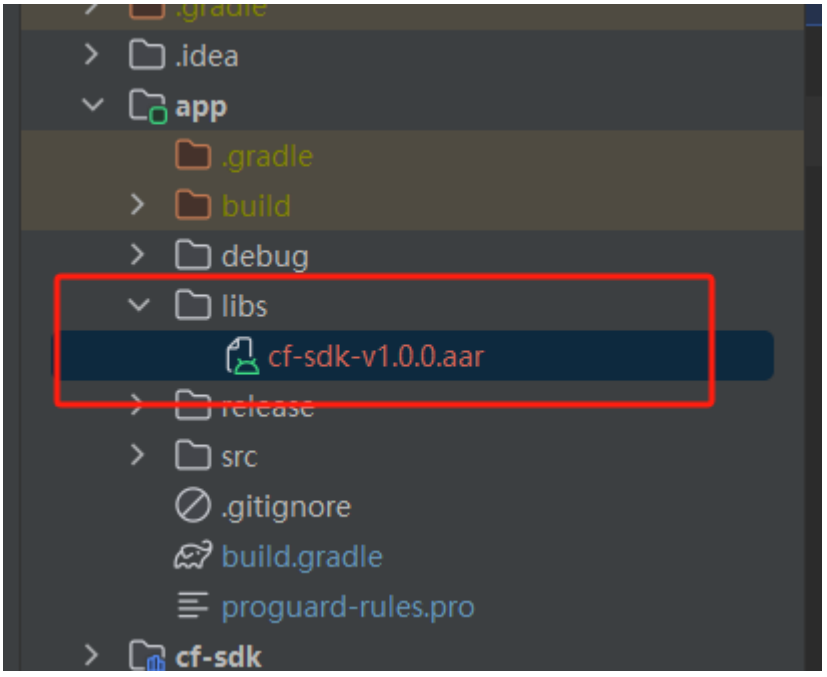
cf-sdk使用手册

概述

cf-sdk是用于Android系统下集成BLE蓝牙、串口、USB、TCP/IP通讯方式的UHF Reader的开发工具包，开发者通过接入cf-sdk可快速集成并使用相关硬件的功能。支持Android 5 . 0 以上系统

使用方法

1、引入sdk



如上图所示新建libs文件夹，并将cf-sdk放入其中，然后在工程中的build.gradle文件添加aar包依赖，如下图所示。

```
70
71  dependencies {
72
73      implementation fileTree(dir: 'libs', include: ['cf-sdk-v1.0.0.aar'])
74
75
76
```

完成上述步骤，刷新工程即可。

2、使用sdk

使用sdk前需对sdk进行初始化，调用***CfSdk.load()***或***CfSdk. load(ExecutorService pExecutorService)***（注意！传入线程池，为了SDK功能的正常运作线程池线程个数应大于等于2个）接口既可完成sdk的初始化，完成初始化动作后，通过调用***CfSdk. get(Class pClass)***接口获取相关的通讯对象，其中入参有***SdkC.BLE***、***SdkC. IP_CONN***、***SdkC. UART***、***SdkC. USB***；分别对应BLE蓝牙、TCP/IP、串口和USB的通讯方式，开发者根据不同的硬件设备，按需获取通讯对象即可。

