

FengMap iOS SDK 开发指南 v1.1.0

● 简介

■ 什么是 FengMap iOS SDK?

FengMap iOS SDK 是一套基于 iOS 7.0 及以上版本设备的应用程序接口，您可以通过该接口实现丰富的室内地图功能。

地图：提供地图（2D、3D）的展示和缩放、平移、旋转、改变视角等地图操作；

主题：提供不同的地图主题风格，体验不同的炫酷效果。开发者可以自定义主题，使地图的风格与 App 之间更加契合；

搜索分析：可根据关键字、类型、ID 等对 POI 数据进行搜索；

路径规划：路径规划支持同层、跨层路径规划；

地图标注物：提供多种地图标注物，满足开发者的各种需求；

离线地图：支持使用离线地图，节省用户流量，同时为用户带来更好的地图体验；

■ 面向的用户

FengMap iOS SDK 是提供给具有一定 iOS 编程经验的开发人员使用。

■ 获取 FengMap iOS SDK

开发者请到 [FengMap SDK](#) 网站下载。

■ 兼容性

FengMap iOS SDK 支持 iOS 7.0 及以上操作系统，支持 armv7、armv7s、arm64 处理器。

● 配置开发环境

■ 开发工具

建议使用 Xcode6.1 及以上版本（Mac OS 系统环境为 10.10 及以上版本）。

■ Xcode 工程配置方法

1. 引入头文件

首先将 FengMap iOS SDK 提供的头文件和静态库文件拷贝到您的工程目录下，在 Xcode 中引入 FengMap iOS SDK 提供的头文件。

在您需要使用 FengMap iOS SDK 的文件中添加以下代码

```
#import "FMMapKit.h"
```

2. 引入静态库文件

FengMap iOS SDK 提供了真机和模拟器两种环境所使用的静态库文件，分别存放在 libs/Release-iphoneos 和 libs/Release-iphonesimulator 文件夹下。有三种方式可以引入静态库文件：

第一种方式：直接将对应平台的.a 文件拖拽至 Xcode 工程左侧的 Groups&Files 中，缺点是每次在真机和模拟器编译时都需要重新添加.a 文件；

第二种方式：使用 lipo 命令将设备和模拟器的.a 合并成一个通用的.a 文件，将合并后的通用.a 文件拖拽至工程中即可，具体操作如下：首先，打开终端，cd 到 libs 文件夹中，然后使用如下命令：lipo -create Release-iphoneos/libFMMapKit.a Release-iphonesimulator/libFMMapKit.a -output libFMMapKit.a，生成新的.a 文件存放在 libs 文件夹中；

