手机蓝牙开门 SDK 开发指南

Android

V2.2.7

Android SDK 版本发布记录

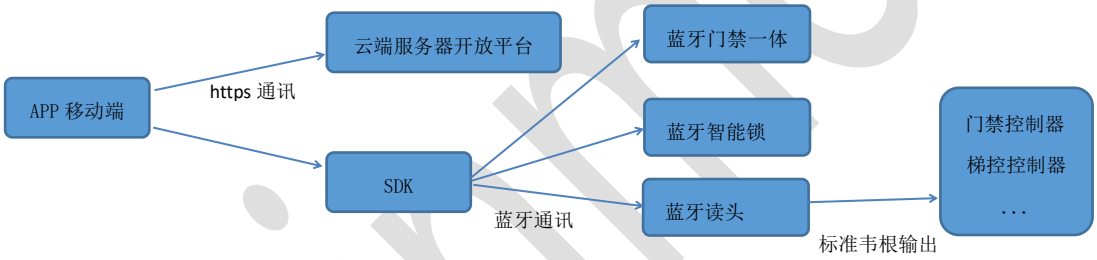
发布版本	发布日期	更新内容	开发人员
V1.4（beta 版本）	2016.02.02	1. SDK 多用户设置、切换。 2. 电子钥匙开门接口优化。	Cola.Zhang
V1.5	2016.03.29	1. 扫描接口添加返回信号值。	Cola.Zhang
V1.6	2016.06.24	1. 去除数据保存，权限管理交于调用者处理	Cola.Zhang
V1.7	2016.12.12	1. 抽离功能接口	Cola.Zhang
V1.8	2016.12.18	1. 增加配置 WiFi 接口 2. 增加配置写卡参数接口	Benson.Chen
V1.9	2016.12.30	1. 传入参数增加判断 2. 修改配置读卡扇区、扇区密钥接口名称 3. 错误码返回值和 IOS 统一	Jason.Huang
V2.0	2017.03.20	1. 添加扫描扩展接口	Jason.Huang
V2.0.2	2017.4.20	1. 扫描接口，扫描可精确到毫秒，最低 600 毫秒 2. 修复读取卡号相关的 bug 3. 扫描排序接口，传入的回调 ExtScanCallBack 改为 SortScanCallBack	Larry.Liu
V2.0.3	2017.05.06	1．优化扫描接口稳定性。 2．修改扫描传入时间参数，单位毫秒，最大 60 秒，传入-1，循环循环扫描。 3．增加扫描 scanDeviceNonfilter 接口，扫描返回的设备 sn 不进行截取 4．增加靠近开门接口	Larry.Liu
V2.0.4	2017.05.09	修改同步时间参数 bug	Larry.Liu
V2.0.5	2017.05.16	1、增加电子钥匙结束时间的验证 2、增加对电子钥匙开始时间的验证	Larry.Liu
V2.0.6	2017.06.07	1、修改自动开门内部逻辑	Benson.Chen
V2.0.7	2017.06.21	1、修改 scanDeviceNonfilter 内部逻辑 2、修改部分读头刷卡失败	Larry.Liu
V2.0.8	2017.07.08	1、修改参数检查时间验证空指针异常	Larry.Liu
V2.0.9	2017.07.11	1. 修复扫描设备时序列号过滤的 bug	Benson.Chen
V2.0.10	2017.07.12	1、优化 SDK 结构 2、增加亮屏自动开门模式	Larry.Liu
V2.1.0	2017.07.18	1、增加蓝牙控制 EXT200 重启设备接口	Larry.Liu
V2.1.1	2017.07.27	1. 优化扫描接口，兼容 Android 7.0 对蓝牙扫描频率的限制 2. 强制打开蓝牙	Benson.chen
V2.1.2	2017.09.05	1. 优化强制打开蓝牙过程	Benson.chen
V2.1.3	2017.09.07	1. 增加蓝牙重启设备接口	Larry.Liu
V2.1.4	2017.09.27	1. 获取系统信息增加获取设备类型、设备编号、设备扇区、扇区密钥（部分设备支持） 2. 增加配置静态 IP 接口（用于 V500 设备的门禁板）	Benson.chen

		3. 解决在子线程调用扫描时出现异常	
V2.1.5	2017.10.09	1. 修复关闭自动开门接口奔溃的 bug	Benson.chen
V2.1.6	2017.10.20	1. 增加配置设备连接服务器地址	Benson.chen
V2.1.7	2017.10.23	1. EXT200 和 EXT210 增加发送卡号到设备	Benson.chen
V2.1.8	2018.01.18	1. 增加配置设备静态 IP 和服务端 IP（V500 设备门禁板） 2. 增加读取卡号接口 3. 增加读取开门记录接口 4. 增加同步指纹数据接口 5. 增加删除所有指纹数据接口 6. 增加删除部分指纹数据接口	Benson.chen
V2.1.9	2018.03.14	1. 获取设备信息增加获取电量	Benson.chen
V2.2.0	2018.05.18	1. EXT220 增加发送卡号到设备	Benson.chen
V2.2.1	2018.06.23	1. 增加获取最近未获取的开门记录 2. 增加删除开门记录的接口	Benson.chen
V2.2.2	2019.02.28	1. 修复 bug	Benson.chen
V2.2.3	2019.03.27	1. 增加 SL100B、SL200B、SL300B 三种设备的开门功能（设备密码放在 cardno 里）	Benson.chen
V2.2.4	2019.04.19	1. 修复发送扫描添加的电子钥匙无法使用的 bug	Benson.chen
V2.2.5	2019.10.28	1. 修复扫描闪退问题	Benson.chen
V2.2.6	2019.11.14	1. 增加 SL400 和 SL500 类型	Benson.chen
V2.2.7	2019.11.29	1. 增加获取信号值的接口	Benson.chen

