**计算机网络编程**

**实验报告**

**班级：07111707**

**组长：1120171189 崔程远**

**成员：1120172149 吴沁璇**

**1120172153 张澈**

**1120172163 王晓媛**

**1120172736 张鉴昊**

**1120172765 曾煜瑾**

**1120173326 曾紫飞**

**北京理工大学**

**计算机学院**

**2020年5月**

**第二章 实验1 EIA RS232C串口通信程序**

**1. 实验目的**

实现RS232串口通讯，加深对串口通讯的理解。

1. **实验内容**

配置文件关键要点：

串口通信设备

SerialPort=COM1或COM2等

串口的波特率

BaudRate=9600

数据位

DataBits=8

校验位

Parity=0或1

停止位

StopBits=1

程序运行屏幕输出要点：

首先显示串口通信的设备名称、串口属性

然后提示键盘输入要发送的数据

串口接收并显示接收的数据

1. **实验原理**

串口是计算机上一种非常通用设备通信的协议，大多数计算机包含基于RS232的串口。串口通信的原理很简单，串口按位发送和接收字节，尽管比按字节的并行通信慢，但是串口可以在使用一根线发送数据的同时用另一根线接收数据，它很简单并且能够实现远距离通信。

典型地，串口用于ASCII码字符的传输。通信使用3根线完成：（1）地线，（2）发送，（3）接收。由于串口通信是异步的，端口能够在一根线上发送数据的同时在另一根线上接收数据。串口通信最重要的参数是波特率、数据位、停止位和奇偶校验。

1. 波特率：这是一个衡量通信速度的参数，它表示每秒钟传送的bit的个数。例如300波特表示每秒钟发送300个bit。当我们提到时钟周期时，就是指波特率。
2. 数据位：这是衡量通信中实际数据位的参数。当计算机发送一个信息包，实际的数据不会是8位的，标准的值是5、7和8位，如何设置取决于你想传送的信息。
3. 停止位：用于表示单个包的最后一位，典型的值为1,1.5和2位。由于数据是在传输线上定时的，并且每一个设备有其自己的时钟，很可能在通信中两台设备间出现了小小的不同步。因此停止位不仅仅是表示传输的结束，并且提供计算机校正时钟同步的机会。适用于停止位的位数越多，不同时钟同步的容忍程度越大，但是数据传输率同时也越慢。
4. 奇偶校验位：在串口通信中一种简单的检错方式。串口会设置校验位（数据位后面的一位），用一个值确保传输的数据有偶个或者奇个逻辑高位。这样使得接收设备能够知道一个位的状态，有机会判断是否有噪声干扰了通信或者是否传输和接收数据是否不同步。
5. **实验环境**

|  |  |  |  |
| --- | --- | --- | --- |
| 语言 | 集成开发环境 | 编译器 | 外部包 |
| C++ | Visual Studio 2017 | gcc version 4.8.1 |  |
| Java | Eclipse 2019 | java version "1.8.0\_65" | RXTXcomm.jar |
| Python | Pycharm 2017 | Python 3.7.0 |  |

1. **实验步骤**

**·C语言实现过程**

C语言实现比较基础，主要难点在于如何打开串口、如何用串口进行数据传输、如何关闭串口

除此以外比较难的地方在于相应的API的使用方法，创建串口的函数如下

hCom = CreateFileA(portname, //串口名

GENERIC\_READ | GENERIC\_WRITE, //支持读写

0, //独占方式，串口不支持共享

NULL,//安全属性指针，默认值为NULL

OPEN\_EXISTING, //打开现有的串口文件

0, //0：同步方式，FILE\_FLAG\_OVERLAPPED：异步方式

NULL);//用于复制文件句柄，默认值为NULL，对串口而言该参数必须置为NULL

然后就是对串口进行参数设置，如

COMMTIMEOUTS TimeOuts;

TimeOuts.ReadIntervalTimeout = 1000; //读间隔超时

TimeOuts.ReadTotalTimeoutMultiplier = 500; //读时间系数

TimeOuts.ReadTotalTimeoutConstant = 5000; //读时间常量

TimeOuts.WriteTotalTimeoutMultiplier = 500; // 写时间系数

TimeOuts.WriteTotalTimeoutConstant = 2000; //写时间常量

SetCommTimeouts(hCom, &TimeOuts);