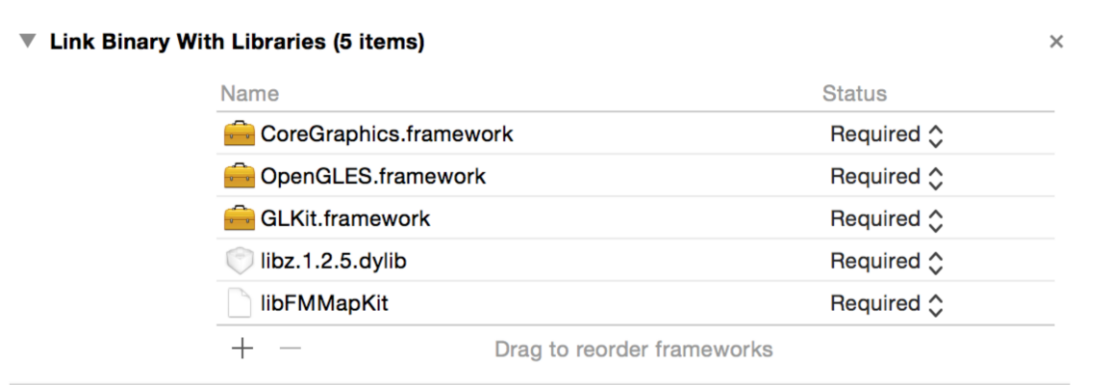
第三种方式：

1. 将 FengMap iOS SDK 的 libs 文件夹拷贝到您的 Application 工程根目录下
2. 在 Xcode 的 Project -> Edit Active Target -> Build -> Linking -> Other Linker Flags 中添加 -ObjC
3. 设置静态库的链接路径，在 Xcode 的 Project -> Edit Active Target -> Build -> Search Path -> Library Search Paths 中添加您的静态库目录，比如"\${SRCROOT}/../libs/Release-\${PLATFORM_NAME}"，\$(SRCROOT)宏代表您的工程文件目录，\$(PLATFORM_NAME)宏代表当前配置是 iOS Device 还是 Simulator。

注：静态库中采用 Objective-C 和 C++ 混合实现，因此需要您保证您工程中至少有一个.mm 后缀的源文件(您可以将任意一个.m 后缀的文件改名为.mm)；

3.引入系统库：

FengMap iOS SDK 使用 OpenGL 渲染，因此您需要在您的 Xcode 工程中引入 CoreGraphics.framework、OpenGL.framework、GLKit.framework。添加方式：在 Xcode 的 Project -> Active Target -> Build Phases -> Link Binary With Libraries，添加即可。



4.TARGETS-Build Settings 设置：

在 TARGETS-Build Settings-Other Linker Flags 中添加如下内容：-ObjC;



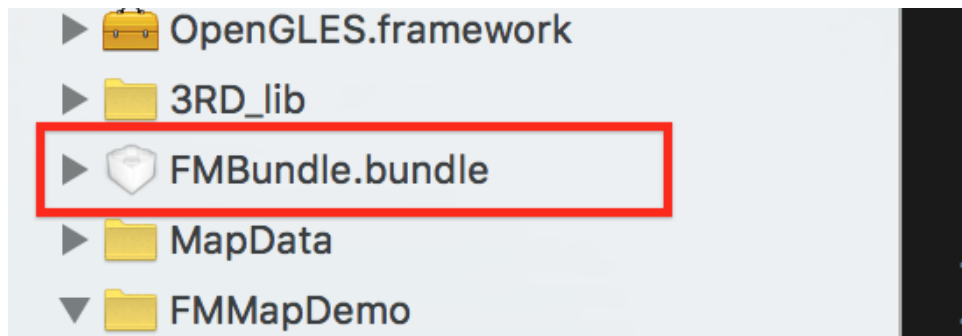
将 TARGETS-Build Settings-C++ Language Dialect 项选择为：GNU++11[-std=gnu++11]

TARGETS-Build Settings-C++ Standard Library 一项改为：

libstdc++(GNU C++ standard library);

5.引入 FMBundle.bundle 资源文件

FMBundle.bundle 中存储了公共设施，定位，指北针及水印等资源图片，还存储了地图的默认主题文件。注意：加载地图会默认设置一套蜂鸟自定义主题，该主题文件不可以删除，否则会导致程序崩溃，如需要添加新的主题，请访问蜂鸟 SDK 官方网站查看主题的获取方式，在程序中更改您的主题。您也可以根据具体需求任意替换该 bundle 中 Resources 文件夹的图片文件。添加方式：将 FMBundle.bundle 拷贝到您的工程目录，直接将该 bundle 文件拖拽至 Xcode 工程左侧的 Groups&Files 中即可。若您需要替换定位、指北针的图标请保留原文件名称，否则不显示替换的新图片。



● Hello FengMap

■ 开发工具

Xcode6.1 及以上版本。

■ 初始化 FengMap SDK

在 AppDelegate.m 加入头文件: #import "FMKMapSDK.h"

在您的 AppDelegate.m 文件中添加如下代码完成对 FengMap SDK 的初始化:

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    FMKMapSDK * sdkManager = [FMKMapSDK shareSDK];
    BOOL ret = [sdkManager start:@"请输入您的密钥" generalDelegate:nil];
    if (!ret) {
        NSLog(@"manager start failed!");
    }
    return YES;
}
```

■ 显示 FengMap

FengMap 地图的显示是由 FMKMapView 完成的, 在类扩展文件中加入如下代码:

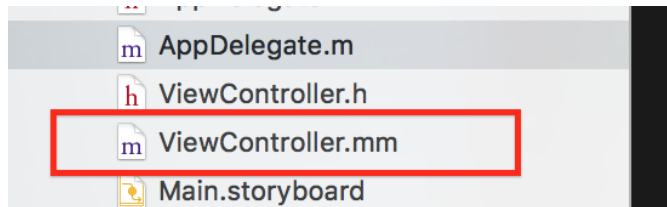
```
#import "ViewController.h"
#import "FMMapKit.h"
```

```

@interface ViewController () <FMKMapViewDelegate>
@property (nonatomic,strong) FMKMapView* mapView;
@end

```

这里需要注意的是，FengMap iOS SDK 是基于 Objective-C 和 C++混合实现的开发工具包，所以凡是引用到底层类的自定义类的.m 文件后缀须改为.mm：



初始化完成之后需要为地图提供一个 ID，此 ID 为对应地图的唯一标识，需要访问 FengMap SDK 官网获取您所需要的信息；用户可通过 SDK 下载类下载地图文件，使用数据类保存文件，详细请参照 FMKMapDataManager,FMKDataDownloader，以下是显示地图代码：

```

self.mapView = [[FMKMapView alloc] initWithFrame:self.view.bounds ID:@"10380" delegate:self];
[self.view addSubview:self.mapView];

```

注：由于iOS9改用更安全的https，为了能够在iOS9中正常使用地图SDK，请在"Info.plist"中进行如下配置，否则影响SDK的使用。

```

<key>NSAppTransportSecurity</key>
<dict>
<key>NSAllowsArbitraryLoads</key>
    <true/>
</dict>

```

运行后效果如下：



● 基础地图

FengMap iOS SDK 提供可视的地图指北针和公司水印，此外地图还提供图层的添加和删除，利用图层的操作可进一步对地图实现元素的添加和删除；

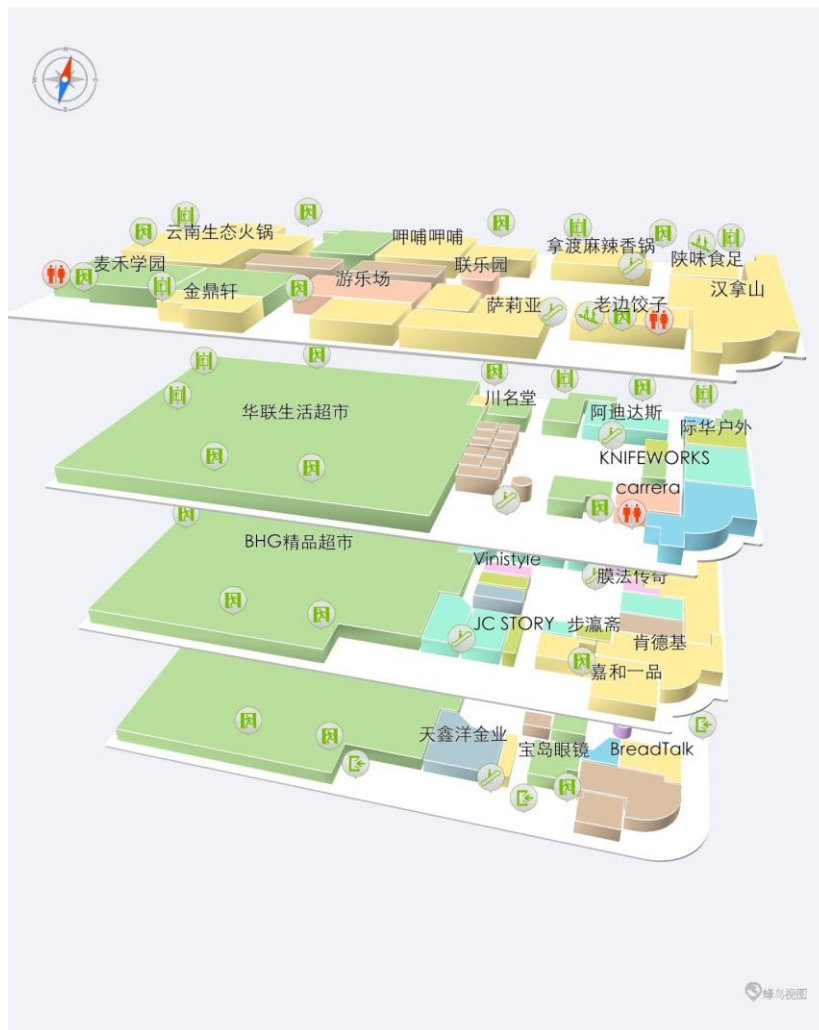
■ 显示设置

1.地图指北针设置

指北针和水印的资源被放在 FMBundle.bundle 中，若要进行修改可直接替换 bundle 中的资源，注意不能修改原来的资源名称。

//显示指北针