文档目的

详细说明 app 移动端开发，SDK 提供的接口、注意事项。本文档作为内部 app 开发以及提供给第三方 app 开发使用。

1. APP 移动端、SDK、服务器开放平台、设备，通讯流程



2.功能说明

APP 移动端，目前支持 Android 手机和 iphone 手机，手机开门功能实际使用的是电子钥匙，电子钥匙的来源从云端服务器中请求获取，APP 获取云端分配的电子钥匙后，通过调用 SDK 接口操作指定设备即可实现手机开门功能。

移动端设备	系统版本	蓝牙版本
Android 手机	Android 4.4 及以上	4.0 及以上

3. SDK 使用注意事项

3.1 支持的设备

支持的 Android 系统最低版本为 Android4.3(API 17)，需同时支持蓝牙版本 4.0

3.2 配置文件

(1)在 AndroidManifest.xml 中需要加入蓝牙权限：

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<uses-feature android:name="android.hardware.bluetooth_le" android:required="true"/>
<!-- 定位权限，高版本的蓝牙可能需要定位配置 -->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

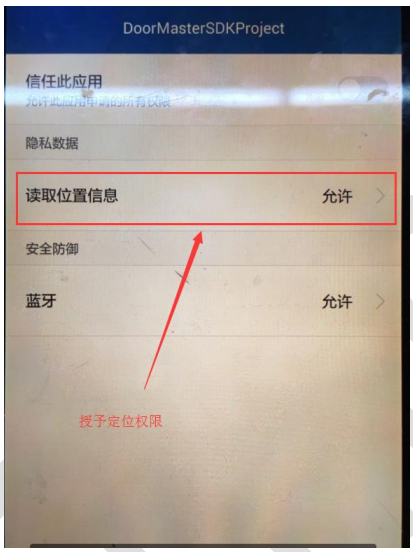
(2)并且需要加入维护的一个蓝牙服务：

```
<service android:name="com.intelligoo.sdk.BluetoothLeService" android:enabled="true" />
```

3.3 常见问题

Q: 蓝牙扫描不到设备，一键开门失败

A: Android 6.0 以上的设备扫描需要定位权限，如下图：



Q: 扫描成功率不高，经常出现-112

A: Android 7.0 限制 30 秒内只能启用扫描功能 5 次

Q: 蓝牙打开了，定位权限也给了，但就是扫描不到设备

A: 可以尝试将扫描回调的 onScanResultAtOnce 接口里的代码注释掉，再不行可以试试将 build.gradle 里的 compileSdkVersion 指定为 22

Q: OPPO 和 vivo 手机扫描不到设备

A: Vivi 和 OPPO 这两种系列手机位置权限需要开启，同时设置界面的 GPS（定位服务）也要开启，其他手机 GPS 可以不用

Q:手机一开始能开，后来扫描不到设备、直接开门也不行

A:目前发现部分手机（Android 7.0）在后台执行开门一定时间后，程序被系统限制，无

法使用蓝牙功能，关闭手机蓝牙，并重启蓝牙可以解决

Q:手机 APP 蓝牙无法扫描、无法开门

A:可能是蓝牙连接数达到最大值，此时可以尝试关闭已匹配的蓝牙设备（如：蓝牙耳环、蓝牙耳机等），或者重启蓝牙

Q:开视频设备（如：V622 等设备），SDK 返回成功，设备的继电器也响了，但是没有提示成功开门

A:查看下开门时传入的 devType 的值，当 devType 为：14、17、18、20 时，需要传入卡号。

4. SDK 接口函数

4.1 openDoor

[功能说明]

LibDevModel 类的静态方法，控制开门。

返回值详细请参见《附表三》。

[接口函数]

public static int openDoor(Context context, LibDevModel device, ManagerCallback callback);

[参数说明]

context：设置开门回调的上下文。

device：需要操作的设备

callback：开门指令操作结束后回调的接口，需要定义出该接口的实现。

[APK 调用 SDK 示例]

```
ManagerCallback callBack = new ManagerCallback(){
        onResult(int ret, Bundle bundle){
        }
};
int ret = LibDevModel.openDoor(context,device, callback);
```

4.2 syncDeviceTime

[功能]

LibDevModel 类的静态方法。同步设备时间，时间格式为”20161212”+”143000”+”01”最后年月日时分秒星期。(01～07:星期一～星期天)

返回值详细请参见《附表三》。

[**接口方法**]

public static int syncDeviceTime(Context context, LibDevModel device, String time, ManagerCallback callback);

[**参数说明**]

context：设置开门回调的上下文。

device: 需要操作的设备

time:同步的时间，例如 2016 年 12 月 12 日 13 点 52 分 33 秒星期一，对应格式为 2016121213523301

callback：操作结束后回调的接口，需要定义出该接口的实现。

[**APP 调用示例**]

```
ManagerCallback callBack = new ManagerCallback(){
    onResult(int ret, Bundle bundle){
    }
};
String syncTime = “2016121213523301”;
int ret = LibDevModel.syncDeviceTime(context,device, syncTime,callback);
```

4.3 modifyPwd

[**功能**]

LibDevModel 类的静态方法。修改有按键的设备六位按键开门密码。

返回值详细请参见《附表三》。

[**接口方法**]

public static int modifyPwd(Context context, LibDevModel device, String oldPwd, String newPwd, ManagerCallback callback);