PurgeComm(hCom, PURGE\_TXCLEAR | PURGE\_RXCLEAR);//清空串口缓冲区

接下来就可以对串口进行数据的发送或接收操作，相应的代码如下：

接收：

int WzSerialPort::receive(void\* buf, int maxlen)

{

HANDLE hCom = \*(HANDLE\*)pHandle;

//同步方式

DWORD wCount = maxlen; //成功读取的数据字节数

BOOL bReadStat = ReadFile(hCom, //串口句柄

buf, //数据首地址

wCount, //要读取的数据最大字节数

&wCount, //DWORD\*,用来接收返回成功读取的数据字节数

NULL); //NULL为同步发送，OVERLAPPED\*为异步发送

if (!bReadStat)

{

return 0;

}

return wCount;

}

发送：

int WzSerialPort::send(const void\* buf, int len)

{

HANDLE hCom = \*(HANDLE\*)pHandle;

// 同步方式

DWORD dwBytesWrite = len; //成功写入的数据字节数

BOOL bWriteStat = WriteFile(hCom, //串口句柄

buf, //数据首地址

dwBytesWrite, //要发送的数据字节数

&dwBytesWrite, //DWORD\*，用来接收返回成功发送的数据字节数

NULL); //NULL为同步发送，OVERLAPPED\*为异步发送

if (!bWriteStat)

{

return 0;

}

return dwBytesWrite;

}

相应的主程序只需要对串口的参数进行初始化以及调用Receive/Send模块对消息进行发送。这里发送端和接收端的代码类似，有区别的地方在于一个调用receiveDemo函数另一个调用sendDemo函数，在发送端发送stop后双方停止传输。

**·Java关键代码**

打开端口

// 获取要打开的端口

**try** {

portId = CommPortIdentifier.*getPortIdentifier*(portname);

} **catch** (NoSuchPortException e) {

showErrMesgbox("抱歉,没有找到" + portname + "串行端口号!");

setComponentsEnabled(**true**);

**return**;

}

// 打开端口

**try** {

serialPort = (SerialPort) portId.open("RS232", 2000);

statusLb.setText(portname + "串口已经打开!");

} **catch** (PortInUseException e) {

showErrMesgbox(portname + "端口已被占用,请检查!");

setComponentsEnabled(**true**);

**return**;

}

设置端口参数

// 设置端口参数

**try** {

**int** rate = Integer.*parseInt*(**this**.rate);

**int** data = Integer.*parseInt*(**this**.data);

**int** stop = stopCombox.getSelectedIndex() + 1;

**int** parity = parityCombox.getSelectedIndex();

serialPort.setSerialPortParams(rate, data, stop, parity);

} **catch** (UnsupportedCommOperationException e) {

showErrMesgbox(e.getMessage());

}

// 打开端口的IO流管道

**try** {

outputStream = serialPort.getOutputStream();

inputStream = serialPort.getInputStream();

} **catch** (IOException e) {

showErrMesgbox(e.getMessage());

}

// 给端口添加监听器

**try** {

serialPort.addEventListener(**this**);

} **catch** (TooManyListenersException e) {

showErrMesgbox(e.getMessage());

}

serialPort.notifyOnDataAvailable(**true**);

}

发送数据

**public** **void** sendDataToSeriaPort() {

**try** {

sendCount++;

outputStream.write(mesg.getBytes());

outputStream.flush();

} **catch** (IOException e) {

showErrMesgbox(e.getMessage());

}

statusLb.setText(" 发送: " + sendCount + " 接收: " + reciveCount);

}

监听事件

**public** **void** actionPerformed(ActionEvent e) {

**if** (e.getSource() == portCombox || e.getSource() == rateCombox || e.getSource() == dataCombox

|| e.getSource() == stopCombox || e.getSource() == parityCombox) {

statusLb.setText(initStatus());

}

**if** (e.getSource() == openPortBtn) {

setComponentsEnabled(**false**);

openSerialPort();

