杰理 OTA 库(Android 版)
开发说明文档

珠海市杰理科技股份有限公司
Zhuhai Jieli Technology Co.,LTD

声明

- 本项目所参考、使用技术必须全部来源于公知技术信息，或自主创新设计。
- 本项目不得使用任何未经授权的第三方知识产权的技术信息。
- 如个人使用未经授权的第三方知识产权的技术信息，造成的经济损失和法律后果由个人承担。

修改记录

版本	更新日期	描述
0.2.0	2022/05/17	1. 制定文档大纲 2. 增加 SDK 框架，工程介绍，开发说明，接口说明,其他的章节 3. 丰富代码示例，错误码说明

目录

一、概述

本文档由**杰理官方**推出，从 SDK 框架/ 工程介绍 / 开发说明 / 接口说明 / 其他 等章节来帮助开发者快速、便捷地使用杰理 OTA 库进行开发。

杰理 OTA 库封装了基于杰理 RCSP 协议的 OTA 流程，因此杰理 OTA 库需要依赖杰理 RCSP 库实现。

杰理 OTA 流程有以下几个优点：

- 成熟稳定可靠
- 支持单双备份升级
- 接入简便快捷

（一）SDK 架构

图 1.1 - 1

说明：

1. **OTA 流程**依赖 **RCSP 协议层**实现，**RCSP 协议层**依赖 **IO 代理层**实现。
2. 用户只需实现 **IO 代理层**的状态和数据回传，发送数据等接口，即可实现杰理 OTA 流程。
3. **OTA 流程**的实现是基于 **RCSP 协议层**，因此用户需要等待 **RCSP 协议层**初始化完成才能使用接口。
4. 用户实现发送数据，需要 MTU 限制。

（二）工程介绍

1. 开发资料文件结构

```
apk  ---  测试 APK 文件夹
 ├── 测试 APK
code ---  参考源码工程文件夹
 ├── 参考 Demo 源码工程
doc ---  开发文档文件夹
 ├── 杰理 OTA 库(Android 版)开发说明外发文档.pdf      ---  讲解 OTA 库的开发使用
libs ---  核心库文件夹
 ├── jl_usb_dongle_Vxxx.aar    ---  杰理 USB 通讯相关
 ├── jl_rcsp_main_Vxxx.aar     ---  杰理 RCSP 协议相关
 └── jl_ota_Vxxx.aar           ---  杰理 OTA 相关
```

2. 示例工程结构

```
com
└── jieli
    └── ota      --- 包名
        ├── data                --- 数据层
        │   ├── constant            --- 常量定义
        │   └── model               --- 数据模型
        ├── tool                --- 功能实现层
        │   ├── config              --- 配置缓存
        │   ├── dongle              --- Dongle 实现
        │   │   ├── callback            --- 回调
        │   │   ├── model               --- 数据模型
        │   │   ├── tool                --- 辅助类
        │   │   ├── util                --- 工具类
        │   │   └── DongleManager       --- 实现类
        │   └── ota                 --- OTA 实现
        │       ├── bean                --- 数据模型
        │       └── OTAManager          --- 实现类
        ├── ui                  --- UI 层
        │   ├── adapter             --- 适配器
        │   ├── dialog              --- 弹窗
        │   ├── home                --- 主页 UI
        │   └── widget              --- 自定义控件
        ├── util                --- 工具层
        └── MainApplication     --- 应用层
```

3. 功能实现参考

重点

- **dongle** 实现，请参考 `com.jieli.ota.tool.dongle.DongleManager`
- **OTA** 实现，请参考 `com.jieli.ota.tool.ota.OTAManager`

