杰理蓝牙控制库
SDK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
0.0.1	2018/11/15	陈森华	初始版本，接口制定
0.0.2	2018/12/17	陈森华	1、修改sys info 格式 2、增加设备音乐 3 、增加Aux功能 4、增加自定义命令发送接口 5、修改StateCode
2.0.0	2019/01/13	陈森华	1、增加固件SDK类型区分：AI SDK和Stard SDK 2、已知bug修复
2.2.0	2019/02/18	陈森华	2、增加FM功能 2、增加EQ调节功能 3、增加OTA升级功能 4、增加歌词接口 5、设备音乐快进快退接口
2.3.0	2019/04/04	陈森华	1、增加RTC功能 2、录音增加断句标志位，参考：StartSpeechParam类 3、增加通知固件app开始播放tts命令 4、修复歌词和目录浏览读取偶现读取不了或者重复问题 5、目录浏览增加 backBrowse接口
2.3.2	2019/10/25	陈森华	1、修复协议MTU导致的目录浏览卡死的问题 2、修复自定义命令接收没有判断是否为回复的问题
2.4.0	2019/11/08	钟卓成 陈森华	1、增加BR22 芯片的弹窗-设置功能命令; 2、增加BLE过滤规则的策略选择 3、修复超出MTU的问题导致数据被抛弃的问题; 4、修复回连异常的问题 5、增加对Androidx的支持和Android 10.0+兼容
2.5.0	2020/06/15	钟卓成	1、修复已知BUG; 2、增加AC693X、AC695X和AC697X的芯片支持; 3、增加ID3显示功能; 4、修改部分接口逻辑; 5、整理文档内容格式
2.5.1	2020/08/05	钟卓成	1、增加查找设备命令 2、增加AC696X的支持 3、增加设置错误码
2.6.0	2020/09/29	钟卓成	1、增加多设备管理功能 2、增加多设备交互接口 3、移除OTA流程(参考OTA外接库) 4、语音解码回调给应用层处理
2.8.0	2020/12/14	陈森华	1、增加直播声卡功能
2.9.0	2021/05/18	陈森华	1、增加音箱SDK的闹钟贪睡模式 2、增加耳机SDK的主动降噪(ANC)设置功能

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。

开发说明

0、蓝牙库导入

0.1、运行环境

类别	兼容范围	备注
系统	Android 5.1以上系统	支持BLE功能
硬件要求	杰理蓝牙产品	AC69X系列
开发平台	Android Studio	建议使用最新版本开发

0.2、权限配置

接入SDK时应在AndroidManifest.xml申请以下权限：

```
    //使用网络权限，因为有可能使用服务器功能
    <uses-permission android:name="android.permission.INTERNET" />
    //使用蓝牙权限
    <uses-permission android:name="android.permission.BLUETOOTH"/>
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
    //定位权限，官方要求使用蓝牙或网络开发，需要位置信息
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    //读写存储权限
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />

    <uses-feature android:name="android.hardware.bluetooth_le"
android:required="true" />
```

注意
在Android 6.0+以上手机系统运行，请增加动态权限管理。

Android Q(10.0)以上支持
因为存储权限改变，需要在AndroidManifest.xml增加以下配置

```
<application
    ...
    android:requestLegacyExternalStorage="true"
    ...>
    ...
</application>
```

0.3、导入AAR包

将jl_bluetooth_rcsp_Vxxx库放到工程目录的libs文件夹下

增加杰理解码库jldecryption_Vxxx-release.aar

0.4 初始化

```
        //配置蓝牙sdk的参数
        BluetoothOption bluetoothOption = new BluetoothOption();
        bluetoothOption.setPriority(BluetoothOption.PREFER_BLE);//通信方式，支持ble和spp
        bluetoothOption.setReconnect(true);//是否断开重连
        bluetoothOption.setBleIntervalMs(500); // BLE 连接间隔时间，默认500ms
        bluetoothOption.setTimeoutMs(2000);// 命令超时时间，默认2000ms
        bluetoothOption.setMtu(512);// 调节蓝牙MTU
        byte[] scanData = "JLAISDK".getBytes();
        bluetoothOption.setScanFilterData(new String(scanData));//设置过滤广播包数据，需要和固件协商

 JL_BluetoothManager.getInstance(ContextUtil.getContext()).configure(bluetoothOption);//配置参数
```

0.4.1 配置参数说明

```
/**
 * 蓝牙配置参数
 *
 */
public class BluetoothOption {

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

    /*==========================================================================*
     * 通用配置
     *==========================================================================*/
    /**
     * 指定通讯方式
     * <p>
     * 可选以下值:
     * {@link #PREFER_BLE} -- BLE协议
     * {@link #PREFER_BLE} -- SPP协议
     * </p>
     */
    private int priority = PREFER_BLE; //连接优先协议
    /**
     * 是否需要重连
     * <p>
     * 说明:异常断开重连机制
     * </p>
     */
    private boolean reconnect;      // 允许重连
    @Deprecated
    private int bleIntervalMs = 500;    // BLE 连接间隔时间，默认500ms
```

```
    /**
     * 命令超时时间
     */
    private int timeoutMs = BluetoothConstant.DEFAULT_SEND_CMD_TIMEOUT;
// 命令超时时间, 默认2000ms
    /**
     * 是否进入低功耗模式
     * <p>
     * 说明: 1. app端主动设置是否进入低功耗模式
     * 2. 如果为低功耗模式, 则不连接经典蓝牙
     * </p>
     */
    private boolean enterLowPowerMode = false;
    /**
     * 是否使用多设备管理
     */
    private boolean isUseMultiDevice = true;

    /**
     * 是否使用设备认证
     */
    private boolean isUseDeviceAuth = true;

    /*=========================================================================*
     * BLE相关配置
     *=========================================================================*/
    /**
     * BLE过滤规则的过滤标识
     */
    private String scanFilterData;  // 搜索过滤数据

    /**
     * BLE扫描策略
     * <p>
     * 说明: 决定设备过滤规则
     * 可选参数:{@link BluetoothConstant#NONE_FILTER}  --- 不使用过滤规则
     * {@link BluetoothConstant#ALL_FILTER} --- 使用全部过滤规则
     * {@link BluetoothConstant#FLAG_FILTER} --- 仅使用标识过滤规则
     * {@link BluetoothConstant#HASH_FILTER} --- 仅使用hash加密过滤规则
     * </p>
     */
    private int bleScanStrategy = BluetoothConstant.ALL_FILTER; //BLE过滤规则

    /**
     * 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;

    /**
     * BLE传输MTU
```

```
     * <p>
     * 说明:Android为了兼容4.0的手机，默认的MTU是23；可用数据长度:20；
     * 随着手机系统的提升，现在的手机的MTU最大为512。建议提升MTU到512。
     * </p>
     */
    private int mtu = BluetoothConstant.BLE_MTU_MIN;  //调节蓝牙MTU

    /**
     * BLE 通讯UUID集合
     * <p>
     * {@link BluetoothConstant#KEY_BLE_SERVICE_UUID}  服务UUID
     * {@link BluetoothConstant#KEY_BLE_WRITE_CHARACTERISTIC_UUID} 写特征UUID
     * {@link BluetoothConstant#KEY_BLE_NOTIFICATION_CHARACTERISTIC_UUID} 通知特征
UUID
     * </p>
     */
    private final Map<String, UUID> bleUUIDMap = new HashMap<>();

    /**
     * 是否使用BLE配对方式
     */
    private boolean isUseBleBondWay = false;
    /**
     * 是否自动连接BLE
     */
    private boolean isAutoConnectBle = false;

    /*==========================================================================*
     * SPP相关配置
     *==========================================================================*/
    /**
     * SPP连接的UUID
     * <p>
     * 默认是标准的SPP UUID
     * </p>
     */
    private UUID sppUUID = BluetoothConstant.UUID_SPP;
    ...
}
```

BLE 过滤规则参数

```
public class BluetoothConstant {
    ...
    /*
============================================================================== *
     * BLE过滤方式
     *
============================================================================== */
    /**
     * 不过滤设备
     */
    public static final int NONE_FILTER = 0;
    /**
     * 使用所有过滤规则
     */
    public static final int ALL_FILTER = 1;
```

```
    /**
     * 只使用标志过滤规则
     * <p>
     *      例如"JLAISDK"作为设备标识
     * </p>
     */
    public static final int FLAG_FILTER = 2;
    /**
     * 只使用Hash加密过滤规则
     */
    public static final int HASH_FILTER = 3;
    ...

     /**
     * BLE连接类型
     */
    public static final int PROTOCOL_TYPE_BLE = 0;
    /**
     * SPP协议类型
     */
    public static final int PROTOCOL_TYPE_SPP = 1;

    ...
}
```

1、 主要接口

1.1、功能接口类IBluetoothManager

```
public interface IBluetoothManager {

    /*-------------------------------------------------------------------------
     * 通用接口
     *-------------------------------------------------------------------------*/

    /**
     * 添加蓝牙事件监听器
     *
     * @param listener 蓝牙事件监听器
     * @return 操作结果
     */
    boolean addEventListener(IBluetoothEventListener listener);     // 监听蓝牙事件、命令、数据等

    /**
     * 移除蓝牙事件监听器
     *
     * @param listener 蓝牙事件监听器
     * @return 操作结果
     */
    boolean removeEventListener(IBluetoothEventListener listener);

    /**
     * 配置蓝牙库参数
     *
     * @param option 蓝牙库参数
```

```
     */
    void configure(BluetoothOption option);     // 配置蓝牙参数

    /*-------------------------------------------------------------------------
     * 蓝牙操作接口
     *-------------------------------------------------------------------------*/

    /**
     * 是否打开蓝牙
     *
     * @return 结果 <p>true -- 打开 false -- 关闭</p>
     */
    boolean isBluetoothEnabled(); //是否打开蓝牙功能

    /**
     * 打开或关闭蓝牙
     * <p>
     * 可以通过{@link IBluetoothEventListener#onAdapterStatus(boolean, boolean)}知
道蓝牙状态
     * </p>
     *
     * @param isEnable 是否使能 <p>true --- 打开  false --- 关闭</p>
     * @return 操作结果
     */
    boolean openOrCloseBluetooth(boolean isEnable); //打开或关闭蓝牙功能

    /**
     * 搜索BLE蓝牙设备
     * <p>
     * 可以通过{@link IBluetoothEventListener#onDiscoveryStatus(boolean, boolean)}
知道搜索状态。
     * 同时可以通过{@link IBluetoothEventListener#onDiscovery(BluetoothDevice,
BleScanMessage)} 获取发现设备。
     * 当然也可以通过{@link IBluetoothEventListener#onDiscovery(BluetoothDevice)}
获取发现设备，但已经不推荐使用。
     * </p>
     *
     * @param timeout 搜索超时时间(单位是毫秒)
     * @return 操作码
     */
    int scan(int timeout); //默认扫描BLE设备, timeout :扫描时长

    /**
     * 搜索蓝牙设备
     * <p>
     * 可以通过{@link IBluetoothEventListener#onDiscoveryStatus(boolean, boolean)}
知道搜索状态。
     * 同时可以通过{@link IBluetoothEventListener#onDiscovery(BluetoothDevice,
BleScanMessage)} 获取发现设备。
     * 当然也可以通过{@link IBluetoothEventListener#onDiscovery(BluetoothDevice)}
获取发现设备，但已经不推荐使用。
     * </p>
     *
     * @param type    搜索设备方式<p>{@link
com.jieli.bluetooth.constant.BluetoothConstant#SCAN_TYPE_BLE}
     *                or {@link
com.jieli.bluetooth.constant.BluetoothConstant#SCAN_TYPE_CLASSIC} </p>
     * @param timeout 搜索超时时间(单位是毫秒)
```

```
     * @return 操作码
     */
    int scan(int type, int timeout); //扫描指定类型蓝牙设备 type:蓝牙设备类型  timeout
:扫描时长

    /**
     * 停止搜索流程
     *
     * @return 结果码
     */
    int stopScan(); //停止当前扫描

    //经典蓝牙操作

    /**
     * 是否连接经典蓝牙协议
     * <p>
     * 经典蓝牙协议：主要是指A2DP和HFP
     * </p>
     *
     * @param device 蓝牙设备
     * @return 是否连接
     */
    boolean isConnectedByProfile(BluetoothDevice device);

    /**
     * 开始连接经典蓝牙设备服务
     * <p>
     * 经典蓝牙协议：主要是指A2DP和HFP
     * </p>
     *
     * @param breDevice 经典蓝牙设备
     */
    void startConnectByBreProfiles(BluetoothDevice breDevice);

    /**
     * 断开已连接经典蓝牙设备服务
     *
     * @param device 经典蓝牙
     * @return 结果
     */
    boolean disconnectByProfiles(BluetoothDevice device);

    /**
     * 快速回连
     * <p>
     * 单设备模式，回连最后连接的设备
     * 多设备模式，回连已连接经典蓝牙的设备和最后连接的设备
     * </p>
     *
     * @return 结果
     */
    boolean fastConnect(); // 与最后连接的设备建立连接，包括打开蓝牙、扫描设备、绑定设备、
建立通信通道等

    /**
     * 断开当前已连接设备
     */
```

```
    void disconnect();  // 断开当前已连接的蓝牙设备

    /**
     * 获取已发现的设备列表
     *
     * @return 已发现的设备列表
     */
    ArrayList<BluetoothDevice> getFoundedBluetoothDevices();  // 获取扫描到的蓝牙设
备列表

    /**
     * 连接蓝牙设备
     *
     * @param device 蓝牙设备
     */
    void connect(BluetoothDevice device);   // 连接指定的蓝牙设备

    /**
     * 断开指定已连接设备
     *
     * @param device 已连接设备
     */
    void disconnect(BluetoothDevice device);

    /**
     * 取消蓝牙设备配对
     *
     * @param device 蓝牙设备
     * @return 结果码
     */
    int unPair(BluetoothDevice device);

    /**
     * 获取已连接且正在使用的设备
     *
     * @return 已连接且正在使用的设备
     */
    BluetoothDevice getConnectedDevice();

    /**
     * 获取连接历史记录列表
     *
     * @return 连接历史记录列表
     */
    List<HistoryBluetoothDevice> getHistoryBluetoothDeviceList(); //获取连接历史记
录

    /**
     * 清除连接历史记录
     */
    void clearHistoryDeviceRecord(); //清除历史设备记录

    /**
     * 移除指定的连接历史记录
     *
     * @param historyBtDevice 连接历史记录
     * @param callback        操作回调
     */
```

```
    void removeHistoryDevice(HistoryBluetoothDevice historyBtDevice,
IActionCallback<HistoryBluetoothDevice> callback); //删除连接历史记录

    /**
     * 获取映射地址
     *
     * @param deviceAddress 设备地址
     * @return 映射地址
     */
    String getMappedDeviceAddress(String deviceAddress); //获取映射地址

    /**
     * 同步系统蓝牙状态
     */
    void synchronizationBtDeviceStatus(); //同步蓝牙设备状态

    /*-------------------------------------------------------------------------
     * 主要用于控制层接收通讯层的数据
     *-------------------------------------------------------------------------*/

    /**
     * 发送数据到指定设备
     *
     * @param device 蓝牙设备
     * @param data   数据包
     * @return 结果
     */
    boolean sendDataToDevice(BluetoothDevice device, byte[] data);   //发送数据接口

    /**
     * 接收到设备的命令数据包
     *
     * @param device 蓝牙设备
     * @param data   命令数据包
     */
    void receiveDataFromDevice(BluetoothDevice device, byte[] data); //接收数据包接
口

    /**
     * 错误事件回调
     *
     * @param error 错误信息
     */
    void errorEventCallback(BaseError error);  //接收错误事件返回

    /*-------------------------------------------------------------------------
     * 命令发送接口
     *-------------------------------------------------------------------------*/

    /**
     * 发送RCSP命令到正在使用设备
     *
     * @param cmd       RCSP命令
     * @param timeoutMs 命令发送超时时间(单位是毫秒)
     * @param callback  命令回调
     * @deprecated 建议使用{@link #sendCommandAsync(BluetoothDevice, CommandBase,
int, RcspCommandCallback)}代替
     */
```

```
    @Deprecated
    void sendCommandAsync(CommandBase cmd, int timeoutMs, CommandCallback
callback);   // 向蓝牙设备发送command, timeoutMs为超时值; 在CommandCallback回调中接收
response

    /**
     * 发送RCSP命令到指定设备
     *
     * @param device     蓝牙设备
     * @param cmd        RCSP命令
     * @param timeoutMs 命令发送超时时间(单位是毫秒)
     * @param callback   命令回调
     */
    void sendCommandAsync(BluetoothDevice device, CommandBase cmd, int
timeoutMs, RcspCommandCallback callback);

    /**
     * 发送数据包到正在使用设备
     *
     * @param cmd  响应RCSP命令
     * @param data 数据包
     */
    void sendData(CommandBase cmd, byte[] data);     // 向蓝牙设备发送数据包, 如ota数
据包等

    /**
     * 发送数据包到指定设备
     *
     * @param device 蓝牙设备
     * @param cmd    响应RCSP命令
     * @param data   数据包
     */
    void sendData(BluetoothDevice device, CommandBase cmd, byte[] data);     //
向蓝牙设备发送数据包, 如ota数据包等

    /**
     * 发送回复RCSP命令到正在使用设备
     *
     * @param cmd 回复RCSP命令
     */
    void sendCommandResponse(CommandBase cmd);  // 接收到设备发送的命令后, 向蓝牙设备
发送response.

    /**
     * 发送回复RCSP命令到指定设备
     *
     * @param device 蓝牙设备
     * @param cmd    回复RCSP命令
     */
    void sendCommandResponse(BluetoothDevice device, CommandBase cmd);  // 接收到
设备发送的命令后, 向蓝牙设备发送response.

    /*--------------------------------------------------------------------------
     * 多设备管理
     *-------------------------------------------------------------------------*/

    /**
     * 获取已连接设备列表
```

```
     *
     * @return 已连接设备列表
     */
    List<BluetoothDevice> getConnectedDeviceList();

    /**
     * 判断设备是否已连接
     *
     * @param device 蓝牙设备
     * @return 是否已连接
     */
    boolean isConnectedBtDevice(BluetoothDevice device);

    /**
     * 判断设备是否正在使用设备
     *
     * @param device 蓝牙设备
     * @return 是否正在使用设备
     */
    boolean isUseBtDevice(BluetoothDevice device);

    /**
     * 切换已连接设备为正在使用设备
     * <p>
     * 可以通过{@link
IBluetoothEventListener#onSwitchConnectedDevice(BluetoothDevice)} 获取切换情况
     * </p>
     *
     * @param device 已连接设备
     */
    void switchConnectedDevice(BluetoothDevice device);

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

1.2、数据监听类IBluetoothEventListener

