

高德地图 Android 室内定位 SDK 开发指南

V6.9

高德软件有限公司
2017 年 4 月·北京

法律声明

版权所有©2014，高德集团。

保留一切权利。

本文档包含的所有内容除特别声明之外，版权均属于高德集团所有，受《中华人民共和国著作权法》及相关法律法规和中国加入的所有知识产权方面的国际条约的保护。未经本公司书面许可，任何单位和个人不得以任何方式（电子或机械，包括影印）翻印或转载本文档的任何部分，否则将视为侵权，高德集团保留依法追究其法律责任的权利。

高德地图 API 的一切有关权利属于高德集团所有。

本文档并不代表供应商或其代理的承诺，高德集团可在不作任何声明的情况下对本文档内容进行修改。

本文档中所涉及的软件产品及其后续升级产品均由高德集团制作并负责全权销售。

本文档中提到的其它公司及其产品的商标所有权属于该商标的所有者。

高德地图

联系邮箱：api@autonavi.com

技术交流论坛：bbs.amap.com

官方微博：<http://weibo.com/gaodedituapi>

高德地图 API 欢迎用户的任何建议或意见。

目录

1 简介	2
1.1 室内定位 SDK	2
1.2 面向的读者	2
1.3 兼容性	2
1.4 申请 API Key	2
2 工程配置	3
2.1 下载	3
2.2 Android Studio 配置工程	3
2.3 添加用户 Key	4
2.4 添加权限	4
3 配置模块	6
4 定位模块	7
4.1 创建对象	7
4.2 引擎初始化	7
4.3 销毁定位模块	8
4.4 监听/停止监听定位	8
4.5 消息类型	9
4.6 定位结果	11

1 简介

1.1 室内定位 SDK

Android 室内定位 SDK 是一套简单的 LBS 服务的室内定位接口，通过室内定位 SDK 开发者可以迅速为应用程序实现室内定位功能。您可以单独使用室内定位 SDK，也可以结合高德地图 Android 室内地图 SDK。

注意：室内定位 SDK 在使用时需要实时访问服务器，只有网络通讯正常才能正常使用室内定位功能。

1.2 面向的读者

高德地图 Android 室内定位 SDK 是提供给具有一定 Android 编程经验和了解面向对象概念的读者使用的。此外，读者还应该对地图产品有一定的了解。用户在使用中遇到任何问题，都可以通过 QQ 群：300265574 或问答[社区](#)反馈给我们。