[**参数说明**]

context：设置开门回调的上下文。

device: 需要操作的设备

oldPwd: 旧密码，6 位数字；设备默认没有初始密码，传入@””空值即可。

newPwd: 新密码，6 位数字。

callback：操作结束后回调的接口，需要定义出该接口的实现。

[**APP 调用示例**]

```
ManagerCallback callBack = new ManagerCallback(){
    onResult(int ret, Bundle bundle){
    }
};
String oldPwd = “”;
String newPwd = “123456”;
int ret = LibDevModel.modifyPwd(context,device, oldPwd, newPwd ,callback);
```

4.4 swipeAddCardModel

[**功能说明**]

LibDevModel 类的静态方法，进入登记卡模式，刷卡登记卡之后需要调用退出登记卡模式接口。

返回值详细请参见《附表三》。

[**接口函数**]

public static int swipeAddCardModel(Context context, LibDevModel device, ManagerCallback callback);

[**参数说明**]

context：设置开门回调的上下文。

device: 需要操作的设备

callback：操作结束后回调的接口，需要定义出该接口的实现。

[**APK 调用 SDK 示例**]

```
ManagerCallback callBack = new ManagerCallback(){
    onResult(int ret, Bundle bundle){
    }
};
int ret = LibDevModel.swipeAddCardModel(context,device, callback);
```

4.5 existSwipeCardAddModel

[**功能说明**]

LibDevModel 类的静态方法，退出当前刷卡登记卡模式。

返回值详细请参见《附表三》。

[**接口函数**]

public static int existSwipeCardAddModel(Context context, LibDevModel device, ManagerCallback callback);

[**参数说明**]

context：设置开门回调的上下文。

device: 需要操作的设备

callback：操作结束后回调的接口，需要定义出该接口的实现。

[**APK 调用 SDK 示例**]

```
ManagerCallback callBack = new ManagerCallback(){
    onResult(int ret, Bundle bundle){
    }
};
int ret = LibDevModel.existSwipeCardAddModel(context,device, callback);
```

4.6 swipeCardDeleteModel

[功能说明]

LibDevModel 类的静态方法，进入删除卡模式，刷卡删除卡之后需要调用退出删除卡模式接口。

返回值详细请参见《附表三》。

[接口函数]

public static int swipeCardDeleteModel(Context context, LibDevModel device, ManagerCallback callback);

[参数说明]

context：设置开门回调的上下文。

device: 需要操作的设备

callback：操作结束后回调的接口，需要定义出该接口的实现。

[APK 调用 SDK 示例]

```
ManagerCallback callBack = new ManagerCallback(){
    onResult(int ret, Bundle bundle){
    }
};
int ret = LibDevModel.swipeCardDeleteModel(context,device, callback);
```

4.7 existSwipeCardDeleteModel

[功能说明]

LibDevModel 类的静态方法，退出当前刷卡删除卡模式。

返回值详细请参见《附表三》。

[接口函数]

public static int existSwipeCardDeleteModel(Context context, LibDevModel device, ManagerCallback callback);

[参数说明]

context：设置开门回调的上下文。

device: 需要操作的设备

callback：操作结束后回调的接口，需要定义出该接口的实现。

[APK 调用 SDK 示例]

```
ManagerCallback callBack = new ManagerCallback(){
    onResult(int ret, Bundle bundle){
    }
};
int ret = LibDevModel.existSwipeCardDeleteModel(context,device, callback);
```

4.8 setDeviceConfig

[功能说明]

LibDevModel 类的静态方法，更新设备配置(韦根格式，开门时间，控锁方式)。

返回值详细请参见《附表三》。

[接口函数]

public static int setDeviceConfig(Context context, LibDevModel device, int wiegand, int openDelay,int controlWay ,ManagerCallback callback);

[参数说明]

context：设置开门回调的上下文。

device: 需要操作的设备

wiegand：设备韦根格式(韦根 26:wiegand = 26,韦根 34:wiegand = 34)

openDelay:开门时长 (openDelay>0:单位为秒)

controlWay:控锁方式（电锁控制 0x00，电气控制 0x01）

callback：操作结束后回调的接口，需要定义出该接口的实现。

[APK 调用 SDK 示例]

```
ManagerCallback callBack = new ManagerCallback(){
    onResult(int ret, Bundle bundle){
    }
};
int wiegand = 26;
int openDelay = 5;
```

int controlWay = 0; (电锁控制开锁锁(继电器)会自动关闭，电气控制锁需要再次调用开门接口才能关闭锁(继电器))

```
int ret = LibDevModel.setDeviceConfig(context,device,wiegand,openDelay, controlWay , callback);
```

4.9 getDeviceConfig

[功能说明]

LibDevModel 类的静态方法，获取当前设备的配置(开门时长，卡登记数，手机登记数，最大用户容量，设备控制方式，韦根格式，设备版本号)。

返回值详细请参见《附表三》。

[接口函数]

public static int getDeviceConfig(Context context, LibDevModel device, ManagerCallback callback);