```
public interface IBluetoothEventListener {

    /**
     * 蓝牙适配器状态回调
     *
     * @param bEnabled 蓝牙是否打开
     * @param bHasBle  是否支持BLE
     */
    void onAdapterStatus(boolean bEnabled, boolean bHasBle);

    /**
     * 发现设备状态回调
     *
     * @param bBle   是否BLE设备发现
     * @param bStart 是否搜索开始
     */
    void onDiscoveryStatus(boolean bBle, boolean bStart);
```

```
    /**
     * 发现设备回调
     *
     * @param device 蓝牙设备
     */
    @Deprecated
    void onDiscovery(BluetoothDevice device);

    /**
     * 发现设备回调
     *
     * @param device          蓝牙设备
     * @param bleScanMessage 广播信息
     */
    void onDiscovery(BluetoothDevice device, BleScanMessage bleScanMessage);

    /**
     * 产品弹窗回调
     *
     * @param device          蓝牙设备
     * @param bleScanMessage 广播信息
     */
    void onShowDialog(BluetoothDevice device, BleScanMessage bleScanMessage);

    /**
     * 蓝牙设备配对状态回调
     *
     * @param device 蓝牙设备
     * @param status 配对状态
     *               <p>参考值如下<br>
     *               {@link BluetoothDevice#BOND_NONE} or
     *               {@link BluetoothDevice#BOND_BONDING} or
     *               {@link BluetoothDevice#BOND_BONDED}
     *               </p>
     */
    void onBondStatus(BluetoothDevice device, int status);

    /**
     * 蓝牙设备连接状态回调
     *
     * @param device 蓝牙设备
     * @param status 连接状态
     *               <p>参考值如下<br>
     *               {@link
com.jieli.bluetooth.constant.StateCode#CONNECTION_DISCONNECT} or
     *               {@link com.jieli.bluetooth.constant.StateCode#CONNECTION_OK}
or
     *               {@link
com.jieli.bluetooth.constant.StateCode#CONNECTION_FAILED} or
     *               {@link
com.jieli.bluetooth.constant.StateCode#CONNECTION_CONNECTING} or
     *               {@link
com.jieli.bluetooth.constant.StateCode#CONNECTION_CONNECTED}
     *               </p>
     */
    void onConnection(BluetoothDevice device, int status);
```

```
    /**
     * 切换已连接且正在使用的设备的回调
     *
     * @param device 已连接设备
     */
    void onSwitchConnectedDevice(BluetoothDevice device);

    /**
     * A2DP服务状态回调
     *
     * @param device 蓝牙设备
     * @param status 连接状态
     *               <p>参考值如下<br>
     *               {@link
android.bluetooth.BluetoothProfile#STATE_DISCONNECTED} or
     *               {@link android.bluetooth.BluetoothProfile#STATE_CONNECTING}
or
     *               {@link android.bluetooth.BluetoothProfile#STATE_CONNECTED}
or
     *               {@link
android.bluetooth.BluetoothProfile#STATE_DISCONNECTING}
     *               </p>
     */
    void onA2dpStatus(BluetoothDevice device, int status);

    /**
     * HFP服务状态回调
     *
     * @param device 蓝牙设备
     * @param status 连接状态
     *               <p>参考值如下<br>
     *               {@link
android.bluetooth.BluetoothProfile#STATE_DISCONNECTED} or
     *               {@link android.bluetooth.BluetoothProfile#STATE_CONNECTING}
or
     *               {@link android.bluetooth.BluetoothProfile#STATE_CONNECTED}
or
     *               {@link
android.bluetooth.BluetoothProfile#STATE_DISCONNECTING}
     *               </p>
     */
    void onHfpStatus(BluetoothDevice device, int status);

    /**
     * SPP连接状态回调
     *
     * @param device 蓝牙设备
     * @param status 连接状态
     *               <p>参考值如下<br>
     *               {@link
com.jieli.bluetooth.constant.StateCode#CONNECTION_DISCONNECT} or
     *               {@link com.jieli.bluetooth.constant.StateCode#CONNECTION_OK}
or
     *               {@link
com.jieli.bluetooth.constant.StateCode#CONNECTION_FAILED} or
     *               {@link
com.jieli.bluetooth.constant.StateCode#CONNECTION_CONNECTING} or
```

```
     *                {@link
com.jieli.bluetooth.constant.StateCode#CONNECTION_CONNECTED}
     *                </p>
     */
    void onSppStatus(BluetoothDevice device, int status);

    /**
     * 接收到设备的RCSP命令数据
     *
     * @param device 蓝牙设备
     * @param cmd    RCSP命令数据
     */
    void onDeviceCommand(BluetoothDevice device, CommandBase cmd);

    /**
     * 接收到设备的数据包
     *
     * @param device 蓝牙设备
     * @param data   数据包
     */
    void onDeviceData(BluetoothDevice device, byte[] data);


    /**
     * 接收到设备的语音数据
     *
     * @param device 蓝牙设备
     * @param data   语音数据
     */
    @Deprecated
    void onDeviceVoiceData(BluetoothDevice device, byte[] data);

    /**
     * 接收到设备的语音数据包
     *
     * @param device 蓝牙设备
     * @param data 语音信息
     */
    void onDeviceVoiceData(BluetoothDevice device, VoiceData data);

    /**
     * 设备VAD结束的回调
     *
     * @param device 蓝牙设备
     */
    void onDeviceVadEnd(BluetoothDevice device);

    /**
     * 错误事件回调
     *
     * @param error 错误信息
     */
    void onError(BaseError error);  //错误事件回调

}
```

1.3、命令回复类CommandCallback

```
public interface CommandCallback {
    /**
     * 命令回复回调
     *
     * @param cmd 回复命令
     *            <p>若是无回复的命令，则返回命令原型；
     *            若是有回复的命令，则返回回复命令</p>
     */
    void onCommandResponse(CommandBase cmd);

    /**
     * 发送错误回调
     *
     * @param error 错误信息
     */
    void onErrCode(BaseError error);
}
```

1.4、RCSP命令回复类RcspCommandCallback

```
public interface RcspCommandCallback {

    /**
     * 命令回复回调
     *
     * @param device 已连接设备
     * @param cmd    回复命令
     *               <p>若是无回复的命令，则返回命令原型；
     *               若是有回复的命令，则返回回复命令</p>
     */
    void onCommandResponse(BluetoothDevice device, CommandBase cmd);

    /**
     * 发送错误回调
     *
     * @param device 已连接设备
     * @param error  错误信息
     */
    void onErrCode(BluetoothDevice device, BaseError error);
}
```

2、 参数解释

2.1、命令参数

命令参数规定为以下几种：

1. **CommandBase** 无参无回复命令
2. **CommandWithParam** 带参无回复命令
3. **CommandWithParamAndResponse** 带参有回复命令
4. **CommandWithResponse** 无参有回复命令

其他命令参数或者自定义命令参数必须为以上基类的派生类

2.2、输入参数

1. 必须继承**BaseParameter**类，且必须实现**getParamData**方法

2.3、输出参数

1. 必须继承**CommonResponse**类

2.4、注意

注意参数的命名规则：
命令类： 最好以Cmd为后缀
参数类： 最好以Param为后缀
回复类： 最好以Response为后缀

2.5、例子

注：内置SDK提供了工具类：**CommandBuilder**，可以快速的生成命令，对于sdk已经实现的功能可以快速的生成命令。

2.5.1、命令类

如**GetSysInfoCmd**类

```
public class GetSysInfoCmd extends CommandWithParamAndResponse<GetSysInfoParam,
SysInfoResponse> {
    public GetSysInfoCmd(GetSysInfoParam param) {
        super(Command.CMD_GET_SYS_INFO, GetSysInfoCmd.class.getSimpleName(),
param);
    }
}
```

2.5.2、输入参数类

如**GetSysInfoParam**类

```
 public class GetSysInfoParam extends BaseParameter {
    private byte function; //功能码
    private int mask; //功能掩码

    public GetSysInfoParam(byte function, int mask) {
        this.mask = mask;
        this.function = function;
    }

    public byte getFunction() {
        return function;
    }

   public GetSysInfoParam setFunction(byte function) {
        this.function = function;
        return this;
    }

    public int getMask() {
        return mask;
```

```
    }

    public GetSysInfoParam setMask(int mask) {
        this.mask = mask;
        return this;
    }

    @Override
    public byte[] getParamData() {
        byte[] bytes = new byte[5];
        bytes[0] = function;
        System.arraycopy(CHexConver.intToBigBytes(mask), 0, bytes, 1, 4);
        return bytes;
    }

    @Override
    public String toString() {
        return "GetSysInfoParam{" +
                "function=" + function +
                "mask=" + mask +
                '}';
    }
}
```

2.5.3、输出回复类

如**SysInfoResponse**类

```
public class SysInfoResponse extends CommonResponse {
    private byte function; //功能码
    private List<AttrBean> attrs; //功能属性


    public void setFunction(byte function) {
        this.function = function;
    }

    public byte getFunction() {
        return function;
    }

    public void setAttrs(List<AttrBean> attrs) {
        if (attrs == null) {
            attrs = new ArrayList<>();
        }
        this.attrs = attrs;
    }

    public List<AttrBean> getAttrs() {
        return attrs;
    }

    @Override
    public String toString() {
        return "SysInfoResponse{" +
                "function=" + function +
                ", attrs=" + attrs +
```