1.3 兼容性

支持 Android 4.0 以上系统，蓝牙定位需 Android 4.3 以上系统。

1.4 申请 API Key

为保证服务可以正常使用，您需要注册成为开发者并申请 Key。每个帐户，最多可以申请 10 个 Key。申请流程请参照[获取密钥](#)。

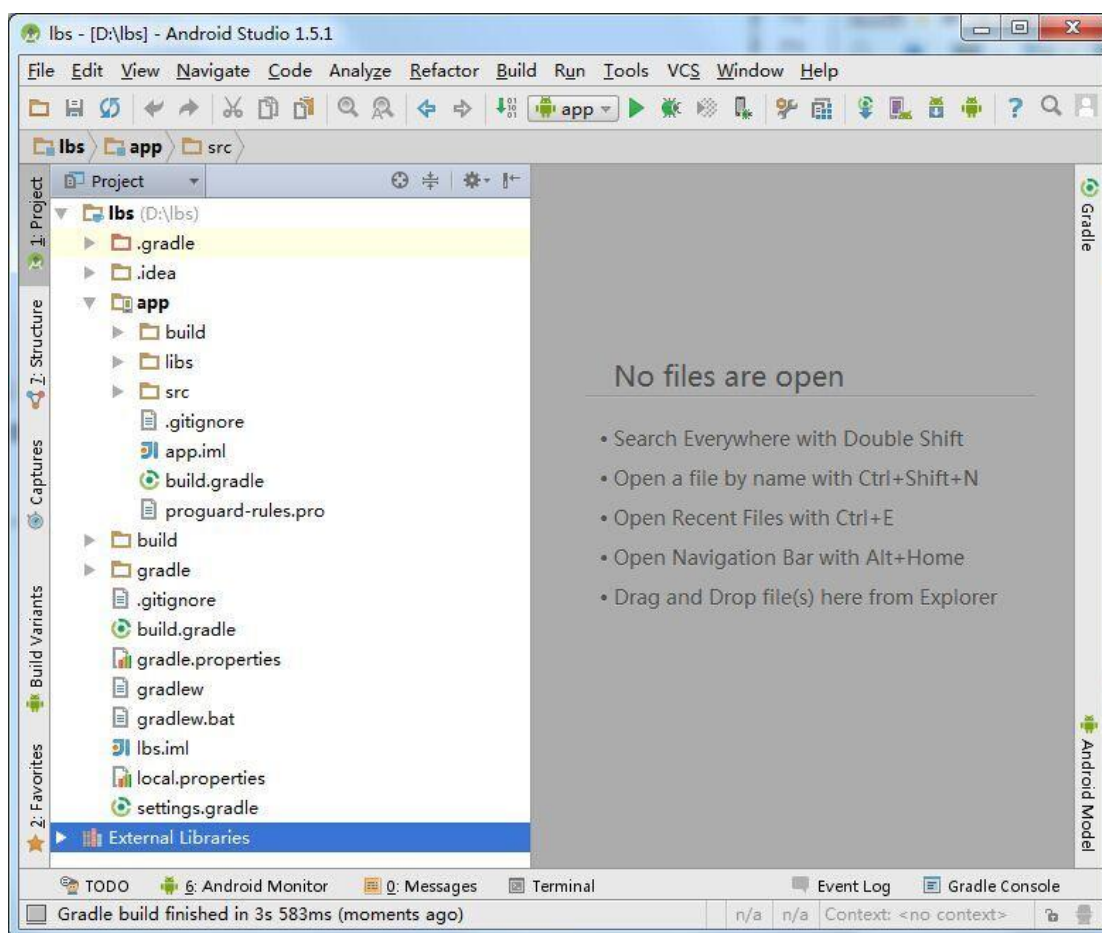
2 工程配置

2.1 下载

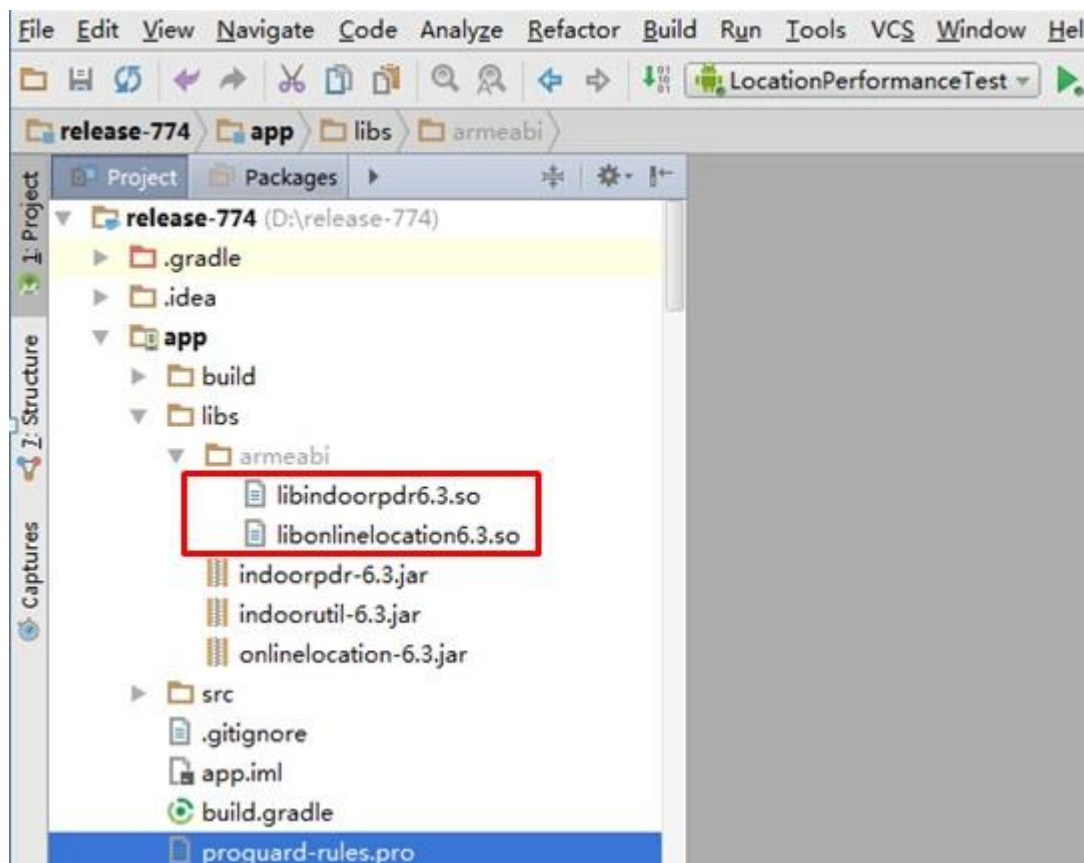
从“[相关下载](#)”页面下载并解压得到 3 个定位包 indoorutil.jar、indoorpdr.jar、onlinelocation.jar,分别是工具包、PDR 包和在线定位包。同时还包括 libindoorpdr6.3.so、libonlinelocation6.3.so 共 2 个 so 库。其中版本号可能跟实际下载略有不同。