}

**if** (e.getSource() == closePortBtn) {

**if** (serialPort != **null**) {

closeSerialPort();

}

setComponentsEnabled(**true**);

}

**if** (e.getSource() == sendMsgBtn) {

**if** (serialPort == **null**) {

showErrMesgbox("请先打开串行端口!");

**return**;

}

mesg = sendTf.getText();

**if** (**null** == mesg || mesg.isEmpty()) {

showErrMesgbox("请输入你要发送的内容!");

**return**;

}

sendDataToSeriaPort();

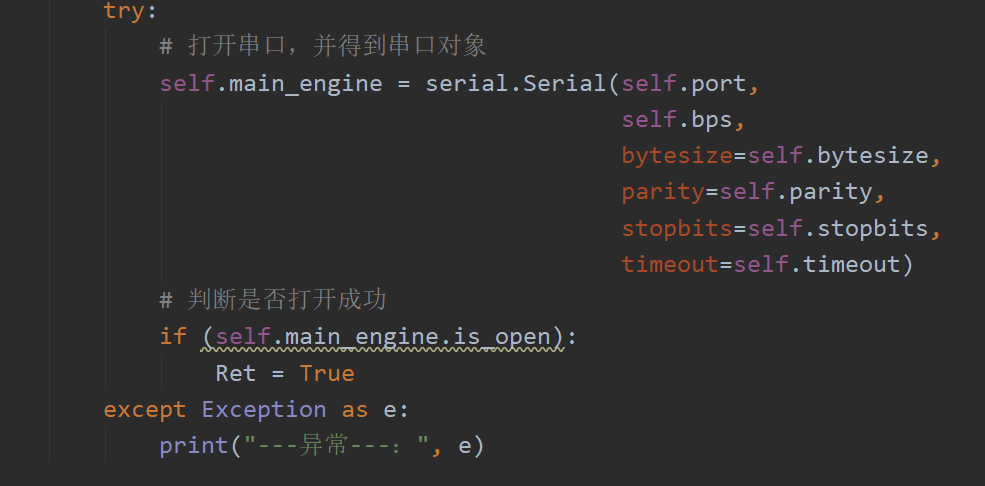
}

}

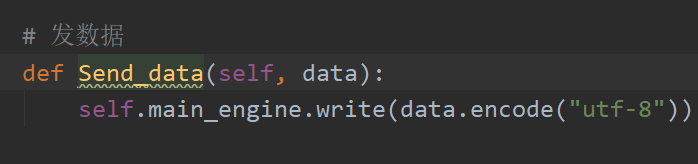
**·Python**

发送端：