二、开发说明

（一）依赖库导入

- jl_ota_Vxxx.aar : OTA 流程封装
- jl_rcsp_main_Vxxx.aar : RCSP 协议封装

提示

- xxx 为版本号，请以最新发布版本为准

（二）接入流程

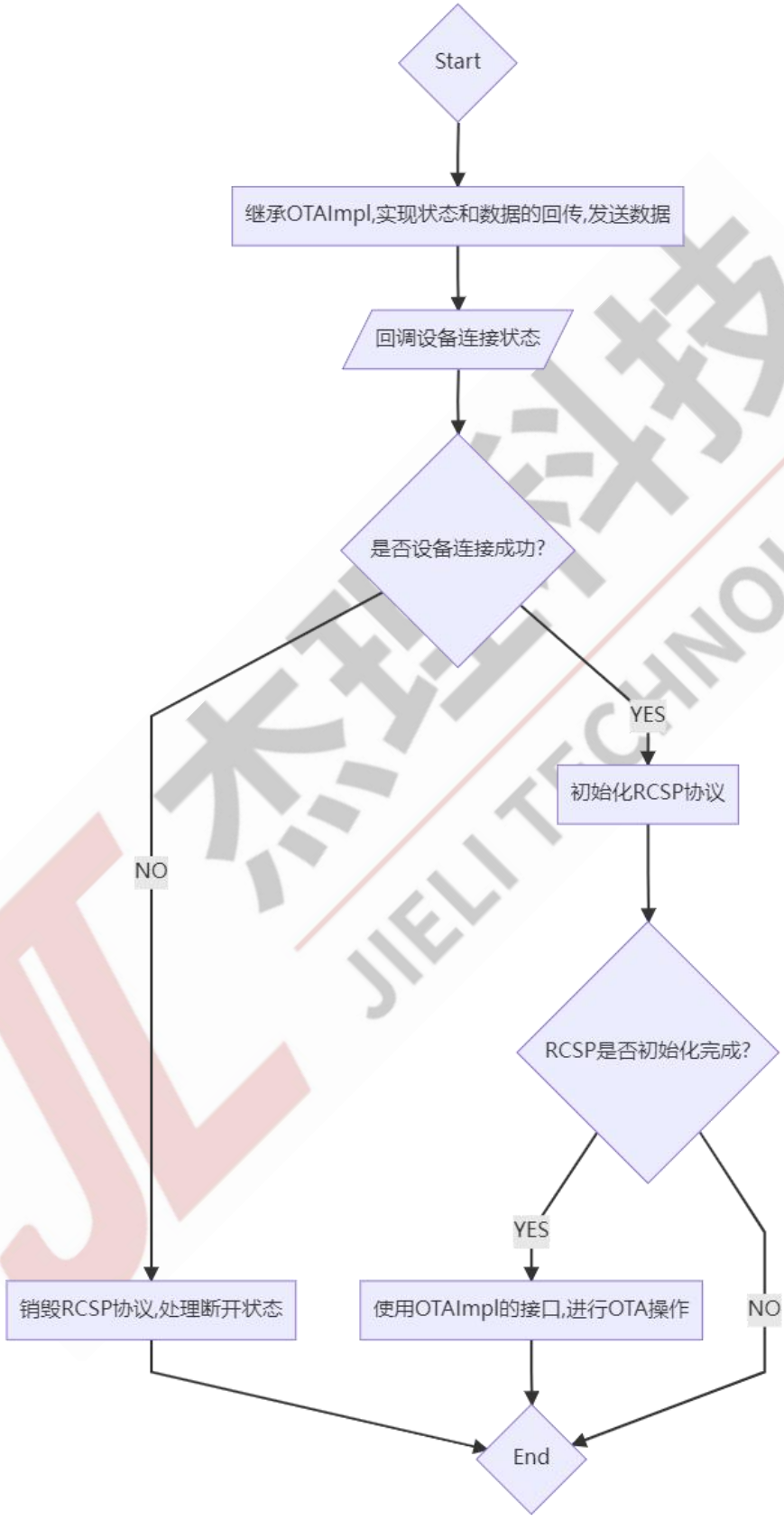
Start
继承OTAImpl,实现状态和数据的回传,发送数据
回调设备连接状态
是否设备连接成功?
NO
YES
初始化RCSP协议
RCSP是否初始化完成?
YES
NO
销毁RCSP协议,处理断开状态
使用OTAImpl的接口,进行OTA操作
End

