# 青莲云Android设备SDK 开发使用文档

| 版本 | 编写/修订说明 | 修订人 | 修订日期 | 备注 |
| --- | --- | --- | --- | --- |
| 3.0.0 | 创建文档 | 王科岩 | 20181220 | |
| 3.5.1 | 增加 OTA 信息回调<br>优化连接稳定性 | 王科岩 | 20190426 | |

# 目录

# 1 概要

青莲云作为领先的物联网安全服务商，为了让开发者不必关心数据的加密传输、网络链路的安全通信，适配了一系列 wifi、蓝牙、NB-IOT 模组。

我们提供了不同平台的嵌入式 SDK，开发者可以通过简单的调试，立即拥有强大的后端云能力，专注于具体业务研发。

各平台 SDK 可适用于联网硬件单品或网关类产品的开发。注意，开发硬件单品时没有“子设备”概念，全部涉及“子设备”的接口，subId 不设置即可。

本 SDK 提供以下功能的接口说明：

1) 安全快速联网
2) 连接状态报告
3) 网络时间同步
4) 获取用户/设备信息
5) 实时数据上报
6) 实时获取命令
7) 实时/离线自定义消息推送
8) OTA 升级
9) 本地存储/加载数据
10) 配置设备信息
11) 子设备上线/下线

## 2 sdk 目录结构

-\android_device_sdk

| | |
|---|---|
| -\debug | |
| -\ssl | debug 版 ssl 动态链接库 |
| -\aes | debug 版 aes 动态链接库 |
| -\release | |
| -\ssl | 正式版 ssl 动态链接库 |
| -\aes | 正式版 aes 动态链接库 |
| -android_device_sdk_demo.zip | Demo 示例 |
| -iot_device_sdk.jar | sdk jar 包 |

# 3 系统函数

## 3.1 初始化

初始化设备与云端交互的上下文环境，并与云端会建立长连接。当长连接断开时，会自动重连。请在 Application 中初始化，并且判断是主进程初始化，其他进程不能调用。

IotDeviceSDK.getInstance().init(Context context, long productId, String productKey, byte[] macAddr, String mcuVersion, int receiveBuffSize, int sendBufferSize, String serverAddr, int serverPort);

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| context | Context | 请传入 ApplicationContext |
| productId | int | 产品 id |
| productKey | String | 产品秘钥，云平台Th成，16 字节的十六进制编码 |
| macAddr | byte[] | Mac 地址 |
| mcuVersion | String | mcu 固件版本，"xx.xx"，0≤x≤9 |
| receiveBuffSize | int | 接收数据 buffer 大小，范围 1024-1048576，默认 1024 |
| sendBufferSize | int | 发送数据 buffer 大小，范围 1024-1048576，默认 1024 |
| serverAddr | String | 云服务器地址 |
| serverPort | int | 云服务器端口 |
| 返回值 | int | 0 ：成功；<br>-1：失败 |

### 3.2 设置设备运行状态

```
IotDeviceSDK.getInstance().resetFactory();

/**
 * @param timeout 超时时间，单位是秒
*/
IotDeviceSDK.getInstance().bindPermit(int timeout);

IotDeviceSDK.getInstance().unbind();
```

| 方法 | 说明 |
| --- | --- |
| resetFactory | 恢复出厂设置，会删除云端设备信息，并解除与 APP 端的绑定关系。 |
| bindPermit | 允许用户绑定设备，可在连云成功后调用。 |
| unbind | 设置解绑，设备会解除与 APP 端的绑定关系。 |

