

**ChipletRing APP SDK说明书**

|  |  |  |  |
| --- | --- | --- | --- |
| 文档版本号： | 1.1.2 | 文档编号： |  |
| 文档密级： |  | 归属部门： | Chiplet研发部 |
| 产品名： | BCL\_603M | 归属项目： |  |
| 编写人： | 郑玉虎 | 编写日期： | 2023.10.19 |

**无锡勇芯科技有限公司 版权所有**

**修订记录：**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本号** | **修订人** | **修订日期** | **修订描述** |
| 1.0.0 | 郑玉虎 | 2023.10.19 | 修改导航目录 |
| 1.0.1 | 郑玉虎 | 2023.10.26 | 修改集成和初始化SDK的描述，修改部分演示代码 |
| 1.0.2 | 郑玉虎 | 2023.10.31 | 补全代码，加入问题解答 |
| 1.0.3 | 郑玉虎 | 2023.11.23 | 更新问题解答，修改读取数据的参数描述 |
| 1.0.4 | 郑玉虎 | 2023.11.29 | 新增Q&A，作为后续简单问题汇总地 |
| 1.1.1 | 郑玉虎 | 2023.12.26 | 1.新增蓝牙扫描时的筛选方法  2.测量心率、测量血氧增加上传数据 |
| 1.1.2 | 郑玉虎 | 2024.01.02 | 增加波形图数据分析 |
| 1.1.3 | 郑玉虎 | 2024.04.19 | 增加心率血氧PPG原始数据解析 |

**目 录**

[一、 文档简介 5](#_Toc10101)

[1、 文档目的 5](#_Toc16332)

[2、 适用范围 5](#_Toc4417)

[3、 功能介绍 5](#_Toc26402)

[二、 快速入门概览 6](#_Toc31153)

[1、 前置条件 6](#_Toc27811)

[2、 使用流程 6](#_Toc7772)

[3、 流程图 7](#_Toc2609)

[三、 集成ChipletRing APP SDK 8](#_Toc8612)

[1、 集成ChipletRing APP SDK 8](#_Toc28833)

[1.1 集成方式 8](#_Toc4381)

[1.1.1 获取到ChipletRing APP SDK的jar包文件 8](#_Toc24321)

[1.1.2 将jar包放在libs目录下 8](#_Toc8250)

[1.1.3 右键设置为Add as Library(添加为类库) 8](#_Toc28948)

[1.1.4 配置所需权限 ，如需存储以及其他权限可自行配置 ，牵扯到动态权限处 ，需要做相关处理 8](#_Toc16743)

[2、 初始化ChipletRing APP SDK 9](#_Toc18045)

[2.1.1 在Application的onCreate方法中进行初始化 9](#_Toc22778)

[2.1.2 在BaseActivity类中 ，启用监听 ，该监听用于监听蓝牙连接状态以及戒指基础指令反馈的数据 9](#_Toc28788)

[3、 使用ChipletRing APP SDK 10](#_Toc24737)

[3.1 蓝牙操作（BLEUtils） 10](#_Toc16894)

[3.1.1 搜索设备 10](#_Toc18223)

[3.1.2 停止搜索 11](#_Toc16118)

[3.1.3 连接设备 11](#_Toc30718)

[3.1.4 断开蓝牙 12](#_Toc2168)

[3.2 指令功能（LmAPI） 12](#_Toc3653)

[3.2.1 同步时间 12](#_Toc22879)

[3.2.2 读取时间 12](#_Toc7170)

[3.2.3 版本信息 13](#_Toc31734)

[3.2.4 电池电量 13](#_Toc26997)

[3.2.5 读取步数 14](#_Toc10774)

[3.2.6 清除步数 14](#_Toc31042)

[3.2.7 恢复出厂设置 15](#_Toc4931)

[3.2.8 采集周期设置 15](#_Toc16037)

[3.2.9 采集周期读取 15](#_Toc24702)

[3.2.10 测量心率 16](#_Toc7787)

[3.2.11 测量血氧 17](#_Toc3050)

[3.2.12 测量温度 18](#_Toc22665)

[3.2.13 历史记录管理 19](#_Toc21122)

[3.2.14 清空历史数据 20](#_Toc10117)

[3.3 固件升级（OTA） 21](#_Toc30817)

[3.3.1 检查版本 21](#_Toc3087)

[3.3.2 开始升级 22](#_Toc7282)

[3.4 数据库相关（DataApi） 23](#_Toc11897)

[3.4.1 查询历史记录 23](#_Toc32386)

[3.4.2 清空历史数据 24](#_Toc19706)

[3.5 逻辑算法相关（LogicalApi） 25](#_Toc29453)

[3.5.1 计算距离、卡路里 25](#_Toc9425)

[3.5.2 计算睡眠数据 25](#_Toc23098)

[四、 其他 27](#_Toc30156)

[1、 筛选相关 27](#_Toc23519)

[1.1 广播数据单元 28](#_Toc31424)

[1.1.1 Raw data: 28](#_Toc30853)

[1.1.2 Details: 29](#_Toc18154)

[1.2 应用 29](#_Toc16333)

[2、 可能会遇到的问题 29](#_Toc14178)

[2.1 版本相关 30](#_Toc16026)

[2.2 Gradle 4.0以上导致Xpopup无法使用 30](#_Toc24827)

[2.3 不需要界面 30](#_Toc3054)

[2.4 OTA类引用未找到 30](#_Toc26180)

[3、 硬件算法逻辑或固件相关 31](#_Toc20834)

[3.1 戒指相关 31](#_Toc16017)

[3.2 算法相关 31](#_Toc21660)

[4、 Q&A 31](#_Toc32295)

