

Vispek Android SDK 使用说明

本文档说明如何在 Android App 中集成 Vispek

SDK，并通过蓝牙连接设备、读取设备信息和获取光谱数据。若需要完整示例，请参考 android_SDK/VispekSDKDemo。

1. SDK 包内容

```
android_SDK
├── sdklibrary-release.aar # Android SDK AAR
├── vispekSDK.pdf # PDF
├── Android_SDK_ .md # Markdown
├── Bluetooth_Communication_Protocol_CN.xlsx #
├── VispekSDKDemo # Android
├── app
├── libs/sdklibrary-release.aar # Demo AAR
└── src/main/java/com/vispek/sdkdemo
```

核心示例文件：

- VispekSDKDemo/app/build.gradle: AAR 和 Demo 依赖配置。
- VispekSDKDemo/app/src/main/AndroidManifest.xml: 蓝牙和定位权限声明。
- VispekSDKDemo/app/src/main/java/com/vispek/sdkdemo/MyApplication.java: 初始化 BluetoothClient。
- VispekSDKDemo/app/src/main/java/com/vispek/sdkdemo/MainActivity.java: 扫描、连接、发送指令和解析回调。

2. 环境要求

当前 Demo 使用以下配置：

- Android Studio
- Gradle Wrapper: gradle-6.5-bin
- Android Gradle Plugin: 4.1.0
- compileSdkVersion 30
- minSdkVersion 23
- Java 8

客户项目可以使用自己的 Gradle 和 Android Gradle Plugin 版本。若 targetSdkVersion 升级到 31 或以上，需要额外适配 Android 12+ 蓝牙运行时权限。

3. 集成 AAR

3.1 拷贝 AAR

将 SDK 包复制到客户项目：

```
app/libs/sdklibrary-release.aar
```

3.2 添加依赖

在 App 模块的 build.gradle 中添加：

```
dependencies {
    implementation files('libs/sdklibrary-release.aar')
}
```

Demo 中还使用了以下 UI 和权限辅助库，客户可根据自己的 UI 实现选择是否复用：

```
implementation 'androidx.appcompat:appcompat:1.2.0'
implementation 'com.google.android.material:material:1.2.1'
implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
implementation 'com.github.li-xiaojun:XPopup:2.7.5'
implementation 'com.github.CymChad:BaseRecyclerViewAdapterHelper:2.9.44'
implementation 'pub.devrel:easypermissions:3.0.0'
```

如果继续使用 Demo 的第三方依赖，请确认项目仓库配置中包含 `google()` 和 `maven { url 'https://jitpack.io' }`。

4. 权限配置

4.1 Android 11 及以下

在 `AndroidManifest.xml` 中声明：

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

运行时需要动态申请定位权限。BLE 扫描在部分 Android 版本中依赖定位权限和系统定位开关，否则可能搜索不到设备。

4.2 Android 12 及以上

如果客户项目设置 `targetSdkVersion >= 31`，建议同时声明并动态申请 Android 12+ 蓝牙权限：

```
<uses-permission android:name="android.permission.BLUETOOTH_SCAN" />
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
```

注意：

- 使用 Android 12+ 权限时，项目通常需要 `compileSdkVersion >= 31`。
- 如果仍需兼容 Android 11 及以下，保留旧版蓝牙权限和定位权限。
- 如果扫描不到设备，请同时检查蓝牙开关、定位开关、运行时权限和设备是否已被其它 App 连接。

5. 初始化 SDK

建议在 `Application` 中初始化 `BluetoothClient`：

```
public class MyApplication extends Application {
    public static Context applicationContext;
    public static BluetoothClient mClient;

    @Override
    public void onCreate() {
        super.onCreate();
        applicationContext = getApplicationContext();
        initClient(applicationContext);
    }

    public BluetoothClient initClient(Context context) {
        if (mClient == null) {
            mClient = new BluetoothClient(context);
        }
        return mClient;
    }
}
```

在 `AndroidManifest.xml` 的 `application` 节点中配置：

```
<application
    android:name=".MyApplication"
    ...>
</application>
```

6. 扫描蓝牙设备

扫描前建议检查：

- 系统蓝牙已开启。
- 系统定位开关已开启。
- App 已获得蓝牙和定位相关运行时权限。

示例：

```
private void searchDevice() {
    SearchRequest request = new SearchRequest.Builder()
        .searchBluetoothLeDevice(5000, 2)
        .build();

    MyApplication.mClient.search(request, mSearchResponse);
}
```