### 3.3 设备运行状态报告

```
// 设置恢复出厂的回调
IotDeviceSDK.getInstance().setResetFactoryListener(new
IotDeviceSDK.ResetFactoryListener()  {
        @Override
        public void onResetFactory() {
            //  恢复出厂成功，请让设备在此重启实现断开再连接的操作
        }
```

```
        });
// 设置连接状态的回调
IotDeviceSDK.getInstance().setConntionListener(new      IotDeviceSDK.ConntionListener()
{
                @Override
                public void onConnectServer() {
                    // 连接成功
                }

                @Override
                public void onDisConnectServer() {
                    // 连接断开
                }
            });
// 设置绑定设备的回调
IotDeviceSDK.getInstance().setBindDeviceListener(
new  IotDeviceSDK.BindDeviceListener() {
                @Override
                public void onBindSuccess() {
                // 用户把绑定成功
```

```
            }


            @Override
            public void onBindFaield() {
            //  用户绑定失败，超时或者设备未连接到网络
            }
        });
// 解绑设备成功
IotDeviceSDK.getInstance().setUserUnbondDeviceListener(new
IotDeviceSDK.UserUnbondDeviceListener()  {
            @Override
            public void onUnbond() {
                // 解绑成功
            }
        });
// 设置分享用户状态的回调
IotDeviceSDK.getInstance().setShareUserListener(new
IotDeviceSDK.ShareUserListener()  {
            @Override
            public void onIncrease() {
                // 分享用户增加
```

```
            }


            @Override
            public void onDecrease() {
                // 分享用户减少
            }
        });
// 设置授权限制回调
IotDeviceSDK.getInstance().setOnAuthLimitListener(new
IotDeviceSDK.OnAuthLimitListener()  {
            @Override
            public void onAuthLimit() {
                //  设备授权数达上限，无法通过云端认证
            }
        });
// 设备 mac 绑定到其他产品的回调
        IotDeviceSDK.getInstance().setOnMacBindOtherProductListener(new
IotDeviceSDK.OnMacBindOtherProductListener()  {
            @Override
            public void onMacBindOtherProduct() {
                //  设备绑定到其他产品上了，无法通过云端认证
```

```
        }

    });
```

### 3.4 获取网络时间

```
int timeStamp = IotDeviceSDK.getInstance().getTimeStamp();
```

| 参数 | 说明 |
| --- | --- |
| 返回值 | 0：时间无效<br>>0：实时网络时间戳(UNIX 时间戳) |

### 3.5 请求设备信息

```
// 请求设备信息

IotDeviceSDK.getInstance().getDeviceInfo();

// 请求绑定用户信息

IotDeviceSDK.getInstance().getMasterUserInfo();

// 请求分享用户信息

IotDeviceSDK.getInstance().getShareUserInfo();
```

### 3.6 接收设备信息

```
// 设置获取设备信息的回调
IotDeviceSDK.getInstance().setOnGetDeviceInfoListener(new
IotDeviceSDK.OnGetDeviceInfoListener() {
        @Override
        public void onGetDeviceInfo(int isBind, String sdkVersion) {
            // isBind  是否绑定  1 绑定  0 未绑定
            // sdkVersion SDK 版本  例如 03.02
        }
    });
// 设置获取绑定用户的回调
IotDeviceSDK.getInstance().setOnGetMasterUserInfoListener(new
IotDeviceSDK.OnGetMasterUserInfoListener() {
        @Override
        public void onGetMasterUserInfo(UserInfo userInfo) {
        UserInfo 字段
        public String id;              用户 id
        public String email;           用户邮箱地址
        public String countryCode;     国家地区区号  中国大陆 0086
        public String phone;           手机号
        public String name;            昵称
    });
// 设置获取分享用户的回调
IotDeviceSDK.getInstance().setOnGetShareUserInfoListener(new
IotDeviceSDK.OnGetShareUserInfoListener() {
        @Override
        public void onGetShareUserInfo(UserInfo[] userInfos) {
        UserInfo 字段
```

```
        public String id;                   用户 id
        public String email;                用户邮箱地址
        public String countryCode;          国家地区区号 中国大陆 0086
        public String phone;                手机号
        public String name;                 昵称
    });
```

## 4 传输数据

发送/接收数据的最大长度与 init 中的 buffer 的大小有关。

如：buffer 设置为 2048，以类型为整型数据点为例，最多可以一次性发送 600 个整型数据。

### 4.1 上传数据

- 开发者需明确每个数据点的 dpid、类型(通过云平台获取)。根据数据点的类型，调用不同的函数将数据点 id、对应数值添加到发送队列中。
- 目前支持的类型包括整数型、布尔型、枚举型、浮点型、字符型、故障型、二进制。
- 上传数据时，需保证在云端创建的数据点是可上报的。

```
// 构造消息并发送

new CloudMessage.Builder()
                .putInt(1, 1) // 发送整型
                .putBool(1, 1) // 发送 bool
                .putEnum(1, 1) // 发送枚举
                .putFloat(1, 1) // 发送浮点型
                .putString(1, "1") // 发送字符串
                .putBinary(1, new byte[3]) // 发送二进制类型
                .putError(1, "1") // 发送错误类型
                .setSubId("子设备 id") // 设置子设备 id(如果当前消息是子设
```

