目录

杰理ac69系列OTA库开发说明

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。

版本历史

版本	日期	修改者	修改记录
1.0.0	2019/03/26	钟卓成	初始版本，接口说明，开发说明
1.0.1	2019/09/09	钟卓成	1、增加特殊字段的说明; 比如，onBtDeviceConnection的status和onMtuChanged的status的区别; 2、IUpgradeCallback增加回连回调和进度类型; 3、兼容ufw的新的升级流程。
1.0.3	2020/04/30	钟卓成	1、增加TWS设备OTA升级的错误码 2、接口更新
1.3.0	2020/09/29	钟卓成	1、重构OTA库结构 2、修改回连机制 3、更新接口
1.4.0	2020/11/05	钟卓成	1、增加错误码说明 2、修改部分错误描述

一、概述

本文档是为了方便后续项目维护和管理、记录开发内容而创建。

二、接口说明

主要讲述比较重要的接口，只是作为参考。如果想详细连接接口使用，请阅读《杰理AC69系列OTA库接口》的javaDoc。

2.1 重要接口 IBluetoothManager

IBluetoothManager是用户必须实现的接口类，需要通过该接口类获取用户SDK的蓝牙控制接口，蓝牙传输数据等重要内容。

```
public interface IBluetoothManager {

    /*
     * TODO:// 注意：以下接口由客户自行实现
     */
    /* ----------------------------------------------*
     * 以下是外部需要回调状态
     */
    /**
     * 连接回调
     * <p>
     * TODO: //需要用户调用，实现数据传输
     * 注意：回调通讯方式的连接状态
     * </p>
     *
     * @param device 蓝牙设备对象
     * @param status 连接状态
     * <p>
     * 注意：status需要转换成库内定义好的连接状态，请参考"StateCode"
     * </p>
     */
    void onBtDeviceConnection(BluetoothDevice device, int status);

    /**
     * 从蓝牙设备接收到的数据
     * <p>
     * TODO: //需要用户调用，实现数据传输
     * 注意：必须回调原始数据
     * </p>
     *
     * @param device 蓝牙设备对象
     * @param data   原始数据
     */
    void onReceiveDeviceData(BluetoothDevice device, byte[] data);

    /**
     * 蓝牙MTU改变回调
     * <p>
     * TODO: //需要用户调用，实现数据传输
     * 说明：走BLE方式需要实现，SPP方式可以忽略
     * </p>
     * @param gatt gatt控制对象
     * @param mtu  mtu
     * @param status 状态
     * <p>
     * 注意：status需要和连接状态回调区分，这里不需要转换，用原始的<BluetoothProfile>
     * </p>
     */
    void onMtuChanged(BluetoothGatt gatt, int mtu, int status);

    /**
     * 错误事件回调
     * <p>
     * TODO: //需要用户调用，实现数据传输
     * </p>
     *
     * @param error 错误事件
     */
    void onError(BaseError error);  //错误事件回调

    /*
```

```
     * 以下需要外部实现接口。
     */

    /**
     * 获取已连接的蓝牙设备对象
     *
     * @return 已连接的蓝牙设备对象
     */
    BluetoothDevice getConnectedDevice(); //获取已连接设备

    /**
     * 获取已连接的BLE的GATT控制对象
     *
     * @return 已连接的BLE的GATT控制对象
     */
    BluetoothGatt getConnectedBluetoothGatt();

    /**
     * 连接蓝牙设备
     *
     * <p>
     *     要求：必须保证单独连接，不会影响到其他连接
     *     比如：BLE的连接，只连接BLE，不连接经典蓝牙
     *     SPP的连接，只连接SPP，不连接A2DP、HFP、BLE等
     * </p>
     *
     * @param device 蓝牙设备对象
     */
    void connectBluetoothDevice(BluetoothDevice device);

    /**
     * 断开蓝牙设备的连接
     *
     * @param device 蓝牙设备对象
     */
    void disconnectBluetoothDevice(BluetoothDevice device);

    /**
     * 向蓝牙设备发送数据
     *
     * @param device 蓝牙设备对象
     * @param data   数据包
     * @return 操作结果
     */
    boolean sendDataToDevice(BluetoothDevice device, byte[] data);   //发送数据接口

    /* ------------------------------------------------------*
     * 由库回调通知外部
     * ------------------------------------------------------*/
    /**
     * 从蓝牙设备接收到的协议数据
     * <p>
     * <pre>
     *      (注意:返回数据是已转换成RCSP协议数据，经过数据过滤和解析的。)
     * </pre>
     *
     * @param device 蓝牙设备对象
     * @param data   协议数据
     */
    void receiveDataFromDevice(BluetoothDevice device, byte[] data); //接收数据包接口

    /**
     * 错误事件回调
     *
     * @param error 错误事件
```

```
     */
    void errorEventCallback(BaseError error);  //接收错误事件返回

    /**
     * 查询是否需要强制升级
     *
     * <p>
  *   建议：每次通讯连接成功都调用一次
     * </p>
     * @param callback 结果回调
     */
    void queryMandatoryUpdate(IActionCallback<TargetInfoResponse> callback); //查询是否需要强制升级

}
```