```
                "} " + super.toString();
    }
}
```

3、 常量定义

3.1、命令集Command

```
public class Command {
    /**********************************************************************
    * 0x01 数据命令
    * 0x02 获取目标信息
    *       ...
    *       ...
    *       ...
    * 0xC0 - 0xCF TWS 命令
    * 0xD0~0xDF 辅助命令
    * 0xE0~0xEF OTA命令
    * 0xF0~0xFF 自定义命令
    **********************************************************************/
    public static final int CMD_DATA = 0x01; //数据命令
    public static final int CMD_GET_TARGET_FEATURE_MAP = 0x02;   //获取支持的特征掩码，返回一个U32的掩码，1代表支持，0代表不支持相关功能
    public static final int CMD_GET_TARGET_INFO = 0x03; //获取目标信息
    public static final int CMD_RECEIVE_SPEECH_START = 0x04;   //开始接收语音数据

    public static final int CMD_RECEIVE_SPEECH_STOP = 0x05;   //停止接收语音数据

    public static final int CMD_DISCONNECT_CLASSIC_BLUETOOTH = 0x06; //请求断开经典蓝牙
    public static final int CMD_GET_SYS_INFO = 0x07; //请求设备设置信息
    public static final int CMD_SET_SYS_INFO = 0x08; //设置设备设置信息
    public static final int CMD_SYS_INFO_AUTO_UPDATE = 0x09; //接收固件推送的状态信息no respond
    public static final int CMD_PHONE_CALL_REQUEST = 0x0A; //发起语音拨号请求
    public static final int CMD_SWITCH_DEVICE_REQUEST = 0x0B; //通知固件切换通信方式

    public static final int CMD_START_FILE_BROWSE = 0x0C; //请求目录浏览
    public static final int CMD_STOP_FILE_BROWSE = 0x0D; //结束目录浏览
    public static final int CMD_FUNCTION = 0x0E; //功能命令
    public static final int CMD_LRC_GET_START = 0x0F; //开始lrc歌词读取
    public static final int CMD_LRC_GET_STOP = 0x10; //结束lrc歌词读取
    public static final int CMD_LRC_PUSH_START_TTS = 0x11; //通知固件app开始播放tts语音
    public static final int CMD_EXTRA_CUSTOM = 0xFF;        // 外部自定义命令
    public static final int CMD_OTA_GET_DEVICE_UPDATE_FILE_INFO_OFFSET = 0xE1; //读取设备所需升级文件标识信息偏移
    public static final int CMD_OTA_INQUIRE_DEVICE_IF_CAN_UPDATE = 0xE2;   //查询设备是否可升级
    public static final int CMD_OTA_ENTER_UPDATE_MODE = 0xE3;   //进入升级模式
    public static final int CMD_OTA_EXIT_UPDATE_MODE = 0xE4;   //退出升级模式
    public static final int CMD_OTA_SEND_FIRMWARE_UPDATE_BLOCK = 0xE5;   //发送升级固件数据块
    public static final int CMD_OTA_GET_DEVICE_REFRESH_FIRMWARE_STATUS = 0xE6; //读取设备升级状态
```

```
    public static final int CMD_REBOOT_DEVICE = 0xE7;   //重启或关闭设备
    public static final int CMD_OTA_NOTIFY_UPDATE_CONTENT_SIZE = 0xE8; //设备通知主机升级文件大小
    public static final int CMD_ADV_SETTINGS = 0xC0;  //TWS设置命令
    public static final int CMD_ADV_GET_INFO = 0xC1;  //获取TWS设置信息
    public static final int CMD_ADV_DEVICE_NOTIFY = 0xC2; //设备通知TWS信息   [seq + connect_flag + battery quantity(3 bytes)]
    public static final int CMD_ADV_NOTIFY_SETTINGS = 0xC3; //设置TWS设备通知 [0 - 关闭通知 1 - 打开通知]
    public static final int CMD_ADV_DEV_REQUEST_OPERATION = 0xC4;
    public static final int CMD_NOTIFY_DEVICE_APP_INFO = 0xD0; //告知设备APP的信息

    public static final int CMD_SETTINGS_COMMUNICATION_MTU = 0xD1; //设置通讯MTU（默认为256）
    public static final int CMD_RECEIVE_SPEECH_CANCEL = 0xD2;   //取消接收语音数据

    public static final int CMD_PHONE_NUMBER_PLAY_MODE = 0xD3; //通知设备播放来电号码的方式(0 - 正常模式  1 - 播放文件模式)
    public static final int CMD_GET_DEV_MD5 = 0xD4; //获取设备MD5命令
    public static final int CMD_GET_LOW_LATENCY_SETTINGS = 0xD5; //获取低延时模式发送参数 （仅用于BLE，SPP不限制发数）
    public static final int CMD_CUSTOM = 0xF0;       //sdk内部自定义命令
    public static final int CMD_START_PERIPHERALS_SCAN = 0x12;//通知固件让固件开启经典蓝牙扫描，在充电仓发射模式下有效
    public static final int CMD_UPDATE_PERIPHERALS_RESULT = 0x13;//固件推送扫描到的蓝牙设备到app，在充电仓发射模式下有效
    public static final int CMD_STOP_PERIPHERALS_SCAN = 0x14;//通知固件让固件停止经典蓝牙扫描，在充电仓发射模式下有效
    @Deprecated public static final int CMD_START_FILE_TRANSFER = 0x16;//开始文件传输(废弃)
    @Deprecated public static final int CMD_STOP_FILE_TRANSFER = 0x17;//结束文件传输(废弃)
    @Deprecated public static final int CMD_NOTIFY_FILE_TRANSFER_OP = 0x18; //通知文件传输操作(废弃)
    public static final int CMD_SEARCH_DEVICE = 0x19; //查找设备
    public static final int CMD_RTC_EXPAND = 0x25;//闹钟拓展命令
    ...
}
```

3.2、错误码ErrorCode

```
package com.jieli.bluetooth.constant;

/**
 * 错误码
 *
 * Created by zqjaosnzhong on 2018/8/6.
 */

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服务连接失败
     */
    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;

    /**
     * 调用反射接口失败
     */
    public static final int SUB_ERR_REFLECT_WAY = 0x1014;
    /**
     * 没有权限
     */
    public static final int SUB_ERR_NO_PERMISSION = 0x1015;

    /* ======================== ERR_STATUS ========================*/
    /**
     * BLE设备未连接
     */
    public static final int SUB_ERR_BLE_CONNECT_FAILED = 0x2001;
    /**
     * 断开经典蓝牙失败
     */
    public static final int SUB_ERR_CLASSIC_BLUETOOTH_IS_CONNECTED = 0x2005;

    /*======================== ERR_COMMUNICATION ========================*/
    /**
     * 连接超时
     */
    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;
    /**
     * 解包错误
     */
    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;
    /**
     * 正在配对中
     */
    public static final int SUB_ERR_DEVICE_PAIRING = 0x300A;
    /**
     * 经典蓝牙连接中
     */
    public static final int SUB_ERR_EDR_CONNECTING = 0x300B;
    /**
     * Spp连接中
     */
    public static final int SUB_ERR_SPP_CONNECTING = 0x300C;
    /**
     * Ble连接中
     */
    public static final int SUB_ERR_BLE_CONNECTING = 0x300D;
    /**
     * 设备连接中
     */
    public static final int SUB_ERR_DEVICE_CONNECTING = 0x300E;

    /* ======================== ERR_OTA ========================*/
    /**
     * OTA升级失败
     */
    public static final int SUB_ERR_OTA_FAILED = 0x4001;

    /* ======================== ERR_OTHER ========================*/
    /**
     * 认证设备失败
     */
    public static final int SUB_ERR_AUTH_DEVICE = 0x5001;


}
```

3.3、状态码StateCode

StateCode包含几部分的常量定义：

```
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
    public final static int SCAN_TYPE_FLAG_PID = 3; //标识VID和PID
    public final static int SCAN_TYPE_FLAG_EDR_MESSAGE = 4; //标识BR/EDR的信息

    /*-------------------------------------------------------------------------
     * TWS
     *-------------------------------------------------------------------------*/
    /* ==================== *
     * 耳机状态
     * ==================== */
    /**
     * 不显示弹窗
     */
    public final static int TWS_HEADSET_STATUS_DIMISS = 0;
    /**
     * 蓝牙未连接
     */
    public final static int TWS_HEADSET_STATUS_DISCONNECTED = 1;
    /**
     * 蓝牙已连接
     */
    public final static int TWS_HEADSET_STATUS_CONNECTED = 2;
    /**
     * 蓝牙正在连接
     */
    public final static int TWS_HEADSET_STATUS_CONNECTING = 3;

    //C0
    public final static int ADV_SETTINGS_ERROR_IN_GAME_MODE = 1; //游戏模式导致设置
失败
    public final static int ADV_SETTINGS_ERROR_DEVICE_NAME_LENGTH_OVER_LIMIT =
2; //蓝牙名长度超出限制
    public final static int ADV_SETTINGS_ERROR_LED_SETTINGS_FAILED = 3; //非蓝牙模
式设置闪灯失败

  }
```

3.4、属性和功能码AttrAndFunCode

AttrAndFunCode包含几部分的常量定义：

```
/**
 * 常量分为以下几部分
 * 1、设备属性：TargetInfo属性
 * 2、系统功能码
 * 3、属性码：属性码是对属性的一个表示，读取属性数据通过32为的掩码读取，掩码为属性的第（属性值）
bit
 * 4、功能码
 */
public class AttrAndFunCode {


        /*------------------------------------------------------------------------
     * TargetInfo属性
     *-----------------------------------------------------------------------*/
    public final static int ATTR_TYPE_PROTOCOL_VERSION = 0; //协议版本标识
    public final static int ATTR_TYPE_POWER_UP_SYS_INFO = 1;   //开机系统信息
    public final static int ATTR_TYPE_EDR_ADDR = 2;   //经典蓝牙地址
    public final static int ATTR_TYPE_PLATFORM = 3;   //ai平台
    public final static int ATTR_TYPE_FUNCTION_INFO = 4;   //功能支持掩码
    public final static int ATTR_TYPE_FIRMWARE_INFO = 5;   //固件版本号掩码
    public final static int ATTR_TYPE_SDK_TYPE = 6; //SDK的平台类型[0是没有AI的
BTMate，1是带AI的AIMate，2是693x，3是695X，4是697X]
    public final static int ATTR_TYPE_UBOOT_VERSION = 7;  //uboot版本标识
    /*
     * 单双备份区别标志
     * Byte0: 是否支持双备份升级 (0 --- 不支持(单备份)  1 --- 支持 (双备份))
     * Byte1: 是否需要下载BootLoader (0 --- 不需要  1 --- 需要)
     * Byte2: 单备份通讯方式 (0 --- 不使能[正常升级流程]  1 --- 强制使用BLE升级  2 ---
强制使用SPP升级)
     */
    public final static int ATTR_TYPE_SUPPORT_DOUBLE_BACKUP = 8;
    /*
     *升级状态
     * Byte0 : SDK和loader标识 (0 : SDK标识  1 : loader标识)
     * Byte1 : 升级方式 (0 : 正常升级 1 : 强制升级)
     */
    public final static int ATTR_TYPE_MANDATORY_UPGRADE_FLAG = 9;  //强制升级标识
    public final static int ATTR_TYPE_VID_AND_PID = 10;  //厂商号和产品号标识
    public final static int ATTR_TYPE_AUTH_KEY = 11; //获取产品的AuthKey
    public final static int ATTR_TYPE_PROJECT_CODE = 12; //获取产品的ProCode
    public final static int ATTR_TYPE_PROTOCOL_MTU = 13; //协议MTU
    //Alexa项目增加
    public final static int ATTR_TYPE_ALLOW_CONNECT = 14; //是否允许连接   0 --- 允
许   1 --- 不允许

    @Deprecated
    public final static int ATTR_TYPE_NAME = 16;   //设备名标识

    public final static int ATTR_TYPE_CONNECT_BLE_ONLY = 17; //是否仅仅连接BLE 和
BLE地址（BR30拓展定义）
    public final static int ATTR_TYPE_PERIPHERALS_SUPPORT = 18; //是否支持外设模式
    /*
     *设备是否支持功能(BR30增加)
```

```
     * Bit0: 是否支持获取MD5(0 --- 不支持 1 --- 支持(使用0xD4命令获取MD5值))
     * Bit1: 是否延时发数模式(0 --- 不是 1 --- 是(使用0xD5命令获取延时发数参数)[仅BLE通讯有效])
     * Bit2: 是否支持查找设备(0 --- 不支持 1 --- 支持)
     * Bit3: 是否支持声卡功能(0 --- 不支持 1 --- 支持)
     * Bit4: 是否禁止APP调节设备音效(0 --- 不禁止 1 --- 禁止)
     * Bit5: 是否支持外挂Flash传输功能(0 --- 不支持 1 --- 支持(使用0xD6命令获取外挂Flash的配置信息))
     */
    public final static int ATTR_TYPE_DEV_SUPPORT_FUNC = 19;
    public final static int ATTR_TYPE_RECODE_FILE_TRANSFER = 20; //录音文件传输(Bit0 : 0 --- 不支持  1 --- 支持)

    //鸿巨昌项目增加
    public final static int ATTR_TYPE_CUSTOM_VER = 31; //自定义版本信息


    /*-------------------------------------------------------------------------
     * Host 属性
     *-------------------------------------------------------------------------*/
    public final static int HOST_ATTR_TYPE_SPEECH_FAST_MODE = 0; //获取主机是否支持语音上报快速模式