```
self.mapView.showCompass = YES;
```



2. 主题设置

主题支持离线加载和在线两种方式：

2.1 加载指定 ID 的主题

```
[self.mapView setThemeWithID:@"1001"];
```

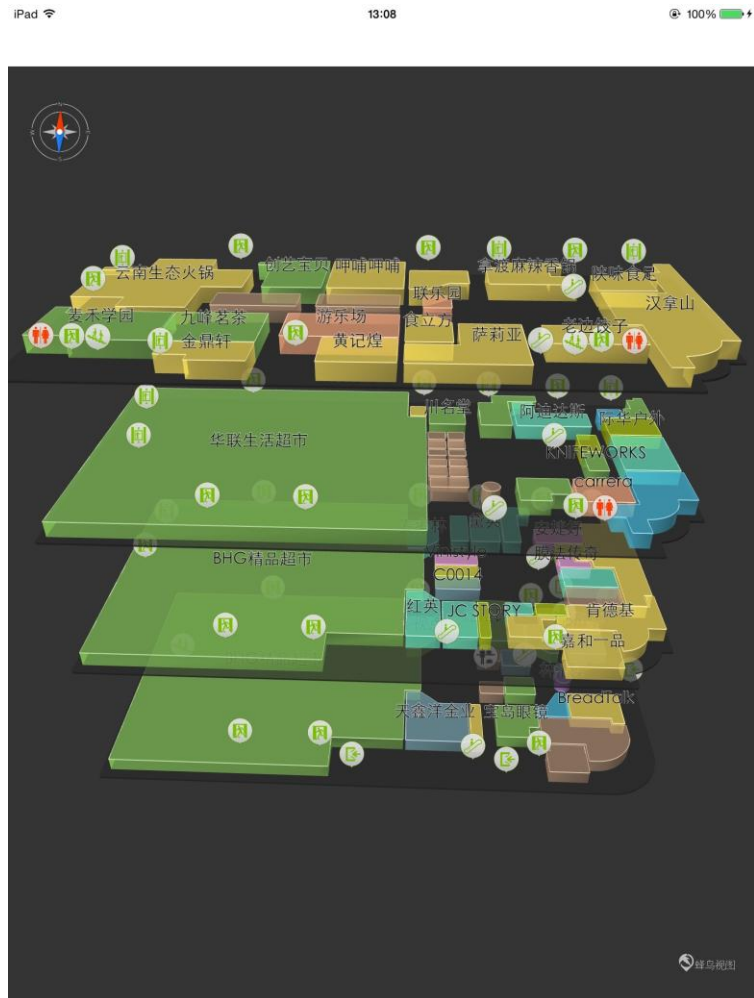
2.2 加载指定路径的主题

//获取地图主题路径

```
NSString* themePath = [MYBUNDLE pathForResource:@"night.theme" ofType:nil];
```

//设置地图主题

```
[self.mapView setThemeWithLocalPath: themePath];
```

3. 多楼层显示设置

用户可根据需要自定义显示的楼层，代码如下：

//获取地图管理节点(楼层节点用于管理地图层的一些操作，另地图的一些信息也可通过该节点获取)