SDK接口介绍

1、BleCore：BLE蓝牙设备操作类

```
/**
 * 初始化
 *
 * @param pContext 上下文
 */
public void init(Context pContext);

/**
 * 是否支持蓝牙模块
 *
 * @return true：表示支持，反之
 */
public boolean isSupportBt();

/**
 * 蓝牙是否打开
 *
 * @return true：表示已经打开，反之
 */
public boolean isEnabled();

/**
 * 是否在连接
 *
 * @return true表示在连接，false断开连接
 */
public boolean isConnect();

/**
 * 开始扫描ble设备
 *
 * @param pIBtScanCallback 扫描结果回调接口，扫描到的ble设备，通过该接口返回
 */
public void startScan(IBtScanCallback pIBtScanCallback);

/**
 * 停止扫描ble设备
 */
public void stopScan();

/**
 * 连接设备
 *
 * @param pDevice       要连接的ble设备
 * @param pContext      上下文对象
 * @param pAutoConnect 是否自动连接，true表示断开后自动连接，false表示不自动连接
 * @return 返回 {@link BluetoothGatt}对象
 */
```

```
public BluetoothGatt connectDevice(BluetoothDevice pDevice, Context pContext, boolean pAutoConnect);

/**
 * 获取当前连接的ble device
 *
 * @return
 */
public BluetoothDevice getConnectedDevice();

/**
 * 打开或关闭通知
 *
 * @param pServiceUuid 服务uuid
 * @param pNotifyUuid  通知uuid
 * @param pOpen        true：打开通知，false：关闭通知
 * @return 返回Boolean值，true表示操作执行成功，false表示操作执行失败
 */
public boolean setNotifyState(UUID pServiceUuid, UUID pNotifyUuid, boolean pOpen);


/**
 * 打开或关闭通知
 *
 * @param pServiceUuid 服务uuid
 * @param pNotifyUuid  通知uuid
 * @param pOpen        true：打开通知，false：关闭通知
 * @param pCallback    打开结果回调
 * @return 返回Boolean值，true表示操作执行成功，false表示操作执行失败
 */
public boolean setNotifyState(UUID pUuid, UUID pNotifyUuid, boolean pOpen, IOnNotifyCallback
pCallback);

/**
 * 设置通知回调，ble设备上报的数据通过IOnNotifyCallback回调，
 * 置空则为注销回调
 *
 * @param pCallback {@link IOnNotifyCallback}
 */
public void setOnNotifyCallback(IOnNotifyCallback pCallback);

/**
 * ble设备断开连接回调，置空则为注销回调
 *
 * @param pIBleDisConnectCallback {@link IBleDisConnectCallback}
 */
public void setIBleDisConnectCallback(IBleDisConnectCallback pIBleDisConnectCallback);

/**
 * 连接完成回调，置空则为注销回调
 *
 * @param pIConnectDoneCallback {@link IConnectDoneCallback}
 */
public void setIConnectDoneCallback(IConnectDoneCallback pIConnectDoneCallback);
```

```
/**
 * 写数据
 *
 * @param pServiceUuid         服务uuid
 * @param pCharacteristicUuid 写特征uuid
 * @param pCmd                 数据数组
 * @return true：执行成功，反之
 */
public boolean writeData(UUID pServiceUuid, UUID pCharacteristicUuid, byte[] pCmd);

/**
 * 读取数据
 *
 * @param pServiceUuid         服务uuid
 * @param pCharacteristicUuid 要读取数据的特征udid
 * @return true：表示操作执行成功，反之
 */
public boolean readData(UUID pServiceUuid, UUID pCharacteristicUuid);

/**
 * 断开已连接的设备
 */
public synchronized void disconnectedDevice();
```

BleCore回调接口说明

```
/**
 * 扫描ble设备回调接口
 */
public interface IBtScanCallback {
    /**
     * 扫描结果
     *
     * @param pResult 扫描结果
     */
    void onBtScanResult(ScanResult pResult);

    /**
     * 扫描失败，默认实现有需要的开发者自行实现
     *
     * @param pErrorCode 错误码
     */
    default void onBtScanFail(int pErrorCode) {
    }
}

/**
 * ble设备断开连接回调，当BLE蓝牙和设备断开连接后，会触发该回调
 */
public interface IBleDisConnectCallback {
    void onBleDisconnect();
}
```

```
/**
 * 设备连接成功回调
 */
public interface IConnectDoneCallback {
    /**
     * @param pB true：为连接成功，反之连接失败
     */
    void onConnectDone(boolean pB);
}


/**
 * ble设备应答回调接口
 */
public interface IOnNotifyCallback {

    /**
     * 指令解析后返回接口，改指令解析只适用于
     * 我司设备的返回指令的解析处理，其他厂商
     * 设备返回的数据，需开发者自行解析处理
     *
     * @param pCmdType 指令类型
     * @param pBleData 数据集
     */
    void onNotify(int pCmdType, BleData pBleData);

    /**
     * 默认实现，有需要的开发者可自行实现并处理BLE蓝牙返回的原数据
     *
     * @param pBytes ble应答原始数据
     */
    default void onNotify(byte[] pBytes) {
    }
}
```

2、IpCore：TCP/IP设备操作类

```
/**
 * 连接
 *
 * @param pIpAddress ip地址
 * @param pPort      端口号
 * @return true：表示连接成功，反之连接失败
 */
public boolean connect(String pIpAddress, int pPort);

/**
 * 写数据
 *
 * @param pBytes 数据byte数组
 * @return true：写出成功，false写出失败
```

```
 */
@WorkerThread
public boolean writeData(byte[] pBytes);

/**
 * 子线程接收数据，
 *
 * @param pCallback 数据回调接口{@link ISocketDataCallback}
 */
public synchronized void receiveData;

/**
 * 断开连接
 */
public void disConnect();
```