2.2 OTA接口IUpgradeManager

主要是实现OTA的流程功能的，由OTA库实现，用户调用对应接口即可实现OTA升级。

```
public interface IUpgradeManager {

    /**
     * 配置OTA的实现参数
     * <p>
     *     注意：执行OTA前必须先配置OTA参数，否则报错
     * </p>
     * @param configure OTA参数
     */
    void configure(BluetoothOTAConfigure configure);

    /**
     * 开始OTA流程
     *
     * @param callback OTA的状态回调
     */
    void startOTA(IUpgradeCallback callback);

    /**
     * 取消OTA流程
     * <p>
     *     说明：如果是单备份方案，此接口无效；如果是双备份方案，此接口才有效
     * </p>
     */
    void cancelOTA();

    /**
     * 释放资源
     */
    void release();
}
```

重点：BluetoothOTAConfigure

BluetoothOTAConfigure必须在OTA前配置，建议在Application#onCreate()内配置OTA参数。

```
public class BluetoothOTAConfigure {

    public static final int PREFER_SPP = 1;  // SPP 优先
    public static final int PREFER_BLE = 0;  // BLE 优先
```

```
    /*==========================================================================*
     * 通用配置
     *==========================================================================*/
    /**
     * OTA的通讯方式
     * <p>
     * * 可选以下值:
     * * {@link #PREFER_BLE} -- BLE协议
     * * {@link #PREFER_BLE} -- SPP协议
     * * </p>
     */
    private int priority = PREFER_BLE;
    /**
     * 是否使用自定义回连方式
     */
    private boolean isUseReconnect = false;
    /**
     * 是否启用设备认证
     * <p>
     * 注意:该设置必须与固件设置一致
     * </p>
     */
    private boolean isUseAuthDevice;      // 是否启用设备认证
    /**
     * BLE 连接间隔时间, 默认500ms
     */
    private int bleIntervalMs = 500;
    /**
     * 命令超时时间, 默认2000ms
     */
    private int timeoutMs = BluetoothConstant.DEFAULT_SEND_CMD_TIMEOUT;
    /**
     * 是否使用JL服务器
     * <p>
     * 说明:当前版本暂不支持杰理服务器
     * </p>
     */
    private boolean isUseJLServer = false; //是否使用JL服务器
    /**
     * 固件本地存储路径
     */
    private String firmwareFilePath; //固件升级文件存放路径

    /*==========================================================================*
     * BLE相关配置
     *==========================================================================*/
    /**
     * BLE广播包过滤数据
     */
    @Deprecated
    private String scanFilterData;  // 搜索过滤数据
    /**
     * BLE调节底层传输MTU
     */
    private int mtu = BluetoothConstant.BLE_MTU_MIN;  //调节BLE协议栈MTU  (20 - 512)
    /**
     * 是否需要调节MTU
     * <p>
     * 默认不调节MTU
     * </p>
     */
    private boolean isNeedChangeMtu = false;
    /**
     * BLE扫描模式
```

```
     * <p>
     * 说明: 1.Android 5.1+才能使用
     * 可选参数: {@link android.bluetooth.le.ScanSettings#SCAN_MODE_LOW_POWER} --- 低功耗模式(默认)
     * {@link android.bluetooth.le.ScanSettings#SCAN_MODE_BALANCED} --- 平衡模式
     * {@link android.bluetooth.le.ScanSettings#SCAN_MODE_LOW_LATENCY} --- 低延迟模式(高功耗,仅前台有效)
     * </p>
     */
    private int bleScanMode = 0;

    ... ...
}
```