```
FMKMap* map = self.mapView.map;
```

//获取所有楼层ID

```
NSArray* gids = [map info].gids;
```

//设置要显示的楼层

```
self.mapView.displayGids = gids;
```

//设置焦点楼层，焦点楼层会显示为高亮

```
self.mapView.focusGid = gids[2];
```



4.2D/3D 视角转换

用户可自行设置 2D/3D 视角的切换:

//设置视角,地图默认为3D视角

self.mapView.enable3D= NO;



■ 地图事件

实现地图的事件代理方法：

```
/**
```

```
 * 当单击地图时
```

```
 *
```

```
 * @param mapView
```

```
 * @param point 单击点
```

```
 */
```

```
- (void)mapView:(FMKMapView *)mapView didSingleTapWithPoint:(CGPoint)point;
```

```
/**
```

```
 * 当双击地图时
```

```
 *
```

```

* @param mapView
*/
- (void)mapView:(FMKMapView *)mapView didDoubleTapWithPoint:(CGPoint)point;

/**
 * 长按地图时
 *
 * @param mapView
 */
- (void)mapView:(FMKMapView *)mapView didLongPressWithPoint:(CGPoint)point;

/**
 * 当移动地图时
 *
 * @param mapView
 */
- (void)mapView:(FMKMapView *)mapView didMovedWithPoint:(CGPoint)point;

```

在地图的代理类中实现以上方法，可以在地图进行单击，双击，长按和移动时对地图进行操作。

■ 模型图层

通过地图的模型图层可以获取地图上的所有模型，并对其进行操作。每个楼层只有一个模型图层，且模型图层只能通过楼层 ID 获取，不可创建。获取方法如下：

//获取地图管理节点

```
FMKMap* map = self.mapView.map;
```

//获取模型图层管理节点的方式如下

```
FMKModelLayer* modelLayer = [map getModelLayerWithGroupID:@"1"];
```

//模型图层的交互通过代理方式实现，若要进行图层交互，则需要设置图层的代理

```
modelLayer.delegate = self;
```

//设置模型的选中状态

```

- (void)onMapClickNode:(FMKNode *)node inLayer:(FMKLayer *)layer
{
    if ([layer isKindOfClass: [FMKModelLayer class]]) {

```

```

FMKModel* model = (FMKModel *)node;
    model.selected = YES;
}
}

```

模型的选中状态的颜色与主题相关;



■ 公共设施图层

通过地图的公共设施图层可以获取地图上的公共设施，并对其进行操作。公共设施图层的获取方法与模型层的相类似，交互方式也是一样，获取方式如下：

//获取地图管理节点

```
FMKMap* map = self.mapView.map;
```

//获取公共设施图层管理节点的方式如下

```
FMKFacilityLayer* facilityLayer = [map getFacilityLayerWithGroupID:@"1"];
```

//公共设施的交互通过代理方式实现，若需要进行交互，需要设置层的代理

```
facilityLayer.delegate = self;
```

■ 图片标注图层

地图标注图层是用户自定义层，用户在自己的自定义层上使用.png 的图片(图片需放置在工程根目录或其子目录下)对地图点进行标注，标注的地图点可通过地图坐标添加；

添加图片代码：

//获取地图管理节点

```
FMKMap* map = self.mapView.map;
```

//为地图添加图片图层

```
FMKImageLayer* imageLayer = [[FMKImageLayer alloc] initWithGroupID:@"4"];
```

//添加图片层

```
[map addLayer:imageLayer];
```

//图片和坐标

```
UIImage* image = [UIImage imageNamed:@"user"];
```

```
FMKMapPoint point = FMKMapPointMake(12961627.000000, 4861811.500000);
```

//图片标注

```
FMKImageMarker* imageMarker = [[FMKImageMarker alloc] initWithImage:image Coord:point];
```

//添加到层