IpCore回调接口说明

```
/**
 * socket数据回调
 */
public interface ISocketDataCallback {
    /**
     * @param pBytes socket返回数据
     */
    void onDataCallback(byte[] pBytes);
}
```

发现设备类UdpFounder，通过发送udp广播进行局域网内设备发现，通过调用UdpFounder.get()函数获取该实例

```
/**
 * 发送单次广播
 *
 * @param pData 广播数据
 * @param pPort 目标端口号
 */
public void sendBroadcast(String pData, int pPort);

/**
 * 发送广播
 *
 * @param pData       广播数据
 * @param pTargetPort 目标端口号
 * @param pTimeOut    发送超时
 */
public void sendBroadcast(byte[] pData, int pTargetPort, int pTimeOut);

/**
 * 停止发送关播
 */
public void stopSendBroadcast();
```

```
/**
 * 接收目标端口号数据
 *
 * @param pPort 目标端口号
 */
public synchronized void receiveData(int pPort);

/**
 * 设置udp广播数据回调，置空为注销回调
 *
 * @param pIUdpDataCallback udp数据返回接口
 */
public void setIUdpDataCallback(IUdpDataCallback pIUdpDataCallback);

/**
 * 释放
 */
public void release();
```

UdpFounder接口回调说明

```
/**
 * udp广播数据回调
 */
public interface IUdpDataCallback {
    /**
     * @param pPacket udp广播应答包
     */
    void onReceiveUdpData(DatagramPacket pPacket);
}
```

3、UartCore：串口设备操作类

```
/**
 * 初始化
 *
 * @param pPath     串口地址
 * @param pBaudRate 波特率
 * @return true：初始化成功，false初始化失败
 */
public boolean init(String pPath, int pBaudRate);

/**
 * 发送数据
 *
 * @param pBytes 要发送数据的byte数组
 * @return true：表示发送成功，false表示发送失败
 */
public boolean sendData(byte[] pBytes);

/**
```

```
 * 接收串口数据
 *
 * @param pSerialDataCallback 串口数据回调接口
 */
public synchronized void receiverData(ISerialDataCallback pSerialDataCallback);

/**
 * 停止接收数据
 */
public void stopReceiverData();

/**
 * 释放串口
 */
public void release();
```

UartCore串口数据回调说明

```
/**
 * 串口数据回调
 */
public interface ISerialDataCallback {
    /**
     * @param pBytes 串口数据
     */
    void onDataCallback(byte[] pBytes);
}
```

4、UsbCore：USB设备操作类

```
/**
 * 初始化
 *
 * @param pContext 上下文对象
 */
public void init(Context pContext);

/**
 * 获取所有usb设备的pid和vid
 *
 * @return 所有设备的pid和vid设备列表
 */
public ArrayList<Pair<Integer, Integer>> getAllDevicePidAndVid();

/**
 * 根据提供的pid和vid寻找目标设备
 *
 * @param pPid
 * @param pVid
 * @return 返回命中的usb设备，否则返回null
 */
public UsbDevice findTargetDevice(int pPid, int pVid);
```

```
/**
 * 连接设备
 *
 * @param pContext          上下文对象
 * @param pUsbDevice        连接的目标设备
 * @param pIUsbConnectDone usb设备连接完成回调
 */
public void connectDevice(Context pContext, UsbDevice pUsbDevice, IUsbConnectDone pIUsbConnectDone);

/**
 * 写数据
 *
 * @param pData     要发送数据byte数组
 * @param pTimeOUt 超时单位ms
 * @return true：写出成功，反之写出失败
 */
public boolean writeData(byte[] pData, int pTimeOUt);

/**
 * 同步获取usb数据
 *
 * @param pTimeOut 获取数据超时
 * @return 返回usb读取的数据
 */
public byte[] readDataSync(int pTimeOut);

/**
 * 异步读取usb数据
 *
 * @param pTimeOut 单次读取数据超时
 */
public synchronized void readDataAsync(int pTimeOut);

/**
 * 设置usb设备数据回调，置空则为注销回调
 *
 * @param pIReadDataCallback
 */
public void setIReadDataCallback(IReadDataCallback pIReadDataCallback);

/**
 * 释放usb设备
 */
public void release();
```

