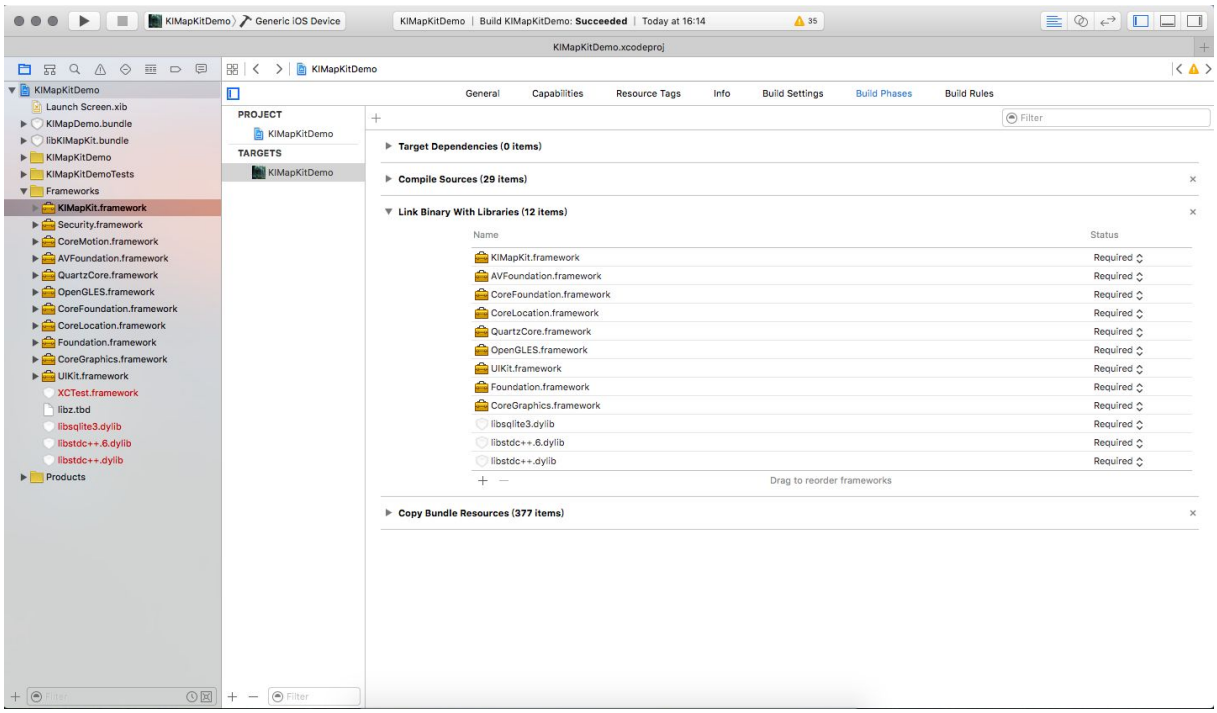

图 4

2.5 环境配置

在 TARGETS->Build Settings->Other Linker Flags 中添加-ObjC。

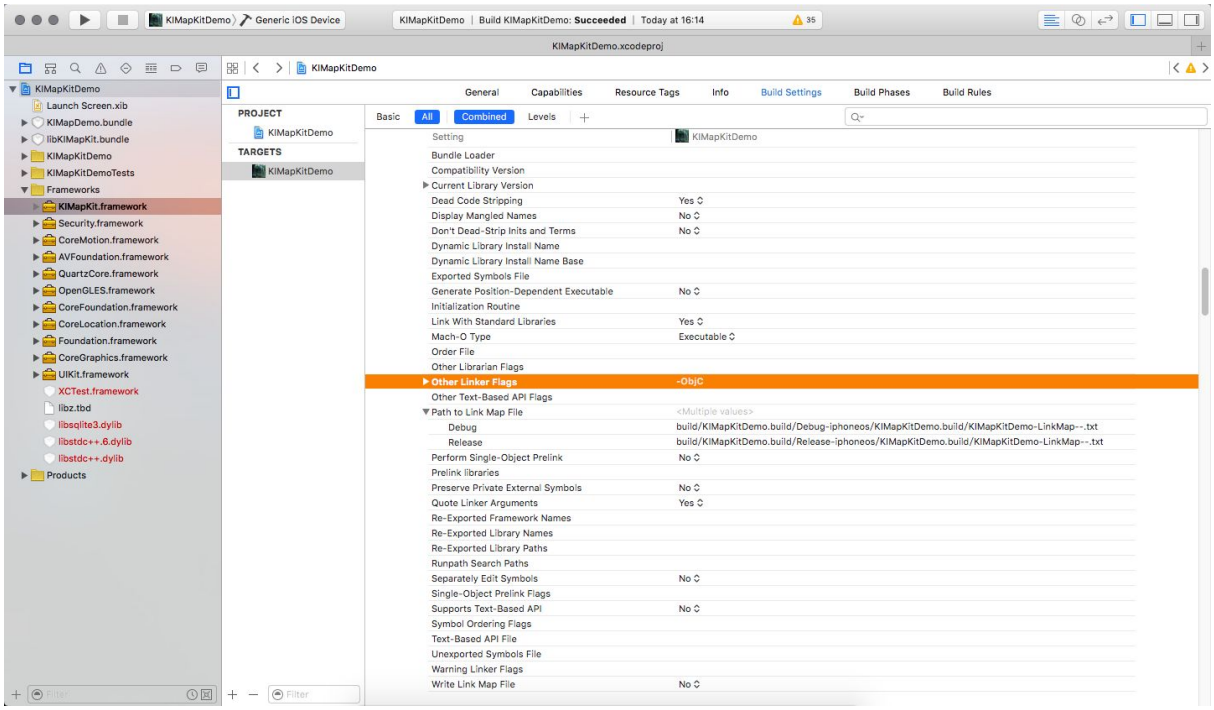

图 5

3 地图显示

3.1 显示主地图

3.1.1 定义 MapView 对象

修改 ViewController.m 文件，引入 KIMapKit.h 文件，继承 KIMapViewDelegate 协议，并定义 KIMapView 对象，示例代码如下所示：

```
#import <ViewController.m>
#import <KIMapKit/KIMapKit.h>
@interface ViewController ()<KIMapViewDelegate>
{
    KIMapView *_mapView;
}
@end
```

3.1.2 MapView 对象初始化

在 ViewController .m 文件相应的方法中进行地图初始化，初始化的步骤：

1. 构造 KIMapView 对象；

2. 设置代理；
3. 将 KIMapView 添加到 Subview 中。

对于 3D 矢量地图，在 viewDidLoad 方法中添加代码：

```
-(void) viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    _mapView = [[KIMapView alloc] initWithFrame:CGRectMake(0, 0,
    CGRectGetWidth(self.view.bounds), CGRectGetHeight(self.view.bounds))];
    _mapView.delegate = self;
    [self.view addSubview:_mapView];
}
```

4 地图图层

iOS SDK 提供三种地图类型 KIMapTypeStandard、KIMapTypeSatellite 和 KIMapTypeStandardNight；

注: KIMapTypeStandard 为标准地图，
KIMapTypeSatellite 为卫星地图，
KIMapTypeStandardNight 为夜景地图。

地图默认显示标准地图，从标准地图切换成卫星地图的方法如下：

```
//显示卫星地图(地图类型切换只需将地图类型设置成相应的类型)
_mapView.mapType = KIMapTypeSatellite;
```

5 地图覆盖物 Overlay

地图覆盖物主要包括标注、折线、多边形、圆、大地曲线和图片覆盖物，您可以在地图上添加这些覆盖物来丰富地图显示，通过设置其属性实现各种功能，优化地图体验。

5.1 标注

iOS 地图 SDK 提供标注点的协议<KIAnnotation>,它包含一个标注的基本信息：标注 View 的中心点坐标、标题和副标题。同时，还封装了一个标注类 KIPointAnnotation，它定义了一个位于指定位置的标注数据对象。