[参数说明]

context：设置开门回调的上下文。

device：需要操作的设备
callback：操作结束后回调的接口，需要定义出该接口的实现。

[APK 调用 SDK 示例]

```
ManagerCallback callBack = new ManagerCallback(){
    onResult(int ret, Bundle bundle){
    /*
    ConstantsUtils.OPEN_DELAY （开门时间）                    （int）
    ConstantsUtils.CONTROL 控制方式（0 电气 1 电锁）          （int）
    ConstantsUtils.REG_CARDS_NUMS 登记的卡数量                （int）
    ConstantsUtils.DEV_SYSTEM_VERSION 固件版本号              （int）
    ConstantsUtils.MAX_CONTAINER 最大容量                     （int）

    以下字段部分设备有（使用之前可以先判断是否有 ConstantsUtils.SECTOR_KEY
字段）：
    ConstantsUtils.SECTOR_KEY          扇区密钥     （字符串）
    ConstantsUtils.DEVICE_DOOR_NO      门编号          （int）
    ConstantsUtils.MIFARE_SECTOR       扇区         （int）
    ConstantsUtils.DEV_TYPE            设备类型     （int）

    ConstantsUtils.SERVER_IP           连接服务器 IP（String）
    ConstantsUtils.SERVER_PORT         连接服务器端口（Int）
    ConstantsUtils.WIFI_NAME           WiFi 名称（String）
    ConstantsUtils.WIFI_PWD            WiFi 密码（String）
    ConstantsUtils.DEVICE_CONFIG_FUNCTION     设备功能参数（Int）
    ConstantsUtils.DEVICE_DATE         设备日期（String）
    ConstantsUtils.DEVICE_ELECTRICITY  设备电量（Int） 1~100

    */
    //在 bundle 中使用以上 Key 获取对应数据
    int openDelay = bundle.getInt(ConstantsUtils.OPEN_DELAY);
    }
};
int ret = LibDevModel.getDeviceConfig(context,device, callback);
```

4.10 deleteDeviceData

[功能说明]

LibDevModel 类的静态方法，删除设备所有数据(卡数据，用户数据)。
返回值详细请参见《附表三》。

[接口函数]

```
public static int deleteDeviceData(Context context, LibDevModel device, ManagerCallback
```

callback);

[参数说明]

context：设置开门回调的上下文。

device: 需要操作的设备

callback：操作结束后回调的接口，需要定义出该接口的实现。

[APK 调用 SDK 示例]

```
ManagerCallback callBack = new ManagerCallback(){
    onResult(int ret, Bundle bundle){
    }
};
int ret = LibDevModel.deleteDeviceData(context,device, callback);
```

4.11 resetDeviceConfig

[功能说明]

LibDevModel 类的静态方法，重置设备配置(开门时长，韦根格式，电锁控制)。

返回值详细请参见《附表三》。

[接口函数]

public static int resetDeviceConfig(Context context, LibDevModel device, ManagerCallback callback);

[参数说明]

context：设置开门回调的上下文。

device: 需要操作的设备

callback：操作结束后回调的接口，需要定义出该接口的实现。

[APK 调用 SDK 示例]

```
ManagerCallback callBack = new ManagerCallback(){
    onResult(int ret, Bundle bundle){
    }
};
int ret = LibDevModel.resetDeviceConfig(context,device, callback);
```

4.12 writeCard

[功能]

写入卡操作接口，将卡号写入设备；

writeCard 函数为 LibDevModel 类的静态方法。

返回值详细请参见《附表三》。

[接口方法]

public static int writeCard(Context context, LibDevModel device,List<String> cardData,final ManagerCallback callback, oolean addAppend);

[**参数说明**]

context：设置上下文。

device: 需要操作的设备。

cardData:卡号数据。

callback:操作结束后回调的接口，需要定义出该接口的实现。

addAppend:true:不进行清空，直接添加卡号;false:清空后添加卡号。

[**APP 调用示例**]

```
ManagerCallback callBack = new ManagerCallback(){
    onResult(int ret, Bundle bundle){
    }
};
ArrayList<String> data = new ArrayList<String>();
data.add(“1976120642”);
data.add(“1976104236”);
data.add(“3320000004”);
data.add(“2584561783”);
int ret = LibDevModel.writeCard(MainActivity.this, device, data, callback,true);
```

4.13 deleteCard

[**功能**]

LibDevModel 类的静态方法，批量删除卡。

返回值详细请参见《附表三》。

[**接口方法**]

public static int deleteCard(Context context, LibDevModel device,List<String> cardData,final ManagerCallback callback);

[**参数说明**]

context：设置上下文。

device: 需要操作的设备。

cardData:卡号数据。

callback:操作结束后回调的接口，需要定义出该接口的实现。

[**APP 调用示例**]

```
ManagerCallback callBack = new ManagerCallback(){
    onResult(int ret, Bundle bundle){
    }
};
ArrayList<String> data = new ArrayList<String>();
data.add(“1976120642”);
```

data.add(“1976104236”);
data.add(“3320000004”);
data.add(“2584561783”);
int ret = LibDevModel.deleteCard(MainActivity.this, device, data, callback);

4.14 cleanCard

[**功能**]
LibDevModel 类的静态方法。清空卡操作接口，删除设备所有卡号；
返回值详细请参见《附表三》。
[**接口方法**]
public static int cleanCard(Context context, LibDevModel device, ManagerCallback callback);
[**参数说明**]
context：设置上下文。
device: 需要操作的设备。
callback:操作结束后回调的接口，需要定义出该接口的实现。

[**APP 调用示例**]
```
ManagerCallback callBack = new ManagerCallback(){
    onResult(int ret, Bundle bundle){
    }
};
int ret = LibDevModel.cleanCard(MainActivity.this, device, callback);
```

4.15 configWifi

[**功能**]
配置支持 wifi 功能的设备连接路由；
configWifi 函数为 LibDevModel 类的静态方法。
返回值详细请参见《附表三》。

[**接口方法**]
public static int configWifi(final Context context, LibDevModel device, Bundle bundle ,final ManagerCallback callback);
[**参数说明**]
context：设置上下文。
device: 需要操作的设备。
bundle: 操作所需配置参数，详细看调用示例；
IP_ADDRESS:服务器 IP 地址
PORT:服务器端口