UsbCore回调接口说明

```
/**
 * usb读取数据回调
 */
public interface IReadDataCallback {
    /**
```

```
     * @param pBytes usb设备返回的数据
     */
    void onDataBack(byte[] pBytes);
}

/**
 * usb连接完成回调
 */
public interface IUsbConnectDone {
    /**
     * @param pB true：表示连接成功，反之连接失败
     */
    void onUsbConnectDone(boolean pB);
}
```

指令构建和指令类型

1、指令构建：通过CmdBuilder构建相应指令

```
/**
* 获取初始化模块指令
*/
public static byte[] buildModuleInitCmd();

/**
* 获取恢复出厂设置指令
*/
public static byte[] buildRebootCmd();

/**
* 获取设置射频输出功率指令
*
* @param pPower 输出功率，单位为：dBm，取值范围为：[0, 26]dBm，大于26 dBm均为26dBm。
* @param pResv  系统预留字段，默认0x00；
*/
public static byte[] buildSetPwrCmd(byte pPower, byte pResv);

/**
* 设置/读取 模块支持的 RF 协议标准
*
* @param pOption 命令控制选项
*                0x01：设置，其后接1Byte长度的RFID ；
*                0x02：读取，其后不接RFID；
*                其他值：无效；
* @param pRfid   协议选项，0x00：ISO 18000-6C； 0x01：GB/T 29768；0x02：GJB 7377.1；当前仅支持 ISO 18000-6C。
* @return
*/
public static byte[] buildSetOrGetRfidCmd(byte pOption, byte pRfid);

/**
* 获取设置所有可配置参数指令
```

```
*
* @param pBean 可配置参数bean类，该类通过发送获取所有参数指令，解析可得
*/
public static byte[] buildSetAllParamCmd(AllParamBean pBean);

/**
* 获取所有可配置参数
*/
public static byte[] buildGetAllParamCmd();

/**
* 获取电池电量指令
*/
public static byte[] buildGetBatteryCapacityCmd();

/**
* 设置/获取蓝牙设备名称指令
*
* @param pOption 命令控制选项
*                0x01：设置，其后接1Byte长度的RFID ；
*                0x02：读取，其后不接RFID；
*                其他值：无效；
* @param pBtName 蓝牙名称
* @return
*/
public static byte[] buildSetOrGetBtNameCmd(byte pOption, String pBtName);

/**
 * 构建设置蓝牙输出模式指令
 *
 * @param pMode 0x00：蓝牙HID输出；0x01：蓝牙透传输出
 * @return
 */
public static byte[] buildSetOutputModeCmd(byte pMode)

/**
* 构建获取蓝牙输出模式指令
*
* @return
*/
public static byte[] buildGetOutputModeCmd()

/**
* 该命令上报按键动作开始结束状态。
*
* @param pKeyState 0x01：开始；
*                  0x02：结束；
*                  其它值：无效。
* @return
*/
public static byte[] buildReportKeyStateCmd(byte pKeyState);

/**
```

```
* 设置设备读取模式。
*
* @param pReadMode 扫描使能参数，1Byte。
*                  0x01：扫描头模式 ；
*                  0x00：RFID模式；
*                  其他值：无效；
* @param pRecev    保留，7Bytes
* @return
*/
public static byte[] buildSetReadModeCmd(byte pReadMode, byte[] pRecev);

/**
* 获取设备读取模式
*
* @return
*/
public static byte[] buildGetReadModeCmd();

/**
* @param pInvType  盘点方式：
*                  0x00：按时间盘点标签，在执行指定时间后停止盘点或接收到停止盘点命令后停止盘点；
*                  0x01: 按照循环次数盘点，在执行指定次数的轮询后或者接到停止盘点指令后停止盘点；
* @param pInvParam 盘点方式参数：
*                  1.若InvType为0x00：
*                  InvParam表示盘点时间，单位为：秒，如果该值为0，则表示持续盘点标签，直到接收到停止盘点命令；
*                  2.若InvType为0x01：
*                  InvParam表示盘点次数，单位为：次，该值必须大于0；
*/
public static byte[] buildInventoryISOContinueCmd(byte pInvType, int pInvParam);

/**
* 停止盘点
*
* @return
*/
public static byte[] buildStopInventoryCmd();

/**
* 获取设备信息
*
* @return
*/
public static byte[] buildGetDeviceInfoCmd();

/**
* 构建选择标签指令
*
* @param pMask 掩码，指标签的EPC号码
* @return
*/
public static byte[] buildSelectMaskCmd(byte[] pMask);

/**
```

```
* 读取标签数据
*
* @param pAccPwd     访问口令，用于标签进入安全态，默认0x00000000；
* @param pMemBank    所要读取标签的存储区，值列表如下：
*                    0x00：Reserved； 0x01：EPC； 0x02：TID； 0x03：User；
* @param pWordPtr    指向逻辑存储区的读取起始地址(字)；
* @param pWordCount 需要读取的字个数，不能为0，默认为4，取值范围[1,120]
* @return
*/
public static byte[] buildReadISOTagCmd(
byte[] pAccPwd,
byte pMemBank,
byte[] pWordPtr,
byte pWordCount
);

/**
* 写入标签数据
*
* @param pAccPwd     访问口令，用于标签进入安全态，默认0x00000000；
* @param pMemBank    所要读取标签的存储区，值列表如下：
*                    0x00：Reserved； 0x01：EPC； 0x02：TID； 0x03：User；
* @param pWordPtr    指向逻辑存储区的读取起始地址（字）；
* @param pWordCount 需要写入标签的数据字个数(1个字为两个字节)，必须大于0；
* @param pData       需要写入标签的数据，长度必须为字的整数倍，长度为1 ~ WordCount个字
* @return
*/
public static byte[] buildWriteISOTagCmd(
byte[] pAccPwd,
byte pMemBank,
byte[] pWordPtr,
byte pWordCount,
byte[] pData
);

/**
* 锁定标签数据
*
* @param pAccPwd 访问口令，用于标签进入安全态，默认0x00000000
* @param pArea    需要锁定的区域，值列表如下：
*                 0x00：灭活密码区；0x01：访问密码区； 0x02：EPC； 0x03：TID； 0x04：User；
* @param pAction 锁定操作类型，值列表如下：
*                 0x00：开放； 0x01：永久开放； 0x02：锁定； 0x03：永久锁定；
*/
public static byte[] buildLockISOTagCmd(byte[] pAccPwd, byte pArea, byte pAction);

/**
* 灭活标签
*
* @param pKLENIPwd 四个字节的灭活密钥
* @return
*/
public static byte[] buildKlenlISOTagCmd(byte[] pKLENIPwd);
```

```
/**
* 获取权限参数
*
* @return
*/
public static byte[] buildGetPermissionParamCmd();

/**
* 设置权限参数
*
* @param pBean 权限参数bean类
* @return
*/
public static byte[] buildSetPermissionParamCmd(PermissionParamBean pBean);
```

