LSslam SDK Android 开发者指南

V1.0.0

深圳市镭神智能系统有限公司

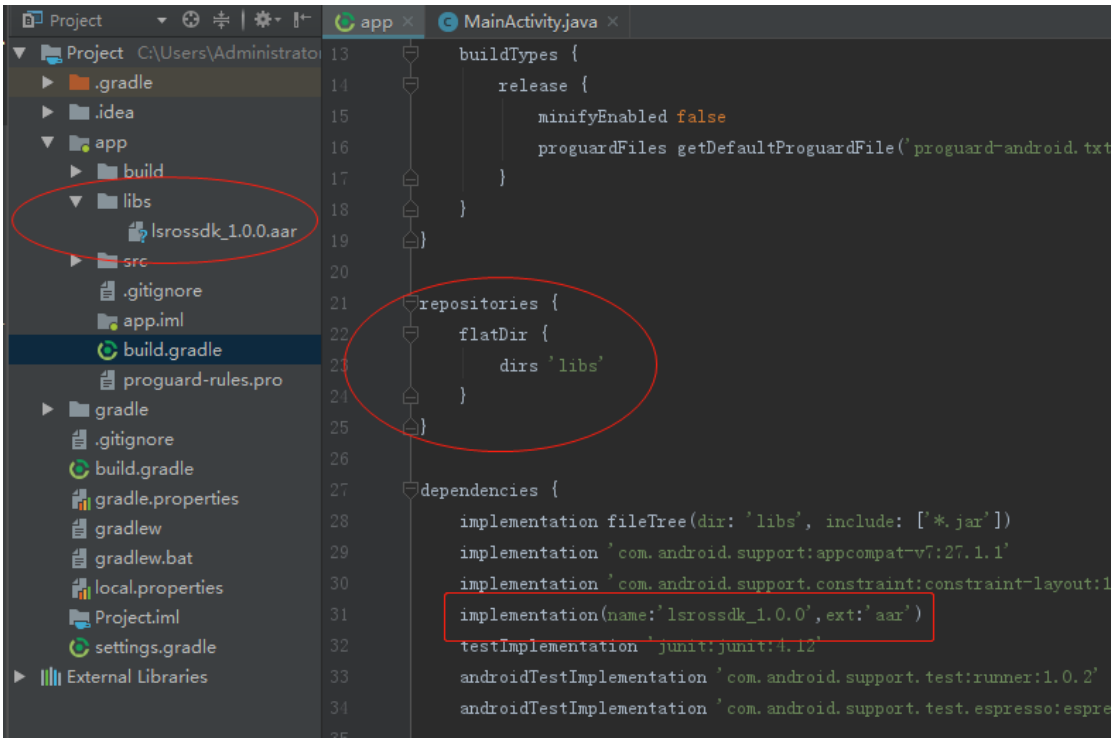
目录

1. Android Studio 配置工程

1.1 添加aar文件

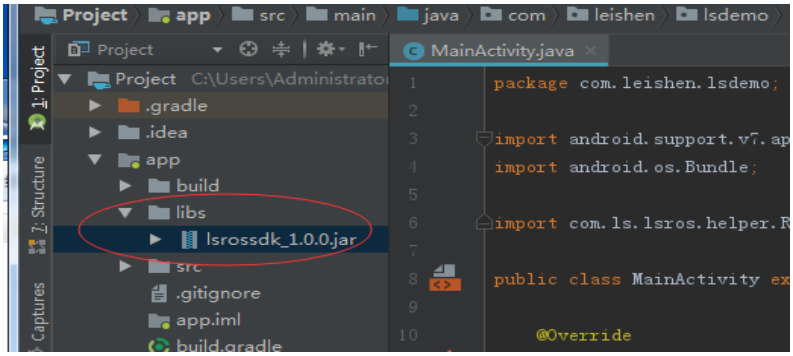
ROS SDK 提供了 jar 文件和 aar 文件,可选择其中之一集成；如选择 aar 文件则需把 aar 包复制到工程的 lib 目录下并在本地引用 aar 文件。如下图所示：