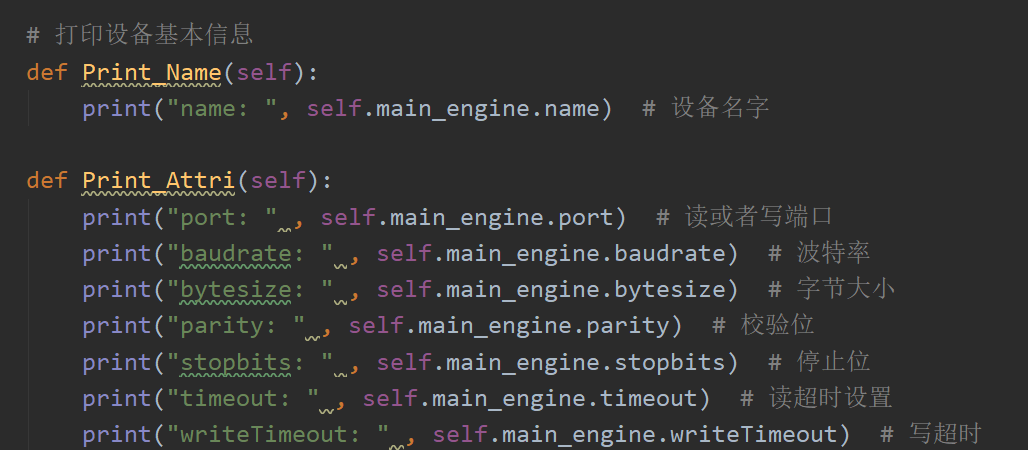
打开串口，得到串口对象：



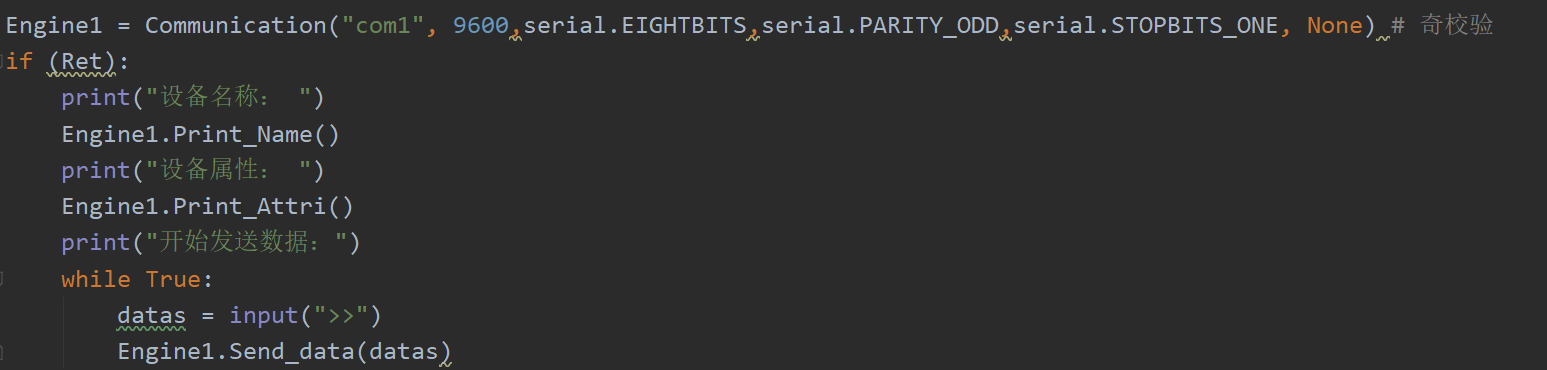
发送数据：



输出设备基本信息：

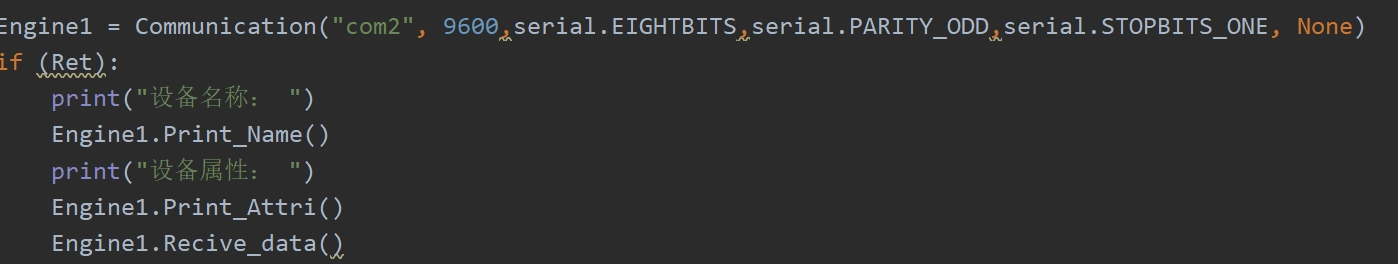


程序通讯字符串界面：

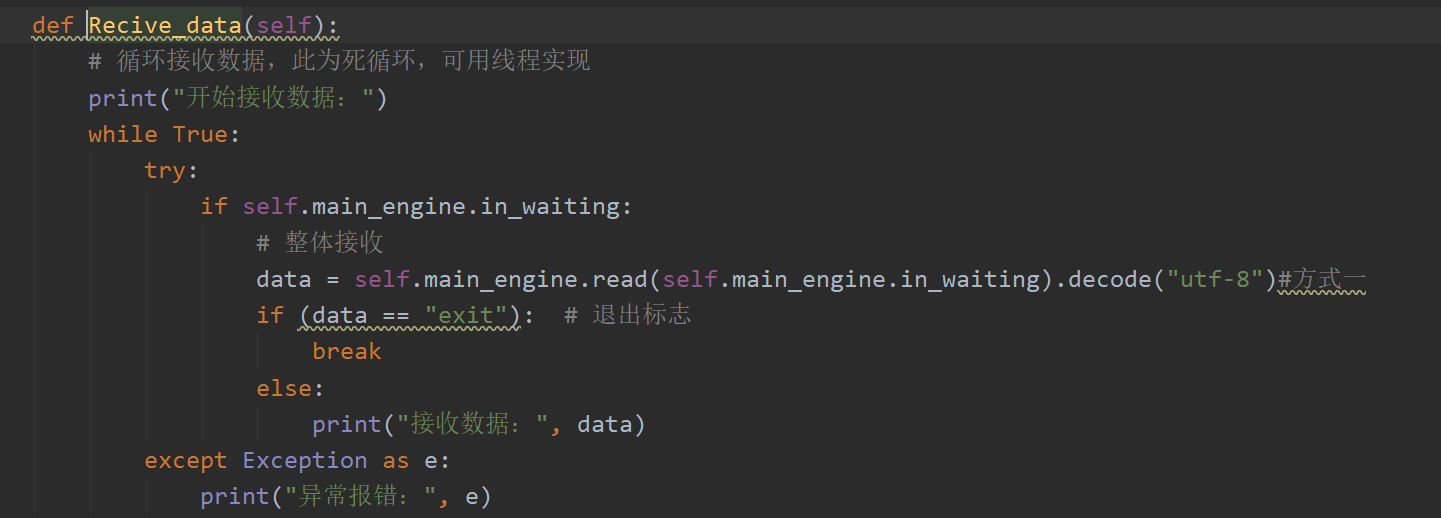


接收端：

其余部分基本同上，区别在于发送端调用发送函数，接收端调用接收函数