2.3 OTA流程回调 -- IUpgradeCallback

IUpgradeCallback是OTA流程的状态回调，用于更新UI。

```
public interface IUpgradeCallback {

    /**
     * OTA开始
     */
    void onStartOTA();

    /**
     * 需要回连的回调
     * <p>注意: 1.仅连接通讯通道 (BLE or  SPP)
     * 2.用于单备份OTA</p>
     *
     * @param addr 回连设备的MAC地址
     */
    void onNeedReconnect(String addr);

    /**
     * 进度回调
     *
     * @param type     类型
     *                 <p>0 -- 下载boot
     *                 or 1 -- 固件升级</p>
     * @param progress 进度
     */
    void onProgress(int type, float progress);

    /**
     * OTA结束
     */
    void onStopOTA();

    /**
     * OTA取消
     */
    void onCancelOTA();

    /**
     * OTA失败
     *
     * @param error 错误信息
     */
    void onError(BaseError error);
}
```

2.4 错误码--ErrorCode

错误码参考**ErrorCode**

```
public class ErrorCode {

    /*主Code值*/
    /**
     *  没有错误
     */
    public static final int ERR_NONE = 0;

    /**
     * 通用错误
     */
    public static final int ERR_COMMON = 1;

    /**
     * 状态错误
     */
    public static final int ERR_STATUS = 2;

    /**
     * 通讯错误
     */
    public static final int ERR_COMMUNICATION = 3;

    /**
     * OTA错误
     */
    public static final int ERR_OTA = 4;

    /**
     * 其他错误
     */
    public static final int ERR_OTHER = 5;

    /**
     * 未知错误
     */
    public static final int ERR_UNKNOWN = -1;

    /*Sub Code 值(格式: 0x[主码值][序号])*/
    /* ======================== ERR_COMMON ========================*/
    /**
     * 参数错误
     */
    public static final int SUB_ERR_PARAMETER = 0x1001;

    /**
     * 不支持BLE功能
     */
    public static final int SUB_ERR_BLE_NOT_SUPPORT = 0x1002;
    /**
     * 蓝牙未打开
     */
    public static final int SUB_ERR_BLUETOOTH_NOT_ENABLE = 0x1003;
    /**
     * 设备未配对
     */
    public static final int SUB_ERR_BLUETOOTH_UN_PAIR_FAILED = 0x1006;
    /**
     * A2DP管理对象未初始化
```

```
     */
    public static final int SUB_ERR_A2DP_NOT_INIT = 0x1007;
    /**
     * A2DP服务连接失败
     */
    public static final int SUB_ERR_A2DP_CONNECT_FAILED = 0x1008;
    /**
     * HFP管理对象未初始化
     */
    public static final int SUB_ERR_HFP_NOT_INIT = 0x1009;
    /**
     * HFP连接失败
     */
    public static final int SUB_ERR_HFP_CONNECT_FAILED = 0x100A;
    /**
     * BLE无服务发现
     */
    public static final int SUB_ERR_NO_SERVER = 0x1010;
    /**
     * 操作失败
     */
    public static final int SUB_ERR_OP_FAILED = 0x1011;
    /**
     * BLE未连接
     */
    public static final int SUB_ERR_BLE_NOT_CONNECTED = 0x1012;
    /**
     * 改变BLE的MTU失败
     */
    public static final int SUB_ERR_CHANGE_BLE_MTU = 0x1013;

    /* ======================== ERR_STATUS ========================*/
    /**
     * 搜索设备失败
     */
