**1、准备开发环境**

使用Android studio。

**2、下载并安装Fengmap Android SDK**

从蜂鸟云平台相关下载中按需下载Fengmap Android SDK开发资源。

Fengmap Android SDK接入步骤参考开发指南中配置工程的详细步骤。

**3、创建Fengmap项目**

按以下步骤新建一个包含地图 Activity 的应用项目：

* 1、启动 Android Studio。
* 2、按以下所述创建新项目:
  + •看到 Welcome to Android Studio 对话框，选择位于对话框右侧“Quick Start”之下的 Start a new Android Studio project。
  + • 否则，请点击 Android Studio 菜单栏中的 File，然后点击 New、New Project。
* 3、输入应用名称、公司域和项目位置。 然后点击 Next。
* 4、选择应用所需的机型。确定自己的需要，然后点击 Next。
* 5、在“Add an activity to Mobile”对话框中选择 Empty Activity。 然后点击 Next。
* 6、按提示输入 Activity 名称、布局名称和标题。 使用默认值即可。 然后点击 Finish。

**4、获取蜂鸟SDK密钥**

首先需要注册成为蜂鸟云用户，登陆成功后在控制台->我的应用中创建应用并添加Android SDK应用密钥。

**5、Hello Fengmap 代码**

**（1）添加密钥（Key）**

在application中添加开发密钥：

<application>

<meta-data android:name="com.fengmap.mapapi.appkey" android:value="开发者 key" />

Java

Copy

**（2）添加所需权限**

如果FMMapSDK使用默认初始化FMMapSDK.init()，不需要读写权限，如果自定义初始化缓存目录FMMapSDK.init(this,path)，需要添加读写权限，添加所需权限，示例如下：

<uses-permission android:name="android.permission.INTERNET"/>

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE"/>

<uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE" />

// 自定义缓存目录添加下面两个

<uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE"/>

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"/>

**（3）在布局xml文件中添加地图控件**

在布局xml文件中添加地图控件：

<com.fengmap.android.map.FMMapView

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:id="@+id/mapview"/>

**（4）初始化 SDK**

在应用程序创建时初始化 SDK引用的Context 全局变量，注意这个MainApplication需要在AndroidManifest.xml里注册：

public class MainApplication extends Application{

@Override

public void onCreate() {

super.onCreate();

//初始化SDK

FMMapSDK.init(this);

// 自定义缓存目录，需要申请读写权限

// FMMapSDK.init(this,path);

}

}

**（5）管理地图**

地图销毁释放底层资源，注意：开发者不能在Activity的onDestory 方法里去销毁地图，原因是底层资源的销毁是异步操作，需要在GL线程里面处理，也就是销毁地图前必须保证GL线程活着，而onDestroy回调执行时并不能保证这一点，大多数情况是此时GL线程已经销毁，底层GLView释放失败，下次再打开地图会出现异常。

public class MainActivity extends Activity {

FMMap mFMMap;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

FMMapView mapView = (FMMapView) findViewById(R.id.mapview);

mFMMap = mapView.getFMMap(); //获取地图操作对象

String bid = "10347"; //地图id

mFMMap.openMapById(bid, true); //打开地图

}

@Override

public void onBackPressed() {

if (mFMMap != null) {

mFMMap.onDestroy();

}

super.onBackPressed();

}

}

**6、连接Android设备**

了解应用实际运行情况的最简单方法是将 Android 设备与计算机相连。 按照说明在Android 设备上启用开发者选项，并配置应用和系统以检测设备。

此外，也可以使用 Android 模拟器来运行应用。 使用 Android Virtual Device (AVD) Manager 来配置一台或多台虚拟设备，可以将设备与 Android 模拟器联用来构建和运行应用。需要的模拟器映像具有的 Google API 平台应基于 Android 4.0 或更高版本。