# 文档简介

## 文档目的

为方便Android端APP与戒指通讯进行二次开发 ，特对通讯协议进行封装 ，以达到简洁明了 ，让开发者不需要关注与戒指通讯层 ，专注业务逻辑交互层面开发。

## 适用范围

本SDK基于安卓原生开发 ，最终提供为jar包库。可用于Android环境下使用。

## 功能介绍

|  |  |  |
| --- | --- | --- |
| 功能模块 | 说明 | 相关文档 |
| 蓝牙基础模块 | * 蓝牙的开关操作 * 蓝牙的搜索链接操作 * 蓝牙的数据写入监听操作 |  |
| 通讯协议模块 | * 时间管理 * 版本号管理 * 电池管理 * 心率测量 * 血氧测量 * 温度测量 * 计步管理 * 历史记录管理 * 系统设置 * 日志管理 |  |

# 快速入门概览

## 前置条件

* Android系统环境 ，系统版本>=5.0
* 必须支持蓝牙5.0
* 使用语言须可以调用jar包

## 使用流程

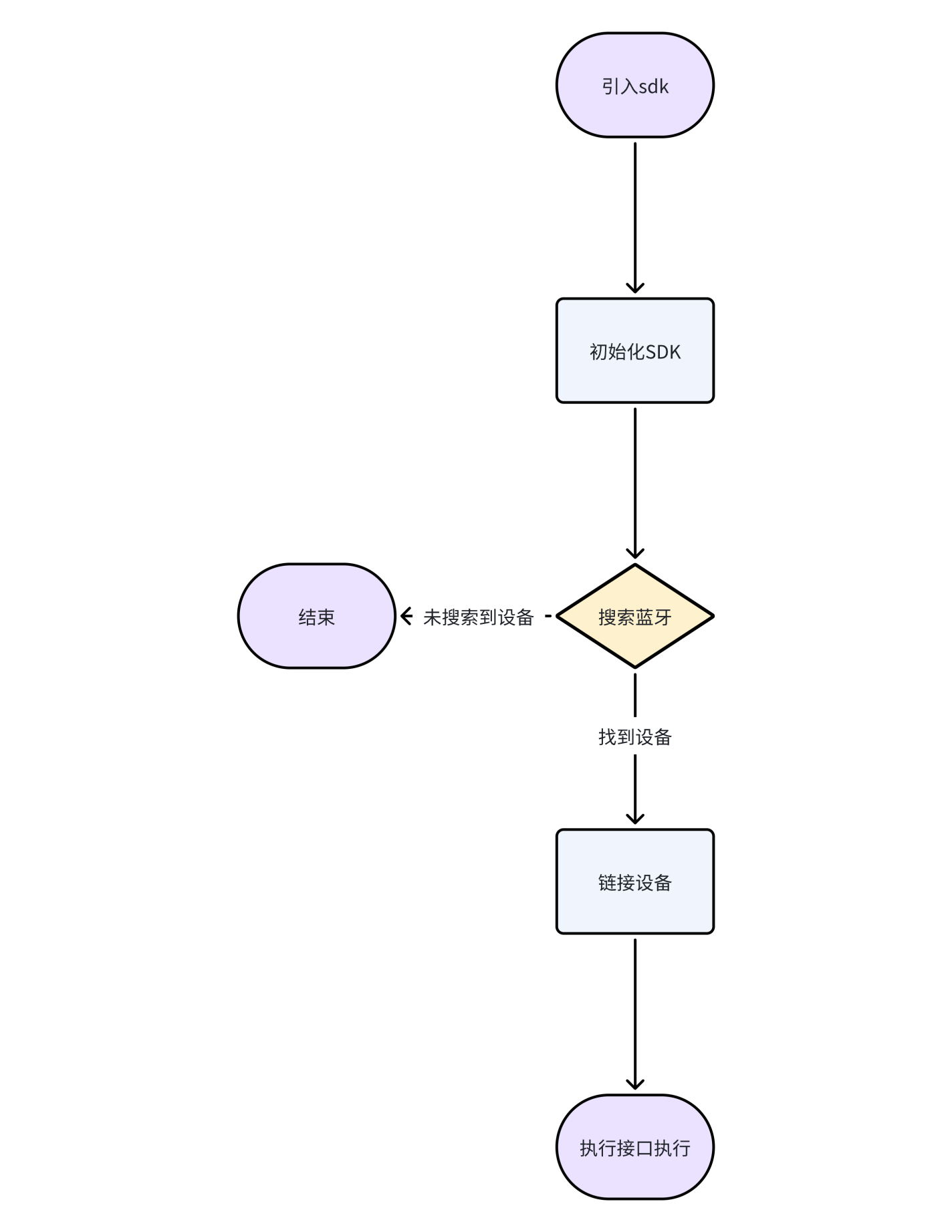
按以下流程使用 SDK：

第—步：集成 SDK

第二步：初始化 SDK

第三步：使用 SDK

## 流程图



# 集成ChipletRing APP SDK

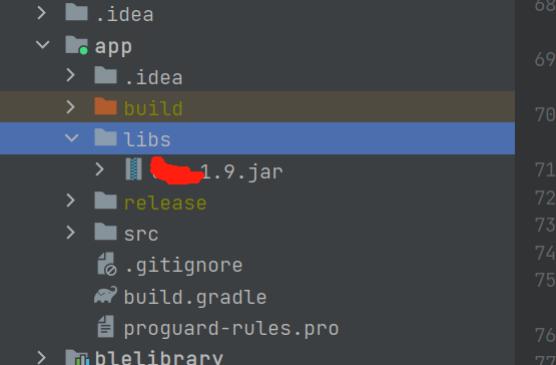
## 集成ChipletRing APP SDK

### 集成方式

#### 获取到ChipletRing APP SDK的jar包文件

#### 将jar包放在libs目录下

#### 右键设置为Add as Library(添加为类库)



#### **配置所需权限 ，如需存储以及其他权限可自行配置 ，牵扯到动态权限**处 ，需要做相关处理

在Manifest.xml中加入以下代码

|  |
| --- |
| 1. <uses-permission android:name="android.permission.BLUETOOTH\_ADVERTISE" /> 2. <uses-permission android:name="android.permission.BLUETOOTH\_CONNECT" /> 3. <uses-permission android:name="android.permission.BLUETOOTH\_SCAN" /> 4. <uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" /> 5. <uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" /> |