AP_NAME:路由名称（SSID 不支持中文）

AP_PASSWORD：路由器密码

callback:操作结束后回调的接口，需要定义出该接口的实现。

[**APP 调用示例**]

```
ManagerCallback callBack = new ManagerCallback(){
    onResult(int ret, Bundle bundle){
    }
};
Bundle bundle = new Bundle();
bundle.putString(ConstantsUtils.IP_ADDRESS, address);
bundle.putInt(ConstantsUtils.PORT, port_int);
bundle.putString(ConstantsUtils.AP_NAME, apName);
bundle.putString(ConstantsUtils.AP_PASSWORD, apPwd);
int ret = LibDevModel.configWifi(MainActivity.this, device, bundle, callback);
```

4.16 setReadSectorKey

[**功能**]

LibDevModel 类的静态方法，配置设备编号、读取卡扇区编号、卡扇区秘钥，用于 M160 门禁一体机等设备 权限写卡扇区使用和所有一体机的临时密码配置。

新版 6 位临时密码需要配置设备门编号和扇区密钥。

返回值详细请参见《附表三》。

[**接口方法**]

public static int setReadSectorKey(final Context context, LibDevModel device, int devId , int mifareSector, String sectorKey, final ManagerCallback callback);

[**参数说明**]

context：设置上下文。

device：需要操作的设备。

devId：设备 ID 编号（值范围：0-255）

mifareSector: Mifare 扇区地址（值范围：0-15）

sectorKey：扇区密钥(十六进制字符串，长度为 12)

callback:操作结束后回调的接口，需要定义出该接口的实现。

[**APP 调用示例**]

```
ManagerCallback callBack = new ManagerCallback(){
    onResult(int ret, Bundle bundle){
    }
};
int devId = 0;
int mifareSector = 1;
String sectorKey = “01234567ffff”;
```

int ret = LibDevModel.setReadSectorKey(context, device, devId， mifareSector, sectorKey, callback);

4.17 scanDevice

[**功能**]

LibDevModel 类的静态方法,扫描附近的蓝牙读头、一体机等，可用于 app 设备列表在线设备高亮显示。

返回值详细请参见《附表三》。

[**接口方法**]

public static int scanDevice(Context context, boolean atOnce, int sec, ScanCallback callBack);

[**参数说明**]

sec：扫描时间，单位：毫秒，范围：0- 60000（或者-1，不限扫描时间）；

atOnce: 扫描到的设备是否立即返回（true 指定时间内扫描到设备即刻返回， false 指定时间结束后，扫描到的所有设备一起返回）

callback: 扫描指令操作结束后回调的接口，返回结果详细请参见《附表三》。

[**APP 调用示例**]

```
ScanCallback callBack = new ScanCallback(){
    onScanResult(ArrayList<LibDevModel> devList, ArrayList<Integer> rssiList){
    //扫描完后之后返回扫描到的设备列表（atOnce = false）
    }
    onScanResult(String devSn, Integer rssi){
    //扫描到设备立即返回设备（atOnce = true）
    }
};
int ret = LibDevModel.scanDevice(context, true, 800, callback);
```

4.18 scanDeviceSort

[**功能**]

LibDevModel 类的静态方法,扫描附近的蓝牙读头、一体机等，可用于 app 设备列表在线设备高亮显示。

返回值详细请参见《附表三》。

[**接口方法**]

public static int scanDeviceSort(Context context, boolean atOnce, int sec, ScanCallbackSort callBack);

[**参数说明**]

sec：扫描时间，单位：秒，范围：0- 60000（或者-1，不限扫描时间）；

callback: 扫描指令操作结束后回调的接口，返回结果详细请参见《附表三》。

[**APP 调用示例**]

```
ScanCallbackSort callBack = new ScanCallbackSort (){
    onScanResult(ArrayList<Map<String, Integer>> devSnRssiList){
//扫描完后之后返回扫描到的设备列表（atOnce = false）
}
OnScanResultAtOnce(String devSn, Integer rssi){
//扫描到设备立即返回设备
}
};
int ret = LibDevModel.scanDevice(context, true, 800, callback);
```

4.19 controlDevice

[**功能**]

LibDevModel 类的静态方法，用于控制设备开门或者其他操作。更多功能请参见《附表二》。

返回值详细请参见《附表三》。

[**接口方法**]

public static int controlDevice(Context context,int operation,LibDevModel device,Bundle bundle, ManagerCallback callback);

[**参数说明**]

context: 当前的上下文

operation: 操作的代码，具体参看《附表二》

device: 操作的设备

bundle: 对应 operation 所需要的参数，以 Key-Vlaue 的方式传输 Key 值请参看附表

callback: 操作结束后回调的接口，返回结果详细请参见《附表三》。

[**APP 调用示例**]

```
ManagerCallback callBack = new ManagerCallback(){
    onResult(int ret, Bundle bundle){
    }
};
//例如开门操作  对应代码为 0x00
int ret = LibDevModel.controlDevice(context, 0, device, null, callback);
```

4.20 startBackgroudMode(Context context,ScanCallBackSort scancallBack)

[**功能**]

LibDevModel 类的静态方法,扫描附近的蓝牙读头、一体机等，可用于靠近开门功能的实现。

返回值详细请参见《附表三》。

[**接口方法**]

public static int (Context context, ScanCallbackSort callback);

[**参数说明**]

context: 当前的上下文

callback: 扫描指令操作结束后回调的接口，返回结果详细请参见《附表三》。

[**APP 调用示例**]

在 Manifest 中申明

```
<service android:name="com.intelligoo.sdk.AutoOpenService" android:enabled="true" />
```

```
//调用代码
Intent autoService = new Intent(MainActivity.this, AutoOpenService.class);
getApplicationContext().startService(autoService);
ScanCallBackSort scancallBack = new ScanCallBackSort() {
@Override
public void onScanResultAtOnce(String devSn, int rssi) {
}
@Override
public void onScanResult(ArrayList<Map<String, Integer>> devSnRssiList) {
//执行开门以及筛选工作
}

AutoOpenService.startBackgroudMode(MainActivity.this, scancallBack);

//停止自动开门：
getApplicationContext().stopService(autoService);
```