备发出的需要设置子设备 id)

```
        .send(new CloudMessage.OnSendListener() {
            @Override
            public void onSuccess() {
                // 发送成功
            }

            @Override
            public void onError(String message) {
                // 发送失败
                // message 错误原因
            }
        });
```

### 4.2 接收数据

- 开发者需明确每个数据点的 dpid、类型(通过云平台获取)。根据数据点的类型，调用不同的转换函数转换成需要的数值。
- 目前支持的类型包括整数型、布尔型、枚举型、浮点型、字符型、故障型、二进制。
- 接收数据时，需保证在云端创建的数据点是可下发的。

```
IotDeviceSDK.getInstance().setOnCommandListener(new
IotDeviceSDK.OnCommandListener() {
        @Override
        public void onCommand(String subId, int dataId, Object value) {
```

```
        // subId 子设备 id 可能为空，当消息对子设备发送的时候不为空
        //dataid 数据点的 id
        // value 数据的值，当前数据是什么类型可以强制转换
        //  例如，DataPointType.getValue 工具方法中会强制转换
    if (dataId == INT_ID) { // 强制转换的数据类型需要和数据点的 id 匹配
            int data = DataPointType.getValue(value);
        } else if (dataId == BYTE_ARRAY_ID) {
            byte[] data = DataPointType.getValue(value);
        } else if (data == FLOAT_ID) {
            float data = DataPointType.getValue(value);
        }               }
});
```

# 5 ota 固件升级

如有远程升级需求，请参考具体 ota 升级流程文档及本小节内容进行实现

## 5.1 设置ota 属性

IotDeviceSDK.getInstance().setOTAOption(int exceptTime, int chunkSize);

| 参数 | 说明 |
| --- | --- |
| exceptTime | 期望升级倒计时，单位秒，范围 120-3600 |
| chunkSize | 云端会将固件按 chunkSize 分块，分块后一次下发一块。<br>2 的 n 次幂，范围 256-1048576，默认 1024 |
| 返回值 | 0 ：设置属性成功，可在 log 中查看实际升级时间<br>-1 ：失败，参数错误 |

## 5.2 接收固件信息

sdk 自动调用该函数，接受云端发来的固件信息说明，包含，固件的种类、固件的大小和固件的 OTA 版本。

```
IotDeviceSDK.getInstance().setOnGetOTAInfoListener(new
IotDeviceSDK.OnGetOTAInfoListener() {
        @Override
        public void onGetOTAInfo(int otaType, long otaFileSize, String
 otaVersion) {
            Log.e("TAG", "otaType：" + otaType); // 0：wifi 固件，1：MCU 固件
            Log.e("TAG", "otaFileSize：" + otaFileSize); // OTA 固件大小
            Log.e("TAG", "otaVersion：" + otaVersion); // OTA 固件版本
        }
    });
```

## 5.3 接收固件数据块

sdk 自动调用该函数，接受云端发来的固件数据块，收到后更新本地固件。回调函数中不可执行太多耗时代码。

```
IotDeviceSDK.getInstance().setOnOTAChunkDataListener(new
IotDeviceSDK.OnOTAChunkDataListener() {
        @Override
        public boolean onOTAChunkData(boolean isLastChunk, long
chunkOffset, byte[] chunkData) {

            isLastChunk 表示是否是最后一个包;

            chunkOffset 表示本次数据包在总数据量的偏移起始位置;

            chunkData 表示本次数据包数据

            // return true 或者 return false
            // 返回 true 表示本次 chunk 包处理成功
            // 返回 false 表示本次 chunk 包处理失败
        }
    });
```

## 5.4 接收升级指令

收到此命令，重启，运行新版本固件。回调函数中不可执行太多耗时代码。

```
IotDeviceSDK.getInstance().setOnOTAUpgradeListener(new
IotDeviceSDK.OnOTAUpgradeListener() {
        @Override
```

```
        public void onOTAUpgrade() {


        }
    });
```