## 初始化ChipletRing APP SDK

#### 在Application的onCreate方法中进行初始化

|  |
| --- |
| 1. LmAPI.init(this); 2. LmAPI.setDebug(true); |

#### 在BaseActivity类中 ，启用监听 ，该监听用于监听蓝牙连接状态以及戒指基础指令反馈的数据

|  |
| --- |
| 1. LmAPI.addWLSCmdListener(this, this); 2. *// 监视蓝牙设备与APP连接的状态* 3. IntentFilter intentFilter = new IntentFilter(); 4. intentFilter.addAction(BluetoothAdapter.ACTION\_STATE\_CHANGED); 5. intentFilter.addAction(BluetoothDevice.ACTION\_ACL\_DISCONNECTED); 6. intentFilter.addAction(BluetoothDevice.ACTION\_ACL\_CONNECTED); 7. registerReceiver(broadcastReceiver,intentFilter); 8. *//使用蓝牙之前，先申请去权限* 9. if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.S) { 10. if (!checkPermissions(new String[]{Manifest.permission.ACCESS\_COARSE\_LOCATION, Manifest.permission.ACCESS\_FINE\_LOCATION, Manifest.permission.BLUETOOTH\_SCAN, Manifest.permission.BLUETOOTH\_CONNECT, Manifest.permission.BLUETOOTH\_ADVERTISE})) { 11. new XPopup.Builder(this).asConfirm(getRsString(R.string.hint), getString(R.string.localtion\_auth), 12. new OnConfirmListener() { 13. @Override 14. public void onConfirm() { 15. requestPermission(new String[]{Manifest.permission.ACCESS\_COARSE\_LOCATION, Manifest.permission.ACCESS\_FINE\_LOCATION, Manifest.permission.BLUETOOTH\_SCAN, Manifest.permission.BLUETOOTH\_CONNECT, Manifest.permission.BLUETOOTH\_ADVERTISE}, 100); 16. } 17. }).show(); 18. return; 19. } 20. } else { 21. if (!checkPermissions(new String[]{Manifest.permission.ACCESS\_COARSE\_LOCATION, Manifest.permission.ACCESS\_FINE\_LOCATION})) { 22. new XPopup.Builder(this).asConfirm(getRsString(R.string.hint), getString(R.string.localtion\_auth), 23. new OnConfirmListener() { 24. @Override 25. public void onConfirm() { 26. requestPermission(new String[]{Manifest.permission.ACCESS\_COARSE\_LOCATION, Manifest.permission.ACCESS\_FINE\_LOCATION}, 100); 27. } 28. }).show(); 29. return; 30. } 31. } |

## 使用ChipletRing APP SDK

### 蓝牙操作（BLEUtils）

此类是使用蓝牙搜索、连接、 断开的公共类 ，统— 由IResponseListener接口反馈。

#### 搜索设备

接口功能：开启蓝牙搜索功能 ，搜索周围的蓝牙设备。

接口声明：

|  |
| --- |
| 1. BLEUtils.startLeScan(Context context, BluetoothAdapter.LeScanCallback leScanCallback); |

参数说明：context：上下文 leScanCallback：蓝牙搜索的回调

返回值（void onLeScan(BluetoothDevice device, intrssi, byte[] bytes)）

该接口的返回值说明如下：

|  |
| --- |
| 1. private BluetoothAdapter.LeScanCallback leScanCallback = new BluetoothAdapter.LeScanCallback() { 2. @Override 3. public void onLeScan(BluetoothDevice device, int rssi, byte[] bytes) { 4. *//处理搜索到的设备* 5. } 6. }; |

注意事项：1.保证蓝牙设备有电

2.如要筛选蓝牙设备（厂商ID == 0xFF01），具体请参考四、其他，筛选相关

#### 停止搜索

接口功能：关闭蓝牙搜索功能。

接口声明：

|  |
| --- |
| 1. BLEUtils.stopLeScan(Context context, BluetoothAdapter.LeScanCallback leScanCallback); |

注意事项：调用此接口 ，需保证与戒指处于连接状态

参数说明：context：上下文 leScanCallback：蓝牙搜索的回调

返回值：无

#### 连接设备

接口功能：发起连接蓝牙设备。

接口声明：

|  |
| --- |
| 1. BLEUtils.connectLockByBLE(Context context, BluetoothDevice bluetoothDevice); |

注意事项：调用此接口 ，需保证与戒指处于连接状态

参数说明：context：上下文 bluetoothDevice ：蓝牙设备

返回值：

|  |
| --- |
| 1. @Override 2. public void lmBleConnecting(int code) { 3. *//正在连接* 4. } 5. @Override 6. public void lmBleConnectionSucceeded(int code) { 7. *//连接成功* 8. } 9. @Override 10. public void lmBleConnectionFailed(int code) { 11. *//连接失败* 12. } |

#### 断开蓝牙

接口功能：断开设备。

接口声明：

|  |
| --- |
| 1. BLEUtils.disconnectBLE(Context context); |

注意事项：调用此接口 ，需保证与戒指处于连接状态

参数说明：context：上下文

返回值：无

### 指令功能（LmAPI）

此类是使用戒指功能的公共类 ，戒指的功能通过该类直接调用即可,数据反馈除了测量和历史记录外 统一 由IResponseListener接口反馈

#### 同步时间

接口功能：同步时间 ，调用此接口 ，会获取手机当前时间同步给戒指 ，保持时间同步。

接口声明：

|  |
| --- |
| 1. LmAPI.SYNC\_TIME(); |

注意事项：调用此接口 ，需保证与戒指处于连接状态

参数说明：无

返回值（void syncTime(byte datum)）

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 示例值 | 说明 |
| datum | byte | 0或其他 | 0代表同步成功  其他代表同步失败 |