扫描回调：

```
private final SearchResponse mSearchResponse = new SearchResponse() {
    @Override
    public void onSearchStarted() {
        // loading
    }

    @Override
    public void onDeviceFounded(SearchResult device) {
        // device.getName() device.getAddress()
    }

    @Override
    public void onSearchStopped() {
        //
    }

    @Override
    public void onSearchCanceled() {
        //
    }
};
```

7. 连接和断开设备

客户选择设备后，保存蓝牙地址和名称：

```
blueToothAddress = device.getAddress();
blueToothName = device.getName();
```

连接示例：

```

private void connectDeviceIfNeeded(String address) {
    if (mConnected) {
        return;
    }

    if (MyApplication.mClient.getConnectStatus(address) == STATUS_DEVICE_CONNECTED) {
        Toast.makeText(this, "                ", Toast.LENGTH_SHORT).show();
        return;
    }

    MyApplication.mClient.registerConnectStatusListener(address, mConnectStatusListener);

    BleConnectOptions options = new BleConnectOptions.Builder()
        .setConnectRetry(3)
        .setConnectTimeout(20000)
        .setServiceDiscoverRetry(3)
        .setServiceDiscoverTimeout(10000)
        .build();

    MyApplication.mClient.connect(address, options, new BleConnectResponse() {
        @Override
        public void onResponse(int code, BleGattProfile profile) {
            if (code == REQUEST_SUCCESS) {
                //                mConnectStatusListener
            }
        }
    });
}

```

连接状态回调：

```

private final BleConnectStatusListener mConnectStatusListener = new
BleConnectStatusListener() {
    @Override
    public void onConnectStatusChanged(String mac, int status) {
        mConnected = (status == STATUS_CONNECTED);

        if (mConnected) {
            registerDeviceDataCallback();
        }
    }
};

```

断开连接：

```

private void disconnectBlueTooth() {
    if (!TextUtils.isEmpty(blueToothAddress)) {
        MyApplication.mClient.disconnect(blueToothAddress);
        MyApplication.mClient.refreshCache(blueToothAddress);
    }
}

```

页面销毁时建议停止搜索、断开连接并注销监听：

```

@Override
protected void onPause() {
    super.onPause();
    MyApplication.mClient.stopSearch();
}

@Override
protected void onDestroy() {
    disconnectBlueTooth();
    MyApplication.mClient.unregisterConnectStatusListener(blueToothAddress,
        mConnectStatusListener);
    super.onDestroy();
}

```

8. 注册数据回调

设备连接成功后，创建 `CodeResponse` 并注册数据回调：

```

private void registerDeviceDataCallback() {
    CodeResponse codeResponse = new CodeResponse(MyApplication.mClient, blueToothAddress);

    codeResponse.registCodeResponseListener((mark, data) -> {
        switch (mark) {
            case CodeResponse.CODE_RESPONSE_DEVICE_NO:
                //
                break;
            case CodeResponse.CODE_RESPONSE_ELEC:
                //
                break;
            case CodeResponse.CODE_RESPONSE_TEMPERATURE:
                //
                break;
            case CodeResponse.CODE_RESPONSE_VERSION:
                //
                break;
            case CodeResponse.CODE_RESPONSE_SCAN:
                //          1,1,1...
                break;
            case CodeResponse.CODE_RESPONSE_SCAN_REFERENCE:
                //          h,h,h...
                break;
            case CodeResponse.CODE_RESPONSE_SCAN_ONE_BY_ONE:
                //
                break;
        }
    });
}

```

9. 发送设备指令

发送前必须确认设备已连接：

```

private void sendCodeIfNeeded(byte[] code) {
    if (!mConnected) {
        Toast.makeText(this, "          ", Toast.LENGTH_SHORT).show();
        return;
    }

    if (MyApplication.mClient != null && !TextUtils.isEmpty(blueToothAddress)) {
        SendCodeUtils.sendCode(MyApplication.mClient, blueToothAddress, code,
            bleWriteResponse);
    }
}

```

常用指令：

```

//
sendCodeIfNeeded(SendCodeUtils.getDeviceSNCode());

//
sendCodeIfNeeded(SendCodeUtils.getTemCode());

//
sendCodeIfNeeded(SendCodeUtils.getElecCode());

//
sendCodeIfNeeded(SendCodeUtils.getDeviceVersionCode());

//
sendCodeIfNeeded(SendCodeUtils.getScanResultCode(blueToothName));

//
sendCodeIfNeeded(SendCodeUtils.getScanResultCodeOneByOne(blueToothName, count));