图 2.2-1

（三）开发流程

我们的开发流程与上图的接入流程图相似

步骤一，继续 OTAImpl，实现状态和数据的回传,发送数据等接口

步骤二，增加 OTA 选项，实例化 OTAImpl 的对象

步骤三，监听 RCSP 状态，等待 RCSP 协议初始化完成

步骤四，进行 OTA 操作

步骤五，移除监听器，销毁 OTAImpl 的对象，回收资源

1．继承 OTAImpl，实现其方法

重点

- 调用接口，实现设备状态和数据的回传
- 实现发送数据接口，注意 **MTU 分包**
- 用户可以完成与设备的交互才回调设备连接成功状态

示例代码

```
1.  public static class OTAHelper extends OTAImpl {
2.      //TODO: 通讯类实现，用户自定义实现，可以是 BLE，SPP 或者 USB 等等
3.      private final DongleManager dongleManager = DongleManager.getInstance()
    ;
4.      //OTA 配置选项
5.      private final OTAOption otaOption; //是否需要设备认证根据固件程序决定，默认
    公版是开启设备认证的
6.      //设备认证处理类
7.      private final RcspAuth rcspAuth;
8.
9.      public OTAHelper(OTAOption option) {
10.         otaOption = option;
11.         rcspAuth = new RcspAuth(this::sendDataToDevice, new RcspAuth.OnRcsp
    AuthListener() {
12.             @Override
13.             public void onInitResult(boolean result) {
14.
```

```
15.             }
16.
17.             @Override
18.             public void onAuthSuccess(Device device) {
19.                 otaOption.setDeviceAuthPass(true);
20.                 //TODO: 回传设备上线
21.                 transmitDeviceStatus(device, Connection.CONNECTION_CONNECTE
    D);
22.             }
23.
24.             @Override
25.             public void onAuthFailed(Device device, int code, String messag
    e) {
26.                 //TODO: 回传设备下线
27.                 callbackDeviceConnectFailed(device);
28.             }
29.         });
30.         dongleManager.addOnDongleEventCallback(mOnDongleEventCallback);
31.     }
32.
33.     @Override
34.     public boolean sendDataToDevice(Device device, byte[] data) {
35.         //TODO: 用户必须实现数据发送
36.         //可以是阻塞方法，也可以是异步函数处理
37.         RemoteDevice remoteDevice = dongleManager.findRemoteDeviceByChannel
    (otaOption.getChannel());
38.         if (null == remoteDevice) return false;
39.         dongleManager.sendRcspData(remoteDevice, data, new OnResultListener
    <Boolean>() {
40.             @Override
41.             public void onResult(Boolean result) {
42.
43.             }
44.
45.             @Override
46.             public void onFailed(int code, String message) {
47.
48.             }
49.         });
50.         return true;
51.     }
52.
53.     @Override
54.     public void destroy() {
```

```
55.         super.destroy();
56.         rcspAuth.destroy();
57.         dongleManager.removeOnDongleEventCallback(mOnDongleEventCallback);
58.     }
59.
60.     //设备是否通过认证
61.     public boolean isDeviceAuthPass() {
62.         if (!otaOption.isUseAuthProgress()) return true;
63.         return otaOption.isDeviceAuthPass();
64.     }
65.
66. //回调设备连接失败
67.     private void callbackDeviceConnectFailed(Device device) {
68.         otaOption.setDeviceAuthPass(false);
69.         transmitDeviceStatus(device, Connection.CONNECTION_DISCONNECT);
70.     }
71.
72.     private final OnDongleEventCallback mOnDongleEventCallback = new OnDong
    leEventCallback() {
73.         @Override
74.         public void onUsbDeviceState(UsbDevice device, boolean isOnLine) {
75.             super.onUsbDeviceState(device, isOnLine);
76.             if (!isOnLine && getUsingDevice() != null) {
77.                 //TODO: 回传设备下线
78.                 callbackDeviceConnectFailed(getUsingDevice());
79.             }
80.         }
81.
82.         @Override
83.         public void onRemoteDevicesChange(List<RemoteDevice> list) {
84.             super.onRemoteDevicesChange(list);
85.             //TODO: 回传设备下线
86.             if (list.isEmpty() && getUsingDevice() != null) {
87.                 callbackDeviceConnectFailed(getUsingDevice());
88.                 return;
89.             }
90.             RemoteDevice device = null;
91.             for (RemoteDevice remoteDevice : list) {
92.                  if (remoteDevice.getChannelID() == otaOption.getChannel())
    {
93.                     device = remoteDevice;
94.                     break;
95.                 }
96.             }
```

```
97.             if (null == device) return;
98.             OTADevice otaDevice = OTADevice.changeOTADevice(device);
99.             if (device.getState() == OTAConstant.STATE_DEVICE_OFFLINE) {
100.                //TODO: 回传设备下线
101.                callbackDeviceConnectFailed(otaDevice);
102.            } else if (device.getState() == OTAConstant.STATE_DEVICE_ONLINE
    ) {
103.                if (isDeviceAuthPass()) { //设备已通过认证
104.                    //TODO: 回传设备上线
105.                    transmitDeviceStatus(otaDevice, Connection.CONNECTION_C
    ONNECTED);
106.                } else { //设备未通过认证
107.                    rcspAuth.stopAuth(otaDevice, false);
108.                    if (!rcspAuth.startAuth(otaDevice)) {
109.                        //TODO: 回传设备下线
110.                        callbackDeviceConnectFailed(otaDevice);
111.                    }
112.                }
113.            }
114.        }
115.
116.        @Override
117.        public void onRcspData(int channel, byte[] data) {
118.            super.onRcspData(channel, data);
119.            if (channel != otaOption.getChannel() || (null == data || data.
    length == 0)) return;
120.            //TODO: 回传接收到的设备数据
121.            if (getUsingDevice() != null) {
122.                if (isDeviceAuthPass()) {  //设备已通过认证
123.                    transmitDeviceData(getUsingDevice(), data);
124.                } else { //设备未通过认证
125.                    rcspAuth.handleAuthData(getUsingDevice(), data);
126.                }
127.            }
128.        }
129.    };
130.}
```