    public static final int SUB_ERR_SCAN_DEVICE_FAILED = 0x2002;

    /*======================== ERR_COMMUNICATION ========================*/

    public static final int SUB_ERR_CONNECT_TIMEOUT = 0x3001;
    /**
     * 发送数据失败
     */
    public static final int SUB_ERR_SEND_FAILED = 0x3002;
    /**
     * 配对超时
     */
    public static final int SUB_ERR_PAIR_TIMEOUT = 0x3003;
    /**
     * 数据格式异常
     */
    public static final int SUB_ERR_DATA_FORMAT = 0x3004;
    /**
     * 解包异常
     */
    public static final int SUB_ERR_PARSE_DATA = 0x3005;
    /**
     * SPP发送数据失败
     */
    public static final int SUB_ERR_SPP_WRITE_DATA_FAIL = 0x3006;
    /**
     * 发送数据超时
     */
    public static final int SUB_ERR_SEND_TIMEOUT = 0x3007;
    /**
```

```
     * 回复失败状态
     */
    public static final int SUB_ERR_RESPONSE_BAD_STATUS = 0x3008;

    /**
     * 不允许连接
     */
    public static final int SUB_ERR_NOT_ALLOW_CONNECT = 0x3009;

    /* ======================== ERR_OTA ======================*/
    /**
     * OTA升级失败
     */
    public static final int SUB_ERR_OTA_FAILED = 0x4001;
    /**
     * 设备低电压
     */
    public static final int SUB_ERR_DEVICE_LOW_VOLTAGE = 0x4002;
    /**
     * 升级文件错误
     */
    public static final int SUB_ERR_CHECK_UPGRADE_FILE = 0x4003;
    /**
     * 读取偏移量失败
     */
    public static final int SUB_ERR_OFFSET_OVER = 0x4004;
    /**
     *数据校验失败
     */
    public static final int SUB_ERR_CHECK_RECEIVED_DATA_FAILED = 0x4005;
    /**
     * 加密key不匹配
     */
    public static final int SUB_ERR_UPGRADE_KEY_NOT_MATCH = 0x4006;
    /**
     * 升级类型出错
     */
    public static final int SUB_ERR_UPGRADE_TYPE_NOT_MATCH = 0x4007;
    /**
     * 升级程序正在进行
     */
    public static final int SUB_ERR_OTA_IN_HANDLE = 0x4008;
    /**
     * 升级过程中出现长度错误
     */
    public static final int SUB_ERR_UPGRADE_DATA_LEN = 0x4009;
    /**
     * flash读写错误
     */
    public static final int SUB_ERR_UPGRADE_FLASH_READ = 0x400A;
    /**
     * 命令超时
     */
    public static final int SUB_ERR_UPGRADE_CMD_TIMEOUT = 0x400B;
    /**
     * 升级文件的固件版本一致
     */
    public static final int SUB_ERR_UPGRADE_FILE_VERSION_SAME = 0x400C;
    /**
     * TWS未连接
     */
    public static final int SUB_ERR_TWS_NOT_CONNECT = 0x400D;
    /**
     * 耳机未在充电仓
     */
```

```
    public static final int SUB_ERR_HEADSET_NOT_IN_CHARGING_BIN = 0x400E;


    /* ======================== ERR_OTHER ========================*/
    /**
     * 认证设备失败
     */
    public static final int SUB_ERR_AUTH_DEVICE = 0x5001;
    /**
     * 未找到升级数据
     */
    public static final int SUB_ERR_FILE_NOT_FOUND = 0x5005;
    /**
     * IO异常
     */
    public static final int SUB_ERR_IO_EXCEPTION = 0x5006;
 ...
}
```

