# 开发指南

清单：

1.RFIDSDK 包含三部分，

（1）armeabi-v7a底层操作的so库文件

（2）RFID.jar 与模块交互的上层接口

2.serialport-module-api Serialport-module.jar 的api文档

3.RFID-api RFID.jar的 api 文档

4.UHFSDKTest简单demo示例

5.RFID开发指南.docx SDK开发指南。

## 1.RFID 开发

### 1.基于省电的考虑，RFID模块可以手动控制上电和掉电操作。RFID模块只有上电时才能工作，不工作时最好做掉电处理，这样做更省电。

示例代码：

set53CGPIOEnabled(true)/模块上电

set53CGPIOEnabled(false);//模块掉电，应用结束时建议掉电

备注：不能在没有释放上电掉电控制设备的情况下在其他应用中执行上电掉电操作

### 2.对模块上电之后下一步就是要连接读写器以实现与读写器的通信，打开连接之后台会启动一个线程启动监听读写器返回数据

示例代码：

ModuleConnector connector = new ReaderConnector();//构建连接器

connector.connectCom("dev/ttyS4",115200);//连接指定串口，返回true表示成功，false失败

### 3.获取RFIDReaderHelper对象，该对象是与RFID模块交互的核心类，可以发送指令到读写器还可以通过注册观察者RXObserver对象监听读写器返回数据

示例代码：

RFIDReaderHelper mReaderHelper = RFIDReaderHelper.getDefaultHelper();//获取RFIDReaderHelper对象

mReaderHelper.realTimeInventory((byte) 0xFF,(byte)0x01);//发送实时盘存指令，更多指令参考API文档，返回的数据在RXObserver 相应的方法中回调。

### 4.获取RFID模块的数据返回，继承RXObserver类覆盖相应的的方法，通过RFIDReaderHelper的registerObserver方法注册到RFIDReaderHelper中，后台线程在读取到RFID模块返回的相应数据的时候会回调对应的方法，作为参数传递出来。因此RXObserver中的各种回调方法运行在子线程中。你没必有覆盖所有的方法，只需覆盖你用到的方法即可。（以下提到的发送指令的函数，均为RFIDReaderHelper中的函数）

示例代码

RXObserver rxObserver = new RXObserver() {

@Override

protected void onExeCMDStatus(byte cmd, byte status) {

//如果指令没有返回额外数据仅包含命令执行的状态码（例如RFIDReaderHelper中的各种以set开头的设置指令函数，）会回调该方法

//如果指令返回数据异常一定会回调该方法 status 为异常码

//cmd可以用来区分具体是哪条命令的返回，命令参考CMD类文档，status指令执行状态码，参考ERROR类文档

}

@Override

protected void refreshSetting(ReaderSetting readerSetting) {

//当发送查询读写器设置指令（例如RFIDReaderHelper中的各种以get开头的查询指令函数）会回调该方法，若有返回值会存储在readerSetting相应字段中

//具体可以参考API文档中ReaderSetting 各个字段的含义

}

@Override

protected void onInventoryTag(RXInventoryTag tag) {

//当发送盘存指令的时候该方法将会回调，盘存指令包括RFIDReaderHelper中inventory，realTimeInventory，customizedSessionTargetInventory,fastSwitchAntInventory 等函数以及扣扳机

//inventory 函数盘存到的标签会先缓存到RFID模块的缓存中，只有调用getInventoryBuffer 或 getAndResetInventoryBuffer 函数是才会回调该方法将数据上传，上传的标签数据无重复

//当盘存到多张标签的时，该方法会多次回调，标签可以重复

}

@Override

protected void onInventoryTagEnd(RXInventoryTag.RXInventoryTagEnd tagEnd) {

//当一条盘存指令执行结束的时候该方法会回调（fastSwitchAntInventory除外fastSwitchAntInventory结束时回调onFastSwitchAntInventoryTagEnd），tagEnd为指令结束时的返回数据，具体各个字段的含义

//可以参考文档中RXInventoryTag.RXInventoryTagEnd各个字段的含义

}

@Override

protected void onFastSwitchAntInventoryTagEnd(RXInventoryTag.RXFastSwitchAntInventoryTagEnd tagEnd) {