    /**
     * 功能码，下面的属性读写和功能操作都要基于对应的功能码操作
     */
    public final static byte SYS_INFO_FUNCTION_PUBLIC = (byte) 0xff; //共用属性
    public final static byte SYS_INFO_FUNCTION_BT = (byte) 0x00; //蓝牙模式相关信息
    public final static byte SYS_INFO_FUNCTION_MUSIC = (byte) 0x01; //设备相关信息
    public final static byte SYS_INFO_FUNCTION_RTC = (byte) 0x02; //时钟相关信息
    public final static byte SYS_INFO_FUNCTION_AUX = (byte) 0x03; //AUX相关
    public final static byte SYS_INFO_FUNCTION_FM = (byte) 0x04; //FM相关
    public final static byte SYS_INFO_FUNCTION_LIGHT = (byte) 0x05; //Light相关
    public final static byte SYS_INFO_FUNCTION_FMTX = (byte) 0x06; //FMTTX, 频点发射 [跳过，复用到公共属性]
    public final static byte SYS_INFO_FUNCTION_EQ = (byte) 0x07; //EQ 相关信息 [跳过，复用到公共属性]

    public final static byte SYS_INFO_FUNCTION_LOW_POWER = (byte) 0x16; //低功耗模式


    //公用属性
    public final static byte SYS_INFO_ATTR_BATTERY = (byte) 0x00; //系统电量
    public final static byte SYS_INFO_ATTR_VOLUME = (byte) 0x01; //系统音量
    public final static byte SYS_INFO_ATTR_MUSIC_DEV_STATUS = (byte) 0x02; //usb/sd状态
    public final static byte SYS_INFO_ATTR_ERR = (byte) 0x03; // 错误
    public final static byte SYS_INFO_ATTR_EQ = (byte) 0x04; // eq
    public final static byte SYS_INFO_ATTR_FILE_TYPE = (byte) 0x05; // //目录浏览文件类型设置
    public final static byte SYS_INFO_ATTR_CUR_MODE_TYPE = (byte) 0x06; // //模式变化
    public final static byte SYS_INFO_ATTR_LIGHT = (byte) 0x07; //灯光状态
    public final static byte SYS_INFO_ATTR_FM_TX = (byte) 0x08; //发射频点
    public final static byte SYS_INFO_ATTR_EMITTER_MODE = (byte) 0x09; //外设模式状态
```

```
    public final static byte SYS_INFO_ATTR_EMITTER_CONNECT_STATUS = (byte) 0x0a;
//外设连接状态
    public final static byte SYS_INFO_ATTR_HIGH_AND_BASS = (byte) 0x0b; //高低音设
置
    public final static byte SYS_INFO_ATTR_EQ_PRESET_VALUE = (byte) 0x0c; //EQ预
设值
    public final static byte SYS_INFO_ATTR_CURRENT_NOISE_MODE = (byte) 0x0d; //耳
机当前噪声处理模式
    public final static byte SYS_INFO_ATTR_ALL_NOISE_MODE = (byte) 0x0e; //耳机所
有噪声处理模式信息
    public final static byte SYS_INFO_ATTR_PHONE_STATUS = (byte) 0x0f; //通话状态
    public final static byte SYS_INFO_ATTR_FIXED_LEN_DATA_FUN = (byte) 0x10; //固
定数据长度的拓展功能
    public final static byte SYS_INFO_ATTR_SOUND_CARD_EQ_FREQ = (byte) 0x11; //声
卡功能EQ频率
    public final static byte SYS_INFO_ATTR_SOUND_CARD_EQ_GAIN = (byte) 0x12; //声
卡功能EQ增益
    public final static byte SYS_INFO_ATTR_SOUND_CARD = (byte) 0x13; //声卡功能
    //BT 属性
    public final static byte SYS_INFO_ATTR_ID3_TITLE = (byte) 0x00;//歌曲名
    public final static byte SYS_INFO_ATTR_ID3_ARTIST = (byte) 0x01;//作者
    public final static byte SYS_INFO_ATTR_ID3_ALBUM = (byte) 0x02;//专辑
    public final static byte SYS_INFO_ATTR_ID3_NUMBER = (byte) 0x03;//序号
    public final static byte SYS_INFO_ATTR_ID3_TOTAL = (byte) 0x04;//总歌单长度
    public final static byte SYS_INFO_ATTR_ID3_GENRE = (byte) 0x05;//类型
    public final static byte SYS_INFO_ATTR_ID3_TOTAL_TIME = (byte) 0x06;//总时长
    public final static byte SYS_INFO_ATTR_ID3_PLAY_STATUS = (byte) 0x07;//播放状
态
    public final static byte SYS_INFO_ATTR_ID3_CURRENT_TIME = (byte) 0x08;//当前
时间


    //MUSIC 属性
    public final static byte SYS_INFO_ATTR_MUSIC_STATUS_INFO = (byte) 0x00; // 音
乐状态
    public final static byte SYS_INFO_ATTR_MUSIC_FILE_NAME_INFO = (byte) 0x01;
//播放文件信息
    public final static byte SYS_INFO_ATTR_MUSIC_PLAY_MODE = (byte) 0x02; //播放
模式

    //RTC属性
    public final static byte SYS_INFO_ATTR_RTC_TIME = (byte) 0x00; // 时间同步
    public final static byte SYS_INFO_ATTR_RTC_ALARM = (byte) 0x01; // 闹钟信息操作
    public final static byte SYS_INFO_ATTR_RTC_CURRENT_ALARM_INDEX = (byte)
0x02; // 当前闹钟序号
    public final static byte SYS_INFO_ATTR_RTC_STOP_ALARM = (byte) 0x03; // 停止
闹钟
    public final static byte SYS_INFO_ATTR_RTC_ALARM_VER = (byte) 0x04; // 闹钟结
构版本
    public final static byte SYS_INFO_ATTR_RTC_ALARM_DEFAULT_BELL_LIST = (byte)
0x05;//默认铃声列表
    public final static byte SYS_INFO_ATTR_RTC_ALARM_AUDITION = 0x06;//闹钟铃声试听
    public final static byte SYS_INFO_ATTR_RTC_ALARM_EXPAND_FLAG = 0x07 ;//闹钟拓
展字段支持标识

    //aux属性
    public final static byte SYS_INFO_ATTR_AUX_STATU = (byte) 0x00; // aux模式
```

```
    //fm属性
    public final static byte SYS_INFO_ATTR_FM_STATU = (byte) 0x00; // fm状态属性
    public final static byte SYS_INFO_ATTR_FM_FRE_INFO = (byte) 0x01; // fm 频道
属性


    //cmd命令码
    //ID3音乐控制命令参数
    public final static byte FUNCTION_BT_CMD_ID3_PLAY_OR_PAUSE = 0x01; // 暂停/播
放
    public final static byte FUNCTION_BT_CMD_ID3_PLAY_PREV = 0x02; // 上一曲
    public final static byte FUNCTION_BT_CMD_ID3_PLAY_NEXT = 0x03; // 下一曲
    public final static byte FUNCTION_BT_CMD_ID3_DATA_PUSH_SWITCH = 0x04; // 数据
推送开关
    //设备音乐控制命令参数
    public final static byte FUNCTION_MUSIC_CMD_PLAY_OR_PAUSE = 0x01; // 暂停/播放
    public final static byte FUNCTION_MUSIC_CMD_PLAY_PREV = 0x02; // 上一曲
    public final static byte FUNCTION_MUSIC_CMD_PLAY_NEXT = 0x03; // 下一曲
    public final static byte FUNCTION_MUSIC_CMD_NEXT_PLAYMODE = 0x04; // 下一个播
放模式
    public final static byte FUNCTION_MUSIC_CMD_NEXT_EQ_MODE = 0x05; // 下一个EQ模
式
    public final static byte FUNCTION_MUSIC_CMD_FAST_FORWARD = 0x07; // 快进
    public final static byte FUNCTION_MUSIC_CMD_RETREAT_QUICKLY = 0x06; // 快退

    //AI下状态恢复命令
    public final static byte FUNCTION_BT_CMD_RESTORE_STATUS = 0x01; // 语音识别完恢
复状态
    //状态恢复命令的附带参数
    public final static byte FUNCTION_BT_CMD_RESTORE_STATUS_RESTORE = 0x00; // 维
持状态不变
    public final static byte FUNCTION_BT_CMD_RESTORE_STATUS_RESTORE_MODE = 0x01;
//跳回原来模式恢复状态
    public final static byte FUNCTION_BT_CMD_RESTORE_STATUS_RESTORE_PLAY_PREV =
0x02; //跳回原来模式上一曲
    public final static byte FUNCTION_BT_CMD_RESTORE_STATUS_RESTORE_PLAY_NEXT =
0x03; //跳回原来模式下一曲
    public final static byte FUNCTION_BT_CMD_RESTORE_STATUS_RESTORE_PAUSE =
0x04; //跳回原来模式暂停
    public final static byte FUNCTION_BT_CMD_RESTORE_STATUS_RESTORE_PLAY = 0x05;
//跳回原来模式播放

    //aux播放控制
    public final static byte FUNCTION_AUX_CMD_PAUSE_OR_PLAY = 0x01; // 下一个EQ模
式


    //fm控制
    public final static byte FUNCTION_FM_CMD_PLAY_OR_PAUSE = 0x01; // 暂停/播放
    public final static byte FUNCTION_FM_CMD_PREV_FREQ = 0x02; // 上一个频点
    public final static byte FUNCTION_FM_CMD_NEXT_FREQ = 0x03; // 下一个频点
    public final static byte FUNCTION_FM_CMD_PREV_CHANNEL = 0x04; // 上一个频道
    public final static byte FUNCTION_FM_CMD_NEXT_CHANNEL = 0x05; // 下一个频道
    public final static byte FUNCTION_FM_CMD_FREQ_SCAN = 0x06; // 频道扫描：0x00：
全段扫描    0x01：向前扫描      0x02：向后扫描    0x03：停止扫描
    public final static byte FUNCTION_FM_CMD_SELECT_CHANNEL = 0x07; // 选择频道
    public final static byte FUNCTION_FM_CMD_DEL_CHANNEL = 0x08; //  删除频道
```

```
    public final static byte FUNCTION_FM_CMD_SELECT_FREQL = 0x09; // 选择频点


    /*-------------------------------------------------------------------
     * ADV Info属性
     *-------------------------------------------------------------------*/
    public final static int ADV_TYPE_BATTERY_QUANTITY = 0;          //电量 [3
bytes; 不可设置]
    public final static int ADV_TYPE_DEVICE_NAME = 1;               //设备名称 [有长
度限制]
    public final static int ADV_TYPE_KEY_SETTINGS = 2;              //按键设置
    public final static int ADV_TYPE_LED_SETTINGS = 3;              //灯光设置
    public final static int ADV_TYPE_MIC_CHANNEL_SETTINGS = 4;      //麦通道设置
    public final static int ADV_TYPE_WORK_MODE = 5;                 //工作模式
    public final static int ADV_TYPE_PRODUCT_MESSAGE = 6;           //产品信息
    public final static int ADV_TYPE_CONNECTED_TIME = 7;            //连接时间
    public final static int ADV_TYPE_IN_EAR_CHECK = 8;              //入耳检测
    public final static int ADV_TYPE_LANGUAGE = 9;                  //语言类型
    public final static int ADV_TYPE_ANC_MODE_LIST = 10;            //ANC模式列表


    /*key specific mode*/
    public final static int KEY_FUNC_ID_SWITCH_ANC_MODE = 255;       //切换ANC模式
功能ID

    /*-------------------------------------------------------------------
     * 固定长度数据功能
     *-------------------------------------------------------------------*/
    public final static int FIXED_LEN_DATA_TYPE_REVERBERATION = 0;          //混响
    public final static int FIXED_LEN_DATA_TYPE_DYNAMIC_LIMITER = 1;          //动
态限幅器
    public static final int FIXED_LEN_DATA_TYPE_SOUND_CARD_EFFECT = 2; // 声卡功能
声音效果
    public static final int FIXED_LEN_DATA_TYPE_SOUND_CARD_ATMOSPHERE = 3; // 声
卡功能气氛
    public static final int FIXED_LEN_DATA_TYPE_SOUND_CARD_MIC_ARG = 4; // 声卡功
能mic参数



