目录

[本文目的： 1](#_Toc2609650)

[功能： 1](#_Toc2609651)

[硬件框图如下： 1](#_Toc2609652)

[测试连接图如下 2](#_Toc2609653)

[对应的两路tty设备如下： 2](#_Toc2609654)

[485 对应的RX TX配置接口如下： 2](#_Toc2609655)

[485TX和RX设置的函数接口如下： 2](#_Toc2609656)

[移植JNI注意事项 3](#_Toc2609657)

[处理业务逻辑的类 4](#_Toc2609658)

[如何判断测试成功？ 6](#_Toc2609659)

[So如何编译的呢？ 7](#_Toc2609660)

## 本文目的：

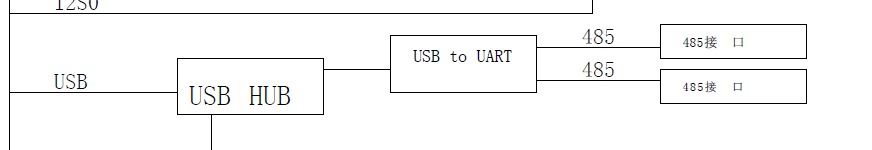
简单说明这个DEMO的功能，还有注意点

## 功能：

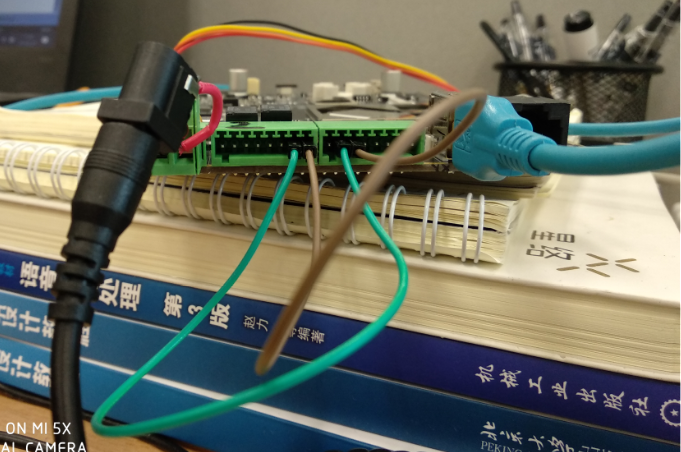
通过两路485的RX TX来验证485的硬件功能是正常的。

## 硬件框图如下：

详细硬件连接图查看对应研二最新的原理图



## 测试连接图如下



## 对应的两路tty设备如下：



## 485 对应的RX TX配置接口如下：



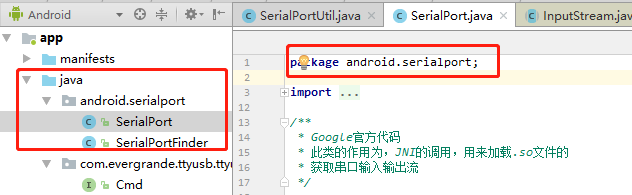
说明：0 表示把对应tty设备设置是RX,1表示把对应tty设备设置为TX。

## 485TX和RX设置的函数接口如下：

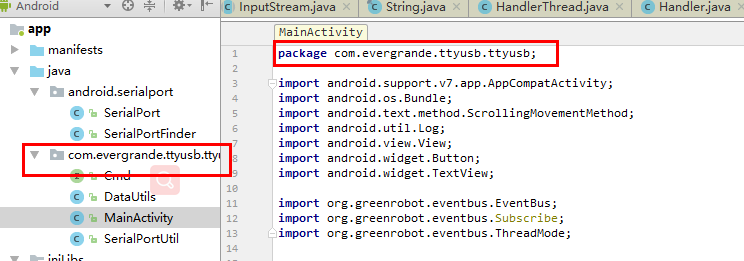
|  |
| --- |
| */\*\*  \* 设置485电源 使能为rx tx  \* 0 表示设置为 rx 1表示设置为tx  \*/* **public void** setTtyusb0InputTtyusb1Ouput(String thvd0,String thvd1){  Log.*d*(***TAG***, "thvd0:"+thvd0+" thvd1:"+thvd1);  setProcStringValue(***THVD0***,thvd0);  setProcStringValue(***THVD1***,thvd1);  } |