**7、构建和运行应用**

在 Android Studio 中，点击 Run 菜单选项（或 play 按钮图标）运行应用。

提示您选择设备时，选择下列选项之一：

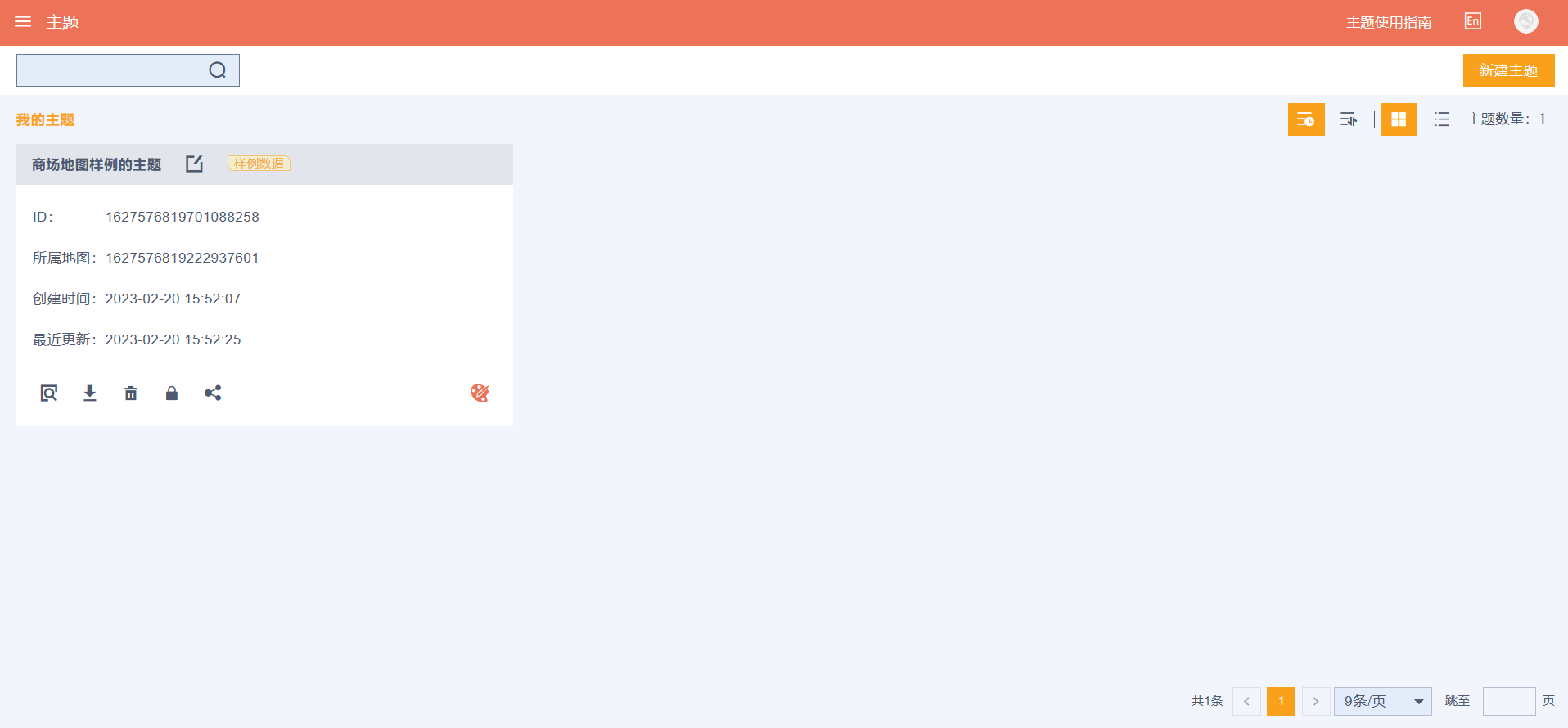
* • 选择与您的计算机相连的 Android 设备
* • 此外，还可以选择 Launch emulator 单选按钮，然后选择您之前配置过的虚拟设备