4.21 getCardNumbersFromDevice

[**功能**]

LibDevModel 类的静态方法，获取设备里存储的卡号数据。

返回值详细请参见《附表三》。

[**接口方法**]

public static int getCardNumbersFromDevice(Context context, LibDevModel device, LibInterface.ReadCardCallback callback);

[**参数说明**]

context: 当前的上下文

device：操作的设备

callback: 读取卡号的回调。

[**APP 调用示例**]

```
//调用代码
final LibInterface.ReadCardCallback readCardCallBack = new LibInterface.ReadCardCallback()
```

```
{
    @Override
    public void onProgress(int cur, int all) {
        //读卡过程的回调，cur：已读取的卡数量   all：所有的卡数量
         }

    @Override
    public void onResult(int result, int all, ArrayList<String> cardnumbers) {
        //读卡结束，result：结果，详见《附表三》，cardnumbers：卡号数据
    }
};

     int setRet = LibDevModel.getCardNumbersFromDevice(MainActivity.this, libDevModel,readCardCallBack);
if (setRet != 0) {
    //失败，详见《附表三》
}
```

4.22 syncFingerPrintToDevice

[**功能**]

LibDevModel 类的静态方法，同步指纹数据到设备里，以实现指纹开门。

返回值详细请参见《附表三》。

[**接口方法**]

public static int syncFingerPrintToDevice(Context context, LibDevModel device, ArrayList<DMFingerprintModel> fingerprints, LibInterface.SyncFingerprintCallback callback)

[**参数说明**]

context: 当前的上下文

device：操作的设备

fingerprints：需要同步的指纹数据

callback: 同步指纹的回调。

[**APP 调用示例**]

```
//调用代码
LibInterface.SyncFingerprintCallback syncFingerprintCallback = new LibInterface.SyncFingerprintCallback()
{

    @Override
    public void onProgress(int cur, int all) {
        //同步过程的回调，cur：已同步的指纹数量，all：所有的指纹数量
    }
```

```
    @Override
    public void onResult(int result, int all) {
        //同步结束，result：结果，详见《附表三》，all：已同步的数量
    }
};

     int setRet = LibDevModel.syncFingerPrintToDevice(MainActivity.this, libDevModel, models, syncFingerprintCallback);
if (setRet != 0) {
    //失败，详见《附表三》
}
```

4.23 getOpenDoorRecordFromDevice

[**功能**]

LibDevModel 类的静态方法，获取设备里存储的开门记录。

返回值详细请参见《附表三》。

[**接口方法**]

public static int getOpenDoorRecordFromDevice(Context context, LibDevModel device, LibInterface.ReadOpenRecordCallback callback)

[**参数说明**]

context: 当前的上下文

device：操作的设备

callback: 读取记录的回调。

[**APP 调用示例**]

```
//调用代码
final LibInterface.ReadOpenRecordCallback readOpenRecordCallback = new LibInterface.ReadOpenRecordCallback(){

    @Override
    public void onProgress(int cur, int all) {
        //读取过程的回调，cur：已获取的记录数量，all：所有的记录数量
    }

    @Override
    public void onResult(int result, int all, ArrayList<Map> records) {
        /*读取结束的回调，result：结果，详见《附表三》，records：开门记录数据，具体字段如下：
            ConstantsUtils.DEVICEOPENDOOR_SERI: 序号，int 类型
            ConstantsUtils.DEVICEOPENDOOR_DATE:日期，String 类型 yyyyMMddHHmmss 格式
            ConstantsUtils.DEVICEOPENDOOR_TYPE：类型，int 类型，1：蓝牙开门、2：刷卡开
```

门、3：密码开门、4：远程开门、5：触摸开门、6：指纹开门

ConstantsUtils.DEVICEOPENDOOR_RESULT：结果 int 类型，0：成功、1：失败

ConstantsUtils.DEVICEOPENDOOR_CARDNO：卡号，String 类型（适用情况：刷卡开门、蓝牙开读头设备）

ConstantsUtils.DEVICEOPENDOOR_IDENTITY：用户 id，String 类型（适用情况：蓝牙开门，指纹开门）

ConstantsUtils.DEVICEOPENDOOR_PASSWORD：密码，String 类型（适用情况：密码开门）*/

}

};

int setRet = LibDevModel.getOpenDoorRecordFromDevice(MainActivity.this, libDevModel, readOpenRecordCallback);

if (setRet != 0) {

//失败，参见《附表三》

}

4.24 getRecentOpenDoorRecordFromDevice

[**功能**]

LibDevModel 类的静态方法，获取设备里存储的开门记录。

返回值详细请参见《附表三》。

[**接口方法**]

public static int getRecentOpenDoorRecordFromDevice(Context context, LibDevModel device, int readCount, LibInterface.ReadOpenRecordCallback callback)

[**参数说明**]

context: 当前的上下文

device：操作的设备

readCount: 调用一次接口读取的最大次数（读取一次最多 15 条记录），即若 readCount 为 5，则此接口最多读取 5 次，即最多返回 75 条记录，若想直接读取完所有未读取过的记录，则 readCount 设置为 0 即可

callback: 读取记录的回调。

[**APP 调用示例**]

可参考 4.23 调用示例，返回的记录格式也是通用的

4.25 cleanAllOpenDoorRecords

[**功能**]