KIAnnotationView 是标注对应的 View，它用于在地图显示标记。

iOS 地图 SDK 提供一个默认的大头针标注 View——KIPinAnnotationView，通过它可设置大头针标注是否可被拾起拖拽、是否以动画效果显示等等。

（1） 修改 ViewController.m 文件，在 viewDidAppear 方法中添加如下所示代码添加标注数据对象。

```
- (void)viewDidAppear:(BOOL)animated
{
[super viewDidAppear:animated];
KIPointAnnotation *pointAnnotation = [[KIPointAnnotation alloc] init];
pointAnnotation.coordinate = CLLocationCoordinate2DMake(39.989631, 116.481018);
pointAnnotation.title = @"方恒国际";
pointAnnotation.subtitle = @"阜通东大街 6 号";
[_mapView addAnnotation:pointAnnotation];
}
```

（2） 实现 KIMapViewDelegate 的 mapView:viewForAnnotation:函数，设置标注样式。

```
- (KIAnnotationView *)mapView:(KIMapView *)mapView
viewForAnnotation:(id<KIAnnotation>)annotation
{
if ([annotation isKindOfClass:[KIPointAnnotation class]])
{
static NSString *pointReuseIndetifier = @"pointReuseIndetifier";
KIPinAnnotationView*annotationView = (KIPinAnnotationView*)[mapView
dequeueReusableAnnotationViewWithIdentifier:pointReuseIndetifier];
if (annotationView == nil)
{
annotationView = [[KIPinAnnotationView alloc] initWithAnnotation:annotation
reuseIdentifier:pointReuseIndetifier];
}
annotationView.canShowCallout= YES;          //设置气泡可以弹出，默认为 NO
annotationView.animatesDrop = YES;           //设置标注动画显示，默认为 NO
annotationView.draggable = YES;              //设置标注可以拖动，默认为 NO
annotationView.pinColor = KIPinAnnotationColorPurple;
return annotationView;
}
return nil;
}
```