#### 读取时间

接口功能：读取时间 ，调用此接口 ，会获取戒指当前时间。

接口声明：

注意事项：调用此接口 ，需保证与戒指处于连接状态

参数说明：无

|  |
| --- |
| 1. LmAPI.READ\_TIME(); |

返回值（void readTime(longtime)）

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 示例值 | 说明 |
| time | long | 1658959025 | 时间戳 |

#### 版本信息

接口功能：版本信息 ，获取戒指的版本信息。

接口声明：

|  |
| --- |
| 1. LmAPI.GET\_VERSION((byte) 0x00);  *//0x00获取软件版本，0x01获取硬件版本* |

注意事项：调用此接口 ，需保证与戒指处于连接状态

参数说明：type：0x00获取软件版本 ，0x01获取硬件版本

返回值（void VERSION(byte type, String version)）

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 示例值 | 说明 |
| type | byte | 0或1 | * 0代表软件 * 1代表硬件 |
| version | String | <1.0.0.1> | * 版本号 |

#### 电池电量

接口功能：获取电池电量、 电池状态。

接口声明：

|  |
| --- |
| 1. LmAPI.GET\_BATTERY((byte) 0x00);  *//0x00获取电量，0x01获取充电状态* |

注意事项：调用此接口 ，需保证与戒指处于连接状态

参数说明：type：0x00获取电量 ，0x01获取充电状态

返回值（void battery(byte status, byte datum)）

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 示例值 | 说明 |
| status | byte | 0或1 | * 0代表电池电量 * 1代表充电状态 |
| datum | byte |  | * 电量 |

#### 读取步数

接口功能：获取当天累计步数。

接口声明：

|  |
| --- |
| 1. LmAPI.STEP\_COUNTING（） |

注意事项：调用此接口 ，需保证与戒指处于连接状态

参数说明：无

返回值（void stepCount(int stepCount)）

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 示例值 | 说明 |
| stepCount | int | 5000 | * 步数 |

#### 清除步数

接口功能：清除步数。

接口声明：

|  |
| --- |
| 1. LmAPI.CLEAR\_COUNTING（） |

注意事项：调用此接口 ，需保证与戒指处于连接状态

参数说明：无

返回值：无

#### 恢复出厂设置

接口功能：恢复出厂设置

接口声明：

|  |
| --- |
| 1. LmAPI.RESET（） |

注意事项：调用此接口 ，需保证与戒指处于连接状态

参数说明：无

返回值：无 ，有回调reset方法即认为成功

#### 采集周期设置

接口功能：采集周期设置

接口声明：

|  |
| --- |
| 1. LmAPI.SET\_COLLECTION（collection）*//采集周期，单位秒* |

注意事项：调用此接口 ，需保证与戒指处于连接状态

参数说明：colection：采集间隔，单位秒

返回值：(void collection(int code))

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 示例值 | 说明 |
| code | int | 0或者1 | 0代表设置成功  1代表设置失败 |

#### 采集周期读取

接口功能：采集周期读取

接口声明：

|  |
| --- |
| 1. LmAPI.GET\_COLLECTION（）*//采集周期，单位秒* |

注意事项：调用此接口 ，需保证与戒指处于连接状态

参数说明：无

返回值：(void collectionResult(int time))

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 示例值 | 说明 |
| time | int | 60 | * 采集时间间隔 ，单位秒 |

#### 测量心率

接口功能：测量心率。

接口声明：

|  |
| --- |
| 1. LmAPI.GET\_HEART\_ROTA（byte waveForm,IHeartListener iHeartListener） |

注意事项：调用此接口 ，需保证与戒指处于连接状态

参数说明：

waveForm：是否配置波形 0不上传 1上传

iHeartListener: 此接口是测量数据的监听

返回值：

|  |
| --- |
| 1. LmAPI.GET\_HEART\_ROTA(new IHeartListener() { 2. @Override 3. public void progress(int progress) { 4. setMessage("正在测量心率..." + String.format 5. ("%02d%%", progress)); 6. } 8. @Override 9. public void resultData(int heart, int heartRota, int yaLi, int temp) { 10. if (colorFragment != null) { 11. colorFragment.heartAndRota(heart, heartRota, yaLi, temp); 12. } 13. } 14. @Override 15. public void waveformData(byte seq, byte number, String waveData) { 16. //心率返回波形图数据分析：waveData 17. } 18. @Override 19. public void error(int value) { 20. switch (value) { 21. case 0: 22. dismissProgressDialog(); 23. ToastUtils.show("未佩戴"); 24. break; 25. case 2: 26. dismissProgressDialog(); 27. ToastUtils.show("充电中不允许采集"); 28. break; 29. case 4: 30. dismissProgressDialog(); 31. ToastUtils.show("繁忙，不执行"); 32. break; 33. default: 34. break; 35. } 36. } 38. @Override 39. public void success() { 40. dismissProgressDialog(); 41. } 42. }); |