三、开发说明

用户在开发时只需要实现部分接口和通过部分接口传输蓝牙数据即可，具体的流程由OTA库实现。详细需要实现的接口请参考IBluetoothManager

3.1. 用户需要通过继承BluetoothOTAManager并实现相应接口。

```
/**
* OTA 管理器实现
*
* <p>
*  建议：单例化使用
*</p>
*/
public class OTAManager extends BluetoothOTAManager {

  public OTAManager(Context context) {
     super(context);
     //完成用户SDK的控制对象初始化和状态监听注册
  }

    /**
    * 获取已连接的蓝牙设备
    * <p>
    *  注意：是通讯方式对应的蓝牙设备对象
    * </p>
    */
    @Override
  public BluetoothDevice getConnectedDevice() {
     return null;
  }

  /**
   * 获取已连接的BluetoothGatt对象
   * <p>
   * 若选择BLE方式OTA，需要实现此方法。反之，SPP方式不需要实现
```

```
     * </p>
     */
    @Override
    public BluetoothGatt getConnectedBluetoothGatt() {
        return null;
    }

     /**
     * 连接蓝牙设备
     * <p>
     *  注意:这里必须是单纯连接蓝牙设备的通讯方式。
     *  例如，BLE方式，只连接BLE，不应联动连接经典蓝牙。
     *  SPP方式，只连接SPP，不能连接A2DP,HFP和BLE。
     * </p>
     * @param device 通讯方式的蓝牙设备
     */
    @Override
    public void connectBluetoothDevice(BluetoothDevice device) {

    }

     /**
     * 断开蓝牙设备的连接
     *
     * @param device 通讯方式的蓝牙设备
     */
    @Override
    public void disconnectBluetoothDevice(BluetoothDevice device) {

    }

     /**
     *发送数据到蓝牙设备
     *<p>
     *  注意: 如果是BLE发送数据，应该注意MTU限制。BLE方式会主动把MTU重新设置为OTA配置参数设置的值。
     *</p>
     * @param device 已连接的蓝牙设备
     * @param data 数据包
     * @return 操作结果
     */
    @Override
    public boolean sendDataToDevice(BluetoothDevice device, byte[] data) {
        return false;
    }

    /**
     * 用户通知OTA库的错误事件
     */
    @Override
    public void errorEventCallback(BaseError error) {

    }

    /**
     * 用于释放资源
     */
    @Override
    public void release() {

    }
}
```

3.2. 实现用户SDK的连接状态, 蓝牙数据的传递

- 2.1 连接状态的传递

```
/**
 *回传连接状态
 *<p>
 *注意：1.用户应该在通讯方式连接状态回调时调用此接口回传连接状态
 *      2.连接状态只需要传递对应通讯方式连接状态即可。比如，BLE方式，只传递BLE连接状态；反之，SPP方式，只传递SPP连接方式
 *      3.连接状态应该统一转换为OTA库通用的StateCode
 *      4.不需要实现，只需要在合适地方调用
 *</p>
 *@param device 蓝牙设备对象(连接中允许为null)
 *@param status 连接状态{@link StateCode}
 */
@Override
public void onBtDeviceConnection(BluetoothDevice device, int status){}
```