5.2 自定义标注

iOS 地图 SDK 可自定义标注的图标和弹出气泡的样式，均可通过 KIAnnotationView 来实现。

5.2.1 自定义标注图标

若大头针样式的标注不能满足您的需求，您可以自定义标注图标。

1）添加标注数据对象，可参考大头针标注的步骤（1）。

2）导入标记图片文件到工程中。这里我们导入一个名为 restauant.png 的图片文件

3）在 KIMapViewDelegate 协议的回调函数 mapView:viewForAnnotation:中修改 KIAnnotationView 对应的标注图片。示例代码如下：

```
- (KIAnnotationView *)mapView:(KIMapView *)mapView
viewForAnnotation:(id<KIAnnotation>)annotation
{
if ([annotation isKindOfClass:[KIPointAnnotation class]])
{
static NSString *reuseIndetifier = @"annotationReuseIndetifier";
KIAnnotationView *annotationView = (KIAnnotationView *)[mapView
dequeueReusableAnnotationViewWithIdentifier:reuseIndetifier];
if (annotationView == nil)
{
annotationView = [[KIAnnotationView alloc] initWithAnnotation:annotation
reuseIdentifier:reuseIndetifier];
}
annotationView.image = [UIImage imageNamed:@"restaurant"];
//设置中　点偏移，使得标注底部中间点成为经纬度对应点
annotationView.centerOffset = CGPointMake(0, -18);
return annotationView;
}
return nil;
}
```

5.2.2 自定义气泡

气泡在 iOS 中又称为 callout，它由背景和气泡内容构成。

每个气泡显示的内容是根据您的需求定义的，这里我们按照如上图所示的气泡介绍实现一个自定义气泡的步骤。

（1） 新建自定义气泡类 CustomCalloutView，继承 UIView。

（2） 在 CustomCalloutView.h 中定义数据属性，包含：图片、商户名和商户地址。

```
@interface CustomCalloutView : UIView
@property (nonatomic, strong) UIImage *image; //商户图
@property (nonatomic, copy) NSString *title; //商户名
@property (nonatomic, copy) NSString *subtitle; //地址
@end
```

（3） 在 CustomCalloutView.m 中重写 UIView 的 drawRect 方法，绘制弹出气泡的背景。

```
#define kArrorHeight        10
- (void)drawRect:(CGRect)rect
```

```
{
[self drawInContext:UIGraphicsGetCurrentContext()];
self.layer.shadowColor = [[UIColor blackColor] CGColor];
self.layer.shadowOpacity = 1.0;
self.layer.shadowOffset = CGSizeMake(0.0f, 0.0f);
}
- (void)drawInContext:(CGContextRef)context
{
CGContextSetLineWidth(context, 2.0);
CGContextSetFillColorWithColor(context, [UIColor colorWithRed:0.3 green:0.3 blue:0.3
alpha:0.8].CGColor);
[self getDrawPath:context];
CGContextFillPath(context);
}
- (void)getDrawPath:(CGContextRef)context
{
CGRect rrect = self.bounds;
CGFloat radius = 6.0;
CGFloat minx = CGRectGetMinX(rrect),
midx = CGRectGetMidX(rrect),
maxx = CGRectGetMaxX(rrect);
CGFloat miny = CGRectGetMinY(rrect),
maxy = CGRectGetMaxY(rrect)-kArrorHeight;
CGContextMoveToPoint(context, midx+kArrorHeight, maxy);
CGContextAddLineToPoint(context,midx, maxy+kArrorHeight);
CGContextAddLineToPoint(context,midx-kArrorHeight, maxy);
CGContextAddArcToPoint(context, minx, maxy, minx, miny, radius);
CGContextAddArcToPoint(context, minx, minx, maxx, miny, radius);
CGContextAddArcToPoint(context, maxx, miny, maxx, maxx, radius);
CGContextAddArcToPoint(context, maxx, maxy, midx, maxy, radius);
CGContextClosePath(context);
}
```