#### 测量血氧

接口功能：测量血氧。

接口声明：

|  |
| --- |
| 1. LmAPI.GET\_HEART\_Q2（byte waveForm,IQ2Listener iQ2Listener） |

注意事项：调用此接口 ，需保证与戒指处于连接状态

参数说明：

waveForm：是否配置波形 0不上传 1上传

IQ2Listener: 此接口是测量数据的监听

返回值：

|  |
| --- |
| 1. LmAPI.GET\_HEART\_Q2(new IQ2Listener() { 2. @Override 3. public void progress(int progress) { 4. setMessage("正在测量血氧..." + String.format("%02d%%", progress)); 5. } 6. @Override 7. public void resultData(int heart, int q2, int temp) { 8. if (colorFragment != null) { 9. colorFragment.updateData(heart, q2, temp); 10. } 11. } 12. @Override 13. public void waveformData (byte seq, byte number, String waveData) { 14. //血氧返回波形图数据分析：waveData 15. } 16. @Override 17. public void error(int value) { 18. switch (value) { 19. case 0: 20. dismissProgressDialog(); 21. ToastUtils.show("未佩戴"); 22. break; 23. case 2: 24. dismissProgressDialog(); 25. ToastUtils.show("充电中不允许采集"); 26. break; 27. case 4: 28. dismissProgressDialog(); 29. ToastUtils.show("繁忙，不执行"); 30. break; 31. default: 32. break; 33. } 34. } 35. @Override 36. public void success() { 37. dismissProgressDialog(); 38. } 39. }); |

#### 测量温度

接口功能： 测量温度。

接口声明：

|  |
| --- |
| 1. LmAPI.GET\_HEART\_Q2（IQ2Listener iQ2Listener） |

注意事项：调用此接口 ，需保证与戒指处于连接状态

参数说明：IQ2Listener: 此接口是测量数据的监听

返回值：同上。测量血氧时同时会返回温度

#### 历史记录管理

接口功能：读取历史记录。

接口声明：

|  |
| --- |
| 1. LmAPI.READ\_HISTORY（int type,IHistoryListener iHistoryListener） |

注意事项：调用此接口 ，需保证与戒指处于连接状态

参数说明：type: 1,获取全部历史记录；0，获取未上传的历史记录

返回值：

|  |
| --- |
| 1. LmAPI.READ\_HISTORY(type, new IHistoryListener() { 2. @Override 3. public void error(int code) { 4. handler.removeMessages(0x99); 5. dismissProgressDialog(); 6. switch (code) { 7. case 0: 8. ToastUtils.show("正在测量中,请稍后重试"); 9. break; 10. case 1: 11. ToastUtils.show("正在上传历史记录,请稍后重试"); 12. break; 13. case 2: 14. ToastUtils.show("正在删除历史记录,请稍后重试"); 15. break; 16. default: 17. break; 18. } 19. } 20. @Override 21. public void success() { 22. *//同步完成* 23. } 24. @Override 25. public void progress(double progress, com.lm.sdk.mode.HistoryDataBean dataBean) { 26. *//处理历史数据* 27. } 28. }); |

#### 清空历史数据

接口功能：清空历史数据。

接口声明：

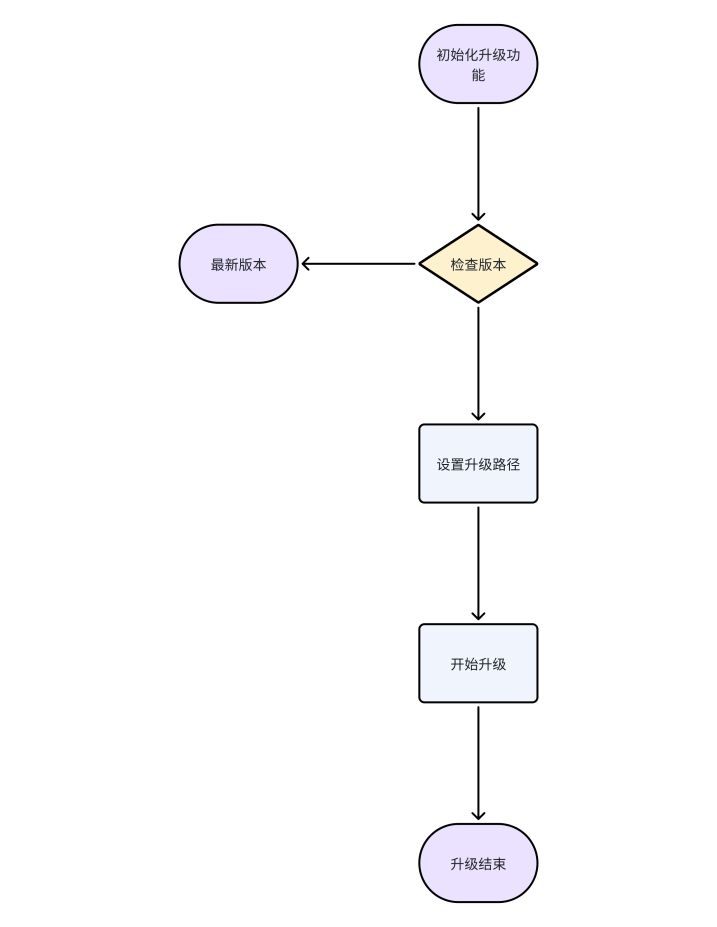
|  |
| --- |
| 1. LmAPI.CLEAN\_HISTORY（） |

注意事项：调用此接口 ，需保证与戒指处于连接状态

参数说明：无

返回值：无

### 固件升级（OTA）



#### 检查版本

接口功能：检查固件版本是否是最新。

接口声明：

|  |
| --- |
| 1. OtaApi.checkVersion(String version, VersionCallback versionCallback); |

注意事项：调用此接口 ，需保证与戒指处于连接状态

参数说明：version：当前戒指的版本号

versionCallback：最新版本信息回调

返回值：

|  |
| --- |
| 1. OtaApi.checkVersion(version, new VersionCallback() { 2. @Override 3. public void success(String newVersion) { 4. *//newVersion：云端最新版本号* 5. if (!StringUtils.isEmpty(newVersion)){ 6. *//有新版本* 7. }else{ 8. *//已是最新版本* 9. } 10. } 11. @Override 12. public void error() { 13. *//获取最新版本失败* 14. } 15. }); |