# 6 高级功能

## 6.1 透传自定义数据

- 可透传任意格式自定义数据，请与 app 开发者自行约定
- 手机端在线，云端不保存数据，直接转发至手机
- 手机端离线，云端最多会保存 20 条数据，待手机上线时发送，发送后清空

```
// 透传自定义消息，不可与前边上报数据同时使用
new  CloudMessage.Builder()
                .setPushData(new  byte[10])
                .push(new  CloudMessage.OnPushListener() {
                    @Override
                    public void onSuccess() {
                        // 发送成功
                    }

                    @Override
                    public void onError(String message) {
                        // 发送失败
                        // message 错误原因
```

```
        }

    });
```

## 6.2 接收自定义数据

- 接收来自 app 的透传数据，格式请与 app 开发者自行约定
- 设备在线，云端收到 app 数据后，直接透传至设备
- 设备离线，云端最多会保存 20 条 app 数据，待设备上线时发送，发送后清空

回调函数中不可执行太多耗时代码。

```
IotDeviceSDK.getInstance().setOnPushReceiveListener(new
IotDeviceSDK.OnPushReceiveListener() {
        @Override
        public void onPushReceive(String subId, int timeStamp, byte[] data) {
            // subId 子设备 id，透传给子设备时该字段不为空
            // timestamp 云端接收到消息的时间戳，如果不需要考虑数据超时则不
用处理该字段
            //data 自定义数据
        }
    });
```

## 6.3 配置设备信息

注意！请在调用 iot_start 前调用此函数。

```
// 设置加密方式
```

IotDeviceSDK.getInstance().setEncryptType(String type);

// 设置设备序列号

IotDeviceSDK.getInstance().setSN(String sn);

| 参数 | 说明 |
| --- | --- |
| type | "AES"：AES 算法（默认）；<br>"SSL"：SSL 算法；<br>"SM4"：SM4 算法 |
| sn | 自定义字符串，数字字母组合，最长 16 字节 |

① sn 表示设置第三方序列号，厂商可设置自定义格式的设备唯一序列号，此功能可用于实现维修设备替换通信模块时，用户设备历史数据不变的功能。

注意！同一产品（product_id 相同）下，该序列号须保证唯一，如需更改 SN，请先删除设备。

例如：IotDeviceSDK.getInstance().setSN("SNDQ201811200003")；设置设备的序列号为 SNDQ201811200003，此序列号可唯一标识一个设备。

# 7 子设备功能

## 7.1 子设备上线

```
// 子设备上线
        new  CloudMessage.Builder()
                .putSubDevice(new SubDevice("Test1", "Test", "01.01",  1))
                .putSubDevice(new SubDevice("Test2", "Test", "01.02", 1))
                .putSubDevice(new SubDevice("Test3", "Test", "01.02", 2))
                .activeSubDevice(new  CloudMessage.OnActiveSubDeviceListener()  {
                    @Override
                    public void onSuccess() {
                        // 发送成功
                    }

                    @Override
                    public void onError(String message) {
                        // 发送失败
                        // message 错误原因
                    }
                });
SubDevice subDevice = new SubDevice(String subId, String subName, String
subversion, intsubType);
```

注：关于一次可以同时上线多少个子设备，代码中不做限制，但是和初始化函数中的 sendBufferSize 参数大小有关。

| 参数 | 说明 |
| --- | --- |
| subId | 自定义的子设备 id，仅限字母数字组合，同一产品下不可重复 |
| subName | 子设备名称，仅限字母数字组合 |

| | |
|---|---|
| subVersion | 子设备固件版本，"xx.xx"，0≤x≤9 |
| subType | 子设备类型，请与 APP 端自行约定 |

### 7.2 子设备离线

子设备离线时，调用此接口。

```
// 子设备离线，每次只能传入一个子设备id
        new  CloudMessage.Builder()
                .setInactiveSubId("Test1")
                .inactiveSubDevice(new
CloudMessage.OnInactiveSubDeviceListener()  {
                    @Override
                    public void onSuccess() {
                        // 发送成功
                    }

                    @Override
                    public void onError(String message) {
                        // 发送失败
                        // message 错误原因
                    }
                });
```