2．OTA 选项配置

重点

- 是否使用设备认证，需要根据固件程序决定，公版程序默认开启
- 是否通过设备认证，判断设备是否已认证

示例代码

```
1. //用户可以自行实现，这个配置选项只是参考作用
2. OTAOption option = new OTAOption(channel); //使用通道
3. option.setUseAuthProgress(true);     //是否使用设备认证流程
4. option.setDeviceAuthPass(false);     //是否通过设备认证
5. //设备认证具体实现，请参考 2.3.2 的实例代码
```

设备认证

```
1.  //初始化 RcspAuth 实现类
2.  //需要实现发送接口 和 添加结果回调监听器
3.  RcspAuth rcspAuth = new RcspAuth(this::sendDataToDevice, new RcspAuth.On
    RcspAuthListener() {
4.      @Override
5.      public void onInitResult(boolean result) {
6.          //回调初始化结果
7.      }
8.
9.       @Override
10.      public void onAuthSuccess(Device device) {
11.          //回调认证成功
12.      }
13.
14.      @Override
15.      public void onAuthFailed(Device device, int code, String message) {
16.          //回调认证失败
17.      }
18. });
19. //认证流程: 开始认证 -> 处理认证数据 -> 回调认证结果
20. //第一步，开始认证
21. rcspAuth.stopAuth(otaDevice, false); //停止认证并不回调结果
22. boolean ret = rcspAuth.startAuth(otaDevice); //开始认证
23. //第二步，处理认证数据
24. rcspAuth.handleAuthData(device, data);
25. //第三步，处理认证结果
26.
27. //可以主动取消设备认证，视为失败
28. rcspAuth.stopAuth(otaDevice);
29. //不需要使用设备认证时，销毁设备认证实现类
30. rcspAuth.destroy();
```