## 移植JNI注意事项

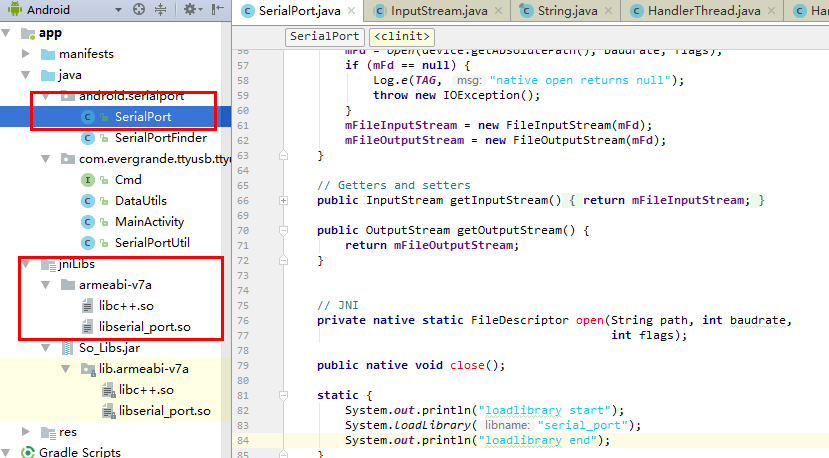
因为编译JNI的时候固定了包名，所以这个名字要固定起来，对应如下图



处理逻辑的包名如下，两个分开来



So的位置要和android.serialport目录在一个位置，要不然加载的时候可能提示错误，很多时候是找不到目标so。如下图所示



## 处理业务逻辑的类

处理发送和接收的核心代码在SerialPortUtil里面。

发送线程

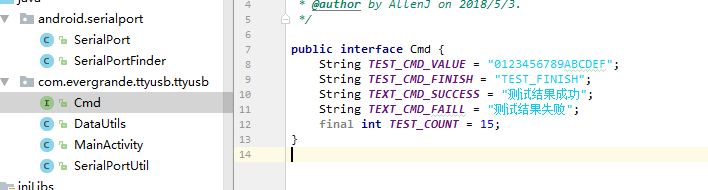
|  |
| --- |
| **private class** SendThread **extends** Thread{   **public void** run() {  **super**.run();  **isRunning** = **true**;  **successCount** =0;  Log.*d*(***TAG***,"开始测试发送数据");   setTtyusb0InputTtyusb1Ouput("0","1");  **CmdUsb** = ***TTYUSB1***;  Log.*d*(***TAG***,**CmdUsb**);  **for**(**int** i = 0; i < Cmd.***TEST\_COUNT***; i++) {  **lock**.lock();  sendSerialPort(***TTYUSB1***, Cmd.***TEST\_CMD\_VALUE***);  **lock**.unlock();  sendTreadSleep(50);  }   **CmdUsb** = ***TTYUSB0***;  Log.*d*(***TAG***,**CmdUsb**);  setTtyusb0InputTtyusb1Ouput("1","0");  **mReceiveThread**.start();  sendTreadSleep(10);  **for**(**int** i = 0; i < Cmd.***TEST\_COUNT***; i++) {  **lock**.lock();  sendSerialPort(***TTYUSB0***,Cmd.***TEST\_CMD\_VALUE***);  **lock**.unlock();  sendTreadSleep(50);  }    */\*结果判断\*/* **if**(**successCount**/2 == Cmd.***TEST\_COUNT***) {  EventBus.*getDefault*().post(Cmd.***TEXT\_CMD\_SUCCESS***);  }**else**{  EventBus.*getDefault*().post(Cmd.***TEXT\_CMD\_FAILL***);  }  EventBus.*getDefault*().post(Cmd.***TEST\_CMD\_FINISH***);   */\*把运行标志位清除\*/* **isRunning** = **false**;  } } |

接收线程

|  |
| --- |
| */\*\*  \* 接收串口数据的线程  \*/* **private class** ReceiveThread **extends** Thread {  @Override  **public void** run() {  **super**.run();  //条件判断，只要条件为true，则一直执行这个线程  **while** (**isStart** == **true**)  {  **if** (**inputStream** == **null** || **inputStream1** == **null**) {  Log.*d*(***TAG***,"未初始化inputStream");  **return**;  }   **byte**[] readData = **new byte**[1024];  String readString = "none";  **int** size =0;  Log.*d*(***TAG***,**CmdUsb**);  **if**(**CmdUsb**.equals(***TTYUSB1***) == **true**)  {  **try**{  size = **inputStream**.read(readData);  }**catch** (IOException e){  e.printStackTrace();  }  **if** (size > 0) {  readString = DataUtils.*ByteArrToHex*(readData, 0, size);  EventBus.*getDefault*().post("[0]" + readString);  Log.*d*(***TAG***, "接收数据[0]:" + readString);  }  }  **if**(**CmdUsb**.equals(***TTYUSB0***) == **true**)  {  **try**{  size = **inputStream1**.read(readData);  }**catch** (IOException e){  e.printStackTrace();  }  **if** (size > 0) {  readString = DataUtils.*ByteArrToHex*(readData, 0, size);  EventBus.*getDefault*().post("[1]" + readString);  Log.*d*(***TAG***, "接收数据[1]:" + readString);  }  }  **if**(readString.equals(Cmd.***TEST\_CMD\_VALUE***)){  **successCount**++;  Log.*d*(***TAG***,"successCount:"+ **successCount**);  }  }   } } |

## 如何判断测试成功？

如果接收到的数据和发送的数据一致，就表示测试成功，目前发送15次，接收15次。一共成功30次，就表明没有问题。

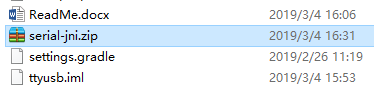


## So如何编译的呢？



直接使用mm –B 进行编译，生成的so目标文件会直接显示在相应的out目录下面。

对应的源码我拷贝在当前目录下了



比较核心的是Android.mk，现在编译的是32位，如果需要修改请baidu之

|  |
| --- |
| #########################################  LOCAL\_PATH := $(call my-dir)  include $(CLEAR\_VARS)  LOCAL\_SRC\_FILES:= SerialPort.c  LOCAL\_MODULE := libSerialPort  LOCAL\_LDFLAGS += -fPIC  LOCAL\_LDLIBS := -lm -llog  LOCAL\_MULTILIB := 32  include $(BUILD\_SHARED\_LIBRARY) |