#### 开始升级

接口功能：执行固件升级。

接口声明：

|  |
| --- |
| 1. void startUpdate(BluetoothDevice bluetoothDevice, int rssi, LmOTACallback otaCallback) |

注意事项：调用此接口 ，需保证与戒指处于连接状态

参数说明：bluetoothDevice： 当前要升级的设备

rssi：设备信号值

otaCallback：升级回调

返回值：

|  |
| --- |
| 1. OtaApi.startUpdate(App.getInstance().getDeviceBean().getDevice(), App.getInstance().getDeviceBean().getRssi(), new LmOTACallback() { 2. @Override 3. public void onDeviceStateChange(int i) { 4. *//设备状态回调* 5. } 6. @Override 7. public void onProgress(int i, int i1) { 8. *//升级进度* 9. } 10. @Override 11. public void onComplete() { 12. *//升级完成* 13. } 14. }); |

### 数据库相关（DataApi）

#### 查询历史记录

接口功能：查询指定时间的历史数据

接口声明：

|  |
| --- |
| 1. *//查询历史数据* 2. List<HistoryDataBean> queryHistoryData(long dayBeginTime,long dayEndTime,String mac) 3. *//查询历史数据按照时间进行正序* 4. List<HistoryDataBean> queryHistoryDataOrderByTimeAsc(long dayBeginTime,long dayEndTime,String mac) 5. *//查询历史数据按照步数进行倒叙* 6. List<HistoryDataBean> queryHistoryDataOrderByStepCountDesc(long dayBeginTime,long dayEndTime,String mac) |

注意事项：需开启相应权限

参数说明：dayBeginTime ：开始时间戳，单位：秒

dayEndTime ：结束时间戳，单位：秒

mac ：设备的MAC地址

返回值：

|  |
| --- |
| 1. public class HistoryDataBean{ 2. @Id 3. private Long id; 4. private String mac; 5. *//总数据包数 4个字节* 6. private long totalNumber; 7. *// 当前第几包 4个字节* 8. private long indexNumber; 9. *// 当前记录时间 4个字节* 10. private long time; 11. *//今天累计步数 2个字节* 12. private int stepCount; 13. *// 心率 1个字节* 14. private int heartRate; 15. *// 血氧 1个字节* 16. private int bloodOxygen; 17. *//心率变异性 1个字节* 18. private int heartRateVariability; 19. *//精神压力指数 1个字节* 20. private int stressIndex; 21. *//温度 2个字节* 22. private int temperature; 23. *//运动激烈程度 1个字节* 24. private int exerciseIntensity; 25. */\*\** 26. \* 睡眠类型 1个字节 27. \* 0：无效 28. \* 1：清醒 29. \* 2：浅睡 30. \* 3：深睡 31. \* 4.眼动期 32. \*/ 33. private int sleepType; 34. *// 保留 2个字节* 35. private int reserve; 36. *//RR间期 1个字节* 37. private int rrCount; 38. *//RR数组数据 1个字节* 39. private byte[] rrBytes; 40. } |

#### 清空历史数据

接口功能：清空全部历史数据。

接口声明：

|  |
| --- |
| 1. void deleteHistoryData()； |

注意事项：需开启相应权限

参数说明：无

返回值：无

### 逻辑算法相关（LogicalApi）

#### 计算距离、卡路里

接口功能：根据步数计算距离、卡路里。

接口声明：

|  |
| --- |
| 1. DistanceCaloriesBean calculateDistance(int stepCount,double height,double weight)； |

注意事项：需开启相应权限

参数说明：stepCount：步数

height：身高 单位cm

weight：体重 单位kg

返回值：

|  |
| --- |
| 1. public class DistanceCaloriesBean { 2. *//距离，单位米* 3. private double distance; 4. *//卡路里，单位卡* 5. private double kcal; 6. } |

#### 计算睡眠数据

接口功能：计算指定时间睡眠数据。

接口声明：

|  |
| --- |
| 1. SleepBean calculateSleep(String date, String mac, int type)； |

注意事项：需开启相应权限

参数说明：date：日期，格式为YYYY-MM-DD HH:mm:ss

mac ：设备蓝牙MAC

type：0表示日数据 ，1表示周数据 ，2表示月数据

返回值：

|  |
| --- |
| 1. public class SleepBean { 2. *// 零星睡眠小时* 3. int hours = 0; 4. *// 零星睡眠分钟* 5. int minutes = 0; 6. *// 全部睡眠小时* 7. int allHours = 0; 8. *//全部睡眠分钟* 9. int allMinutes =0; 10. *// 深度睡眠时间* 11. long highTime = 0; 12. *//浅度睡眠时间* 13. long lowTime = 0; 14. *// 眼动时间* 15. long ydTime = 0; 16. *// 清醒时间* 17. long qxTime = 0; 18. *// 入睡时间戳* 19. long startTime = 0; 20. *//清醒时间戳* 21. long endTime = 0; 22. *//睡眠数据* 23. private List<HistoryDataBean> historyDataBeanList; 24. } |

数据库中数据

|  |
| --- |
| 1. public class HistoryDataBean{ 2. @Id 3. private Long id; 4. private String mac; 5. *// 总数据包数 4个字节* 6. private long totalNumber; 7. *// 当前第几包 4个字节* 8. private long indexNumber; 9. *// 当前记录时间 4个字节* 10. private long time; 11. *// 今天累计步数 2个字节* 12. private int stepCount; 13. *// 心率 1个字节* 14. private int heartRate; 15. *// 血氧 1个字节* 16. private int bloodOxygen; 17. *// 心率变异性 1个字节* 18. private int heartRateVariability; 19. *// 精神压力指数 1个字节* 20. private int stressIndex; 21. *// 温度 2个字节* 22. private int temperature; 23. *// 运动激烈程度 1个字节* 24. private int exerciseIntensity; 25. */\*\** 26. \* 睡眠类型 1个字节 27. \* 0：无效 28. \* 1：清醒 29. \* 2：浅睡 30. \* 3：深睡 31. \* 4.眼动期 32. \*/ 33. private int sleepType; 34. *// 保留 2个字节* 35. private int reserve; 36. *// RR间期 1个字节* 37. private int rrCount; 38. *// RR数组数据 1个字节* 39. private byte[] rrBytes; 40. } |