```
[imageLayer addImageMarker:imageMarker];
```

//删除图片标注

```
[imageLayer removeImageMarker:imageMarker];
```

//删除图片层

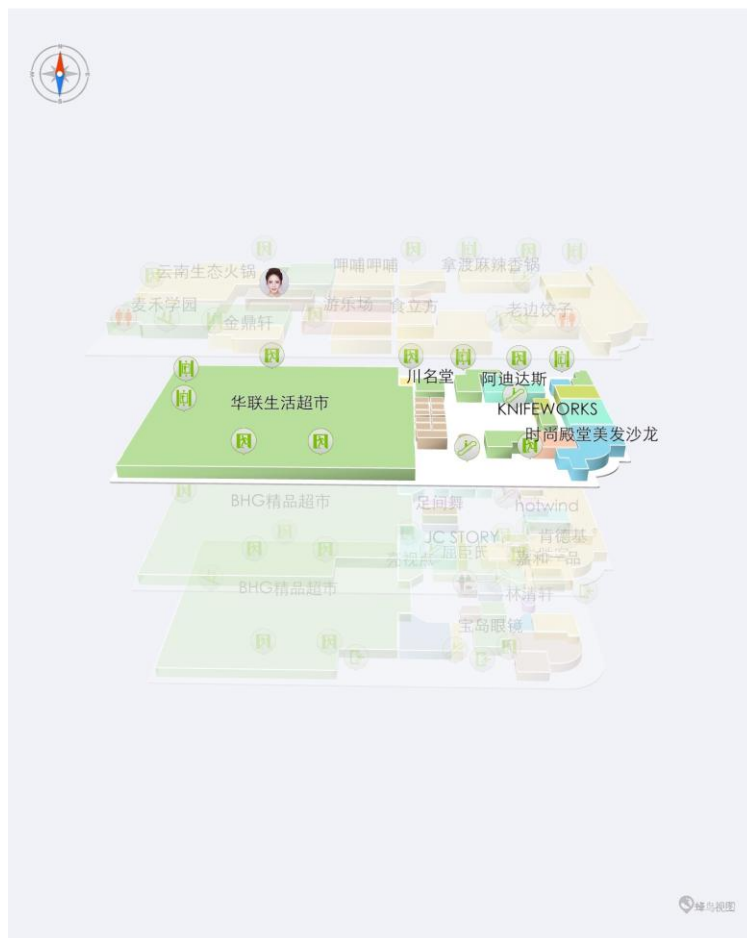
```
[map removeLayer:imageLayer];
```

我们还为用户提供了地图元素的点击交互功能，通过点击地图实现地图上模型的选中、添加图片标注等功能。在添加图片层的类扩展中遵循协议 FMKLayerDelegate,设置图片层的代理。

在图片层的代理类中实现方法：

```
- (void)onMapClickNode:(FMKNode *)node inLayer:(FMKLayer *)layer
{
    if ([layer isKindOfClass:[FMKImageLayer class]]) {

    }
}
```



■ 定位图层

地图的定位图层获取方式与模型图层的相类似，图层获取后可自行添加可定位的点，定位点可添加多个，同时定位点提供位置和方向变化的设置方法；

//获取定位标注层

```
FMKLocationLayer* locationLayer = self.mapView.map.locateLayer;
```

//添加定位标注点

```
FMKLocationMarker* locateMarker = [[FMKLocationMarker alloc]
initWithPointerImageName:@"location" DomelImageName:@"dome"];
[locationLayer addLocationMarker:locateMarker];
```

//改变定位点位置

```
FMKGeoCoord coord = FMKGeoCoordMake(3, FMKMapPointMake(12961595.000000,
4861800.000000));
[locateMarker locateWithGeoCoord:coord];
```

//改变定位点方向

```
[locateMarker updateRotate:(30)%(2*180)];
```

■ 坐标转换

//屏幕坐标转换为地图坐标

```
FMKGeoCoord coord = [self.mapView coverPoint:CGPointMake(120, 100)];
```

//地图坐标转换为屏幕坐标

//地图中绘制元素的Z值的枚举类型。在地图坐标转屏幕坐标时，需要使用这里面的值