LibDevModel 类的静态方法，清除设备里存储的开门记录。

返回值详细请参见《附表三》。

[**接口方法**]

public static int cleanAllOpenDoorRecords(Context context, LibDevModel device, ManagerCallback callback)

[参数说明]

context: 当前的上下文

device：操作的设备

callback: 清除记录的回调。

4.26 setDeviceStaticIP

[功能]

LibDevModel 类的静态方法，设置设备的静态 IP，适用于 V500 等设备。

返回值详细请参见《附表三》。

[接口方法]

public static int setDeviceStaticIP(Context context, LibDevModel device, Bundle bundle, final ManagerCallback callback)

[参数说明]

context: 当前的上下文

device：操作的设备

bundle：操作所需配置参数，详细看调用示例；

ConstantsUtils.DHCP_ENABLE：是否开启 DHCP，int 型，1：使用静态 IP，则需要以下 4 个字段，0：使用 DHCP，不需设置以下 4 个字段

ConstantsUtils.STATIC_IP: 静态 IP

ConstantsUtils.SUBNET_MASK: 子网掩码

ConstantsUtils.GATEWAY: 网关

ConstantsUtils.DNS_SERVER：DNS 服务器地址

callback: 回调。

[APP 调用示例]

```
//调用代码
ManagerCallback callBack = new ManagerCallback(){
    onResult(int ret, Bundle bundle){
    }
};
Bundle bundle = new Bundle();
bundle.putInt(ConstantsUtils.DHCP_ENABLE, 1);
bundle.putString(ConstantsUtils.STATIC_IP, "192.168.1.251");
bundle.putString(ConstantsUtils.SUBNET_MASK, "255.255.255.0");
bundle.putString(ConstantsUtils.GATEWAY, "192.168.1.1");
bundle.putString(ConstantsUtils.DNS_SERVER, "8.8.8.8");
int ret1 = LibDevModel.setDeviceStaticIP(MainActivity.this,device,bundle, callback);
if (ret1 != 0) {
    //失败，参见《附表三》
}
```

4.27 setServerIP

[功能]

LibDevModel 类的静态方法，设置设备连接的服务器地址，适用于 V500 等设备。

返回值详细请参见《附表三》。

[接口方法]

public static int setServerIP(Context context, LibDevModel device, Bundle bundle, final ManagerCallback callback)

[参数说明]

context: 当前的上下文

device：操作的设备

bundle：操作所需配置参数，详细看调用示例；

ConstantsUtils.SERVER_IP：服务器 IP

ConstantsUtils.SERVER_PORT: 服务器端口

callback: 回调。

[APP 调用示例]

```
//调用代码
ManagerCallback callBack = new ManagerCallback(){
    onResult(int ret, Bundle bundle){
    }
};
Bundle bundle = new Bundle();
bundle.putString(ConstantsUtils.SERVER_IP, "192.168.1.251");
bundle.putInt(ConstantsUtils.SERVER_PORT, 8089);
int ret = LibDevModel.setServerIP(MainActivity.this,libDev,bundle, callback);
if (ret != 0) {
    //失败，参见《附表三》
}
```

4.28 getDeviceSignal

[功能]

LibDevModel 类的静态方法，获取设备的信号值。

返回值详细请参见《附表三》。

[接口方法]

public static int getDeviceSignal(Context context, LibDevModel device, final ManagerCallback callback)

[参数说明]

context: 当前的上下文

device：操作的设备

callback：回调。

[**APP 调用示例**]

```
//调用代码
ManagerCallback callBack = new ManagerCallback(){
    onResult(int ret, Bundle bundle){
        if(ret == 0){
            //ConstantsUtils.DEVICE_SIGNAL：设备的信号值  int 类型
    }
};
int ret = LibDevModel.getDeviceSignal(MainActivity.this,libDev, callback);
if (ret != 0) {
    //失败，参见《附表三》
}
```

5. 常见问题及解决办法

(1)LibDevModel 的参数设置：sn，mac，ekey 的值不能为空。
(2)操作设备时，一直返回 48(连接超时)、回调无返回值:原因可能如下：

1)、设备不在周围

2)、AndroidManifest.xml 未配置蓝牙服务

3)、检查 controlDevice 方法的返回值是否为 0x00，否则无法进行设备操作

4)、设备正在通信过程中，controlDevice 返回 0x99（十进制 153），仅允许设备单次通信

6. 附录

附表一

Class	**LibDevModel**		
Attribute	**Column**	**DataType**	**Remark**
	devSn*	NSString/String	设备序列号，主键
	devMac*	NSString/String	设备 MAC
	devType*	int/int	设备类型（1 门禁读头，2 门禁一体机，3 梯控读头，4 无线联网锁，5 蓝牙遥控模块，6 门禁控制器，7 触摸开关门禁，8 二维码一体机，9 二维码读头，10 一体机（写卡），11 触摸开关门禁（WiFi），12 门禁一体机（WiFi、写卡），13 门禁一体机（WiFi）），必须
	privilege	int/int	管理者权限（1 超级管理员，2 管理员，4 普通用户），一体机使用
	openType	int/int	开门方式（1 手机，2 手机+卡，3 手机+密码）-- 目前不开放配置，默认手机开门
	verified	int/int	验证方式（1 有效期，2 次数，3 有效期+次数），一体机使用
	startDate	NSString/String	有效开始时间，格式：年月日时分秒，如：20151001121000
	endDate	NSString/String	冻结时间，格式：年月日时分秒，如：20161001121000
	useCount	int/int	使用次数，一体机使用
	eKey*	NSString/String	用户钥匙，传入云端给的 eKey
Static	public static int openDoor(Context context, LibDevModel device,		