点击 OK。 Android Studio 将调用 Gradle 构建应用，然后在设备或模拟器上显示结果。应用打开可能需要几分钟的时间。

您可以看到在您的项目里有地图展示。如果未看到地图，那请检查是否联网，因为**蜂鸟SDK密钥需要在应用首次运行时连接网络**，或者检查是否在AndroidManifest.xml中添加了蜂鸟Key。

**（1）地图显示**

在<https://designer.fengmap.com/home>网址中创建地图、主题，并生成相关地图ID、主题ID，创建后如下：



@Override  
protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
 FMMapView mapView = (FMMapView) findViewById(R.id.*mapview*);  
 mFMMap = mapView.getFMMap(); //获取地图操作对象  
 String bid = "1627576819222937601"; //地图id  
 mFMMap.openMapById(bid, true); //打开地图  
 openMapByPath();  
}

*/\*\*  
 \* 加载地图数据  
 \*/*private void openMapByPath() {  
 mMapView = (FMMapView) findViewById(R.id.*mapview*);  
 mFMMap = mMapView.getFMMap();  
 mFMMap.setOnFMMapInitListener(this);  
 //加载离线数据  
 String path = FileUtils.*getDefaultMapPath*(this);  
 mFMMap.openMapByPath(path);  
}

**（2）单/多楼层显示**

*/\*\*  
 \* 初始化多楼层控件位置  
 \*/*private void initMultiFloorControllerComponent() {  
 //创建多楼层显示按钮  
 mMultiFloorBtn = new FMMultiFloorControllerButton(this);  
 //初始状态  
 mMultiFloorBtn.initState(false);  
 mMultiFloorBtn.measure(0, 0);  
 int width = mMultiFloorBtn.getMeasuredWidth();  
 //单/多层楼层切换控件  
 mMapView.addComponent(mMultiFloorBtn, FMDevice.*getDeviceWidth*() - 10 - width, 60);  
 //单、多楼层点击切换  
 mMultiFloorBtn.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 FMTextButton button = (FMTextButton) v;  
 if (button.isSelected()) {  
 button.setSelected(false);  
 setSingleDisplay();  
 } else {  
 button.setSelected(true);  
 setMultiDisplay();  
 }  
 }  
 });  
}  
  
*/\*\*  
 \* 单层显示  
 \*/*private void setSingleDisplay() {  
 int[] gids = {mFMMap.getFocusGroupId()}; //获取当前地图焦点层id  
 mFMMap.setMultiDisplay(gids, 0, null);  
}  
  
*/\*\*  
 \* 多层显示  
 \*/*private void setMultiDisplay() {  
 int[] gids = mFMMap.getMapGroupIds(); //获取地图所有的group  
 int focus = 0;  
 for (int i = 0; i < gids.length; i++) {  
 if (gids[i] == mFMMap.getFocusGroupId()) {  
 focus = i;  
 break;  
 }  
 }  
 mFMMap.setMultiDisplay(gids, focus, null);  
}  
  
*/\*\*  
 \* 切换楼层  
 \*  
 \** ***@param*** *groupId 楼层id  
 \*/*void switchFloor(int groupId) {  
 mFMMap.setFocusByGroupIdAnimated(groupId, new FMLinearInterpolator(), (OnFMSwitchGroupListener) this);  
}

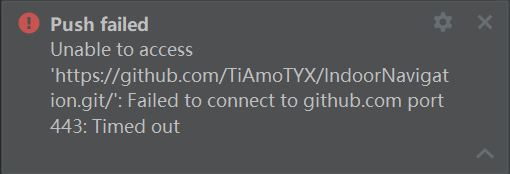
*/\*\*  
 \* 楼层切换控件初始化  
 \*/*private void initFloorControllerComponent() {  
 // 创建楼层切换控件  
 mFloorComponent = new FMFloorControllerComponent(this);  
 mFloorComponent.setMaxItemCount(4);//设置楼层最大显示数量  
 //楼层切换事件监听  
 mFloorComponent.setOnFMFloorControllerComponentListener  
 (new FMFloorControllerComponent.OnFMFloorControllerComponentListener() {  
 @Override  
 public void onSwitchFloorMode(View view, FMFloorControllerComponent.FMFloorMode currentMode) {  
 if (currentMode == FMFloorControllerComponent.FMFloorMode.*SINGLE*) {  
  