把 aar 文件复制到 libs 目录下，并在 app 的 build.gradle 中添加如下内容。



1.2 添加jar文件

ROS SDK 提供了 jar 文件和 aar 文件,可选择其中之一集成；如选择 jar 文件则需把 jar 包复制到工程的 lib 目录下并同步 gradle 来引用 jar 文件，同步后如下图。



2. SDK初始化

首先对 SDK 组件进行初始化，推荐创建项目的 Application，调用 RosSDKInitHelper 类 init 方法进行初始化。

```
public class MyApplication extends Application {
    @Override
    public void onCreate() {
        // 初始化 SDK
        RosSDKInitHelper.init(this);
        super.onCreate();
    }
}
```

3. 连接服务器

管理连接服务器对应的类为 ROSConnectHelper，调用 getInstance 函数得到其单例对象。

注意：SDK 的所有业务均需连接到服务器后才可以使用。

调用 connect 函数连接服务器；其中函数第一个参数为服务器 ip，第二个参数为端口号，第三个参数为连接服务器的状态回调 ROSClient.ConnectionStatusListener 对象。如下代码示例：

```
ROSConnectHelper.getInstance().connect("192.168.1.200", 8080, new
ROSClient.ConnectionStatusListener() {

    @Override
    public void onConnect() {
        Log.i(TAG, "连接至机器人服务器成功！");
    }

    @Override
    public void onDisconnect(boolean normal, String reason, int code) {
        Log.e(TAG, "断开机器人服务器连接");
    }

    @Override
    public void onError(Exception ex) {
        ex.printStackTrace();
        Log.e(TAG, "与机器人服务器连接失败");
    }
});
```

4. 机器人运动控制

管理机器人运动对应的类为 RobotMovingHelper，可调用 getInstance 函数得到其单例对象。

调用对应接口函数控制机器人移动；机器人运动控制包括向前、向后、左转、右转、左前方、左后方、右前方、右后方、停止等，具体可参考 API 文档；对应函数传入的参数有线速度和角速度。如下代码示例：

```
// 向前移动，参数值范围为 0.0 到 0.5
RobotMovingHelper.getInstance().moveForward((float) 0.4);

// 向后移动，参数值为负数，参数范围为 -0.5 到 0.0
RobotMovingHelper.getInstance().moveBackward((float)-0.4);

// 右转，线速度传入 0.0，对应传入角速度即可；角速度参数范围 -0.5 到 0.0
RobotMovingHelper.getInstance().turnRightFront((float)0.0, angle);

// 左转，线速度传入 0.0，对应传入角速度即可；角速度参数范围 0.0 到 0.5
RobotMovingHelper.getInstance().turnLeftFront((float)0.0, angle);

// 向左前方前进，参数对应为线速度与角速度
RobotMovingHelper.getInstance().turnLeftFront(0.2f, 0.2f);

// 向左后方后退，参数对应为线速度与角速度
RobotMovingHelper.getInstance().turnLeftRear(-0.2f, -0.2f);

// 向右前方前进，参数对应为线速度与角速度
RobotMovingHelper.getInstance().turnRightFront(0.2f, -0.2f);

// 向右后方后退，参数对应为线速度与角速度
RobotMovingHelper.getInstance().turnRightRear(-0.2f, 0.2f);
```

5. 机器人导航

管理机器人导航对应的类为 RobotMovingHelper，可调用 getInstance 函数得到其单例对象。

注意：开始导航前要先导入地图后导航才会生效，导入地图参考 6. 地图操作，另外导航接口的回调函数在子线程，如需更新 UI 要回到主线程更新。

5.1 设置初始位置

调用 setInitPose 函数设置导航初始位置，位置坐标为世界坐标，方法对应的参数为：x 值、y 值、yaw 航向角；

```
// 设置导航初始位置
RobotNavigationHelper.getInstance().setInitPose(0.161087304354, -1.66296613216,
0.996426557807);
```