2、指令类型：通过CmdType类，进行指令类型引用

```
/**
 * 盘点标签
 */
public static final int TYPE_INVENTORY = 0x0001;
/**
 * 停止盘点标签
 */
public static final int TYPE_STOP_INVENTORY = 0x0002;
/**
 * 读取标签
 */
public static final int TYPE_READ_TAG = 0x0003;
/**
 * 写标签
 */
public static final int TYPE_WRITE_TAG = 0x0004;
/**
 * 锁定标签
 */
public static final int TYPE_LOCK_TAG = 0x0005;
/**
 * 灭活标签
 */
public static final int TYPE_KILL_TAG = 0x0006;
/**
 * 选定标签
 */
public static final int TYPE_SELECT_MASK = 0x0007;
/**
 * 模块初始化
 */
public static final int TYPE_MODULE_INIT = 0x0050;
/**
 * 恢复出厂设置
```

```
 */
public static final int TYPE_REBOOT = 0x0052;
/**
 * 网络参数接口参数
 */
public static final int TYPE_REMOTE_NET_PARA = 0x005F;
/**
 * 获取设备信息
 */
public static final int TYPE_GET_DEVICE_INFO = 0x0070;
/**
 * 设置所有参数
 */
public static final int TYPE_SET_ALL_PARAM = 0x0071;
/**
 * 获取所有参数
 */
public static final int TYPE_GET_ALL_PARAM = 0x0072;
/**
 * 设置或获取权限参数
 */
public static final int TYPE_GET_OR_SET_PERMISSION = 0x0076;
/**
 * 获取电池信息
 */
public static final int TYPE_GET_BATTERY_CAPACITY = 0x0083;
/**
 * 设置或获取设备名称
 */
public static final int TYPE_SET_OR_GET_BT_NAME = 0x0086;
/**
 * 输出模式
 */
public static final int TYPE_OUT_MODE = 0x0088;
/**
 * 按键状态
 */
public static final int TYPE_KEY_STATE = 0x0089;
```