 } else {  
  
 }  
 }  
  
 @Override  
 public boolean onItemSelected(int groupId, String floorName) {  
 if (mAnimateEnded) {  
 switchFloor(groupId);  
 return true;  
 }  
 return false;  
 }  
 });  
 //设置为单层模式  
 mFloorComponent.setFloorMode(FMFloorControllerComponent.FMFloorMode.*SINGLE*);  
 int groupId = 1; //设置默认显示楼层id  
 mFloorComponent.setFloorDataFromFMMapInfo(mFMMap.getFMMapInfo(), groupId);  
 mMapView.addComponent(mFloorComponent, 10, 400);//添加楼层控件并设置位置  
}

**（3）导航路线规划显示**

*/\*\*  
 \* 地图销毁调用  
 \*/*@Override  
public void onBackPressed () {  
 if (mFMMap != null) {  
 mFMMap.onDestroy();  
 }  
 super.onBackPressed();  
}  
  
*/\*\*  
 \* 地图加载成功回调事件  
 \*  
 \** ***@param*** *path 地图所在sdcard路径  
 \*/*@Override  
public void onMapInitSuccess(String path) {  
 //加载离线主题文件  
 mFMMap.loadThemeByPath(FileUtils.*getDefaultThemePath*(this));  
 initMultiFloorControllerComponent();  
 if (mFloorComponent == null) {  
 initFloorControllerComponent();  
 }  
 // 创建模拟导航对象  
 mNavigation = new FMSimulateNavigation(mFMMap);  
 // 创建真实导航对象  
 // mNavigation = new FMActualNavigation(mFMMap);  
 // 设置导航文字语种，目前支持中文英文两种("zh"，"en")  
 // mNavigation.setNaviLanguage(this, "en");  
 // 设置是否启用走过导航线变化,默认开启  
 // mNavigation. setNaviAcrossChange(true);  
 // 创建导航配置对象  
 mNaviOption = new FMNaviOption();  
 // 设置跟随模式，默认跟随  
 mNaviOption.setFollowPosition(true);  
 // 设置跟随角度（第一人视角），默认跟随  
 mNaviOption.setFollowAngle(true);  
 // 点移距离视图中心点超过最大距离5米，就会触发移动动画；若设为0，则实时居中  
 mNaviOption.setNeedMoveToCenterMaxDistance(*NAVI\_MOVE\_CENTER\_MAX\_DISTANCE*);  
 // 设置导航开始时的缩放级别，true: 导航结束时恢复开始前的缩放级别，false：保持现状  
 mNaviOption.setZoomLevel(*NAVI\_ZOOM\_LEVEL*, false);  
 // 设置配置  
 mNavigation.setNaviOption(mNaviOption);  
 initNavi();  
}  
  