# 其他

注：使用戒指API前，应先查看戒指状态

## 筛选相关

血氧戒指设备以XXXXXX的名字进行广播。XXX为任何字符，广播间隔为500ms。本例中XXX为BCL603。

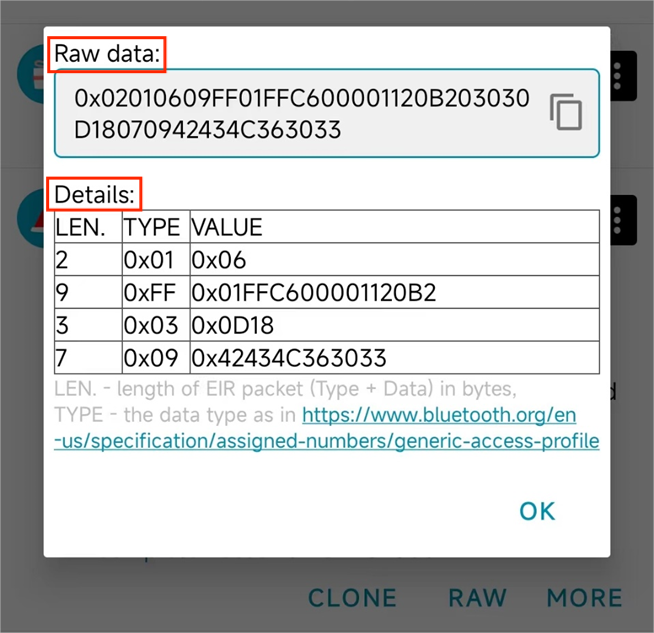
广播包的长度必须是31个字节，如果不到31个字节 ，则剩下的全用0填充 补全，这部分的数据是无效的

### 广播数据单元

广播包中包含若干个广播数据单元，广播数据单元也称为 AD Structure。

**广播数据单元 = 长度值Length + AD type + AD Data**

长度值Length只占**一个字节**，并且位于广播数据单元的**第一个字节**。



#### Raw data:

0x代表这串字符串是十六进制的字符串。两位十六进制数代表一个字节。因为两个字符组成的十六进制字符串最大为FF，即255，而Java中byte类型的取值范围是-128到127，刚好可以表示一个255的大小。所以两个十六进制的字符串表示一个字节。

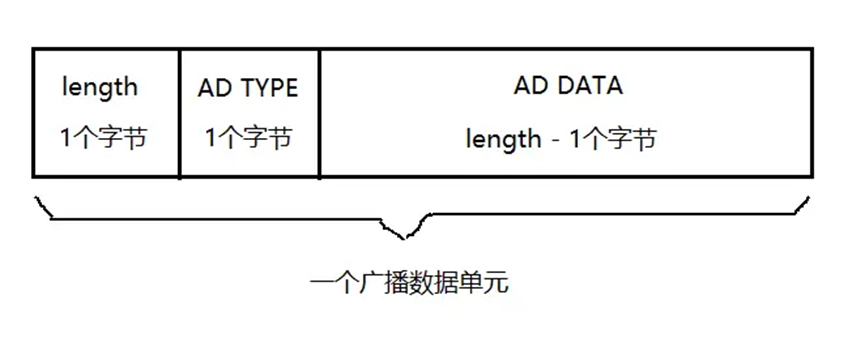
继续查看报文内容，开始读取第一个广播数据单元。读取第一个字节:0x02,转换为十进制就是2，即表示后面的2个字节是这个广播数据单元的数据内容。超过这2个字节的数据内容后，表示是一个新的广播数据单元。

而第二个广播数据单元，第一个字节的值是0x09,转换为十进制就是9，表示后面9个字节为第二个广播数据单元。

而第三个广播数据单元，第一个字节的值是0x03,转换为十进制就是3，表示后面3个字节为第三个广播数据单元。

以此类推。

在广播数据单元的数据部分中，第一个字节代表数据类型（AD type），决定数据部分表示的是什么数据。（即广播数据单元第二个字节为AD type）



#### Details:

(1)、Type = 0x01 表示设备LE物理连接。

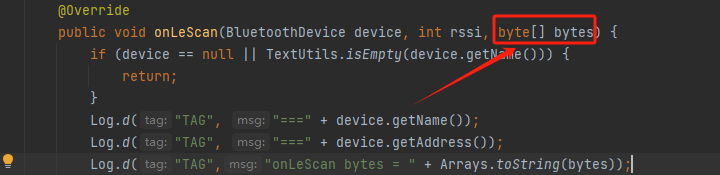
(2)、Type = 0xFF 表示厂商数据。前两个字节表示厂商ID,即厂商ID为0xFF01。后面的为厂商数据，具体由用户自行定义

(3)、Type = 0x03 表示完整的16bit UUID。其值为0x0D18。

(4)、Type = 0x09 表示设备的全名，例如：0x42434C363033转byte[]再转字符串即为“BCL603”

### 应用

注：数据传输方式为小端模式



调用蓝牙扫描时找到返回的byte[],这里具体为[2, 1, 6, 9, -1, 1, -1, -58, 0, 0, 17, 32, -78, 3, 3, 13, 24, 7, 9, 66, 67, 76, 54, 48, 51, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

注：byte数据需转换为16进制

已知该数据的格式和含义，根据规则设置筛选条件为厂商ID == “FF01”即可

或直接对接收到的数据验证“01FF”