指令解析返回类

1、CmdData

接口返回的数据都通过这个类返回，通过调用getDataType()函数知道返回的数据类型

```
/**
 * CmdData所包含的数据类型
 */
private GeneralBean mGeneralBean;
private TagInfoBean mTagInfoBean;
private AllParamBean mAllParamBean;
private BatteryCapacityBean mBatteryCapacityBean;
```

```
private DeviceNameBean mDeviceNameBean;
private DeviceInfoBean mDeviceInfoBean;
private OutputModeBean mOutputModeBean;
private PermissionParamBean mPermissionParamBean;
private TagOperationBean mTagOperationBean;
private KeyStateBean mKeyStateBean;
private RemoteNetParaBean mRemoteNetParaBean;


/**
 * 通过调用getDataType()获取当前赋值数据的类型
 *
 * @return 返回当前数据类型对象
 */
public Object getData()

/**
 * 获取数据类型
 *
 * @return 返回数据类型的class对象
 */
public Class<?> getDataType()
```

2、GeneralBean

指令状态通用类，用于记录应答状态和应答信息

```
/**
 * 状态码
 */
public int mStatus;
/**
 * 状态信息
 */
public String mMsg;
```

3、AllParamBean

所有参数类，通过该类获取和设置所有配置参数

```
/**
 * 设备的通信地址，默认为0x00。这个地址不能为0xFF。如果设置为0xFF，则读写模块将返回参数出错信息。
 */
public byte mAddr;

/**
 * 设备射频RFID的协议标准规范，
 * 0x00：ISO 18000-6C；
 * 0x01：GB/T 29768；0x02：
 * GJB 7377.1；
 * 当前仅支持 ISO 18000-6C。
```

```
 */
public byte mRFIDPRO;

/**
 * 设备的工作模式，默认值0
 * 0    应答模式
 * 1    主动模式
 * 2    触发模式
 */
public byte mWorkMode;

/**
 * 设备的通信接口，默认值0x80，具体释义如下：
 * 0x80 RS232
 * 0x40 RS485
 * 0x20 RJ45
 * 0x10 WiFi
 * 0x01 USB
 * 0x02 keyboard
 * 0x04 CDC_COM
 */
public byte mInterface;

/**
 * 串口波特率，默认值为4，具体释义如下：
 * 0    9600bps
 * 1    19200 bps
 * 2    38400 bps
 * 3    57600 bps
 * 4    115200 bps
 */
public byte mBaudrate;

/**
 * 韦根数据输出接口的配置参数，默认值0x00，具体释义如下：
 * WGSet      Bit7              Bit6     Bit5        Bit4    Bit3    Bit2    Bit1    Bit0
 * Bit位释义   0：关闭韦根输出    0：wg26   0：低位在前   备用    备用     备用     备用     备用
 * -----------1：开启韦根输出    1：wg34   1：高位在前
 */
public byte mWGSet;

/**
 * 设备所有的天线号，按位表示选择使用的天线，对应Bit位值为1则表示使用该天线，值为0则表示不使用该天线；
 * 从低位开始，第0位表示1号天线，第1位表示2号天线，以此类推，最多能表示8个天线；
 * 不同模块支持不同个天线，视具体情况而定；
 * 默认值0x01，表示1号天线。
 */
public byte mAnt;

/**
 * 设备的RFID频率相关参数，用于选择频段及各频段中的上限频点和下限频点，长度8Bytes
 */
public RfidFreq mRfidFreq;
```

```
/**
 * 设备的RFID输出功率，单位为：dBm，取值范围为：[0, 33]dBm，其他无效。
 */
public byte mRfidPower;

/**
 * 设备要访问标签的存储区区域。
 * 0x00：保留区；
 * 0x01(默认)：EPC存储区；
 * 0x02：TID存储区；
 * 0x03：USER存储区；
 * 0x04：EPC+TID；
 * 0x05：EPC+USER；
 * 0x06：EPC+TID+USER；
 * 其它值保留。若命令中出现了其它值，将返回参数出错的消息。
 */
public byte mInquiryArea;

/**
 * 询查EPC标签时使用的初始Q值，