5.2 设置导航目标位置

调用 sendGoal 函数设置导航目标位置，位置坐标为世界坐标，方法对应的参数为：x 值、y 值、yaw 航向角；

```
// 设置导航目标位置
RobotNavigationHelper.getInstance().sendGoal(1.161087304354, 1.67296613216,
0.996426557807);
```

5.3 开始导航

调用 startNav 函数开始导航，函数对应的参数为回调 CallBack<NavigationResult>对象，从回调函数返回的 NavigationResult 对象可得到开启导航的结果 success 及返回码 code；

```
// 开始导航
RobotNavigationHelper.getInstance().startNav(new CallBack<NavigationResult>() {
    @Override
    public void call(NavigationResult data) {
        Log.i(TAG, "开始导航-->" + data);
    }
});
```

5.4 结束导航

调用 stopNav 函数结束导航，函数对应的参数为回调 CallBack<NavigationResult>对象，从回调函数返回的 NavigationResult 对象可得到结束导航的结果 success 及返回码 code。

```
// 结束导航
RobotNavigationHelper.getInstance().stopNav(new CallBack<NavigationResult>() {
    @Override
    public void call(NavigationResult data) {
        Log.i(TAG, "结束导航-->" + data);
    }
});
```

6. 地图操作

地图操作对应的类为 RobotMapOperationHelper，可调用 getInstance 函数得到其单例对象。

注意：地图操作接口的回调函数在子线程，如需更新 UI 要回到主线程更新。

6.1 开始创建地图

调用 startMap 函数开始创建地图，函数对应的参数为回调 CallBack<MapOperationResult>对象，从回调函数返回的 MapOperationResult 对象可得到请求创建地图的结果 success 及返回码 code。

```
// 开始创建地图
RobotMapOperationHelper.getInstance().startMap(new
CallBack<MapOperationResult>() {
    @Override
    public void call(MapOperationResult data) {
        Log.i(TAG, "请求创建地图结果-->" + data);
    }
});
```

6.2 结束创建地图

调用 endMap 函数停止创建地图，函数对应的三个参数为：isSave 为是否需要保存地图、saveUrl 为保存路径（为算法板上路径），如果需要保存地图则该参数必须填写、callback 为回调函数。

```
// 结束创建地图
```

```
RobotMapOperationHelper.getInstance().endMap(true, "/home/fu/testmap", new
CallBack<MapOperationResult>() {
    @Override
    public void call(MapOperationResult data) {
        Log.i(TAG, "请求结束创建地图结果-->" + data);
    }
});
```

6.3 导入地图

调用 importMap 函数导入地图，函数对应的两个参数为：url 为导入地图路径的 URL，一般为算法板上的路径、callback 为回调对象。

```
// 导入地图
RobotMapOperationHelper.getInstance().importMap("/home/fu/testmap", new
CallBack<MapOperationResult>() {
    @Override
    public void call(MapOperationResult data) {
        Log.i(TAG, "导入地图--->" + data);
    }
});
```

6.4 获取地图信息

调用 getMap 函数获取地图信息，函数参数 callback 为回调对象；从回调函数返回的 ImageInfo 对象包含了地图的信息，如地图对象 bitmap、地图高度、地图宽度等。

注意：获取地图信息接口为订阅模式，回调函数会多次执行上报，如不需频繁获取可调用停止获取地图信息接口。

```
// 获取地图信息
RobotMapOperationHelper.getInstance().getMap(new CallBack<ImageInfo>() {
    @Override
    public void call(final ImageInfo data) {
        Log.i(TAG, "请求获取地图结果--->" + data);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                imageView.setImageBitmap(data.getBitmap());
            }
        });
    }
});
```

6.5 停止获取地图信息

调用 stopGetMap 函数停止获取地图信息，调用此函数之后表示不再接收地图信息。

```
// 停止获取地图信息
RobotMapOperationHelper.getInstance().stopGetMap();
```