}
```

4、接口详细介绍

4.1 通用接口

4.1.1 addEventListener

类型	内容	备注
方法原形	boolean addEventListener(IBluetoothEventListener listener)	
作用描述	注册蓝牙事件监听器	
参数解释	listener：蓝牙事件监听器	用于监听蓝牙事件的所有回调
结果说明	操作结果	

4.1.2 removeEventListener

类型	内容	备注
方法原形	boolean removeEventListener(IBluetoothEventListener listener)	
作用描述	注销蓝牙事件监听器	
参数解释	listener：蓝牙事件监听器	
结果说明	操作结果	

4.1.3 configure

类型	内容	备注
方法原形	void configure(BluetoothOption option)	建议在Application初始化的时候调用
作用描述	配置蓝牙通讯参数	
参数解释	option：蓝牙通讯参数	
结果说明	操作结果	

4.2 蓝牙操作接口

4.2.1 isBluetoothEnabled

类型	内容	备注
方法原形	boolean isBluetoothEnabled()	
作用描述	是否打开蓝牙模块	
参数解释		
结果说明	布尔值	

4.2.2 openOrCloseBluetooth

类型	内容	备注
方法原形	boolean openOrCloseBluetooth(boolean isEnable)	
作用描述	打开或关闭蓝牙模块	
参数解释	isEnable : 是否使能	
结果说明	操作结果	

4.2.3 scan

类型	内容	备注
方法原形	int scan(int timeout)	调用时请确保蓝牙已开启
作用描述	扫描BLE蓝牙设备	
参数解释	timeout : 扫描超时时长(单位:ms)	
结果说明	操作结果	操作成功后，等待回调 IBluetoothEventListener#onDiscoveryStatus IBluetoothEventListener#onDiscovery

类型	内容	备注
方法原形	int scan(int type, int timeout)	调用时请确保蓝牙已开启
作用描述	扫描蓝牙设备	
参数解释	type : 扫描类型(0 : BLE 1 : classic) timeout : 扫描超时时长(单位:ms)	
结果说明	操作结果	操作成功后，等待回调 IBluetoothEventListener#onDiscoveryStatus IBluetoothEventListener#onDiscovery

4.2.4 stopScan

类型	内容	备注
方法原形	int stopScan()	
作用描述	停止蓝牙设备扫描	
参数解释		
结果说明	操作结果	IBluetoothEventListener#onDiscoveryStatus 会回调

4.2.5 isConnectedByProfile

类型	内容	备注
方法原形	boolean isConnectedByProfile(BluetoothDevice device)	
作用描述	经典蓝牙是否已连接	
参数解释	device : 蓝牙设备	
结果说明	布尔值	

4.2.6 startConnectByBreProfiles

类型	内容	备注
方法原形	void startConnectByBreProfiles(BluetoothDevice device)	
作用描述	开始连接经典蓝牙设备	
参数解释	device： 蓝牙设备	
结果说明		

4.2.7 disconnectByProfiles

类型	内容	备注
方法原形	boolean disconnectByProfiles(BluetoothDevice device)	
作用描述	断开指定的经典蓝牙设备	
参数解释	device : 经典蓝牙设备	
结果说明	操作结果	

4.2.8 fastConnect

类型	内容	备注
方法原形	boolean fastConnect()	
作用描述	快速连接	(与最近连接历史设备建立连接，包括打开蓝牙、扫描设备、绑定设备、建立通信通道等)
参数解释		
结果说明	布尔值	true : 开始快速连接 或 设备已连接 false : 操作失败 没有历史设备记录

4.2.9 disconnect

类型	内容	备注
方法原形	void disconnect()	
作用描述	断开当前已连接的蓝牙设备	
参数解释		
结果说明		

类型	内容	备注
方法原形	void disconnect(BluetoothDevice device)	
作用描述	断开指定的蓝牙设备	
参数解释	device : 蓝牙设备	
结果说明		

4.2.10 getFoundedBluetoothDevices

类型	内容	备注
方法原形	ArrayList<BluetoothDevice> getFoundedBluetoothDevices()	
作用描述	获取扫描到的蓝牙设备列表	
参数解释		
结果说明	扫描到的蓝牙设备列表	

4.2.11 connect

类型	内容	备注
方法原形	void connect(BluetoothDevice device)	
作用描述	连接指定的蓝牙设备	
参数解释	device : 蓝牙设备	
结果说明		IBluetoothEventListener#onConnection 回调连接状态

4.2.12 disconnect

类型	内容	备注
方法原形	void disconnect(BluetoothDevice device)	
作用描述	断开指定蓝牙设备的连接	
参数解释	device : 蓝牙设备	
结果说明		IBluetoothEventListener#onConnection 回调连接状态

4.2.13 unPair

类型	内容	备注
方法原形	int unPair(BluetoothDevice device)	
作用描述	取消蓝牙设备配对	
参数解释	device : 蓝牙设备	
结果说明	操作结果	

4.2.14 getConnectedDevice

类型	内容	备份
方法原形	BluetoothDevice getConnectedDevice()	
作用描述	获取已连接蓝牙设备	
参数解释		
结果说明	蓝牙设备	

4.2.15 getHistoryBleDeviceList

类型	内容	备注
方法原形	List<HistoryBluetoothDevice> getHistoryBleDeviceList()	
作用描述	获取连接历史纪录	
参数解释		
结果说明	连接历史纪录	

4.2.16 clearHistoryDeviceRecord

类型	内容	备注
方法原形	void clearHistoryDeviceRecord()	
作用描述	清除历史设备记录	
参数解释		
结果说明		

4.2.17 getMappedDeviceAddress

类型	内容	备注
方法原形	String getMappedDeviceAddress(String deviceAddress)	
作用描述	获取映射关系的设备地址	
参数解释		
结果说明		

4.2.18 synchronizationBtDeviceStatus

类型	内容	备注
方法原形	void synchronizationBtDeviceStatus()	
作用描述	同步当前设备连接状态	
参数解释		
结果说明		

4.3 命令数据操作接口

4.3.1 sendCommandAsync

类型	内容	备注
方法原形	void sendCommandAsync(CommandBase cmd, int timeoutMs, CommandCallback callback)	
作用描述	异步发送命令数据	建议使用'sendCommandAsync(BluetoothDevice, CommandBase, int, RcspCommandCallback)'代替
参数解释	cmd : 命令 timeoutMs : 超时时间(单位:ms) callback : 命令回复的回调	参考CommandCallback
结果说明		

类型	内容	备注
方法原形	void sendCommandAsync(BluetoothDevice device, CommandBase cmd, int timeoutMs, RcspCommandCallback callback)	多设备交互接口，指定发送设备
作用描述	指定设备异步发送命令数据	
参数解释	device: 已连接设备 cmd : RCSP命令 timeoutMs : 超时时间(单位:ms) callback : rscp命令回复的回调	参考RcspCommandCallback
结果说明		

4.3.2 sendData

类型	内容	备注
方法原形	void sendData(CommandBase cmd, byte[] data)	
作用描述	发送数据包	
参数解释	cmd : 响应的命令 data : 数据包	**注意：cmd 是需要响应的命令, 与其他命令接口参数区分** 例如，语音数据 是响应 开始语音命令的
结果说明		

类型	内容	备注
方法原形	void sendData(BluetoothDevice device, CommandBase cmd, byte[] data)	多设备交互接口，指定发送设备
作用描述	指定设备发送数据包	
参数解释	device: 已连接设备 cmd : 响应的命令 data : 数据包	**注意：cmd 是需要响应的命令, 与其他命令接口参数区分** 例如，语音数据 是响应 开始语音命令的
结果说明		

4.3.3 sendCommandResponse

类型	内容	备注
方法原形	void sendCommandResponse(CommandBase cmd)	
作用描述	发送命令的回复	
参数解释	cmd : 命令	需要添加状态码
结果说明		

类型	内容	备注
方法原形	void sendCommandResponse(BluetoothDevice device, CommandBase cmd)	多设备交互接口，指定发送设备
作用描述	发送命令的回复	
参数解释	device: 已连接设备 cmd : 命令	需要添加状态码
结果说明		

4.4 过渡数据接口

此接口主要用于控制层接收通讯层的数据，尽量不要使用

4.4.1 sendDataToDevice

类型	内容	备注
方法原形	boolean sendDataToDevice(BluetoothDevice device, byte[] data)	
作用描述	发送数据包到已连接的蓝牙设备	
参数解释	device : 蓝牙设备 data : 数据包	
结果说明	操作结果	

4.4.2 receiveDataFromDevice

类型	内容	备注
方法原形	void receiveDataFromDevice(BluetoothDevice device, byte[] data)	
作用描述	接收蓝牙设备返回的数据	
参数解释	device : 蓝牙设备 data : 数据包	
结果说明		

4.4.3 errorEventCallback

类型	内容	备注
方法原形	void errorEventCallback(BaseError error)	一般是发送数据超时或发送失败的回调
作用描述	错误事件返回	
参数解释	error : 错误事件	参考ErrorCode
结果说明		

4.5 多设备管理接口

4.5.1 getConnectedDeviceList

类型	内容	备注
方法原形	List<BluetoothDevice> getConnectedDeviceList()	
作用描述	获取已连接设备列表	
参数解释		
结果说明	已连接设备列表	

4.5.2 isConnectedBtDevice

类型	内容	备注
方法原形	boolean isConnectedBtDevice(BluetoothDevice device)	
作用描述	判断设备是否已连接	
参数解释	device : 蓝牙设备	
结果说明	是否已连接	

4.5.3 isUseBtDevice

类型	内容	备注
方法原形	boolean isUseBtDevice(BluetoothDevice device)	
作用描述	判断设备是否正在使用设备	
参数解释	device : 蓝牙设备	
结果说明	是否正在使用设备	

4.5.4 switchConnectedDevice

类型	内容	备注
方法原形	void switchConnectedDevice(BluetoothDevice device)	
作用描述	切换已连接设备为正在使用设备	
参数解释	device : 已连接设备	
结果说明		可以通过IBluetoothEventListener#IBluetoothEventListener获取切换情况

5、具体功能说明

5.1、属性数据结构

属性AttrBean

字节长度(Byte)	名称	备注
1	size	属性长度，不包括本字节长度
1	attr type	属性类型(参考AttrAndFunCode)
size - 1	data	属性内容

在GetTargetInfoCmd、GetSysInfoCmd、UpdateSysInfoCmd、SetSysInfoCmd等指令的参数或者响应是通过属性数据结构组装的，attr type的值为掩码的bit位（例如：bit1的attr type为1）。sdk为该数据结构提供了AttrBean类，

5.2、设备配置

命令原型	GetTargetInfoCmd
作用	获取设备信息
方向	APP --> 固件
是否需要回复	需要
参数	GetTargetInfoParam mask : 查询固件配置32位掩码(参考AttrAndFunCode#ATTR_TYPE_XXX))
回复	TargetInfoResponse

获取设备信息的参数GetTargetInfoParam

```
public class GetTargetInfoParam extends BaseParameter {    private int mask;//配置信息查询掩码    private byte platform;//手机平台 0 : android 1: iOS  2 : web app ... }
```

设备信息的回复TargetInfoResponse

```
public class TargetInfoResponse extends CommonResponse {
    private String versionName; //设备版本名称
    private int versionCode;    // 设备版本信息
    private String protocolVersion;//协议版本     //经典蓝牙相关信息
    private String edrAddr;//经典蓝牙地址
    private int edrStatus = 0;//经典蓝牙的连接状态
    private int edrProfile = 0;//经典蓝牙支持的协议     //BLE相关信息
    private String bleAddr; //BLE地址     //ai平台相关参数
    private int platform = -1;//AI平台类型（0：图灵；1：deepbrain；2：小米）
    private String license;//设备唯一标示     //SysInfo属性
    private int volume;// 设备音量
    private int maxVol;
    private int quantity;   // 设备电量
    private int functionMask;// 功能支持掩码
    private byte curFunction;// 当前模式
    private int sdkType; //是否为标准sdk 默认false即默认为ai sdk
    private String name;    //名字
    private int pid;        // 产品ID
    private int vid;        // 厂商ID
    private int mandatoryUpgradeFlag; //强制升级
    private int ubootVersionCode; //uboot 版本号
    private String ubootVersionName;  //uboot 版本名称
    private boolean isSupportDoubleBackup; //是否支持双备份升级（单备份[false]，需要断开回连过程；双备份[true]，不需要断开回连过程）
    private boolean isNeedBootLoader; //是否需要下载boot loader
    private int singleBackupOtaWay; //单备份OTA连接方式(00 -- 不使能 01 -- BLE  02 -- SPP)
    private int allowConnectFlag; //是否允许连接 0 -- 允许 1 -- 不允许     //用于服务器校验产品信息
    private String authKey; //认证秘钥
    private String projectCode; //项目标识码     //用于自定义版本信息
    private String customVersionMsg;    private boolean bleOnly; //是否仅仅连接ble设备，
    private boolean emitterSupport;//是否支持外设模式
    private int emitterStatus;//0x00:普通模式  0x01:外设模式
    private boolean isSupportMD5; //是否支持MD5读取
    private boolean isGameMode; //是否游戏模式
    private boolean isSupportSearchDevice; //是否支持查找设备
    private boolean supportOfflineShow = false;//是否支持usb、sd、linein不在线时显示功能图标
    private boolean supportUsb = true; //是否支持usb
    private boolean supportSd0 = true;//是否支持sd0
    private boolean supportSd1 = true;//是否支持sd1
    private boolean supportVolumeSync = false;//是否支持音量同步
    ...
}
```

5.3、设备参数

命令原型	GetSysInfoCmd
作用	获取设备的状态信息
方向	APP --> 固件
是否需要回复	需要
参数	GetSysInfoParam function : 功能码 (参考AttrAndFunCode#SYS_INFO_FUNCTION_XXX) mask : 查询固件状态32位掩码(参考AttrAndFunCode#SYS_INFO_ATTR_XXX))
回复	SysInfoResponse function : 功能码(参考AttrAndFunCode#SYS_INFO_FUNCTION_XXX) attrs:功能属性(参考AttrBean)

命令原型	SetSysInfoCmd
作用	设置设备的状态信息
方向	APP --> 固件
是否需要回复	需要
参数	SetSysInfoParam function : 功能码(参考AttrAndFunCode#SYS_INFO_FUNCTION_XXX) attrs:功能属性(参考AttrBean)
回复	CommonResponse(通用回复，无有效参数)

命令原型	UpdateSysInfoCmd
作用	推送设备的状态信息
方向	固件 --> APP
是否需要回复	不需要
参数	UpdateSysInfoParam function : 功能码(参考AttrAndFunCode#SYS_INFO_FUNCTION_XXX) attrs:功能属性(参考AttrBean)
回复	

获取设备状态信息的参数GetSysInfoParam

```
public class GetSysInfoParam extends BaseParameter {    private byte function;
//功能码    private int mask; //功能掩码    ...}
```

设备状态信息的回复SysInfoResponse

```
public class SysInfoResponse extends CommonResponse {    private byte function;
//功能码    private List<AttrBean> attrs; //功能属性    ...}
```

设置设备状态信息的参数SetSysInfoParam

```
public class SetSysInfoParam extends BaseParameter {    private byte function;
//功能码    private List<AttrBean> attrBeanList; //功能属性 ... }
```

推送设备状态信息的参数UpdateSysInfoParam

```
public class UpdateSysInfoParam extends BaseParameter {    private
List<AttrBean> attrBeanList; //功能列表    private byte function; //当前模式 ...}
```

5.4、录音开始命令

命令原型	StartSpeechCmd
作用	开始语音传输
方向	APP --> 固件
是否需要回复	需要
参数	StartSpeechParam
回复	CommonResponse(通用回复，无有效参数)

开始语音传输的参数StartSpeechParam