 * Q值的设置应为场内的标签数量约等于2Q。
 * Q值的默认值为4，Q值的范围为0～15，
 * 若命令中出现了其它值，将返回参数出错的消息。
 */
public byte mQValue;

/**
 * 询查EPC标签时使用的Session值，
 * 默认为0，取值范围[0，3]，其它值，
 * 将返回参数出错的消息。
 * 0    Session使用S0
 * 1    Session使用S1
 * 2    Session使用S2
 * 3    Session使用S3
 */
public byte mSession;

/**
 * 设备要访问标签存储区的起始地址，单位 ：Byte，
 * 默认值：0x00；
 * 访问EPC区时，0x00表示访问EPC区除CRC和PC段的EPC号码段起始地址；
 * 访问其他存储区时，0x00表示该存储区的起始地址。
 */
public byte mAcsAddr;

/**
 * 设备要访问标签存储区的数据长度，单位 ：Byte，默认值：0x00.
 */
public byte mAcsDataLen;

/**
 * 过滤时间，在读取成功一张标签数据后的该值时间内，
```

```
 * 过滤掉有相同数据的标签。单位为：S，
 * 取值范围为：[0, 255]，其他无效；默认值为0，没有过滤。
 */
public byte mFilterTime;

/**
 * 设备收到触发信号后的询查时长，单位为：S，默认值为1，取值范围为：[0, 255]，其他无效。
 */
public byte mTriggerTime;

/**
 * 设备执行成功后蜂鸣器鸣叫时长，单位为：10ms，
 * 取值范围为：[0, 255]，其他无效；默认为1，
 * 当为0时，表示蜂鸣器不鸣叫。
 */
public byte mBuzzerTime;

/**
 * 询查间隔时间，单位为：10ms，取值范围为：[0, 255]，其他无效，默认为1。
 */
public byte mPollingInterval;

/**
 * 各国频段范围：
 * 0x00:  用户根据需求自定义；
 * 0x01：US [902.75~927.25]
 * 0x02：Korea [917.1~923.5]
 * 0x03：EU [865.1~868.1]
 * 0x04：JAPAN [952.2~953.6]
 * 0x05：MALAYSIA [919.5~922.5]
 * 0x06：EU3 [865.7~867,5]
 * 0x07：CHINA_BAND1 [840.125~844.875]
 * 0x08：CHINA_BAND2 [920.125~924.875]
 */
public byte mREGION;

/**
 * 长度两位
 * 兆赫兹起始频率的整数部分；如920.125MHz，STRATFREI = 920 = 0x0398，高字节=0x03，低字节=0x98；
 */
public byte[] mSTRATFREI;

/**
 * 长度两位
 * 兆赫兹起始频率的小数部分；如 920.125MHz，STRATFRED =125，高字节=0x00，低字节=0x7D
 */
public byte[] mSTRATFRED;

/**
 * 长度两位
 * 频率步进（KHz），需参考各频段计算公式；如125KHz，STEPFRE =125，高字节=0x00，低字节=0x7D；
 */
public byte[] mSTEPFRE;
```

```
/**
 * 信道数;
 */
public byte mCN;
```

4、BatteryCapacityBean

电池电量类，获取设备电量信息

```
/**
 * 电池电量百分比
 */
public byte mBatteryCapacity;
```

5、DeviceInfoBean

设备信息类，获取设备信息

```
/**
 * 硬件版本
 */
public String mHwVer;
/**
 * 固件版本
 */
public String mFirmVer;
/**
 * 序列号
 */
public String mSn;
/**
 * RFID模块版本
 */
public String mRFIDModeVer;
/**
 * RFID模块名称
 */
public String mRFIDModeName;
/**
 * RFID模块序列号
 */
public String mRFIDModeSn;
```

6、DeviceNameBean

设备名称类

```
/**
 * 操作类型
 * 0x01：设置
 * 0x02：读取
 */
public byte mOption;
/**
 * 设备名称
 */
public String mDeviceName;
```

7、KeyStateBean

按键状态类

```
/**
 * 按键状态
 * 0x01：开始；
 * 0x02：结束；
 */
public byte mKeyState;
```

8、OutputModeBean

输出模式类

```
/**
 * 操作类型
 * 0x01 设置
 * 0x02 获取
 */
public byte mOption;
/**
 * 输出模式
 * 0x00：蓝牙HID输出
 * 0x01：蓝牙透传输出
 */
public byte mMode = -1;
```