（4） 定义用于显示气泡内容的控件，并添加到 SubView 中。我们需要一个 UIImageView 和两个 UILabel，添加方法如下：

```
#define kPortraitMargin      5
#define kPortraitWidth       70
#define kPortraitHeight      50
#define kTitleWidth          120
#define kTitleHeight         20
@interface CustomCalloutView ()
```

```
@property (nonatomic, strong) UIImageView *portraitView;
@property (nonatomic, strong) UILabel *subtitleLabel;
@property (nonatomic, strong) UILabel *titleLabel;
@end
@implementation CustomCalloutView
- (id)initWithFrame:(CGRect)frame
{
self = [super initWithFrame:frame];
if (self)
{
self.backgroundColor = [UIColor clearColor];
[self initSubViews];
}
return self;
}
- (void)initSubViews
{
//  添加图片，即商户图
self.portraitView = [[UIImageView alloc] initWithFrame:CGRectMake(kPortraitMargin,
kPortraitMargin, kPortraitWidth, kPortraitHeight)];
self.portraitView.backgroundColor = [UIColor blackColor];
[self addSubview:self.portraitView];
//  添加标题，即商户名
self.titleLabel = [[UILabel alloc] initWithFrame:CGRectMake(kPortraitMargin * 2 +
kPortraitWidth, kPortraitMargin, kTitleWidth, kTitleHeight)];
self.titleLabel.font = [UIFont boldSystemFontOfSize:14];
self.titleLabel.textColor = [UIColor whiteColor];
self.titleLabel.text = @"titletitletitletitle";
[self addSubview:self.titleLabel];
//  添加副标题，即商户地址
self.subtitleLabel = [[UILabel alloc] initWithFrame:CGRectMake(kPortraitMargin * 2 +
kPortraitWidth, kPortraitMargin * 2 + kTitleHeight, kTitleWidth, kTitleHeight)];
self.subtitleLabel.font = [UIFont systemFontOfSize:12];
self.subtitleLabel.textColor = [UIColor lightGrayColor];
self.subtitleLabel.text = @"subtitleLabelsubtitleLabelsubtitleLabel";
[self addSubview:self.subtitleLabel];
}
```

（5） 在 CustomCalloutView.m 中给控件传入数据。

```
- (void)setTitle:(NSString *)title
{
```

```
self.titleLabel.text = title;
}
- (void)setSubtitle:(NSString *)subtitle
{
self.subtitleLabel.text = subtitle;
}
- (void)setImage:(UIImage *)image
{
self.portraitView.image = image;
}
```

以上就是自定义气泡的全部过程，但是为了在点击标注时，弹出自定义的气泡，还需要自定义 AnnotationView。步骤如下：

（1）新建类 CustomAnnotationView，继承 KIAnnotationView 或 KIPinAnnotationView。若继承 KIAnnotationView，则需要设置标注图标；若继承 KIPinAnnotationView，使用默认的大头针标注。

（2）在 CustomAnnotationView.h 中定义自定义气泡属性，代码如下所示：

```
#import "CustomCalloutView.h"
@interface CustomAnnotationView : KIAnnotationView
@property (nonatomic, readonly) CustomCalloutView *calloutView;
@end
```

（3）在 CustomAnnotationView.m 中修改 calloutView 属性，如下：

```
@interface CustomAnnotationView ()
@property (nonatomic, strong, readwrite) CustomCalloutView *calloutView;
@end
```

（4）重写选中方法 setSelected。选中时新建并添加 calloutView，传入数据；非选中时删除 calloutView。

```
#define kCalloutWidth       200.0
#define kCalloutHeight      70.0
- (void)setSelected:(BOOL)selected animated:(BOOL)animated
{
if (self.selected == selected)
{
return;
}
if (selected)
{
if (self.calloutView == nil)
```

```
{
self.calloutView = [[CustomCalloutView alloc] initWithFrame:CGRectMake(0, 0,
kCalloutWidth, kCalloutHeight)];
self.calloutView.center = CGPointMake(CGRectGetWidth(self.bounds) / 2.f +
self.calloutOffset.x,
-CGRectGetHeight(self.calloutView.bounds) / 2.f + self.calloutOffset.y);
}
self.calloutView.image = [UIImage imageNamed:@"building"];
self.calloutView.title = self.annotation.title;
self.calloutView.subtitle = self.annotation.subtitle;
[self addSubview:self.calloutView];
}
else
{
[self.calloutView removeFromSuperview];
}
[super setSelected:selected animated:animated];
}
```

（5）修改 ViewController.m，在 KIMapViewDelegate 的回调方法 mapView:viewForAnnotation 中的修改 annotationView 的类型，代码如下：

```
#import “CustomAnnotationView.h”
- (KIAnnotationView *)mapView:(KIMapView *)mapView
viewForAnnotation:(id<KIAnnotation>)annotation
{
if ([annotation isKindOfClass:[KIPointAnnotation class]])
{
static NSString *reuseIndetifier = @"annotationReuseIndetifier";
CustomAnnotationView *annotationView = (CustomAnnotationView *)[mapView
dequeueReusableAnnotationViewWithIdentifier:reuseIndetifier];
if (annotationView == nil)
{
annotationView = [[CustomAnnotationView alloc] initWithAnnotation:annotation
reuseIdentifier:reuseIndetifier];
}
annotationView.image = [UIImage imageNamed:@"restaurant"];
//  设置为 NO，用以调用自定义的 calloutView
annotationView.canShowCallout = NO;
//  设置中心点偏移，使得标注底部中间点成为经纬度对应点
annotationView.centerOffset = CGPointMake(0, -18);
return annotationView;
```

```
}
return nil;
}
```