private void initNavi() {  
 // 创建模拟导航对象  
 mNavigation = new FMSimulateNavigation(mFMMap);  
 // 设置导航文字语种，目前支持中文（CH）英文(EN)两种模式，不区分大小写，默认中文  
 mNavigation.setNaviLanguage(this, "Ch");  
 // 创建模拟导航配置对象  
 mNaviOption = new FMNaviOption();  
 // 设置跟随模式，默认跟随  
 mNaviOption.setFollowPosition(mHasFollowed);  
 // 设置跟随角度（第一人视角），默认跟随  
 mNaviOption.setFollowAngle(mIsFirstView);  
 // 点移距离视图中心点超过最大距离5米，就会触发移动动画；若设为0，则实时居中  
 mNaviOption.setNeedMoveToCenterMaxDistance(*NAVI\_MOVE\_CENTER\_MAX\_DISTANCE*);  
 // 设置导航开始时的缩放级别，true: 导航结束时恢复开始前的缩放级别，false：保持现状  
 mNaviOption.setZoomLevel(*NAVI\_ZOOM\_LEVEL*, false);  
 // 设置配置  
 mNavigation.setNaviOption(mNaviOption);  
 // 设置导航监听接口  
 mNavigation.setOnNavigationListener(this);  
 // 路径规划  
 analyzeNavigation(mStartCoord, mEndCoord);  
 // 总长  
 mTotalDistance = mNavigation.getSceneRouteLength();  
 isMapLoaded = true;  
}  
  
//导航  
void analyzeNavigation(FMGeoCoord startPt, FMGeoCoord endPt) {  
 // 设置起点  
 Bitmap startBmp = BitmapFactory.*decodeResource*(getResources(), R.drawable.*start*);  
 mStartOption.setBitmap(startBmp);  
 mNavigation.setStartPoint(startPt);  
 mNavigation.setStartOption(mStartOption);  
  
 // 设置终点  
 Bitmap endBmp = BitmapFactory.*decodeResource*(getResources(), R.drawable.*end*);  
 mEndOption.setBitmap(endBmp);  
 mNavigation.setEndPoint(endPt);  
 mNavigation.setEndOption(mEndOption);  
  
 // 路径规划  
 int ret = mNavigation.analyseRoute(FMNaviAnalyser.FMNaviModule.*MODULE\_SHORTEST*);  
 if (ret == FMNaviAnalyser.FMRouteCalcuResult.*ROUTE\_SUCCESS*) {  
 mNavigation.drawNaviLine();  
 isRouteCalculated = true;  
 } else {  
 FMLog.*le*("failed",FMNaviAnalyser.FMRouteCalcuResult.*getErrorMsg*(ret));  
 }  
}

**遇到的问题、错误、bug以及解决方案**

Android Studio 中使用可视化界面提交失败，失败报错如下：

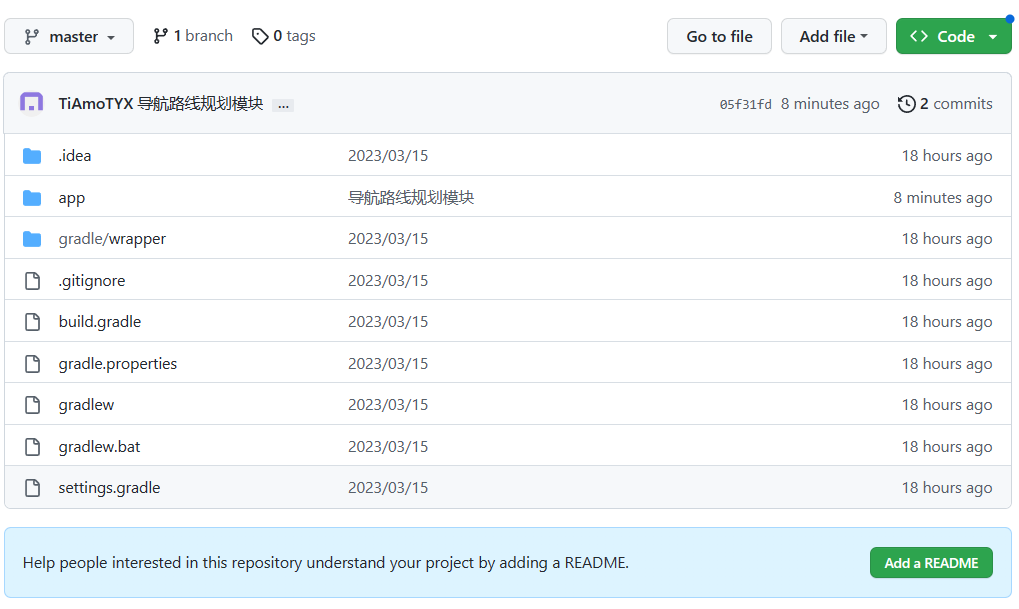


考虑使用 Git 命令提交;