9、PermissionParamBean

权限参数类

```
/**
 * 命令控制选项，0x01设置，0x02 读取，其它值无效。
 */
public byte mOption;
/**
 * :密码功能使能参数，长度1字节。0x01启用，0x00 不启用。默认0x00。
 */
```

```
public byte mCodeEN;
/**
 * 标签的访问密码，长度4字节。默认[0x00,0x00,0x00,0x00]。
 */
public byte[] mCodes;
/**
 * 掩码功能使能参数，长度1字节。0x01启用，0x00 不启用。默认0x00。
 */
public byte mMaskEN;
/**
 * 掩码起始地址，长度1字节，单位字节。默认0x00。
 */
public byte mStartAdd;
/**
 * 掩码长度，长度1字节，单位字节，最大12。默认 0x00。
 */
public byte mMaskLen;
/**
 * 掩码数据，长度 31字节，掩码长度不足 31字节时，后面字节数据以0补充。
 * 默认数据都为0x00。
 */
public byte[] mMaskData;
/**
 * :掩码条件，长度1字节。
 * 0x00:密码或者掩码符合;
 * 0x01:密码和掩码同时符合。
 * 默认0x00。
 */
public byte mMaskCondition;
```

10、RemoteNetParaBean

局域网设备发现，应答数据类

```
/**
 * 操作项
 * 0x01：设置； 0x02：获取
 */
public byte mOption;
/**
 * 4个字节
 * 设备IP地址，若IP地址为192.168.1.1则数据为[0xC0,A8,0x01,0x01]
 */
public byte[] mIpAddr;
/**
 * 6个字节
 * Mac地址，如果多个设备在同一现场工作，请使用不同的MAC地址；
 * 例如：mac[6]={0x00,0x08,0xdc,0x11,0x11,0x11}，mac地址为00-08-DC-11-11-11
 */
public byte[] mMacAddr;
/**
 * 2个字节
```

```
 * 监听端口，取值为[0,65536]，默认为5000；
 */
public byte[] mPort;
/**
 * 4个字节
 * 子网掩码，默认为[0xFF,0xFF,0xFF,0x00]
 */
public byte[] mNetMask;
/**
 * 4个字节
 * 默认网关，默认为[0xC0,0xA8,0x01,0x01]
 */
public byte[] mGateWay;
```

11、TagInfoBean

标签信息类

```
/**
 * 标签ACK响应的RSSI，单位为dBm，带符号数，负数使用补码格式；
 */
public int mRSSI;
/**
 * 从哪个天线端口接收到的标签数据，值范围为：1~4，分别表示1~4号天线
 */
public int mAntenna;
/**
 * 从哪个信道接收到的标签数据，值从0开始，0表示0信道，1表示1信道，以此类推；
 */
public int mChannel;
/**
 * 标签的EPC号码长度（字节）；
 */
public int mEPCLen;
/**
 * 标签的EPC号码；
 */
public byte[] mEPCNum;
```

12、TagOperationBean

标签操作应答类

```
/**
 * 标签状态
 */
public byte mTagStatus;
/**
 * 从哪个天线端口接收到的标签数据，值范围为：1~4，分别表示1~4号天线
 */
public byte mAntenna;
/**
```

```
 * 2个字节
 * 标签响应数据中的CRC数据
 */
public byte[] mCRC;
/**
 * 2个字节
 * 标签响应数据中的PC数据
 */
public byte[] mPC;
/**
 * 标签的EPC号码长度（字节）
 */
public byte mEPCLen;
/**
 * 标签的EPC号码;
 */
public byte[] mEPCNum;

//以下两个字段，只有读标签时才有值
/**
 * 成功读取到的标签数据字个数;
 */
public byte mWordCount;
/**
 * N个字节
 * 成功读取到的标签数据，长度为WordCount×2个字节
 */
public byte[] mData;
```

附录

附录A. STATUS的定义

STATUS	错误描述
0x00	执行成功（此处只表示模块成功接收到标签响应数据，如果标签响应中有标签执行状态，则还应该进一步判断标签执行状态是否正确）
0x01	参数值错误或越界，或者模块不支持该参数值
0x02	由于模块内部错误导致的命令执行失败 （设置频率或者设置功率）
0x03	保留
0x12	整个盘点命令执行完成
0x13	没有盘点到标签
0x14	标签响应超时
0x15	解调标签响应错误
0x16	协议认证失败

STATUS	错误描述
0x17	口令错误
0xFF	没有更多数据了