```
public class StartSpeechParam extends BaseParameter {

    private byte type;//0 : pcm  1 : speex  2 : opus
    private byte freq;//8 : 8K   16 : 16K
    private byte way;//0:固件断句，1：app断句
    ...
}
```

5.5、录音结束命令

命令原型	StopSpeechCmd
作用	结束语音传输
方向	APP --> 固件
是否需要回复	需要
参数	StopSpeechParam
回复	CommonResponse(通用回复，无有效参数)

结束语音传输的参数StopSpeechParam

```
public class StopSpeechParam extends BaseParameter {

    private byte reason; //停止原因（0:正常，2：取消输入）
    ...
}
```

5.6、电话拨号命令

命令原型	PhoneCallRequestCmd
作用	请求拨打电话
方向	APP --> 固件
是否需要回复	需要
参数	PhoneCallRequestParam
回复	CommonResponse(通用回复，无有效参数)

电话播放的参数PhoneCallRequestParam

```
public class PhoneCallRequestParam extends BaseParameter {
    private int type;  //拨号方式  0: 重拨  1 : 拨打号码
    private String number; //电话号码
    ...
}
```

5.7、自定义命令

命令原型	CustomCmd
作用	自定义命令(用于客户拓展)
方向	APP <--> 固件
是否需要回复	需要
参数	CustomParam(数据需要客户自行处理)
回复	CustomResponse(数据需要客户自行处理)

自定义命令的参数CustomParam

```
public class CustomParam extends BaseParameter {
    private byte[] data;//自定义数据
    ...
}
```

自定义命令的回复CustomResponse

```
public class CustomResponse extends CommonResponse {
    private byte[] data; //自定义回复数据
    ...
}
```

5.8、设置耳机功能命令

命令原型	SetADVInfoCmd
作用	设置耳机功能
方向	APP --> 固件
是否需要回复	需要
参数	SetADVInfoParam（参考AttrAndFunCode#ADV_TYPE_XXX)
回复	SetADVResponse result : 结果码(0 -- 成功，1 -- 游戏模式导致设置失败)

设置耳机功能的参数SetADVInfoParam

```
public class SetADVInfoParam extends BaseParameter {    private byte[]
payload;//设置数据    ...}
```

payload的数据格式是属性数据结构(参考AttrBean)

- ADV_TYPE_DEVICE_NAME : size + type + name(最大31Byte)
- ADV_TYPE_KEY_SETTINGS : size + type + value(n Bytes)
  - value 可以是一个或多个，格式 : keyId(1Byte) + actionId(1Byte) + funId(1Byte)
- ADV_TYPE_LED_SETTINGS: size + type + value(n Bytes)
  - value可以是一个或多个，格式：sceneId(1Byte) + funId(1Byte)
- ADV_TYPE_WORK_MODE: size + type + funId(1Byte)
- ADV_TYPE_MIC_CHANNEL_SETTINGS: size + type + funId(1Byte)
- ADV_TYPE_CONNECTED_TIME : size + type + time(4Bytes,以秒为单位)

设置耳机功能的回复SetADVResponse

```
public class SetADVResponse extends CommonResponse {    private int result;//结果码 （0 --- 成功，其他是失败码）  ...}
```

5.9、获取耳机信息命令

命令原型	GetADVInfoCmd
作用	获取耳机信息
方向	APP --> 固件
是否需要回复	需要
参数	GetADVInfoParam（参考AttrAndFunCode#ADV_TYPE_XXX）
回复	ADVInfoResponse

获取耳机信息的参数GetADVInfoParam

```
public class GetADVInfoParam extends BaseParameter {    private int mask; //功能掩码    ...}
```

获取耳机信息的回复ADVInfoResponse

```
public class ADVInfoResponse extends CommonResponse implements Parcelable {
    /**
     * 厂商ID
     */
    private int pid;
    /**
     * 产品ID
     */
    private int vid;
    /**
     * 客户ID
```

```
     */
    private int uid;
    /**
     * 左设备（设备1）电量
     */
    private int leftDeviceQuantity;
    /**
     * 左设备（设备1）充电标识
     */
    private boolean isLeftCharging;
    /**
     * 右设备（设备2）电量
     */
    private int rightDeviceQuantity;
    /**
     * 右设备（设备2）充电标识
     */
    private boolean isRightCharging;
    /**
     * 充电仓（设备3）电量
     */
    private int chargingBinQuantity;
    /**
     * 充电仓（设备3）充电标识
     */
    private boolean isDeviceCharging;
    /**
     * 设备名
     */
    private String deviceName;
    /**
     * 麦通道
     */
    private int micChannel;
    /**
     * 工作模式
     */
    private int workModel;
    /**
     * 按键设置
     */
    private List<KeySettings> mKeySettingsList;
    /**
     * 灯光设置
     */
    private List<LedSettings> mLedSettingsList;
    /**
     * 入耳检测设置
     */
    private int inEarSettings;
    ...
}

/**
 *按键设置
 */
 public static class KeySettings implements Parcelable {
        private int keyNum; //按键序号
```

```
        private int action;//动作标识
        private int function;//功能标识
        ...
}

/**
 * 灯光设置
 */
 public static class LedSettings implements Parcelable {
       private int scene; //场景标识
       private int effect; //效果标识
       ...
}
```

5.10、推送耳机信息命令

命令原型	NotifyAdvInfoCmd
作用	获取耳机信息
方向	固件 --> APP
是否需要回复	不需要
参数	NotifyAdvInfoParam
回复	

推送耳机信息的参数NotifyAdvInfoParam

```
public class NotifyAdvInfoParam extends BaseParameter {
    /**
     * 厂商ID
     */
    private int pid;
    /**
     * 产品ID
     */
    private int vid;
    /**
     * 客户ID
     */
    private int uid;
    /**
     * 芯片类型
     */
    private int chipType;
    /**
     * 协议版本
     */
    private int version;
    /**
     * 弹窗标志
     * <p>
     * 说明: 1 - true 0 - false
```

```
     * </p>
     */
    private boolean showDialog;
    /**
     * 设备的经典蓝牙地址
     */
    private String edrAddr;
    /**
     * 序列号
     */
    private int seq;
    /**
     * 行为
     */
    private int action;
    /**
     * 左设备（设备1）电量
     */
    private int leftDeviceQuantity;
    /**
     * 左设备（设备1）充电标识
     */
    private boolean isLeftCharging;
    /**
     * 右设备（设备2）电量
     */
    private int rightDeviceQuantity;
    /**
     * 右设备（设备2）充电标识
     */
    private boolean isRightCharging;
    /**
     * 充电仓（设备3）电量
     */
    private int chargingBinQuantity;
    /**
     * 充电仓（设备3）充电标识
     */
    private boolean isDeviceCharging;
    ...
}
```

5.11、控制模拟信息推送命令

命令原型	**SetDeviceNotifyAdvInfoCmd**
作用	控制模拟信息推送
方向	APP --> 固件
是否需要回复	需要
参数	SetDeviceNotifyAdvInfoParam
回复	SetDeviceNotifyAdvInfoResponse

控制模拟信息推送的参数SetDeviceNotifyAdvInfoParam

```
public class SetDeviceNotifyAdvInfoParam extends BaseParameter {
    private int op;//设置广播数据是否发送  0 : 关闭  1 : 打开
    ...
}
```

控制模拟信息推送的回复*SetDeviceNotifyAdvInfoResponse*

```
public class SetDeviceNotifyAdvInfoResponse extends CommonResponse {

    private int result;//结果码
    ...
}
```

5.12、设备请求操作命令

命令原型	**RequestAdvOpCmd**
作用	设备请求操作
方向	固件 --> APP
是否需要回复	需要
参数	RequestAdvOpParam op:操作码 0 -- 主动更新配置信息 1 -- 更新配置信息，需要重启生效 2 -- 请求同步时间 3 -- 请求回连设备
回复	CommonResponse(通用回复，无有效参数)

设备请求操作的参数RequestAdvOpParam

```
public class RequestAdvOpParam extends BaseParameter {
    private int op; //请求操作码
    ...
}
```

5.13、查找设备命令

命令原型	SearchDevCmd
作用	查找设备命令
方向	固件<--> APP
是否需要回复	需要
参数	SearchDevParam
回复	SearchDevResponse

查找设备的参数SearchDevParam

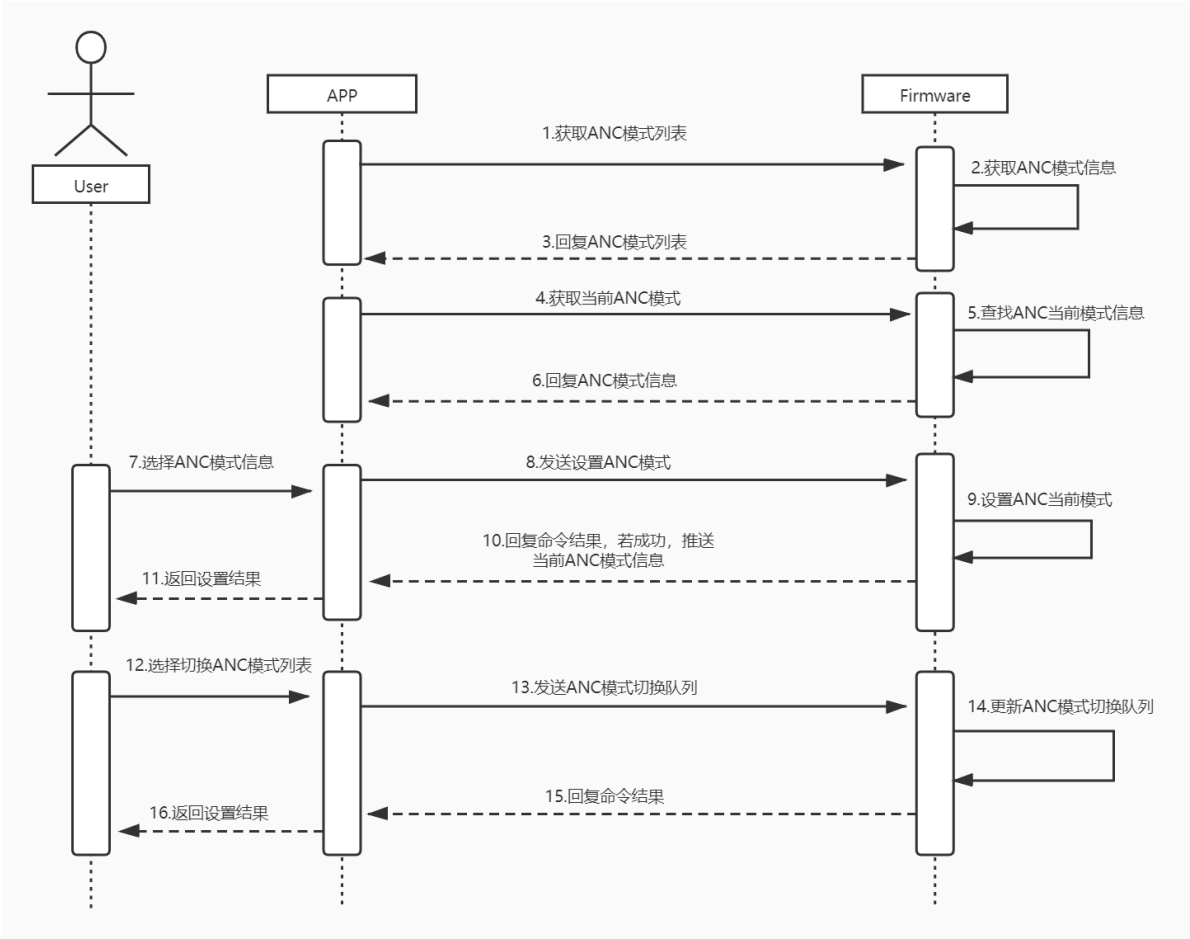
```
public class SearchDevParam extends BaseParameter {
    private int type; //查找类型 ： 0 --- 查找手机  1 --- 查找设备
    private int op; // 铃声操作 ： 0 --- 关闭铃声  1 --- 播放铃声
    private int timeoutSec; // 超时时间 ： 0 --- 不限制时间
    ...
}
```

查找设备的回复SearchDevResponse

```
public class SearchDevResponse extends CommonResponse {

    private int result;//结果码
    ...
}
```

5.14、ANC设置功能



5.15、命令生成工具类

命令工具类CommandBuilder

```
/**
 * 命令生成工具类
 */
public class CommandBuilder {

    /**
     * 创建Data命令
     *
     * @param cmd  响应命令
     * @param data 数据包
     * @return Data命令
     */
    public static DataCmd buildDataCmd(CommandBase cmd, byte[] data);

    /**
     * 创建带回复的Data命令
     *
     * @param responseCmd 响应命令
     * @param data        数据包
     * @return 带回复的Data命令
     */
    public static DataHasResponseCmd buildDataCmdWithResponse(CommandBase
responseCmd, byte[] data);

    //-------------------------------------------------------------------
```

```
    /**
     * 创建获取设备信息的命令
     *
     * @param mask 功能掩码
     * @return 命令对象
     */
    public static GetTargetInfoCmd buildGetTargetInfoCmd(int mask);

    /**
     * 创建获取设备信息的命令
     *
     * @param mask     功能掩码
     * @param platform 平台标记
     *                 <p>参考{@link com.jieli.bluetooth.constant.Constants}</p>
     * @return 命令对象
     */
    public static GetTargetInfoCmd buildGetTargetInfoCmd(int mask, byte
platform);

     //------------------------------------------------------------------------

    /**
     * 读取设备电量
     *
     * @return
     */
    public static CommandBase buildGetBatteryCmd() ;

    /**
     * 设置音量命令
     *
     * @param volume 音量值
     * @return
     */
    public static CommandBase buildSetVolumeCmd(int volume);

    /**
     * 读取设备音量命令
     *
     * @return
     */
    public static CommandBase buildGetVolumeCmd();

    /**
     * 读取设备的sd/usb等状态命令
     *
     * @return
     */
    public static CommandBase buildGetDevInfoCmd() ;