接收函数：



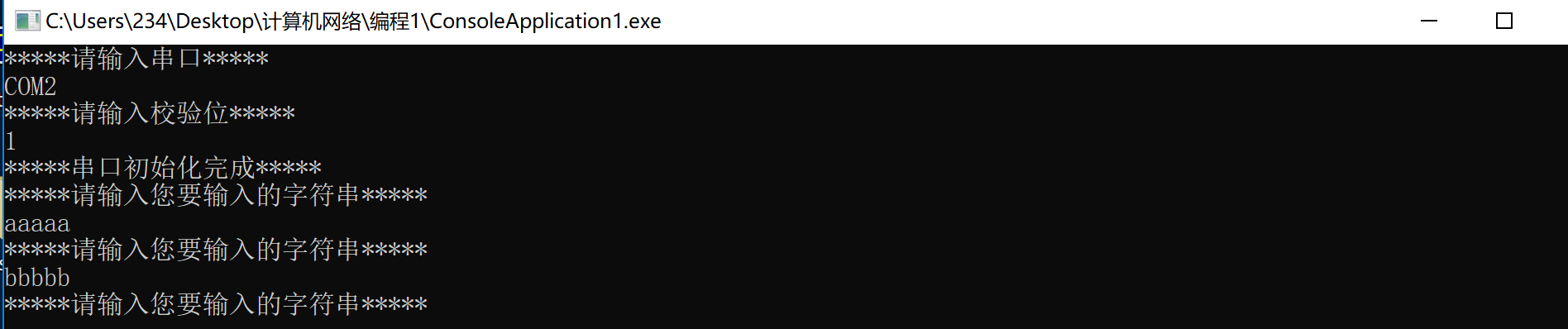
1. **运行结果**

**·C**

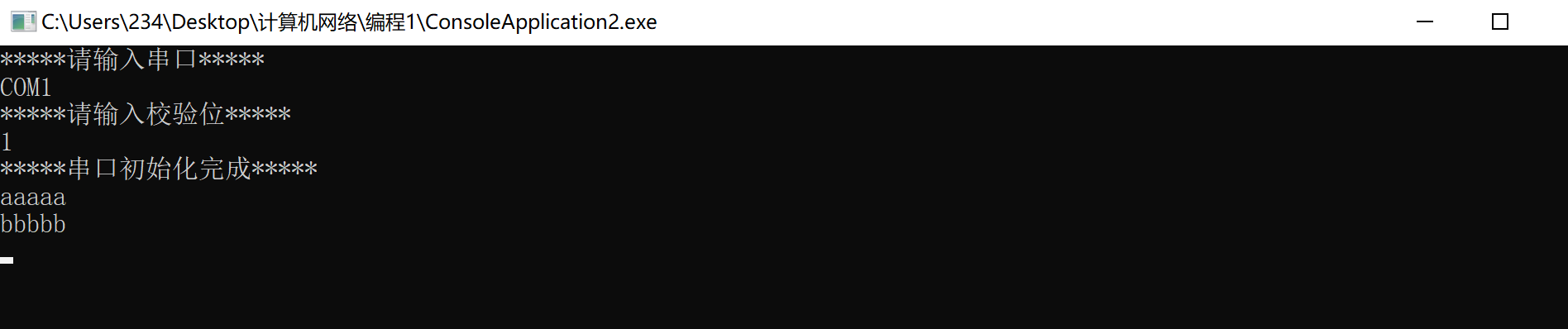
**串口配置为COM1和COM2：**

**校验位可以是0（无校验）或1（奇校验）或2偶校验**

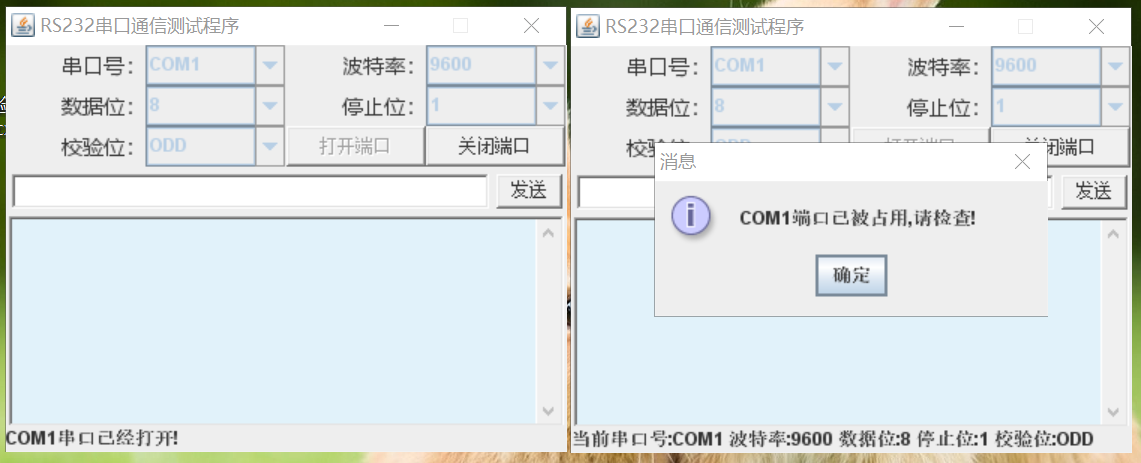
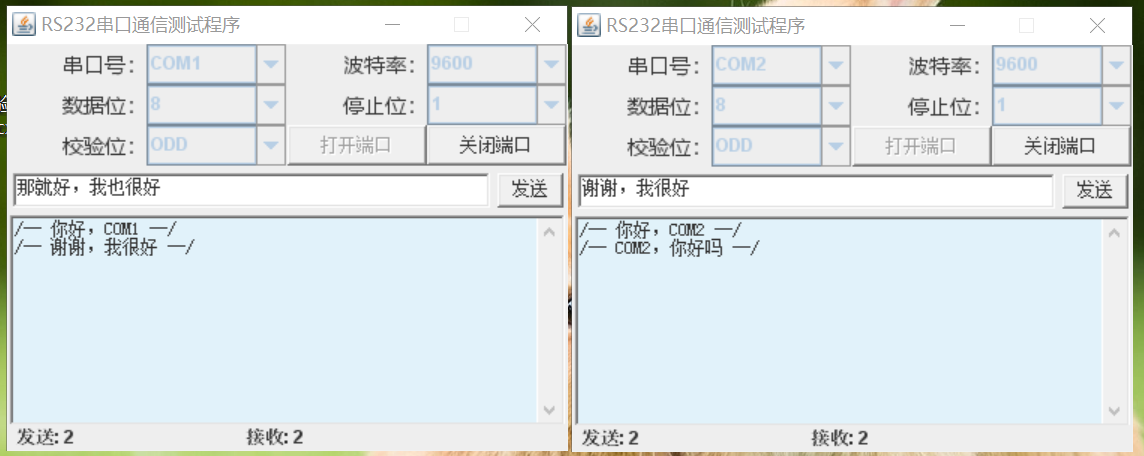
**发送端运行结果如下：**