2.2 Android Studio 配置工程

1. 新建工程如图所示：



2. 将下载的开发包解压，拷贝到 app.libs 下面即可，如图所示：



2.3 添加用户 Key

添加您的用户 Key。示例代码如下：

```
Configuration.Builder mConfigBuilder= new Configuration.Builder(context);  
mConfigBuilder.setLBSParam("用户的 key");
```

2.4 添加权限

添加权限，请直接拷贝。

```
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />  
<uses-permission android:name="android.permission.BLUETOOTH" />  
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />  
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />  
<uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE" />  
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />  
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />  
<uses-permission android:name="android.permission.READ_PHONE_STATE" />  
<uses-permission android:name="android.permission.INTERNET" />  
<uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS  
" />  
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

```
<uses-feature android:name="android.hardware.bluetooth_le" android:required="true" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
```

部分手机系统对 app 权限做了额外限制，导致开发的 app 不能正常定位，这时候需要用户自己在手机的应用程序设置里面放开相应的权限。

3 配置模块

运用定位引擎实现室内定位，就必须提前设置好一些定位需要的参数，必须在定位最开始的时候设置进去，一旦设置之后配置将不能再修改，直到用户销毁定位模块。

配置模块使用 Builder 模式用来构建，首先创建 Configuration.Builder 的对象，然后调用对象的一系列 set 方法设置各种参数，最后调用 build() 方法构造最后的配置对象，设置给定位引擎。

客户端需要关注的设置选项有：

函数名	参数类型	说明
setLBSParam	String	设置访问服务时需要的参数 KEY
setLocationProvider	LocationProvider	选择是蓝牙/还是 WIFI ,【WIFI BLE】 【FUSION_WIFI_BLE】 暂不支持

示例代码：

```
Configuration.Builder mConfigBuilder=new Configuration.Builder(context);  
LocationManager.getInstance().init(mConfigBuilder.build());
```

用户也可以如上直接创建对象，SDK 会直接提供一组相关的默认值。此外，在开发过程中，如果开发者所处环境不支持室内定位，那么我们可以通过模拟定位方式来测试开发应用的定位效果。要使用模拟定位，首先需要从 Android 室内定位 SDK 的“相关下载”页面下载模拟定位的日志文件（地址为：<http://lbs.amap.com/api/android-indoorlocation-sdk/download/>），保存在手机上，然后把详细路径设置到配置对象的 public String mSimulateFile 成员上，这样，再次启动定位的时候，SDK 将使用模拟文件进行测试。

说明：模拟定位日志文件，暂提供“北京太阳宫凯德 mall（B000A8WAWL）”的 WIFI 和蓝牙两种定位方式日志文件，以供模拟调试使用。

4 定位模块

定位模块根据客户端采集的信息进行定位服务。

4.1 创建对象

只需要把 indoorutil.jar、indoorpdr.jar 和 onlinelocation.jar 几个 jar 包引入进来，同时包含 libindoorpdr.so 和 libonlinelocation.so 两个 so 库即可。在创建定位引擎的时候使用 onlinelocation.jar 中的 LocationManager.getInstance()。

4.2 引擎初始化

函数体：

```
public void init(String bid, final Configuration config, final Handler initHandler);
```

功能诠释：

初始化的室内定位模块。

参数诠释：

参数	数据类型	用处
bid	String	要进行的室内定位建筑，如果是纯在线定位这个参数可以为空，为空的时候引擎做粗定位，成功之后会把建筑 bid 返回。如果不为空则表示在本建筑之内进行室内定位；如果不是
config	Configuration	配置信息，如上一节所述。
initHandler	Handler	初始化结果 handler，初始化过程中的一些错误和处理结果都通过这个回调函数返回过来。可能的消息有：MessageCode.MSG_THREAD_PREPARED、MSG_WIFI_NOT_ENABLED 等，具体的错误消息参考其他章节。

用法备注：

示例代码：

```
ILocationManager mLocationManager;  
// 获取到实例  
mLocationManager = LocationManager.getInstance();  
SDKInitHandler mSDKInitHandler = new SDKInitHandler(this);
```

```
// 初始化定位引擎
mLocationManager.init(this.strBuildNameId, mConfigBuilder.build(), mSDKInitHandle
r);
```

这里需要注意的是，程序初始化的结果不是直接返回，而是异步的通过 initHandler 反馈给客户端。所以一般需要在 initHandler 中收到了 MessageCode.MSG_THREAD_PREPARED 消息之后再进行其他的操作，比如注册定位结果监听等操作。在初始化失败的情况下，SDK 也会通过这种方式把错误消息反馈给客户端，此时会影响后面的定位结果。

4.3 销毁定位模块

函数体：