进入 Terminal 面板 , 进入工程根目录 , 执行如下命令 , 即可向 GitHub 提交代码 ;

git push --set-upstream origin master

问题解决，查看Github项目仓库更新



基于Android平台的室内导航系统的设计与实现

设计思路：

1.确定需求和功能：明确室内导航系统需要提供哪些功能，如定位、路径规划、导航等，同时考虑到用户体验和易用性，设计用户界面和交互流程。

2.采集室内地图数据：使用激光扫描或者摄像头等技术，对室内地图进行采集和处理，生成地图数据，包括建筑结构、房间分布、门窗位置、楼层高度等。

3.确定定位方式：在室内环境中，GPS定位信号较弱，因此需要采用其他定位方式，如Wi-Fi信号、蓝牙信号、红外线等。通过定位算法，确定用户当前的位置和方向。

4.实现路径规划和导航：根据用户的起点和终点位置，使用路径规划算法，计算出最优路径，同时考虑到各种限制条件，如楼层高度、走廊长度、门的状态等。在导航过程中，根据用户的位置和方向，实时更新导航路径，并提供语音提示、震动提示等辅助功能。

5.软件开发：根据需求和功能，使用Android平台上的开发工具和语言，如Android Studio、Java等，进行软件开发和编码。

6.测试和优化：在开发完成后，进行测试和优化，发现和解决软件中的问题和bug，提高软件的稳定性和性能。

总之，基于Android平台的室内导航系统需要进行多方面的设计和实现，包括采集地图数据、定位、路径规划、导航等，同时需要考虑到用户体验和易用性，进行软件开发和测试，最终上线和发布。

利用iBeacon开发的室内导航Android App可以通过以下步骤实现：

1. 确定需求和功能：明确室内导航系统需要提供哪些功能，如定位、路径规划、导航等，同时考虑到用户体验和易用性，设计用户界面和交互流程。
2. 购买iBeacon设备：iBeacon设备是一种可以发射蓝牙信号的小型硬件，可以用于室内定位。购买一些iBeacon设备，部署到室内空间中，用于发射信号。
3. 采集室内地图数据：使用激光扫描或者摄像头等技术，对室内地图进行采集和处理，生成地图数据，包括建筑结构、房间分布、门窗位置、楼层高度等。
4. 确定定位方式：通过iBeacon设备发射的蓝牙信号，可以获取用户的位置信息。使用定位算法，确定用户当前的位置和方向。
5. 实现路径规划和导航：根据用户的起点和终点位置，使用路径规划算法，计算出最优路径，同时考虑到各种限制条件，如楼层高度、走廊长度、门的状态等。在导航过程中，根据用户的位置和方向，实时更新导航路径，并提供语音提示、震动提示等辅助功能。
6. 软件开发：根据需求和功能，使用Android平台上的开发工具和语言，如Android Studio、Java等，进行软件开发和编码。通过iBeacon的API，获取蓝牙信号信息，并进行位置定位和路径规划。
7. 测试和优化：在开发完成后，进行测试和优化，发现和解决软件中的问题和bug，提高软件的稳定性和性能。

摘要：

本文介绍了一种基于iBeacon技术开发的Android平台的室内导航系统的设计与实现方法。首先，介绍了iBeacon技术的原理和优势，以及室内导航系统的需求和功能。其次，提出了基于iBeacon的定位和路径规划算法，并采集了室内地图数据，对地图进行三维建模。接着，使用Android平台的开发工具和语言进行软件开发，并实现了路径规划和导航功能。最后，进行了系统测试和优化。该系统具有较高的定位精度和路径规划准确性，能够为用户提供高质量的室内导航服务。