3. 初始化 OTA 管理类

重点

- 需要监听 RCSP 库是否初始化成功，初始化成功后才能操作 OTA 接口

示例代码

```
OTAOption option = new OTAOption(channel); //使用通道
option.setUseAuthProgress(true);    //是否使用设备认证流程
option.setDeviceAuthPass(false);    //是否通过设备认证
final OTAHelper otaHelper = new OTAHelper(option);
otaHelper.addOnRcspCallback(new OnRcspCallback() {
    @Override
    public void onRcspInit(Device device, boolean isInit) {
        super.onRcspInit(device, isInit);
        //TODO: 判断 RCSP 库初始化是否成功
        if (isInit) {
            //操作 OTA 接口
//          otaHelper.startOTA();
        }
    }
});
otaHelper.isOTA(); //是否正在 OTA
otaHelper.isDeviceConnected(); //是否已连接并初始化 RCSP 成功
//   otaHelper.isTargetDevice(Device device); //是否目标设备
```

4. OTA 操作

重点

- 需要监听 RCSP 库是否初始化成功，初始化成功后才能操作 OTA 接口

示例代码

```
1.  OTAOption option = new OTAOption(channel); //使用通道
2.  option.setUseAuthProgress(true);     //是否使用设备认证流程
3.  option.setDeviceAuthPass(false);     //是否通过设备认证
4.  final OTAHelper otaHelper = new OTAHelper(option);
5.  OTAConfig config = new OTAConfig();
6.  config.setCommunicationWay(OTAConfig.COMMUNICATION_WAY_BLE);  //通讯方式
7.  config.setUpdateFilePath(filePath);                          //升级文件存放路径
8.  //        config.setUpdateFileData(fileData);                //升级文件数据， 如果设置升级数据，优先此方式，两者选其一
9.  config.setSupportNewRebootWay(false);                        //是否支持新的回连方式，根据固件程序设置
10. otaHelper.startOTA(config, new OnUpgradeCallback() {
11.     @Override
12.     public void onStartOTA(Device device) {
13.         //回调开始OTA
14.     }
15.
16.     @Override
17.     public void onNeedReconnect(Device device, ReConnectMsg reConnectMsg) {
18.         //回调需要回连设备
19.     }
20.
21.     @Override
22.     public void onProgress(Device device, UpgradeType type, float progress) {
23.         //回调升级进度
24. //        UpgradeType.UPGRADE_TYPE_CHECK_FILE     //可能是校验文件，也可能是下载boot loader
25. //        UpgradeType.UPGRADE_TYPE_FIRMWARE       //升级固件程序
26.     }
27.
```

```
28.     @Override
29.     public void onStopOTA(Device device) {
30.         //回调升级完成 -- 意味着OTA升级成功
31.     }
32.
33.     @Override
34.     public void onCancelOTA(Device device) {
35.         //回调用户取消OTA
36.     }
37.
38.     @Override
39.     public void onError(Device device, int error, String message) {
40.         //回调升级过程的异常情况
41.         //error 参考 OTAError 类
42.     }
43. });
44.
45. //用户主动取消OTA
46. //返回操作结果
47. //如果设备是单备份OTA，不允许主动取消OTA，操作无效
48. //如果设备是双备份OTA，则可以主动取消OTA
49. boolean ret = otaHelper.cancelOTA();
50.
51. otaHelper.isOTA();//是否正在OTA
52. otaHelper.isDeviceConnected(); //是否已连接并初始化RCSP成功
53. //不再使用OTA功能时，销毁OTA实现类，释放资源
54. otaHelper.destroy();
```

三、接口说明

（一）IO 代理接口

非常重要

用于代理 IO 操作， 透传设备的连接状态和数据，实现与设备的交互，是数据交互的基础

```
1.  public interface IOProxy {
2.
3.      /* ============================================================ *
4.       * TODO: //需要用户调用，实现数据传输
5.       * ============================================================ */
6.
7.  /**
8.       * 传递设备的状态
9.       *
10.      * @param device 操作设备
11.      * @param status 设备状态
12.      *               <p>说明:
13.      *               1. 设备状态由库内定义，参考{@link }。
14.      *               2. 用户必须传入对应的状态码</p>
15.      */
16.     void transmitDeviceStatus(Device device, Connection status);
17.
18.     /**
19.      * 传递从设备接收到的数据
20.      *
21.      * @param device 操作设备
22.      * @param data   原始数据
23.      */
24.     void transmitDeviceData(Device device, byte[] data);
25.
26.
27.     /* ============================================================ *
28.      * TODO: //需要用户实现
29.      * ============================================================ */
```

```
30.
31.     /**
32.      * 向设备发送 RCSP 数据包
33.      *
34.      * @param device 操作设备
35.      * @param data   RCSP 数据包
36.      * @return 结果
37.      * <p>
38.      * 说明:
39.      * 1. 该方法需要用户自行实现
40.      * 2. 该方法运行在子线程，允许阻塞处理
41.      * 3. 该方法会回调完整的一包 RCSP 数据，用户实现需要根据实际发送 MTU 进行分包处理
42.      * </p>
43.      */
44.     boolean sendDataToDevice(Device device, byte[] data);
45. }
```