5.3 折线

折线类为 KIPolyline，由一组经纬度坐标组成，并以有序序列形式建立一系列的线段。iOS SDK 支持在 3D 矢量地图上绘制带箭头或有纹理等样式的折线。在地图添加折线的步骤如下：

（1）修改 ViewController.m 文件，在 viewDidLoad 方法中构造折线数据对象（一组经纬度坐标点）。

```
-(void) viewDidLoad
{
//构造折线数据对象
CLLocationCoordinate2D commonPolylineCoords[4];
commonPolylineCoords[0].latitude = 39.832136;
commonPolylineCoords[0].longitude = 116.34095;
commonPolylineCoords[1].latitude = 39.832136;
commonPolylineCoords[1].longitude = 116.42095;
commonPolylineCoords[2].latitude = 39.902136;
commonPolylineCoords[2].longitude = 116.42095;
commonPolylineCoords[3].latitude = 39.902136;
commonPolylineCoords[3].longitude = 116.44095;
//构造折线对象
KIPolyline *commonPolyline = [KIPolyline
polylineWithCoordinates:commonPolylineCoords count:4];
//在地图上添加折线对象
[_mapView addOverlay: commonPolyline];
}
```

（2）在 ViewController.m 文件中，实现 KIMapViewDelegate 的 mapView:viewForOverlay:函数，设置折线的样式。示例代码如下：

```
- (KIOverlayView *)mapView:(KIMapView *)mapView viewForOverlay:(id
<KIOverlay>)overlay
{
if ([overlay isKindOfClass:[KIPolyline class]])
{
KIPolylineView *polylineView = [[KIPolylineView alloc] initWithPolyline:overlay];
polylineView.lineWidth = 10.f;
polylineView.strokeColor = [UIColor colorWithRed:0 green:0 blue:1 alpha:0.6];
```

```
polylineView.lineJoinType = kKILineJoinRound;//连接类型
polylineView.lineCapType = kKILineCapRound;//端点类型
return polylineView;
}
return nil;
}
```

在该回调函数中， 调用 KIPolylineView 的 loadStrokeTextureImage 方法可以设置折线的纹理图片。

5.4 多边形

多边形类为 KIPolygon，与 KIPolyline 类似，包括有序序列的一系列坐标，但是多边形包含有内部区域。

在地图添加多边形的步骤如下：

（1）修改 ViewController.m 文件，在 viewDidLoad 方法中构造多边形的数据对象（一组经纬度坐标点）。

```
-(void) viewDidLoad
{
//构造多边形数据对象
CLLocationCoordinate2D coordinates[4];
coordinates[0].latitude = 39.810892;
coordinates[0].longitude = 116.233413;
coordinates[1].latitude = 39.816600;
coordinates[1].longitude = 116.331842;
coordinates[2].latitude = 39.762187;
coordinates[2].longitude = 116.357932;
coordinates[3].latitude = 39.733653;
coordinates[3].longitude = 116.278255;
KIPolygon *polygon = [KIPolygon polygonWithCoordinates:coordinates count:4];
//在地图上添加折线对象
[_mapView addOverlay: polygon];
}
```

（2）在 ViewController.m 文件中，实现 KIMapViewDelegate 的 mapView:viewForOverlay:函数，设置多边形的样式。示例代码如下：

```
- (KIOverlayView *)mapView:(KIMapView *)mapView viewForOverlay:(id
<KIOverlay>)overlay
{
if ([overlay isKindOfClass:[KIPolygon class]])
{
KIPolygonView *polygonView = [[KIPolygonView alloc] initWithPolygon:overlay];
polygonView.lineWidth = 5.f;
```

```
polygonView.strokeColor = [UIColor colorWithRed:0.6 green:0.6 blue:0.6 alpha:0.8];
polygonView.fillColor = [UIColor colorWithRed:0.77 green:0.88 blue:0.94 alpha:0.8];
polygonView.lineJoinType = kKILineJoinMiter;//连接类型
return polygonView;
}
return nil;
}
```