```
public void destroy();
```

功能诠释：

释放所有定位相关资源，并停止、销毁室内定位引擎。

用法备注：

销毁之后，室内定位引擎将不能再被使用，用户需要重新初始化定位引擎。程序最后退出的时候，或者用户需要修改定位模块配置信息的时候需要调用此函数。

4.4 监听/停止监听定位

函数体：

```
public void requestLocationUpdates(Handler handler);
```

```
public void removeUpdates(Handler handler)
```

功能诠释：

注册或删除定位监听，参数 handler 是回调接口，定位引擎发送定位结果、定位错误等信息到这个 handler。在第一次添加监听的时候，引擎会自动启动定位服务，当最后一个监听被删除之后，引擎将自动停止定位模块。

示例代码：

设置监听：

```
mLocationManager.requestLocationUpdates(mInnerHandler);
```

取消监听：

```
mLocationManager.removeUpdates(mInnerHandler);
```

4.5 消息类型

类 `MessageCode` 中定义了定位引擎所需要的所有消息类型。主要的消息有：

消息名	含义
<code>MSG_THREAD_PREPARED</code>	定位模块初始化完成
<code>MSG_REPORT_ONLINE_LOCATION</code>	在线定位成功返回。此时 <code>msg.obj</code> 是一个 <code>LocationResult</code> 对象
<code>MSG_SENSOR_MISSING</code>	手机缺少步导需要的传感器：加速度、磁力计、重力计等
<code>MSG_WIFI_NOT_ENABLED</code>	没有打开 wifi
<code>MSG_BLE_NOT_ENABLED</code>	没有打开蓝牙
<code>MSG_WIFI_NOT_PERMITTED</code>	wifi 没有授权
<code>MSG_BLE_NOT_PERMITTED</code>	蓝牙没有授权
<code>MSG_BLE_NO_SCAN</code>	一段时间内没有蓝牙扫描
<code>MSG_WIFI_NO_SCAN</code>	一段时间内没有 WIFI 扫描
<code>MSG_NETWORK_ERROR</code>	网络通讯失败
<code>MSG_SERVER_ERROR</code>	服务器端联接失败
<code>MSG_PRESSURE_CHANGED</code>	气压计有变化。此时 <code>msg.obj</code> 是一个 <code>PressData</code> 对象
<code>MSG_REPORT_PED</code>	步数，方向返回值。此时 <code>msg.obj</code> 是一个 <code>PedData</code> 对象
<code>MSG_LBS_ERROR</code>	用户 key 非法或过期

示例代码：

```
// 创建 Handler 对象，处理回调消息
public void handleMessage(Message msg)
{
    switch (msg.what) {
        // 获取到了在线定位结果
        case MessageCode.MSG_REPORT_ONLINE_LOCATION: {
            LocationResult result = (LocationResult)msg.obj;

            //result 中保存了定位结果 xyz
            break;
        }
        // 传感器检测到错误
        case MessageCode.MSG_SENSOR_MISSING: {
            AlertDialog.show(mParent, "MSG_SENSOR_MISSING",
"手机缺少步导需要的传感器：加速度、磁力计、重力计等");
            break;
        }
    }
}
```

```
    }
    // 蓝牙检测到错误
    case MessageCode.MSG_BLE_NO_SCAN: {
        AlertDialog.show(mParent, "MSG_BLE_NO_SCAN", "一
段时间内没有蓝牙扫描");

        break;
    }
    // wifi 检测到错误
    case MessageCode.MSG_WIFI_NO_SCAN: {
        AlertDialog.show(mParent, "MSG_WIFI_NO_SCAN", "
一段时间内没有 WIFI 扫描");

        break;
    }
    // 网路错误
    case MessageCode.MSG_NETWORK_ERROR: {
        AlertDialog.show(mParent, "MSG_NETWORK_ERROR", "
网络错误");

        break;
    }
    case MessageCode.MSG_SERVER_ERROR: {
        AlertDialog.show(mParent, "MSG_SERVER_ERROR", "
服务器端错误");

        break;
    }
    // 气压计变化
    case MessageCode.MSG_PRESSURE_CHANGED: {
        PressData pressure = (PressData) msg.obj;
        L.d("MSG_PRESSURE_CHANGED, press:" + pressur
e.mPress);

        break;
    }
    // 步导模块返回
    case MessageCode.MSG_REPORT_PED: {
        PedData pedData = (PedData) msg.obj;
        L.d("MSG_REPORT_PED, step:" + pedData.mStep +
" angle:" + pedData.mAngle);

        break;
    }
}
}
```

4.6 定位结果

定位成功之后，引擎发送消息 MSG_REPORT_LOCATION 或 MSG_REPORT_ONLINE_LOCATION 给上层 其中定位结果以 LocationResult 对象方式保存在 msg.obj 中 具体 LocationResult 的成员主要有：

参数	数据类型	说明
x	double	经度
y	double	纬度
z	int	楼层
r	float	精度，单位米
a	float	角度，单位度(0-360)