```

发送结果回调：

```
private final BleWriteResponse bleWriteResponse = new BleWriteResponse() {
@Override
public void onResponse(int code) {
if (code == REQUEST_SUCCESS) {
// CodeResponse
} else {
//
}
}
};
```

10. 光谱数据规则

SDK 回调中会分别返回两组光谱值。接口名沿用 SDK 现有命名，但按协议含义应理解为高/低电压光谱值：

- `CodeResponse.CODE_RESPONSE_SCAN`：低电压光谱值，示例格式 `l,l,l,...`
- `CodeResponse.CODE_RESPONSE_SCAN_REFERENCE`：高电压光谱值，示例格式 `h,h,h,...`

如需查看字节级协议字段、命令帧和 CRC 规则，请参考本 SDK 包内的 `Bluetooth_Communication_Protocol_CN.xlsx`。

如果客户不调用 Vispek 后台接口，而是自建算法平台：

- 可按自建算法平台的输入要求自行组织高电压和低电压光谱值。
- 建议在算法侧明确两组数组与 LED/通道序号的对应关系，便于后续计算和数据追溯。

如果客户调用 Vispek 后台接口：

- 必须按 Vispek 指定顺序拼接完整光谱数据提交。
- 指定顺序为“低电压光谱值在前，高电压光谱值在后”：`l1,l2,...,l22,h1,h2,...,h22`。

11. 常见问题

11.1 搜索不到设备

优先检查：

- App 是否获得蓝牙和定位权限。
- 系统蓝牙开关是否打开。
- 系统定位开关是否打开。
- 设备是否通电并处于可连接状态。
- 设备是否已被其它 App 或系统连接。
- Android 12+ 项目是否申请了 `BLUETOOTH_SCAN` 和 `BLUETOOTH_CONNECT`。

11.2 连接失败或连接后马上断开

建议检查：

- 是否使用了正确的 `device.getAddress()`。
- 是否在连接前注册了 `BleConnectStatusListener`。
- 是否重复连接同一设备。
- 是否在页面切换或后台时调用了 `stopSearch()` 或 `disconnect()`。

11.3 指令发送成功但没有数据

建议检查：

- 是否已经连接成功，并且 `mConnected == true`。
- 是否在连接成功后注册了 `CodeResponse`。

- 是否使用了正确的蓝牙地址 `blueToothAddress`。
- 光谱指令是否传入了正确的 `blueToothName`。

12. AI 接入速查

给其它 AI 或开发助手的快速规则：

- Android SDK 包路径： `android_SDK/sdklibrary-release.aar`
- Demo AAR 路径： `android_SDK/VispekSDKDemo/app/libs/sdklibrary-release.aar`
- SDK 主要包名： `com.vispec.sdklibrary`
- Demo 包名： `com.vispec.sdkdemo`
- 初始化入口： `MyApplication.mClient = new BluetoothClient(context)`
- 扫描入口： `MyApplication.mClient.search(request, mSearchResponse)`
- 连接入口： `MyApplication.mClient.connect(address, options, response)`
- 连接状态： `BleConnectStatusListener.onConnectStatusChanged`
- 发送指令： `SendCodeUtils.sendCode(...)`
- 数据回调： `CodeResponse.registCodeResponseListener(...)`
- 蓝牙协议表： `android_SDK/Bluetooth_Communication_Protocol_CN.xlsx`

AI 修改或生成客户接入代码时必须保留以下顺序：

- 声明并动态申请权限。
- 初始化 `BluetoothClient`。
- 检查蓝牙和定位开关。
- 扫描设备并保存 `SearchResult.getAddress()` 与 `SearchResult.getName()`。
- 注册连接状态监听。
- 连接设备。
- 连接成功后注册 `CodeResponse`。
- 使用 `SendCodeUtils` 生成指令并发送。
- 在 `CodeResponse` 中解析设备返回值。
- 页面退出时停止扫描、断开连接并注销监听。

不要做的事：

- 不要修改或重新打包 `sdklibrary-release.aar`。
- 不要跳过 SDK 的 `SendCodeUtils`，直接手写蓝牙协议指令，除非客户明确提供新协议。
- 不要把 `BleWriteResponse` 当作设备数据回调，它只表示指令写入结果。
- 不要改动光谱数据拼接规则。
- 不要只声明 Manifest 权限而忘记运行时权限申请。