Device： 标识设备， 必须有唯一标识, mac

（二）升级流程的回调

应用于开始 OTA 的接口，回调 OTA 流程中各种事件

```
1.  public interface OnUpgradeCallback {
2.
3.      /**
4.       * OTA 开始
5.       */
6.      void onStartOTA(Device device);
7.
8.      /**
9.       * 需要回连的回调
10.      * <p>
11.      * 注意：1.仅连接通讯通道（BLE or  SPP）
12.      * 2.用于单备份 OTA</p>
13.      *
14.      * @param reConnectMsg 回连设备信息
15.      */
16.     void onNeedReconnect(Device device, ReConnectMsg reConnectMsg);
17.
18.     /**
19.      * 进度回调
20.      *
21.      * @param type     类型
22.      * @param progress 进度
23.      */
24.     void onProgress(Device device, UpgradeType type, float progress);
25.
26.     /**
27.      * OTA 结束
28.      */
29.     void onStopOTA(Device device);
30.
31.     /**
32.      * OTA 取消
33.      */
34.     void onCancelOTA(Device device);
35.
36.     /**
```

```
37.      * OTA 失败
38.      *
39.      * @param error   错误码
40.      * @param message 错误信息
41.      */
42.     void onError(Device device, int error, String message);
43. }
```

（三）RCSP 状态回调

作用于回调 RCSP 的状态，设备状态，接收到的设备命令等

```
1.  public abstract class OnRcspCallback {
2.
3.      /**
4.       * Rcsp 协议初始化回调
5.       *
6.       * @param device 已连接设备
7.       * @param isInit 初始化结果
8.       */
9.      public void onRcspInit(Device device, boolean isInit) {
10.
11.     }
12.
13.     /**
14.      * 设备发送的 rcsp 命令回调
15.      *
16.      * @param device  已连接设备
17.      * @param command RCSP 命令
18.      */
19.     public void onRcspCommand(Device device, Command command) {
20.
21.     }
22.
23.     /**
24.      * 设备发送的数据命令回调
25.      *
26.      * @param device  已连接设备
27.      * @param dataCmd 数据命令
28.      */
29.     public void onRcspDataCmd(Device device, Command dataCmd) {
30.
31.     }
32.
33.     /**
34.      * RCSP 错误事件回调
35.      *
36.      * @param device  设备对象
```

```
37.      * @param error   错误码 (参考
   {@link com.jieli.rcsp.data.constant.ErrorCode})
38.      * @param message 错误描述
39.      */
40.     public void onRcspError(Device device, int error, String message) {
41.
42.     }
43.
44.     /**
45.      * 需要强制升级回调
46.      *
47.      * @param device 需要强制升级的设备
48.      */
49.     public void onMandatoryUpgrade(Device device) {
50.
51.     }
52.
53.     /**
54.      * 设备连接状态
55.      *
56.      * @param device 蓝牙设备
57.      * @param status 连接状态
58.      */
59.     public void onConnectStateChange(Device device, Connection status) {
60.
61.     }
62. }
```

（四）命令结果回调

它是发送 RCSP 命令的结果回调

```
1.  public interface CommandCallback<C extends Command> {
2.      /**
3.       * 回调回复命令
4.       *
5.       * @param device  操作对象
6.       * @param command 回复命令
7.       *                <p>说明:
8.       *                1. 若有回复的命令, 返回的是带回复数据的命令对象
9.       *                2. 若无回复的命令, 返回的是命令原型</p>
10.      */
11.     void onCmdResponse(Device device, C command);
12.
13.     /**
14.      * 回调错误事件
15.      *
16.      * @param device  操作对象
17.      * @param code    错误码 (参考
    {@link com.jieli.rcsp.data.constant.ErrorCode})
18.      * @param message 错误描述
19.      */
20.     void onError(Device device, int code, String message);
21. }
```