StateCode

```
public class StateCode {

    /*-------------------------------------------------------------------
     *连接状态
     * ------------------------------------------------------------------*/
    public final static int CONNECTION_OK = 1; //连接成功
    public final static int CONNECTION_FAILED = 2;//连接失败
    public final static int CONNECTION_DISCONNECT = 0;//断开连接
    public final static int CONNECTION_CONNECTING = 3;//连接中
    public final static int CONNECTION_CONNECTED = 4;//已连接

    /*-------------------------------------------------------------------
     * Response Status
     * ------------------------------------------------------------------*/
    public final static int STATUS_SUCCESS = 0;   //成功状态
    public final static int STATUS_FAIL = 1;      //失败状态
    public final static int STATUS_UNKOWN_CMD = 2; //未知命令
    public final static int STATUS_BUSY = 3;       //繁忙状态
    public final static int STATUS_NO_RESOURCE = 4; //没有资源
    public final static int STATUS_CRC_ERROR = 5;   //CRC校验错误
    public final static int STATUS_ALL_DATA_CRC_ERROR = 6;//全部数据CRC错误
    public final static int STATUS_PARAMETER_ERROR = 7; //参数错误
    public final static int STATUS_RESPONSE_DATA_OVER_LIMIT = 8; //回复数据超出限制

    /*-------------------------------------------------------------------
     * OTA
     *------------------------------------------------------------------*/
   //E3
   public final static int RESULT_OK = 0x00; //成功
    public final static int RESULT_FAIL = 0x01; //失败
 //E2
    public final static int RESULT_CAN_UPDATE = 0x00; //可以更新
    public final static int RESULT_DEVICE_LOW_VOLTAGE_EQUIPMENT = 0x01; //设备低电压
    public final static int RESULT_FIRMWARE_INFO_ERROR = 0x02; //升级固件信息错误
 public final static int RESULT_FIRMWARE_VERSION_NO_CHANGE = 0x03; //升级文件的固件版本一致
 public final static int RESULT_TWS_NOT_CONNECT = 0x04; //TWS未连接
    public final static int RESULT_HEADSET_NOT_IN_CHARGING_BIN = 0x05; //耳机未在充电仓

 //E6
```

```
    public final static int UPGRADE_RESULT_COMPLETE = 0x00; //升级完成
    public final static int UPGRADE_RESULT_DATA_CHECK_ERROR = 0x01; //升级数据校验出错
    public final static int UPGRADE_RESULT_FAIL = 0x02; //升级失败
    public final static int UPGRADE_RESULT_ENCRYPTED_KEY_NOT_MATCH = 0x03; //加密key不匹配
    public final static int UPGRADE_RESULT_UPGRADE_FILE_ERROR = 0x04;  //升级文件出错
    public final static int UPGRADE_RESULT_UPGRADE_TYPE_ERROR = 0x05;  //升级类型出错
 public final static int UPGRADE_RESULT_ERROR_LENGTH = 0x06; //升级过程中出现长度错误
    public final static int UPGRADE_RESULT_FLASH_READ = 0x07; //出现flash读写错误
    public final static int UPGRADE_RESULT_CMD_TIMEOUT = 0x08; //升级过程中指令超时
    public final static int UPGRADE_RESULT_DOWNLOAD_BOOT_LOADER_SUCCESS = 0x80; //下载boot loader完成


    /*-------------------------------------------------------------------
     * SCAN 属性
     *-------------------------------------------------------------------*/
    public final static int SCAN_TYPE_FLAG_CONTENT = 0; //标识内容
    public final static int SCAN_TYPE_FLAG_PAIRED = 1;  //标识设备是否已配对
    public final static int SCAN_TYPE_FLAG_PHONE_VIRTUAL_ADDRESS = 2; //标识手机虚拟地址

}
```

- 2.2 蓝牙数据的传递

```
/**
 *回传蓝牙数据
 *<p>
 * 注意:1.数据包必须保证是原始数据。可以是通过用户SDK过滤后的原始数据，或者未经过滤的原始数据
 *      2.用户应在选择OTA方式的对应数据回调处调用改接口传递蓝牙数据
 *      3.不需要实现，只需要在合适地方调用
 *</p>
 */
@Override
public void onReceiveDeviceData(BluetoothDevice device, byte[] data){}
```

- 2.3 BLE的MTU改变的传递

```
/**
 * 蓝牙MTU改变回调
 * <p>
 * 注意: 1.用户若选择BLE方式，应在BLE回调MTU改变处调用此接口回调。SPP方式，不需要调用
 *       2.不需要实现，只需要在合适地方调用
 * </p>
 * @param gatt gatt控制对象
 * @param mtu  mtu值
 * @param status 状态
 */
 void onMtuChanged(BluetoothGatt gatt, int mtu, int status)
```

- 2.4 用户SDK错误回调

```
/**
 * 错误事件回调
 * <p>
 * 注意:1.用户SDK发送异常时可以通过此接口传递
 *      2.此接口调用可能会停止OTA流程，请慎用
 *      3.不需要实现，只需要在合适地方调用
 * </p>
```

```
 *
 * @param error 错误事件
  */
 void onError(BaseError error);
```