5.5 圆

圆类为 KICircle，圆对象由中心点（经纬度）和半径（米）构成。
在地图绘制圆的步骤如下：

（1） 在 ViewController.m 的 viewDidLoad 方法中根据中心点和半径构造圆对象。

```
-(void) viewDidLoad
{
//构造圆
KICircle *circle = [KICircle
circleWithCenterCoordinate:CLLocationCoordinate2DMake(39.952136, 116.50095)
radius:5000];
//在地图上添加圆
[_mapView addOverlay: circle];
}
```

（2） 在 ViewController.m 文件中，实现 KIMapViewDelegate 的 mapView:viewForOverlay: 函数，设置圆的样式。示例代码如下：

```
- (KIOverlayView *)mapView:(KIMapView *)mapView viewForOverlay:(id
<KIOverlay>)overlay
{
if ([overlay isKindOfClass:[KICircle class]])
{
KICircleView *circleView = [[KICircleView alloc] initWithCircle:overlay];
circleView.lineWidth = 5.f;
circleView.strokeColor = [UIColor colorWithRed:0.6 green:0.6 blue:0.6 alpha:0.8];
circleView.fillColor = [UIColor colorWithRed:1.0 green:0.8 blue:0.0 alpha:0.8];
circleView.lineDash = YES;
return circleView;
}
return nil;
}
```

5.6 大地曲线

大地曲线类为 KIGeodesicPolyline，继承自 KIPolyline。

在地图添加大地曲线的步骤如下：

（1） 修改 ViewController.m 文件，在 viewDidLoad 方法中构造折大地曲线数据对象（一组经纬度坐标点或地图投影点），下面以经纬度点为例：

```
- (void) viewDidLoad
{
CLLocationCoordinate2D geodesicCoords[2];
geodesicCoords[0].latitude = 39.905151;
geodesicCoords[0].longitude = 116.401726;
geodesicCoords[1].latitude = 38.905151;
geodesicCoords[1].longitude = 70.401726;
//构造大地曲线对象
KIGeodesicPolyline *geodesicPolyline = [KIGeodesicPolyline
polylineWithCoordinates:geodesicCoords count:2];
[_mapView addOverlay:geodesicPolyline];
}
```

（2） ViewController.m 文件中，实现 KIMapViewDelegate 的 mapView:viewForOverlay:函数，设置曲线的样式。示例代码如下：

```
- (KIOverlayView *)mapView:(KIMapView *)mapView viewForOverlay:(id
<KIOverlay>)overlay
{
    if ([overlay isKindOfClass:[KIPolyline class]])
    {
        KIPolylineView *polylineView = [[KIPolylineView alloc] initWithPolyline:overlay];
        polylineView.lineWidth = 8.f;
        polylineView.strokeColor = [UIColor colorWithRed:0 green:0 blue:1 alpha:0.8];
        return polylineView;
    }
    return nil;
}
```

5.7 图片覆盖物

图片覆盖物类为 KIGroundOverlay， 可完成将一张图片以合适的大小贴在地图指定的位置上的功能。

绘制图片覆盖物的步骤如下：

（1）在 ViewController.m 的 viewDidLoad 方法中根据范围和图片构造图片覆盖物。

```
- (void) viewDidLoad
{
    KICoordinateBounds  coordinateBounds  =
```

```
KICoordinateBoundsMake(CLLocationCoordinate2DMake
(39.939577, 116.388331),CLLocationCoordinate2DMake(39.935029, 116.384377));
KIGroundOverlay *groundOverlay = [KIGroundOverlay
groundOverlayWithBounds:coordinateBounds icon:[UIImage imageNamed:@"GWF"]];
[_mapView addOverlay:groundOverlay];
_mapView.visibleMapRect = groundOverlay.boundingMapRect;
}
```

（2）实现 KIMapViewDelegate 的 mapView:viewForOverlay:函数，以在地图上显示图片覆盖物。

```
- (KIOverlayView *)mapView:(KIMapView *)mapView viewForOverlay:(id
<KIOverlay>)overlay{
if ([overlay isKindOfClass:[KIGroundOverlay class]])
{
KIGroundOverlayView *groundOverlayView = [[KIGroundOverlayView alloc]
initWithGroundOverlay:overlay];
return groundOverlayView;
}
return nil;
}
```

5.8 覆盖物层级控制(3D)

3D 矢量地图提供覆盖物的层级控制接口，可帮助将您添加的覆盖物绘制到地图底图标注之下。示例代码如下：

```
//将折线绘制在地图底图标注和兴趣点图标之下
[_mapView addOverlay:polyline level:KIOverlayLevelAboveRoads];
```