（五）错误码

标识错误内容

对应类：`com.jieli.ota.data.constant.OTAError`

错误码	对应值	描述
-0x01	ERROR_UNKNOWN	未知错误
0x00	ERROR_NONE	没有错误
-0x02	ERROR_INVALID_PARAM	无效参数
-0x03	ERROR_DATA_FORMAT	数据格式错误
-0x04	ERROR_NOT_FOUND_RESOURCE	没有找到资源
-0x20	ERROR_UNKNOWN_DEVICE	未知设备
-0x21	ERROR_DEVICE_OFFLINE	设备下线
-0x23	ERROR_IO_EXCEPTION	IO 异常
-0x24	ERROR_REPEAT_STATUS	重复状态
-0x40	ERROR_RESPONSE_TIMEOUT	等待回复命令超时
-0x41	ERROR_REPLY_BAD_STATUS	设备回复不好的状态
-0x42	ERROR_REPLY_BAD_RESULT	设备回复错误的结果
-0x43	ERROR_NONE_PARSER	没有对应的解析器
-0x61	ERROR_OTA_LOW_POWER	设备低电压
-0x62	ERROR_OTA_UPDATE_FILE	升级固件信息错误
-0x63	ERROR_OTA_FIRMWARE_VERSION_NO_CHANGE	升级文件的固件版本一致
-0x64	ERROR_OTA_TWS_NOT_CONNECT	TWS 未连接
-0x65	ERROR_OTA_HEADSET_NOT_IN_CHARGING_BIN	耳机未在充电仓
-0x66	ERROR_OTA_DATA_CHECK_ERROR	升级数据校验出错

-0x67	ERROR_OTA_FAIL	升级失败
-0x68	ERROR_OTA_ENCRYPTED_KEY_NOT_MATCH	加密 key 不匹配
-0x69	ERROR_OTA_UPGRADE_FILE_ERROR	升级文件出错
-0x6A	ERROR_OTA_UPGRADE_TYPE_ERROR	升级类型出错
-0x6B	ERROR_OTA_LENGTH_OVER	升级过程中出现长度错误
-0x6C	ERROR_OTA_FLASH_IO_EXCEPTION	出现 flash 读写错误
-0x6D	ERROR_OTA_CMD_TIMEOUT	升级过程中指令超时
-0x6E	ERROR_OTA_IN_PROGRESS	正在 OTA
-0x6F	ERROR_OTA_COMMAND_TIMEOUT	等待命令超时
-0x70	ERROR_OTA_RECONNECT_DEVICE_TIMEOUT	回连设备超时
-0x71	ERROR_OTA_USE_CANCEL	用户取消升级
-0x72	ERROR_OTA_SAME_UPDATE_FILE	相同升级文件

四、其他

（一）调试说明

打印类 **RcspLog** ：com.jieli.rcsp.util.RcspLog

- 开启打印

```
1.  boolean isLog = RcspLog.isLog();  //是否开启打印
2.
3.  RcspLog.setTagPrefix("ota"); //设置打印前缀，方便过滤打印信息
4.  RcspLog.setIsLog(true);      //设置是否开启打印
```

- 打印等级

```
1.  RcspLog.v("tag", "message");   //打印 v 等级 log，verbose
2.  RcspLog.d("tag", "message");   //打印 d 等级 log，debug
3.  RcspLog.i("tag", "message");   //打印 i 等级 log，info
4.  RcspLog.w("tag", "message");   //打印 w 等级 log，warning
5.  RcspLog.e("tag", "message");   //打印 e 等级 log，error
```

- 输出 log

```
1.  RcspLog.setLogOutput(new RcspLog.ILogOutput() {
2.      @Override
3.      public void output(String logcat) {
4.          //回调打印信息，可以保存到文件中，方便查找问题
5.      }
6.  })
```