    /**
     * 错误命令
     *
     * @param err 错误码
     * @return
     */
    public static CommandBase buildErrCmd(byte err);
```

```
    /**
     * 读取music 模式的EQ信息命令
     *
     * @return
     */
    public static CommandBase buildGetEqValueCmd() ;


    /**
     * 设置music 模式的EQ 值命令
     * @param eqMode 模式：0：NORMAL  1：ROCK  2：POP  3：CLASSIC  4：JAZZ  5：
COUNTRY  6：CUSTOM
     * @param   eqMode=CUSTOM时后面10bytes才有作用，可以为null
     * @return 命令对象
     */
    public static CommandBase buildSetEqValueCmd(byte[] data);


    /**
     * 读取模式的目录浏览的文件类型命令
     *
     * @return
     */
    public static CommandBase buildGetBrowseFileTypeCmd();


    /**
     * 设置music 模式的目录浏览的文件类型命令
     *
     * @param type 后缀名字符串序列  例：“MP3\WMA\WAV\FLA\APE”(以斜杠分割)
     * @return
     */
    public static CommandBase buildSetBrowseFileTypeCmd(String type);


    //-------------------------------------------------------------------------

    /**
     * 读取music 模式的播放状态信息命令
     *
     * @return
     */
    public static CommandBase buildGetMusicStatusInfoCmd();

    /**
     * 读取music 模式的播放文件信息命令
     *
     * @return
     */
    public static CommandBase buildGetMusicFileInfoCmd();

    /**
     * 设置music 模式的播放模式命令
     *
     * @param playMode 播放模式
     * @return
     */
```

```
    public static CommandBase buildSetMusicPlayModeCmd(byte playMode);

    /**
     * 读取music 模式的播放模式命令
     *
     * @return
     */
    public static CommandBase buildGetMusicPlayModeCmd();

    //aux

    /**
     * 读取aux的播放状态
     * @return
     */
    public static CommandBase buildGetAuxPlayStatueCmd();

    //fm

    /**
     * 读取FM状态
     *
     * @return 命令对象
     */
    public static CommandBase buildGetFmStatueCmd();

    /**
     * 读取频道状态
     *
     * @return 命令对象
     */
    public static CommandBase buildGetFmChannelCmd();


    /**
     * 读取频道状态
     *
     * @return 命令对象
     */
    public static CommandBase buildGetFmSysInfoCmd(byte mask) ;

    //--------------------------------------------------------------------------

    /**
     * 同步手机当前时间到设备命令
     *
     * @return
     */
    public static CommandBase buildSyncTimeCmd();

    //--------------------------------------------------------------------------

    /**
     * 同步指定的时间到设备命令
     *
     * @return
     */
```

```
    public static CommandBase buildSyncTimeCmd(Calendar calendar);

    /**
     * 读取闹钟命令
     * @return
     */

    public static CommandBase buildGetAlarmCmd();

    /**
     * 删除闹钟
     *
     * @param attrBean
     * @return
     */
     public static CommandBase buildDelAlarmCmd(AttrBean attrBean)


    /**
     * 设置修改闹钟
     *
     * @return
     */

    public static CommandBase buildSetAlarmCmd();
    /**
     * 闹钟铃声试听
     *
     * @return 命令对象
     */

    public static CommandBase buildAuditionAlarmBellCmd(byte type, byte dev, int
cluster);
      /**
     * 闹钟铃声试听
     *
     * @return 命令对象
     */

    public static CommandBase buildStopAuditionAlarmBellCmd();
      /**
     * 获取默认闹钟铃声列表
     * @return
     */
    public static CommandBase buildGetDefaultBellList()

    //-------------------------------------------------------------------------

    /**
     * 电话拨号命令
     *
     * @param phone 电话号码
     * @return
     */
    public static CommandBase buildPhoneCallCmd(String phone);

    //-------------------------------------------------------------------------
```

```
    /**
     * 生成文件路径命令
     *
     * @param data
     * @return
     */
    public static CommandBase buildSendPathDataCmd(PathData data);

    //-------------------------------------------------------------------------

    /**
     * 读取全部共有属性
     *
     * @return
     */
    public static CommandBase buildGetPublicSysInfoCmd();

    /**
     * 读取指定的共有属性
     *
     * @param mask 属性掩码:
     *             BIT0 系统电量
     *             BIT1 系统音量
     *             BIT2 DEV INFO
     *             BIT3 ERR REPORT
     * @return
     */
    public static CommandBase buildGetPublicSysInfoCmd(int mask);

    /**
     * 设置共有的属性命令
     *
     * @return
     */
    public static CommandBase buildSetPublicSysInfoCmd(List<AttrBean> list) ;

    //-------------------------------------------------------------------------

    /**
     * 设置bt模式的属性命令
     *
     * @param list
     * @return
     */
    public static CommandBase buildSetBtSysInfoCmd(List<AttrBean> list);

    /**
     * 读取全部bt模式的属性命令
     *
     * @return
     */
    public static CommandBase buildGetBtSysInfoCmd();

    /**
     * 读取指定的bt模式的属性
     *
     * @param mask 属性掩码:
     *             BIT0 reserve
```

```
     *              BIT1 reserve
     * @return
     */
    public static CommandBase buildGetBtSysInfoCmd(int mask);


    /**
     * 设置MUSIC模式的属性命令
     *
     * @param list
     * @return
     */
    public static CommandBase buildSetMusicSysInfoCmd(List<AttrBean> list);

    /**
     * 读取全部MUSIC模式的属性命令
     *
     * @return
     */
    public static CommandBase buildGetMusicSysInfoCmd();

    /**
     * 读取指定的MUSIC模式的属性命令
     *
     * @param mask 属性掩码:
     *              BIT0 MUSIC STATUS INFO
     *              BIT1 MUSIC FILE NAME INFO
     *              BIT2 EQ INFO
     *              BIT3 目录浏览文件类型设置
     *              BIT4 PLAY MODE INFO
     * @return
     */
    public static CommandBase buildGetMusicSysInfoCmd(int mask);

    //-------------------------------------------------------------------------
    /**
     * 读取当前发射频点信息的属性命令
     *
     * @return 命令对象
     */
    public static CommandBase buildGetFrequencyTxInfoCmd();

     /**
     * 设置当前发射频点信息的属性命令
     *
     * @param payload 有效数据
     * @return 命令对象
     */
    public static CommandBase buildSetFrequencyTxInfoCmd(byte[] payload);

    /**
     * 设置当前发射频点信息的属性命令
     *
     * @param value 频点
     * @return 命令对象
     */
    public static CommandBase buildSetFrequencyTxInfoCmd(float value);
```

```
    //-------------------------------------------------------------------------


    /**
     * sys info设置指令
     *
     * @param function 功能：bt/音乐/rtc/aux/公共/fm
     * @param list     属性值：
     * @return
     */
    public static CommandBase buildSetSysInfoCmd(byte function, List<AttrBean>
list) ;
    /**
     * sys info属性读取指令
     *
     * @param function  功能：bt/音乐/rtc/aux/公共/fm
     * @param mask  属性掩码
     * @return
     */
    public static CommandBase buildGetSysInfoCmd(byte function, int mask) ;


    //==================播放控制===========

    /**
     * ID3音乐暂停/播放
     *
     * @return 命令对象
     */

    public static CommandBase buildID3PlayOrPauseCmd();

    /**
     * ID3音乐下一曲
     *
     * @return 命令对象
     */

    public static CommandBase buildID3PlayNextCmd();

    /**
     * ID3音乐上一曲
     *
     * @return 命令对象
     */

    public static CommandBase buildID3PlayPrevCmd();

    /**
     * ID3数据推送开关
     *
     * @return 命令对象
     */

    public static CommandBase buildID3DataPushSwitch(byte isOpen);

    //-------------------------------------------------------------------------
```

```
    /**
     * 暂停/播放
     *
     * @return
     */

    public static CommandBase buildPlayOrPauseCmd();

    /**
     * 上一曲
     *
     * @return
     */

    public static CommandBase buildPlayNextCmd();


    /**
     * 下一曲
     *
     * @return
     */

    public static CommandBase buildPlayPrevCmd() ;


    /**
     * 下一个播放模式
     *
     * @return
     */

    public static CommandBase buildNextPlaymodeCmd();


    /**
     * 下一个EQ模式
     *
     * @return
     */

    public static CommandBase buildNextEqModeCmd();

    /**
     * 快进
     *
     * @return
     */

    public static CommandBase buildFastForwardCmd(short time) ;


    /**
     * 快退
     *
     * @return
     */
```

```
    public static CommandBase buildRetreatQuicklyCmd( short time) ;



    //aux


    /**
     * aux播放/暂停
     *
     * @return
     */

    public static CommandBase buildAuxPlayOrPauseCmd();


     //===========fm==========

    /**
     * FM暂停播放
     * @return
     */
    public static CommandBase buildFmPlayOrPauseCmd( ) ;


    /**
     * 上一个频点
     * @return
     */
    public static CommandBase buildFmPrevFreqCmd( ) ;

    /**
     * 下一个频点
     * @return
     */
    public static CommandBase buildFmNextFreqCmd( ) ;


    /**
     * 下一个频道
     * @return
     */
    public static CommandBase buildFmNextChannelCmd();


      /**
     * 上一个频道
     * @return
     */
    public static CommandBase buildFmPrevChannelCmd( );


    /**
     * 选择频道
     * @return
     */
    public static CommandBase buildFmSelChannelCmd( byte channel);
```

```
    /**
     * 删除频道
     * @return
     */
    public static CommandBase buildFmSelectFreqCmd(float freq);


    /**
     * 删除频道
     * @return
     */
    public static CommandBase buildFmDelChannelCmd( byte channel);


    /**
     * 频道扫描：0x00：全段扫描    0x01：向前扫描      0x02：向后扫描    0x03：停止扫描
     * @return
     */
    public static CommandBase buildFmScanCmd( byte fun);

    /**
     *  fm功能指令
     * @param cmd
     * @param extend
     * @return
     */
    public static CommandBase buildFmFunctionCmd(byte cmd,byte [] extend);


    /**
     * 切换到蓝牙模式
     */

    public static CommandBase buildSwitchBtModeCmd();

    /**
     * 切换到音乐模式
     */

    public static CommandBase buildSwitchMusicModeCmd();


    /**
     * 切换到RTC模式
     */

    public static CommandBase buildSwitchRtcModeCmd();

    /**
     * 切换到Aux模式
     */

    public static CommandBase buildSwitchAuxModeCmd();


  /**
     * 切换到FM模式
```

```
     *
     * @return 命令对象
     */

    public static CommandBase buildSwitchFMModeCmd();



    /**
     * 切换功能模式
     */

    public static CommandBase buildSwitchModeCmd(byte fun);

    /**
     * 功能操作指令
     * @param function 功能：bt/音乐/rtc/aux/公共/fm
     * @param cmd ： 命令 shu
     * @param extend：附带参数
     * @return
     */

    public static CommandBase buildFunctionCmd(byte function, byte cmd, byte[]
extend);


    //AI 模式恢复


    /**
     * 维持状态不变
     *
     * @return
     */
    public static CommandBase buildRestoreCmd();

    /**
     * 跳回原来模式恢复状态
     *
     * @return
     */
    public static CommandBase buildRestoreModeCmd();

    /**
     * 跳回原来模式上一曲
     *
     * @return
     */
    public static CommandBase buildRestorePlayPrevCmd();

    /**
     * 跳回原来模式下一曲
     *
     * @return
     */
    public static CommandBase buildRestorePlayNextCmd();

    /**
```

```
     * 跳回原来模式暂停
     *
     * @return
     */
    public static CommandBase buildRestorePauseCmd();

    /**
     * 跳回原来模式播放
     *
     * @return
     */
    public static CommandBase buildRestorePlayCmd();


    /**
     * ai模式恢复命令
     *
     * @param status 恢复的状态：
     *     0x00; // 维持状态不变
     *     0x01; //跳回原来模式恢复状态
     *     0x02; //跳回原来模式上一曲
     *     0x03; //跳回原来模式下一曲
     *     0x04; //跳回原来模式暂停
     *     0x05;
     * @return
     */
    public static CommandBase buildRestoreCmd(byte status);



    /**
     * 通知固件app开始播放tts语音
     * @return
     */
    public static CommandBase buildPushStartTtsCmd()
    /**
     * 生成自定义命令
     * @param data 命令附带的参数
     * @return
     */
    public static CommandBase buildCustomCmd(byte [] data);

  //tws 命令

  /**
   * 获取耳机信息 (全部功能)
   *
   * @return 命令对象
   */
   public static CommandBase buildGetADVInfoCmdWithAll();

   /**
     * 获取耳机信息
     *
     * @param mask 功能掩码
     * @return 命令对象
     */
    public static CommandBase buildGetADVInfoCmd(int mask);
```

```
    /**
     * 设置耳机功能
     *
     * @param payload 设置数据
     * @return 命令对象
     */
    public static CommandBase buildSetADVInfoCmd(byte[] payload);

    /**
     * 停止设备发送耳机信息广播包命令
     *
     * @return 命令对象
     */
    public static CommandBase buildStopDeviceNotifyADVInfoCmd();

    /**
     * 控制耳机广播命令
     *
     * @param op 控制操作 {@link Constants#ADV_OP_CLOSE_NOTIFY} or {@link
Constants#ADV_OP_OPEN_NOTIFY}
     * @return 命令对象
     */
    public static CommandBase buildSetDeviceNotifyADVInfoCmd(int op);

    //EQ
    /**
     * 设置固件高低音
     * high:高音
     * bass 低音
     *
     * @return 命令对象
     */
    public static CommandBase buildSetHighAndBassCmd(int high, int bass);


//直播声卡
    /**
     * 读取声卡功能的eq信息：增益和频率
     *
     * @return
     */
    public static CommandBase buildGetSoundCardEqInfo() ;

    /**
     * 设置声卡功能的eq增益
     *
     * @param eqValue 增益
     * @return
     */
    public static CommandBase buildSetSoundCardEqValue(byte[] eqValue)