//因为fastSwitchAntInventory 函数返回的结束数据特殊，因此其单独回调这个函数

// RXInventoryTag.RXFastSwitchAntInventoryTagEnd中各字段的含义参考API文档

}

@Override

protected void onGetInventoryBufferTagCount(int nTagCount) {

//通过函数getInventoryBufferTagCount 得到缓存中盘存标签的数量，数据是通过inventory盘存到读写器缓存区中标签数量，无重复标签的数量

}

@Override

protected void onOperationTag(RXOperationTag tag) {

//当执行readTag,writeTag,lockTag 或者 killTag 等操作标签指令函数时会回调该方法，当一次操作多张标签时会多次回调

//返回数据RXOperationTag tag 参考API文档

}

@Override

protected void onOperationTagEnd(int operationTagCount) {

//当执行readTag,writeTag,lockTag 或者 killTag 等操作标签指令函数结束时会回调该方法

//operationTagCount 为操作的标签数量

}

@Override

protected void onInventory6BTag(byte nAntID, String strUID) {

//当执行iso180006BInventory时会回调该方法，如果盘存到多张标签会回调多次

//nAntID 盘存的标签的天线号，strUID盘存到6B标签的UID

}

@Override

protected void onInventory6BTagEnd(int nTagCount) {

//当iso180006BInventory函数执行结束，所有盘存到的6B标签数据上传完毕，会回调该方法，并传回盘存的6B标签数量

//nTagCount 为盘存到6B标签的数量

}

@Override

protected void onRead6BTag(byte antID, String strData) {

//当执行iso180006BReadTag函数时该方法会回调

//

}

@Override

protected void onWrite6BTag(byte nAntID, byte nWriteLen) {

//当执行iso180006BWriteTag函数时该方法会回调

}

@Override

protected void onLock6BTag(byte nAntID, byte nStatus) {

//当执行iso180006BLockTag函数时该方法会回调

//nAntID 天线号 nStatus 标签Lock状态

}

@Override

protected void onLockQuery6BTag(byte nAntID, byte nStatus) {

//当执行iso180006BQueryLockTag函数时该方法会回调

//nAntID 天线号 nStatus 标签Lock状态

}

@Override

protected void onConfigTagMask(MessageTran msgTran) {

//当执行setTagMask，getTagMask，clearTagMask函数时改方法会回调

//返回数据msgTran 具体数据参考MessageTran API说明与Select指令格式

}

};

//注册RXObserver对象到RFIDReaderHelper，只有这样一旦RFID模块有数据返回才会回调RXObserver中的相应方法。

mReader.registerObserver(rxObserver);

### 5.释放资源

退出应用的时候一定要释放相应的资源 示例代码：

//移除指定RXObserver监听

mReader.unRegisterObserver(rxObserver);

//停止相应的线程，关闭相应I/O资源，调用该方法后RFIDReaderHelper 的各种方法不能用，否则//会报异常，同时数据也不能同时接收。必须重新调用//connector.connectCom("dev/ttyS4",115200); RFIDReaderHelper才能正常工作

Connector.disConnect();

//释放读写器上电掉电控制设备

ModuleManager.newInstance().release();

### 6.高级

#### (1).监听发送和接收数据，以及模块的链接状态。实现RXTXListener接口将其设置到RFIDReaderHelper类中

示例代码： //实现RXTXListener接口 RXTXListener mListener = new RXTXListener() {

@Override

public void reciveData(byte[] btAryReceiveData) {

// TODO Auto-generated method stub

//获取从RFID模块接收到的数据

}

@Override

public void sendData(byte[] btArySendData) {

// TODO Auto-generated method stub

//获取发送到RFID模块的数据

}

@Override

public void onLostConnect() {

// TODO Auto-generated method stub

//链接断开会回调该方法。

}

};

//将RXTXListener注册到RFIDReaderHelper以便监听相关的数据

mReader.setRXTXListener(mListener);

#### （2).自定义相关的实现，如果你想自定义类去实现与RFID模块的交互，可以继承或实现com.module.interaction包中的类或接口定义自己的实现，具体可以参考文档以及我们的实现。