3.3. 用户必须先配置OTA参数，然后在使用BluetoothOTAManager的接口。建议在Application#onCreate()内实现OTA参数配置。

```
public class MainApplication extends Application{
      @Override
    public void onCreate() {
        super.onCreate();
        OTAManager.init(getApplicationContext());
        initOTAManager();
    }

 private void initOTAManager(){
        mOTAManager = OTAManager.getInstance();
        BluetoothOTAConfigure bluetoothOption = new BluetoothOTAConfigure();
        bluetoothOption.setPriority(BluetoothOTAConfigure.PREFER_BLE); //请按照项目需要选择
        bluetoothOption.setUseAuthDevice(false); //非AC692X_AI_SDK的，建议不使用认证流程。具体根据固件的配置选择
        bluetoothOption.setBleIntervalMs(500); //默认是500毫秒
        bluetoothOption.setTimeoutMs(3000); //超时时间
        bluetoothOption.setMtu(512); //BLE底层通讯MTU值，会影响BLE传输数据的速率。建议用512。该MTU值会使OTA库在BLE连接时改变MTU，所以用户SDK需要对此处理。
  bluetoothOption.setNeedChangeMtu(false); //不需要调整MTU，建议客户连接时调整好BLE的MTU
        byte[] scanData = "JLAISDK".getBytes();//根据固件配置
        bluetoothOption.setScanFilterData(new String(scanData)); //用于设备过滤回连，已弃用，由用户主动回连
        bluetoothOption.setUseJLServer(false); //是否选择JL服务器（暂不支持）
        String firmwarePath = FileUtil.splicingFilePath(getApplicationContext().getPackageName(), DIR_UPDATE, null, null)
                + "/" + FIRMWARE_UPGRADE_FILE;
        bluetoothOption.setFirmwareFilePath(firmwarePath); //设置本地存储OTA文件的路径
        mOTAManager.configure(bluetoothOption); //设置OTA参数
    }
}
```

3.4. 强制升级的检查

该接口的作用是通过获取设备信息来判断是否需要强制升级。也可以通过该接口获取设备信息。

```
/**
 * 查询强制升级状态
 * <p>
 * 说明：每次通讯方式连接成功都应该调用一次
 * </p>
 */
queryMandatoryUpdate(new IActionCallback<TargetInfoResponse>() {
            @Override
            public void onSuccess(TargetInfoResponse targetInfoResponse) {
                /*
                 * TODO:说明设备需要强制升级，请跳转到OTA界面，引导用户升级固件
                 */
```

```
            }

            @Override
            public void onError(BaseError baseError) {
                /*
                 *可以不用处理
                 */
            }
        });
```

举例说明

```
... ...
 @Override
public void onConnection(BluetoothDevice device, int status){
   if (status == StateCode.CONNECTION_OK) {
        queryMandatoryUpdate(new IActionCallback<TargetInfoResponse>() {
            @Override
            public void onSuccess(TargetInfoResponse targetInfoResponse) {
               /*
                * TODO:说明设备需要强制升级，请跳转到OTA界面，引导用户升级固件
                */
            }

            @Override
            public void onError(BaseError baseError) {
      //可以在这里获取设备信息，做版本校验。
                if (baseError != null && baseError.getCode() == ErrorCode.ERR_NONE) { //没有错误，说明发送命
令成功并回复了
                TargetInfoResponse deviceInfo =
DeviceStatusManager.getInstance().getDeviceInfo(getConnectedDevice()); //获取当前连接设备的信息
                if (deviceInfo != null) {
                    Debug.i("ota", String.format(Locale.getDefault(), "device msg >> version : %d,
version_name : %s, project_code : %s",
                            deviceInfo.getVersionCode(), //设备版本号
                            deviceInfo.getVersionName(),  //设备版本名
                            deviceInfo.getProjectCode())); //设备产品ID(默认是0，如果设备支持会改变)
                }
                }
            }
        });
      }
}
... ...
```

注意事项

1. 为了防止升级失败导致固件变"砖"，因此固件升级失败后会进入强制升级模式
2. 用户SDK或APP应该具备回连上一次连接的蓝牙设备的功能，用于强制升级时自动回连设备，再通过OTA升级更新固件
3. 单备份方案，进入uboot后，将不存在A2DP和HFP。因此SPP方式的回连，不能通过经典蓝牙回连实现。
4. 双备份方案，不需要回连过程，直接开始升级流程。