    /**
     * 读取声卡功能的功能状态信息
     *
     * @return
     */
```

```
    public static CommandBase buildGetSoundCardStatusInfo()

         /**
     * @param index 功能index标识
     * @param value 功能的值，如果是按钮类型值则填0
     * @return
     */
    public static CommandBase buildSetSoundCardInfo(byte index, int value)

   //查找设备
   /**
     * 创建查找设备命令
     *
     * @param op          查找方式
     * @param timeoutSec 超时时间
     * @return 命令对象
     */
    public static CommandBase buildSearchDevCmd(int op, int timeoutSec);

   /**
     * 创建查找设备命令
     *
     * @param op          查找方式
     * @param timeoutSec 超时时间
     * @param way         播放方式
     * @param player      播放源
     * @return 命令对象
     */
    public static CommandBase buildSearchDevCmd(int op, int timeoutSec, int way,
int player);


    /**
     * 获取固定长度功能的状态信息
     *
     * @return 命令对象
     */
    public static CommandBase buildGetFixedLenDataCmd();
    /**
     * 设置固定长度功能的状态信息
     *
     * @param mask 是否操作结束
     * @param dataArray
     * Byte0-3:掩码位(mask)
     *Bit0(mask)：是否支持混响(len = 5)
     *byte0:开关状态(1-打开/0-关闭)
     *byte1-2:深度数值 ， 取值范围:[0,100]
     *byte3-4:强度数值 ， 取值范围:[0,100]
     *Bit1(mask)：是否支持动态限幅器(len=2)
     *byte0-1:限幅数值 ,取值范围:[-60,0]
     * Byte4-N:拼接数据
     * 注释:
     * 拼接数据，根据mask取出对应的固定长度数据
     * Byte0-N:代表的是固定数据长度的功能的全部数据
     * byte0-N:代表的是该bit位代表的功能的对应数据
     * @return 命令对象
     */
```

```
    public static CommandBase buildSetFixedLenDataCmd(int mask, byte[]
dataArray);
    /**
     * 获取灯光功能的功能状态信息
     *
     * @param isOpEnd 是否操作结束
     * @return 命令对象
     */
    public static CommandBase buildGetLightControlInfoCmd();
    /**
     * 设置灯光功能的功能状态
     *
     * @param dataArray 灯光状态信息（
     * Byte0:
     * **bit0-1**:open(开关状态) 0:关闭，1:打开，2:设置模式(app->固件)，3:保留
     * **bit2-4**:mode(模式) 0:彩色（数据:Byte1-3,Byte7-10），1:闪烁（数据:Byte4-5），
2:情景模式(数据:Byte6)，其他:保留
     * **bit5-7**:保留
     * Byte1：Red ，
     * Byte2：Green，
     * Byte3：Blue，
     * Byte4：闪烁模式index，0：七彩闪烁1：红色闪烁2：橙色闪烁3：黄色闪烁4：绿色闪烁5：青色
闪烁6：蓝色闪烁7：紫色闪烁
     * Byte5：闪烁频率，0：快闪1：慢闪2：缓闪3：音乐闪烁
     * Byte6：情景模式index，0：彩虹1：心跳2：烛火3：夜灯4：舞台5：漫彩呼吸6：漫红呼吸7：漫
绿呼吸8：漫蓝呼吸9：绿色心情10：夕阳美景11：音乐律动
     * Byte7-8：Hue(色调，范围0-360)(HSL)，
     * Byte9：Saturation（饱和度，0-100）(HSL)，
     * Byte10：Luminance（亮度，0-100）(HSL）
     * @return 命令对象
     */
    public static CommandBase buildSetLightControlInfoCmd(byte[] dataArray);


    /**
     * 获取当前噪声处理模式信息命令
     *
     * @return 命令对象
     */
    public static CommandBase buildGetCurrentVoiceMode()

    /**
     * 设置当前噪声处理模式信息命令
     *
     * @param voiceMode 噪声处理模式
     * @return 命令对象
     */
    public static CommandBase buildSetCurrentVoiceMode(VoiceMode voiceMode)

    /**
     * 获取所有噪声处理模式信息命令
     *
     * @return 命令对象
     */
    public static CommandBase buildGetAllVoiceModes()

    /**
     * 获取闹钟铃声的设置参数
```

```
     * @param mask
     * @return
     */
    public static CommandBase buildReadBellArgsCmd(byte mask)

    /**
     * 设置闹钟铃声的参数
     * @param bellArg
     * @return
     */
    public static CommandBase buildSetBellArgsCmd(AlarmExpandCmd.BellArg
bellArg)


     /**
     * 获取当前噪声处理模式信息命令
     *
     * @return 命令对象
     */
    public static CommandBase buildGetCurrentVoiceMode();

    /**
     * 设置当前噪声处理模式信息命令
     *
     * @param voiceMode 噪声处理模式
     * @return 命令对象
     */
    public static CommandBase buildSetCurrentVoiceMode(VoiceMode voiceMode);

    /**
     * 获取所有噪声处理模式信息命令
     *
     * @return 命令对象
     */
    public static CommandBase buildGetAllVoiceModes();
```

6、目录浏览

鉴于目录浏览功能较为复杂。独立出来说明；

6.1、使用方式：

1. 使用杰理蓝牙库通信
2. 不使用杰理蓝牙库通信，但使用sdk的缓存
3. 不使用杰理蓝牙库通信，只使用命令的数据结构

6.2、使用方式1

使用杰理蓝牙库通信

```
    //第一步：注册观察者，回调详情参考FileObserver
    FileBrowseManager.getInstance().addFileObserver(fileObserver);
```

```
  // 第2步：获取在线设备列表，可以通过fileObserver处理设备状态变化
   FileBrowseManager.getInstance()。getOnlineDev();

  // 第3步：读取当前设备正在读的当前目录
   Folder currentFolder =
FileBrowseManager.getInstance().getCurrentReadFile(sdCardBean);

   //第4步：获取当前目录下已经读了但在缓存中的子文件
   List<FileStruct> fileStructs= currentFolder.getChildFileStructs()

   //第5步：具体操作
       a、FileBrowseManager.getInstance().loadMore(sdCardBean);//加载更多
       b、
FileBrowseManager.getInstance().backBrowse(sdCardBean);//FileBrowseManager.getIn
stance().backBrowse(sdCardBean，hasEvent);//返回上一级目录没有列表回调
       c、FileBrowseManager.getInstance().appenBrowse(fileStruct, sdCardBean);//
进入下一级目录
       d、FileBrowseManager.getInstance().playFile(fileStruct, sdCardBean);//点播
文件

   //第6步 参考demo的使用方式
  //如果您想自己发送数据可以实现自己的发送接口FileBrowseOperator ，然后设置发送数据实现
   /**
     * 设置数据发送的实现
     * @param fileBrowseOperator
     */

    public void setFileBrowseOperator(FileBrowseOperator fileBrowseOperator);
```

6.3、使用方式2

不使用杰理蓝牙库通信，但使用sdk的缓存

```
   1、使用过程同方式1，但需要设置FileBrowseOperator的实现
   2、增加对FileBrowseManager状态管理，FileBrowseManager是OnFileBrowseCallback的子
类。
       当您发送命令和收到数据或者状态变化时需要回调相应的方法，详情查看OnFileBrowseCallback介
绍
```

6.4、使用方式3

不使用杰理蓝牙库通信，只使用命令的数据结构

```
  //通过路径生成命令
   /**
     * 将文件路径数据转化为data数据，用于用户使用自己的传输协议的情况，如果使用杰理sdk ，
     * 可以使用CommandBuilder类buildStartFileBrowse方法生成命令
     * @param pathData 路径数据
     * @return
     */
    byte[] cmd= FileBrowseUtil.coverPathDataToCmd(PathData pathData);
```

```
//解析数据

   /**
 * 解析文件数据，数据包含packet info
 * @param data 文件数据，数据包含packet info
 * @return
 */

 List<FileStruct> list= FileBrowseUtil.parseDataHasPacket(byte[] data);
```

6.5、目录浏览观察者

```
public interface FileObserver {

    /**
     * 收到目录文件数据后回调
     * @param fileStructs
     */
    void onFileReceiver(List<FileStruct> fileStructs);

    /**
     * 一次文件读取结束
     * @param isEnd
     */

    void onFileReadStop(boolean isEnd);

    /**
     * 文件读取开始
     */

    void onFileReadStart();

    /**
     * 文件读取失败
     * @param reason
     */
    void onFileReadFailed(int reason);

    /**
     * 设备的存储设备状态变化
     * @param onLineCards
     */

    void onSdCardStatusChange(List<SDCardBean> onLineCards);

    /**
     *  文件点播成功回调
     */

    void OnFlayCallback(boolean success);
```

```
}
```

6.6、FileBrowseOperator接口

```
/**
 * 目录浏览功能数据发送接口：
 *  * 情况1：使用杰理蓝牙sdk的客户，无需关心这个接口的实现，sdk已内置实现
 *  * 情况2：客户使用自己的蓝牙库，开发人员实现这个接口向设备发送命令数据。
 *          通过 FileBrowseManager的setFileBrowseOperator方法设置FileBrowseManager
的数据发送实现类
 */

public interface FileBrowseOperator {

    /**
     * 发送目录浏览开始命令
     * @param pathData 文件路径数据，适用使用我司Android sdk的开发客户，可以把PathData通
过CommandBuilder的工具方法生成命令（情况1）。
     *                 当pathData 为目录路径则为读取文件，为文件时则时点播歌曲
     *
     * @param data 包装后的可直接发送的路径数据，适用于不使用我司Android sdk的开发客户（情
况2）。
     * @param callback 操作回调
     */
    void sendPathDataCmd(PathData pathData, byte[] data, OperatCallback
callback);


    /**
     *
     * @return 收到的数据是否带有协议头，这里不是使用我司的sdk 则需要返回tuer（情况2）
     */

    boolean  dataHasPacket();

}
```

6.7、OnFileBrowseCallback

```
/**
 * 目录浏览状态变化回调类，FileBrowseManager实现了该接口
 * 情况1：使用杰理蓝牙sdk的客户，无需关心这个类的操作，在目录浏览的时候会自动回调这类中的方法
 * 情况2：客户使用自己的蓝牙库，开发人员需要根据目录浏览的状态变化调用对应的方法
 */

public interface OnFileBrowseCallback {

    /**
     * 收到目录浏览数据
```

```
     *
     * @param data
     */

    void onFileDataReceive(byte[] data);

    /**
     * 目录浏览结束
     *
     * @param isEnd
     */

    void onFileReadStop(boolean isEnd);

    /**
     * 目录浏览开始
     */
    void onFileReadStart();


    /**
     * 目录浏览过程发生错误
     *
     * @param reason
     */

    void onFileReadFailed(int reason);

    /**
     *  点播成功回调
     */

    void OnFlayCallback(boolean success);


    /**
     * 卡状态变化
     *
     * @param list
     */
    void onSdCardChange(List<SDCardBean> list);


    /**
     * 连接状态变化
     *
     * @param device 蓝牙设备
     * @param status 连接状态
     */
    void onBluetoothConnectionChange(BluetoothDevice device, int status);


}
```

6.8、SDCardBean属性

```
    public static final int SD = 0;
    public static final int USB = 1;
    public static final int FLASH = 2;
    public static final int LINEIN = 3;
    private int index;//索引 SD/USB/FLASH/LINEIN
    private int type;//类型
    private String name; //名称
    private int devHandler;//handler
    private boolean online;//是否在线
```

6.9 、PathData属性

```
    public static final byte PATH_TYPE_FlODER=0; //目录类型
    public static final byte PATH_TYPE_FILE=1;   //文件类型

    private byte type=PATH_TYPE_FlODER;//类型 PATH_TYPE_FlODER/PATH_TYPE_FILE
    private byte readNum=10; // 读文件个数，ype=PATH_TYPE_FlODER有效
    private short startIndex=1;// 读开始位置，ype=PATH_TYPE_FlODER有效
    private int devHandler=0;//句柄
    private List<Integer> path; //路径簇号
    private int repeatTimes=3;//sdk默认实现忙状态重发次数
```

6.10、工具类FileBrowseUtil

```
    /**
     * 解析文件数据，数据包含packet info
     * @param data 文件数据，数据包含packet info
     * @return
     */
    public static List<FileStruct> parseDataHasPacket(byte[] data)

      /**
     * 将文件路径数据转化为data数据，用于用户使用自己的传输协议的情况，如果使用杰理sdk ，
     * 可以使用CommandBuilder类buildStartFileBrowse方法生成命令
     * @param pathData 路径数据
     * @return
     */
    public static byte[] coverPathDataToCmd(PathData pathData)

     /**
     * 获取设备名称
     * @param devIndex
     * @return
     */

    public static String getDevName(int devIndex)


       /**
```

```
     * 将文件路径数据转化为data数据，用于用户使用自己的传输协议的情况，如果使用杰理sdk ，
     * 可以使用CommandBuilder类buildStartFileBrowse方法生成命令
     * @param pathData 路径数据
     * @return
     */
    public static byte[] coverPathDataToCmd(PathData pathData)
```

6.11、歌词读取

```
//1、通过FileBrowseManager注册LrcReadObserver观察者
    FileBrowseManager.getInstance().addLrcReadObserver(lrcobserver)
// 2、当设备音乐的播放歌曲变化时调用
  FileBrowseManager.getInstance().startLrcRead(fileStruct)
  //读取歌词。读取的状态在LrcReadObserver回调
```

6.12、LrcReadObserver类

```
  public interface LrcReadObserver {

    /**
     * 文件读取结束
     * @param path 歌词文件保存路径
     */

    void onLrcReadStop(String path);
    /**
     * 文件读取开始
     */
    void onLrcReadStart();
    /**
     * 文件读取失败
     * @param reason
     */
    void onLrcReadFailed(int reason);
}
```

7、注意事项

1. SDK不能重复初始化，建议是单例模式使用。
2. SDK使用完后需要及时释放资源。