说明：level 传入一个枚举，其中：

KIOverlayLevelAboveRoads 表示“在地图底图标注和兴趣点图标之下绘制 overlay”，

KIOverlayLevelAboveLabels 是默认值表示“在地图底图标注和兴趣点图标之上绘制 overlay”。

6 地图控件

iOS 地图 SDK 提供 “地图 Logo”、“指南针”、“比例尺”、“缩放按钮”、“定位按钮”五种地图控件。

说明：以下示例代码都基于“显示地图”中初始化并添加到 Subview 中的_mapView 对象。

6.1 地图 Logo

地图 Logo 不能移除，但可通过 KIMapView.logoCenter 属性来调整 Logo 的显示位置。在 ViewController.m 的 viewDidLoad 方法添加如下：

```
_mapView.logoCenter = CGPointMake(CGRectGetWidth(self.view.bounds)-55, 450);
```

6.2 指南针

指南针默认是开启状态，显示在地图的右上角。

通过 KIMapView 的 showsCompass 属性用来控制指南针的可见性。compassOrigin 属性可改变指南针的显示位置。在 ViewController.m 的 viewDidLoad 方法添加如下：

```
_mapView.showsCompass= YES; //  设置成 NO 表示关闭指南针；YES 表示显示指南针
_mapView.compassOrigin= CGPointMake(_mapView.compassOrigin.x, 22); //设置指南针位置
```

6.3 比例尺

比例尺表示地图上两点间距离与实际与之对应的两点距离的比，在不同的缩放级别下，比例尺代表的长度也是不同的。

在 iOS 地图 SDK 中，比例尺默认显示在地图的左上角。KIMapView 的 showScale 属性用来控制比例尺的可见性，scaleOrigin 属性用来改变比例尺的显示位置。在 ViewController.m 的 viewDidLoad 方法添加如下代码：

```
_mapView.showScale= YES;      //设置成 NO 表示不显示比例尺；YES 表示显示比例尺
_mapView.scaleOrigin= CGPointMake(_mapView.scaleOrigin.x, 22);
```

7 地图操作及手势

7.1 手势控制

iOS 地图 SDK 支持丰富的地图交互手势功能。手势功能默认都是开启的。

7.1.1 缩放手势

缩放手势可改变地图的缩放级别，地图响应的手势如下：

双击地图可以使缩放级别增加 1 (放大)

两个手指捏/拉伸

通过 KIMapView 的 scrollEnabled 属性可以禁用或启用缩放手势。这不会影响用户使用地图上的缩放控制按钮。禁用缩放手势的代码如下：

```
- (void)viewDidLoad {
{
    [super viewDidLoad];
    _mapView.zoomEnabled = NO;      //NO 表示禁用缩放手势，YES 表示开启
}
```

7.1.2 平移（滑动）手势

用户可以用手指拖动地图四处滚动（平移）或用手指滑动地图（动画效果）。通过 KIMapView 的 scrollEnabled 属性可以禁用或开启平移（滑动）手势。以下介绍展示如何禁用缩放手势，示例代码如下：

```
- (void)viewDidLoad {
{
```

```
    [super viewDidLoad];
    _mapView.scrollEnabled = NO;      //NO 表示禁用滑动手势，YES 表示开启
}
```

7.1.3 旋转手势(3D)

用户可以用两个手指在地图上转动，可以旋转 3D 矢量地图。通过调用类 KIMapView 的 rotateEnabled 属性禁用或开启旋转手势。

```
- (void)viewDidLoad {
{
    [super viewDidLoad];
    _mapView.rotateEnabled= NO;      //NO 表示禁用旋转手势，YES 表示开启
}
```

7.1.4 倾斜手势(3D)

用户可以在地图上放置两个手指，移动它们一起向下或向上去增加或减小倾斜角。通过 KIMapView 的 rotateCameraEnabled 属性禁用或启用倾斜手势。

```
- (void)viewDidLoad {
{
    [super viewDidLoad];
    _mapView.rotateEnabled= NO;      //NO 表示禁用倾斜手势，YES 表示开启
}
```

7.2 地图操作

通过 iOS 地图 SDK 中操作地图方法来进行操作地图显示。

7.2.1 地图缩放

地图的缩放级别的范围是[3-19]，调用 KIMapView 的 setZoomLevel 方法设置地图的缩放级别，用来缩放地图。示例代码如下：

```
[_mapView setZoomLevel:17.5 animated:YES];
```

7.2.2 地图平移

地图平移时，缩放级别不变，可通过改变地图的中心点来移动地图，示例代码如下：

```
[_mapView setCenterCoordinate:center animated:YES];
```

7.2.3 地图旋转（3D）

旋转角度的范围是[0.f 360.f]，以逆时针为正向。调用 KIMapView 的 setRotationDegree 方法设置地图的旋转角度。示例代码如下：

```
[_mapView setRotationDegree:60.f animated:YES duration:0.5];
```

7.2.4 地图倾斜（3D）

倾斜角度范围为[0.f, 45.f]，调用 KIMapView 的 setCameraDegree 方法设置地图的倾斜角度。示例代码如下：

```
[_mapView setCameraDegree:30.f animated:YES duration:0.5];
```