****

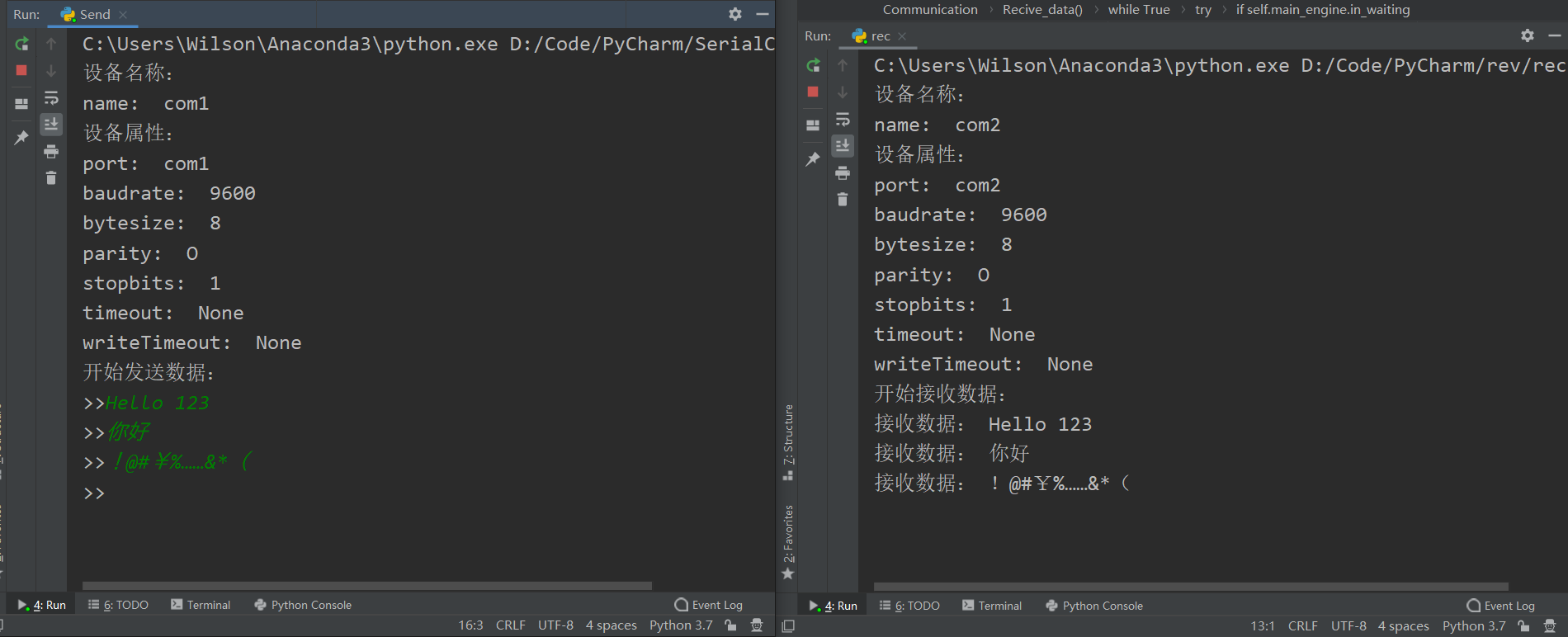
**接收端程序运行结果如下：**

****

**·java**



**·Python**



1. **实验总结**

经过这次试验我了解了串口通讯的基本内容和原理，对理解物理层的通讯协议和标准大有裨益。