```
FMKMapCoordZType type = FMKMAPCOORDZ_EXTENT;
```

```
CGPoint point = [self.mapView coverCoord:coord zType:type];
```

● 路径规划

■ 路径计算

路径规划是通过对起点和终点的描述获取地图信息的过程，此过程需要使用FMNaviAnalyser类进行获取；在调用分析类的头文件或类扩展中遵循协议FMKNaviAnalyserDelegate；SDK支持同层和跨层路径规划。

//添加起始点

```
FMKGeoCoord start,end;
```

```
start = FMKGeoCoordMake(4,FMKMapPointMake(12961663.000000, 4861869.500000));
```

```
end = FMKGeoCoordMake(3,FMKMapPointMake(12961591.000000, 4861822.500000));
```

//路线分析

//创建路径分析对象


```
FMKNaviAnalyser* analyser = [[FMKNaviAnalyser alloc] initWithMapID:mapID];

//路径分析会产生结果，成功的分析才会得到信息

analyser.delegate = self;
RouteCalculateResultType type =
[analyser analyseRouteWithStartCoord:start endCoord:end type:MODULE_SHORTEST ];

//若结果为IROUTE_SUCCESS,则路径规划成功
```

此外，用户还可以通过以下方式，直接进行结果数组的获取，结果中为FMKNaviResult类型的对象：

```
//创建路径分析对象

FMKNaviAnalyser* analyser = [[FMKNaviAnalyser alloc] initWithMapID:mapID];

//设置路径分析代理

analyser.delegate = self;

//路径规划结果数组，内为FMKNaviResult型对象

NSMutableArray* result = [NSMutableArray array];

//进行路径分析

FMKRouteCalculateResultType type;
type = [analyser analyseRouteWithStartCoord:start end:end
type:MODULE_SHORTESTrouteResult:&result];

//若路径计算结果不成功，直接返回

if(type != IROUTE_SUCCESS) return;
```

■ 路径计算结果绘制

```
//利用计算结果添加线

FMKLineMarker* line = [[FMKLineMarker alloc] init];
for (inti=0; i<result.count; i++) {
if ([result[i] isKindOfClass:[FMKNaviResult class]]) {

//计算结果

FMKNaviResult* navi = result[i];

//利用计算结果添加segment

FMKSegment* segment = [[FMKSegment alloc] initWithGroupID:navi.groupID
```

```
pointArray:navi.pointArray];

//在直线中添加segment

    [line addSegment:segment];
}
}
line.width = 3;

//路径绘制线段类型

line.type = FMKLINE_DOTTED;
line.color = [UIColor blackColor];
[self.mapView.map.lineLayer addLineMarker:line];
```



路径分析的结果分单层和跨层，用户根据需要可自行进行数据获取；

//路径结果将代理方法中返回

//同层结果计算，路径坐标集合

- (void)getFloorNaviGeoCoords:(NSArray *)coords;

//同层结果计算，路径长度

- (void)getFloorRouteLength:(double)length;

//跨层结果计算，获取经过楼层的id

```

- (void)getMutilFloorNaviGroupIDs:(NSArray *)gids;

//跨层结果计算，获取各层路径的长度

- (void)getMutilFloorRouteLength:(double)length inFloor:(NSString *)groupID;

//跨层结果计算，返回路径上的点

- (void)getMutilFloorNaviGeoCoords:(NSArray *)coords inFloor:(NSString *)groupID;

//跨层路径总长度

- (void)getMutilFloorTotalLength:(double)length;

```

● 搜索分析

目前只支持模型和公共设施的搜索分析。

■ 模型分析

```

//创建模型搜索分析对象

FMKSearchAnalyser* searchAnalyser = [[FMKSearchAnalyser alloc] initWithMapID:mapID];

//设置搜索分析代理

searchAnalyser.delegate = self;

创建搜索分析请求对象

//创建模型搜索请求对象

FMKModelSearchRequest* modelRequest = [[FMKModelSearchRequest alloc] init];

```

1.FID搜索