Method	ManagerCallback callback);
	public static int syncDeviceTime(Context context, LibDevModel device, String time, ManagerCallback callback);
	public static int modifyPwd(Context context, LibDevModel device, String oldPwd, String newPwd, ManagerCallback callback);
	public static int swipeAddCardModel(Context context, LibDevModel device, ManagerCallback callback);
	public static int existSwipeCardAddModel(Context context, LibDevModel device, ManagerCallback callback);
	public static int swipeCardDeleteModel(Context context, LibDevModel device, ManagerCallback callback);
	public static int existSwipeCardDeleteModel(Context context, LibDevModel device, ManagerCallback callback);
	public static int setDeviceConfig(Context context, LibDevModel device, int wiegand, int openDelay,int controlWay ,ManagerCallback callback);
	public static int getDeviceConfig(Context context, LibDevModel device, ManagerCallback callback);
	public static int deleteDeviceData(Context context, LibDevModel device, ManagerCallback callback);
	public static int resetDeviceConfig(Context context, LibDevModel device, ManagerCallback callback);
	public static int writeCard(Context context, LibDevModel device,List<String> cardData,final ManagerCallback callback, oolean addAppend) ;
	public static int deleteCard(Context context, LibDevModel device,List<String> cardData,final ManagerCallback callback) ;
	public static int cleanCard(Context context, LibDevModel device, ManagerCallback callback);
	public static int configWifi(final Context context, LibDevModel device, Bundle bundle ,final ManagerCallback callback);
	public static int setReadSectorKey(final Context context, LibDevModel device, int devId , int mifareSector, String sectorKey, final ManagerCallback callback);
	public static int scanDevice(Context context, boolean atOnce, int sec, ScanCallback callBack);
	public static int controlDevice(Context context,int operation,LibDevModel device,Bundle bundle, ManagerCallback callback);

注：红色标记的 Column 是必须要设置的

附表二

operation	功能	说明

0x00	开门	
0x04	设置设备时间	ConstantsUtils.SYC_SET_TIME
0x05	修改密码	ConstantsUtils.MODIFY_OLD_PWD ConstantsUtils.MODIFY_OLD_PWD
0x06	进入登记卡模式	
0x07	进入删除卡模式	
0x08	退出卡登记模式	
0x09	退出卡删除模式	
0x0a	设置设备配置	ConstantsUtils.OPEN_DELAY （开门时间） ConstantsUtils.CONTROL （控制方式）
0x0b	获取设备配置	ConstantsUtils.OPEN_DELAY （开门时间） ConstantsUtils.CONTROL 控制方式（0 电气 1 电锁） ConstantsUtils.REG_CARDS_NUMS 登记的卡数量 ConstantsUtils.DEV_SYSTEM_VERSION 固件版本号 ConstantsUtils.MAX_CONTAINER 最大容量
0x0c	删除所有卡用户信息	
0x0d	系统配置初始化	
0x0e		
0x0f	批量登记卡	ConstantsUtils.CARD_NUMBER
0x10	批量删除卡（全部清空建议使用 0x0c）	ConstantsUtils.CARD_NUMBER

注:1、设置的参数和获取的参数使用 bundle 来传递，说明中给出的是 KEY 值，例如：设置开门时长：

int open_delay = 5;

bundle.putInt(ConstantsUtils.OPEN_DELAY, open_delay);

2、批量登记设备卡号时，需要使用韦根 36 读卡器读取卡号。

附表三 （返回值说明）

返回值	错误说明
通信返回值类	
0	操作成功
1	CRC 校验错误
2	通信命令格式错误
3	设备管理密码错误
4	ERROR_POWER(仅适用于锁)
5	数据读写错误
6	用户未注册在设备中
7	随机数检测错误

8	获取随机数错误
9	命令长度不匹配
10	未进入添加设备模式
11	devKey 检测错误
12	功能不支持
13	容量超过限制
48	通信连接超时
49	蓝牙服务未发现
50	通信数据长度错误，重新添加
51	接受数据为空
52	命令解析错误
53	未获取到随机数
54	未获取到配置子命令
55	未获取数据操作子命令
参数设置返回值	
-1	卡号为空
-2	Sn 为空
-3	Mac 为空
-4	E-Key 为空
-5	设备类型为空
-6	设备权限为空
-7	开门方式值错误
-8	验证方式值错误
-9	起始时间格式错误
-10	冻结时间格式错误
-11	使用次数未设置
-12	值未定义
-13	operation 其他功能未开放
-14	非法时间开门，即不在有效期内开门错误
-15	超过设置的开门距离
-16	韦根格式错误，当前仅支持 26 和 34
-17	开门时长值范围错误，仅支持 1-254 秒
-18	电器开关参数值错误，仅支持 0 电锁控制，1 电器开关
-19	密码必须为 6 位数字
-20	卡号列表不能为空
-21	卡号写入设备，每次不能大于 200 张卡
-22	扇区密钥必须是 16 进制字符串，并且长度为 12
-23	设备编号范围只能是 0-255
-24	卡扇区编号范围只能是 0-15
-25	scanTime 参数不能为空
-41	device 不能为 null

-42	context 不能为 null
-43	Device 过期
-44	Device 未到使用时间
其余错误返回值	
-100	不支持 BLE
-101	BLE 未打开
-102	指定的 SN 不存在
-103	蓝牙通信返回值为空
-104	开门失败
-105	设备未反应
-106	设备不在附近
-107	设备正在操作中
-108	sec 扫描时间单位错误
-109	设置扫描秒数超出范围
-110	设备已经存在超级用户，必须先初始化设备才能添加设备
-111	设备 MAC 地址错误
-112	使用蓝牙扫描太频繁（Android 7.0 的限制）