## 可能会遇到的问题

资料中带有简单demo，可以先查看简单demo使用SDK的逻辑，再进行自己的开发

### 版本相关

Gradle版本。可在gradle-wrapper.properties里修改

|  |
| --- |
| 1. distributionBase=GRADLE\_USER\_HOME 2. distributionPath=wrapper/dists 3. distributionUrl=https\://services.gradle.org/distributions/gradle-8.0-all.zip 4. zipStoreBase=GRADLE\_USER\_HOME 5. zipStorePath=wrapper/dists |

可在以下网址下载所需版本，将zip放在wrapper/dists对应的路径下（放在随机码文件夹下，记得清空原有内容），重新sync一下

[Gradle Distributions](https://services.gradle.org/distributions/)

### Gradle 4.0以上导致Xpopup无法使用

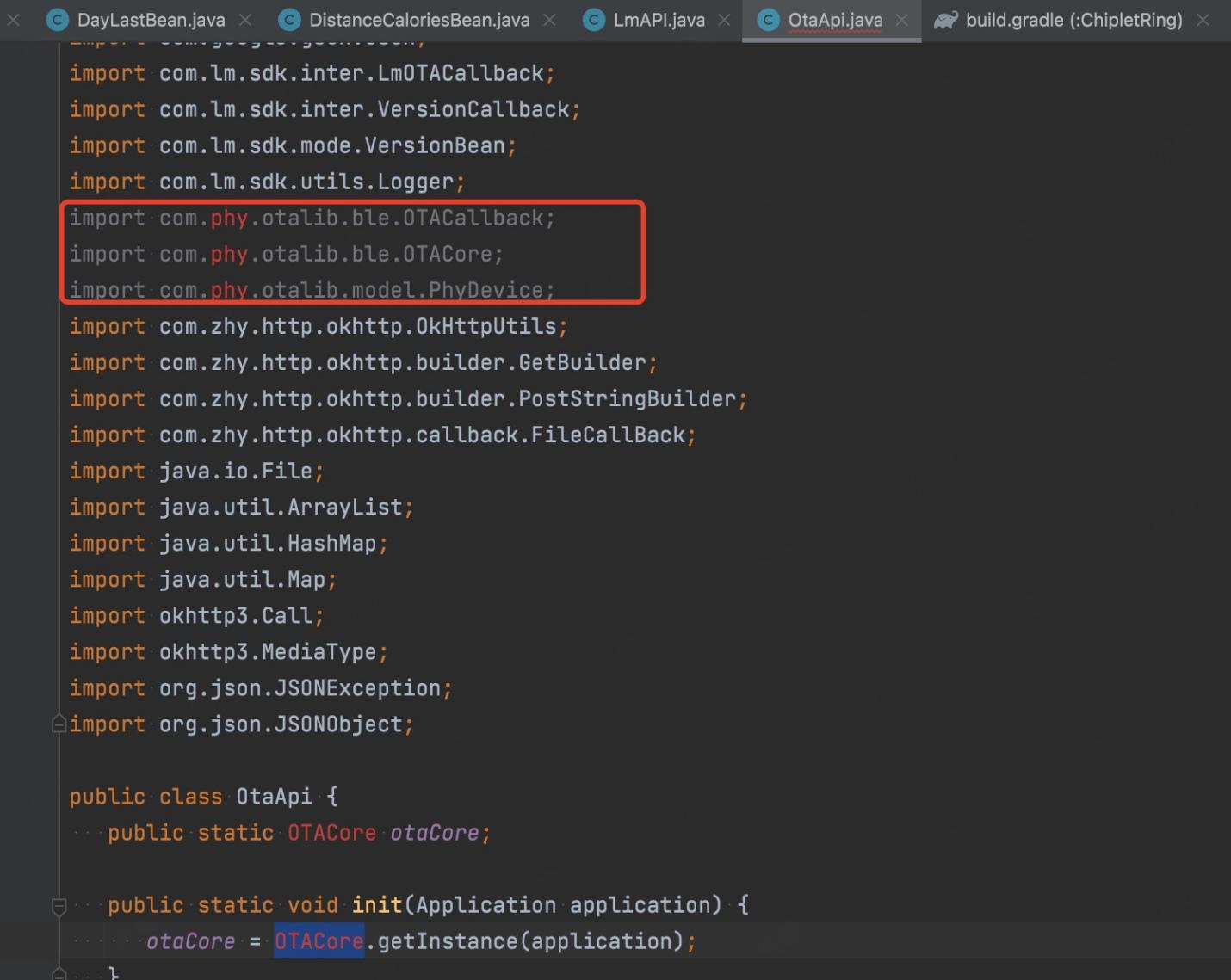
Xpopup是个第三方弹窗框架，换为普通弹窗可以解决问题

### 不需要界面

ringSDK1.0.2已支持不需要界面，在service做扫描、连接等功能

### OTA类引用未找到

可能会出现以下情况：



官方的其他依赖库，若找不到，可联系勇芯开发人员，要原始类

原始jar包已放到SDK的OTA类文件夹下

## 硬件算法逻辑或固件相关

### 戒指相关

问：戒指多久存一次数据

答：5分钟

问：OTA升级会清除数据吗

答：会

问：戒指里的数据可以存几天

答：7天，7天后自动覆盖

问：恢复出厂设置是只恢复戒指吗

答：对，只针对戒指硬件进行恢复

### 算法相关

问：为什么使用SDK的心率/血氧测量时总显示超时

答：戒指充电时，无法进行心率血氧测量

## Q&A

Q：读历史记录过程中，是上报一条标记一条已同步，还是全部上传了整体标记？

A：一条一条

Q：采集周期设置有什么限制

A: 采集周期单位为秒，正常值最小为60s，为0时代表关闭采集。