7.3 地图截屏

iOS 地图 SDK 支持对选定的屏幕地图区域(CGRect)进行截屏，截取的内容包括：地图、覆盖物、弹出气泡。

使用 KIMapView 中的 takeSnapshotInRect 方法进行截屏，该方法返回 UIImage 对象。示例代码如下：

```
CGRect inRect = CGRectMake(80,142,160,284);
UIImage *screenshotImage = [_mapView takeSnapshotInRet:inRect] ;
```

8 定位

在使用定位功能前需要在工程的 info.plist 中追加 NSLocationWhenInUseUsageDescription 或 NSLocationAlwaysUsageDescription 字段，以申请相应的权限。

- NSLocationWhenInUseUsageDescription 表示应用在前台的时候可以搜到更新的位置信息。
- NSLocationAlwaysUsageDescription 表示应用在前台和后台（suspend 或 terminated）都可以获取到更新的位置数据。

9 离线地图

地图 iOS SDK 支持 3D 矢量地图数据的下载和更新。

离线地图数据以 KIOfflineItem 为单位进行下载，每个 KIOfflineItem 对象包含城市编码、城市名称、数据状态等基本信息，是离线数据省信息（KIOfflineProvince）和离线数据城市信息（KIOfflineCity）的基类。离线数据城市信息（KIOfflineCity）又派生出三个子类，分别是全国概要图（KIOfflineItemNationWide）、直辖市（KIOfflineItemMunicipality）和普通城市（KIOfflineItemCommonCity）。各类的关系如下图所示：

按照下载项分项下载的离线数据效果如下图所示：

获取离线数据项的示例代码如下：

```
- (void)setupCities
{
    self.sectionTitles = @[@"全国", @"直辖市", @"省份"];
    self.cities = [KIOfflineMap sharedOfflineMap].cities;//普通城市和直辖市
    self.provinces = [KIOfflineMap sharedOfflineMap].provinces;//省
    self.municipalities = [KIOfflineMap sharedOfflineMap].municipalities;//直辖市
}
- (KIOfflineItem *)itemForIndexPath:(NSIndexPath *)indexPath
{
```

```
KIOfflineItem *item = nil;
switch (indexPath.section)
{
    case 0:
{
    item = [KIOfflineMap sharedOfflineMap].nationWide;//全国概要图
    break;
}
case 1:
{
    item = self.municipalities[indexPath.row];//直辖市
    break;
}
case 2:
{
    item = nil;
    break;
}
default:
{
    KIOfflineProvince *pro = self.provinces[indexPath.section -
    self.sectionTitles.count];
    if (indexPath.row == 0)
    {
        item = pro; //整个省
    }
    else
    {
    item = pro.cities[indexPath.row - 1]; //市
    }
    break;
    }
    }
    return item;
}
```

离线下载某个城市(参数 city)地图数据包示例代码如下：

```
- (void)download: (KIOfflineCity *) city
{
[[KIOfflineMap sharedOfflineMap] downloadCity:city
downloadBlock:^(KIOfflineMapDownloadStatus downloadStatus, id info) {
dispatch_async(dispatch_get_main_queue(), ^{
if (downloadStatus == KIOfflineMapDownloadStatusWaiting)
```

```
{
NSLog(@"状态为: %@", @"等待下载");
}
else if(downloadStatus == KIOfflineMapDownloadStatusStart)
{
NSLog(@"状态为: %@", @"开始下载");
}
else if(downloadStatus == KIOfflineMapDownloadStatusProgress)
{
NSLog(@"状态为: %@", @"正在下载");
}
else if(downloadStatus == KIOfflineMapDownloadStatusCancelled) {
NSLog(@"状态为: %@", @"取消下载");
}
else if(downloadStatus == KIOfflineMapDownloadStatusCompleted) {
NSLog(@"状态为: %@", @"下载完成");
}
else if(downloadStatus == KIOfflineMapDownloadStatusUnzip) {
NSLog(@"状态为: %@", @"下载完成，正在解压缩");
}
else if(downloadStatus == KIOfflineMapDownloadStatusError) {
NSLog(@"状态为: %@", @"下载错误");
}
else if(downloadStatus == KIOfflineMapDownloadStatusFinished) {
NSLog(@"状态为: %@", @"全部完成");
[self.mapView reloadMap];                    //激活离线地图
}
}];
}
```

暂停某个城市(参数 city)离线地图下载的示例代码如下：

```
- (void)pause:(KIOfflineItem *)item
{
NSLog(@"pause :%@", item.name);
[[KIOfflineMap sharedOfflineMap] pauseItem:item];
}
```

暂停所有离线地图下载的函数为 ：[[KIOfflineMap sharedOfflineMap] cancelAll]。