7. 获取机器人本体信息

获取机器人本体信息的类为 RobotInfoHelper，可调用 getInstance 函数得到其单例对象。

注意：获取机器人信息接口的回调函数在子线程，如需更新 UI 要回到主线程更新。

7.1 获取机器人导航状态

调用 getNaviStatus 函数获取导航状态，函数对应的两个参数为：throttleRate，指订阅消息发送的时间间隔(ms)、callBack 为回调对象；从回调函数中返回的 RobotStatusResult 对象的 status 属性可获得导航状态。

```
// 获取机器人导航状态
RobotInfoHelper.getInstance().getNaviStatus(2 * 1000, new
CallBack<RobotStatusResult>() {
    @Override
    public void call(RobotStatusResult data) {
        Log.e(TAG, "获取机器人导航状态--->" + data);
    }
});
```

7.2 停止获取机器人导航状态

调用 stopGetNaviStatus 函数停止获取导航状态，调用该函数后将不再收到推送的导航状态信息。

```
// 停止获取机器人导航状态
RobotInfoHelper.getInstance().stopGetNaviStatus();
```

7.3 获取机器人位置信息

调用 getRobotPose 函数获取位置信息，函数对应的两个参数为：throttleRate，指订阅消息发送的时间间隔(ms)、callBack 为回调对象；从回调函数中返回的 RobotPoseResult 对象的 Position 与 Orientation 属性即可获得相应的位置信息。

```
// 获取机器人的位置信息
RobotInfoHelper.getInstance().getRobotPose(2 * 1000, new
CallBack<RobotPoseResult>() {
    @Override
    public void call(RobotPoseResult data) {
        Log.e(TAG, "获取机器人位置信息--->" + data);
    }
});
```

7.4 停止获取机器人的位置信息

调用 stopGetRobotPose 函数停止获取机器人位置，调用该函数后将不再收到推送的位置信息。

```
// 停止获取机器人的位置信息
RobotInfoHelper.getInstance().stopGetRobotPose();
```

7.5 获取机器人传感器数据

调用 getRobotSensor 函数获取传感器数据，函数对应的两个参数为：throttleRate，指消息发送的时间间隔(ms)、callBack 为回调对象；从回调函数中返回的 RobotSensorResult 对象的 rightEncoder、leftEncoder、analogInputs 属性可分别获得右编码、左编码及超声波数据。

```
// 获取机器人传感器数据
RobotInfoHelper.getInstance().getRobotSensor(2 * 1000, new
CallBack<RobotSensorResult>() {
    @Override
    public void call(RobotSensorResult data) {
        Log.e(TAG, "获取机器人传感器信息--->" + data);
    }
});
```

7.6 停止获取机器人传感器数据

调用 stopGetRobotSensor 函数停止获取传感器数据，调用该函数后将不再收到推送的传感器数据。

```
// 停止获取传感器数据
RobotInfoHelper.getInstance().stopGetRobotSensor();
```

8. 工具类说明

8.1 欧拉角与四元数互相转换

（1）在 QEUtils 工具类中，函数 eulerAngle2Quaternion（EulerAngle eulerAngle）欧拉角转换至四元数；根据欧拉角返回四元数，计算公式采用 3D 笛卡尔坐标系。

(2) 在 QEUtils 工具类中，函数 quaternion2EulerAngle（Quaternion quaternion）四元数转换至欧拉角；根据四元数计算返回欧拉角，计算公式采用 3D 笛卡尔坐标系。

8.2 像素点坐标和世界坐标互相转换

(1) 在 CoordinatesUtils 工具类中，函数 world2Pix()将世界坐标点转换至像素坐标点；根据传入的世界坐标点，图片宽度、图片高度、计算像素坐标点。

（2）在 CoordinatesUtils 工具类中，函数 pix2World ()将像素坐标点转换至世界坐标点；根据传入的像素坐标点，图片宽度、图片高度、计算世界坐标点。

9. 修订历史

版本号	修订日期	修订内容	修订人	备注
V1.0.0	2018.06.12	初始版本	Huang	