```

//设置搜索条件，搜索多个用"|"号分开

modelRequest.fid = @"103800140|103800133";

```

//设置要搜索的楼层，如果不设置则默认为全楼层

```
modelRequest.groupIDs = @[@"1",@"2"];
```

//执行搜索，查询结果将在代理函数中返回

```
[searchAnalyser executeFMKSearchRequestByFid:modelRequest];
```

```
- (void)onModelSearchDone:(FMKModelSearchRequest *)request result:(NSArray *)resultArray{
```

//返回的结果是FMKModelSearchResult的数组

```
NSLog(@"%@",resultArray);
```

```
}
```

2.TYPE查询

//设置搜索条件，搜索多个用"|"号分开

```
modelRequest.type = @"100100|100101";
```

//执行搜索，搜索结果将在代理函数中返回

```
[searchAnalyser executeFMKSearchRequestByKeyType:modelRequest];
```

```
- (void)onModelSearchDone:(FMKModelSearchRequest *)request result:(NSArray *)resultArray{
```

//返回的结果是FMKModelSearchResult的数组

```
NSLog(@"%@",resultArray);
```

```
}
```

3.关键字查询

//设置搜索条件，搜索多个用"|"号分开

```
modelRequest.keyWords = @"k|l";
```

//执行搜索，搜索结果将在代理函数中返回

```
[searchAnalyser executeFMKSearchRequestByKeyWords:modelRequest];
```

//返回model搜索结果方法

```
- (void)onModelSearchDone:(FMKModelSearchRequest *)request result:(NSArray *)resultArray{
```

//返回的结果是FMKModelSearchResult的数组

```
NSLog(@"%@",resultArray);
```

```
}
```

■ 公共设施分析

//创建公共设施搜索请求对象

```
FMKFacilitySearchRequest* facilityRequest = [[FMKFacilitySearchRequest alloc] init];
```

//设置搜索条件，多个查询值中间用"|"隔开

```
facilityRequest.type = @"1000045|1000035";
```

//执行搜索，搜索结果将在代理函数中返回

```
[searchAnalyser executeFMKSearchRequestByType:facilityRequest];
```

//返回公共设施搜索结果方法

```
- (void)onFacilitySearchDone:(FMKFacilitySearchRequest *)request result:(NSArray *)resultArray{
```

//返回的结果是FMKFacilitySearchResult的数组

```
NSLog(@"%@",resultArray);
```

```
}
```

● 数据下载

用户可通过 SDK 网站自行下载地图数据和相关的主题文件，地图数据和主题数据会自动保存在沙盒指定目录；

■ 地图的下载与保存

1. 开发者下载由蜂鸟提供的公共地图数据
开发者到[蜂鸟 SDK 网站](#)获取地图数据。
2. 开发者下载私有数据
开发者登录蜂鸟 SDK 网站后到[我的地图](#)页面下载私有数据。
3. 开发者通过 SDK 接口下载地图数据

//相应的地图ID

```
NSString* mapID = @"10380";
```

//通过地图ID使用地图下载模块下载对应的地图数据

```
[FMKDataDownloader asynRequestMapWithID:mapID finish:^(FMKHttpRequest *request, NSData
```

```

*responseData) {

//保存地图数据

[[FMKMapDataManager sharedInstance] saveMapData:responseDataID:mapID];

} failure:^(NSError *error) {

});

```

■ 主题的下载与保存

1. 开发者自行下载数据

开发者到 [SDK 网站](#) 下载数据，注意保存文件名不变。

2. 使用 SDK 相关接口下载数据

(1) 获取主题文件的 id

开发者到 [SDK 网站](#) 查看主题 id，记录下来。

```

//相应的主题ID
NSString* themeID = @"1001";
//通过主题ID使用主题下载模块下载对应的主题数据
[FMKDataDownloader downloadThemeWithID:themeID finish:^(FMKHttpRequest *request, NSData
*responseData) {

//保存主题数据

[[FMKThemeDataManager sharedInstance] saveThemeData:responseData ID:themeID];

} failure:^(NSError *error) {

});

```