在编码之前先来理一理串口通信的主要环节：

1. 计算机首先需要进行硬件check,查找是否有可用的COM端口，并对该对端口进行简要判断，包括这些端口是否是串口，是否正在使用。以下是部分主要代码：

/\*类方法 不可改变 不接受继承

\* 扫描获取可用的串口

\* 将可用串口添加至list并保存至list

\*/

public static final ArrayList<String> uartPortUseAblefind()

{

//获取当前所有可用串口

//由CommPortIdentifier类提供方法

Enumeration<CommPortIdentifier> portList=CommPortIdentifier.getPortIdentifiers();

ArrayList<String> portNameList=new ArrayList();

//添加并返回ArrayList

while(portList.hasMoreElements())

{

String portName=portList.nextElement().getName();

portNameList.add(portName);

}

return portNameList;

}

以下是测试类的测试实例：

ArrayList<String> arraylist=UARTParameterSetup.uartPortUseAblefind();

int useAbleLen=arraylist.size();

if(useAbleLen==0)

{

System.out.println("没有找到可用的串口端口，请check设备！");

}

else

{

System.out.println("已查询到该计算机上有以下端口可以使用：");

for(int index=0;index<arraylist.size();index++)

{

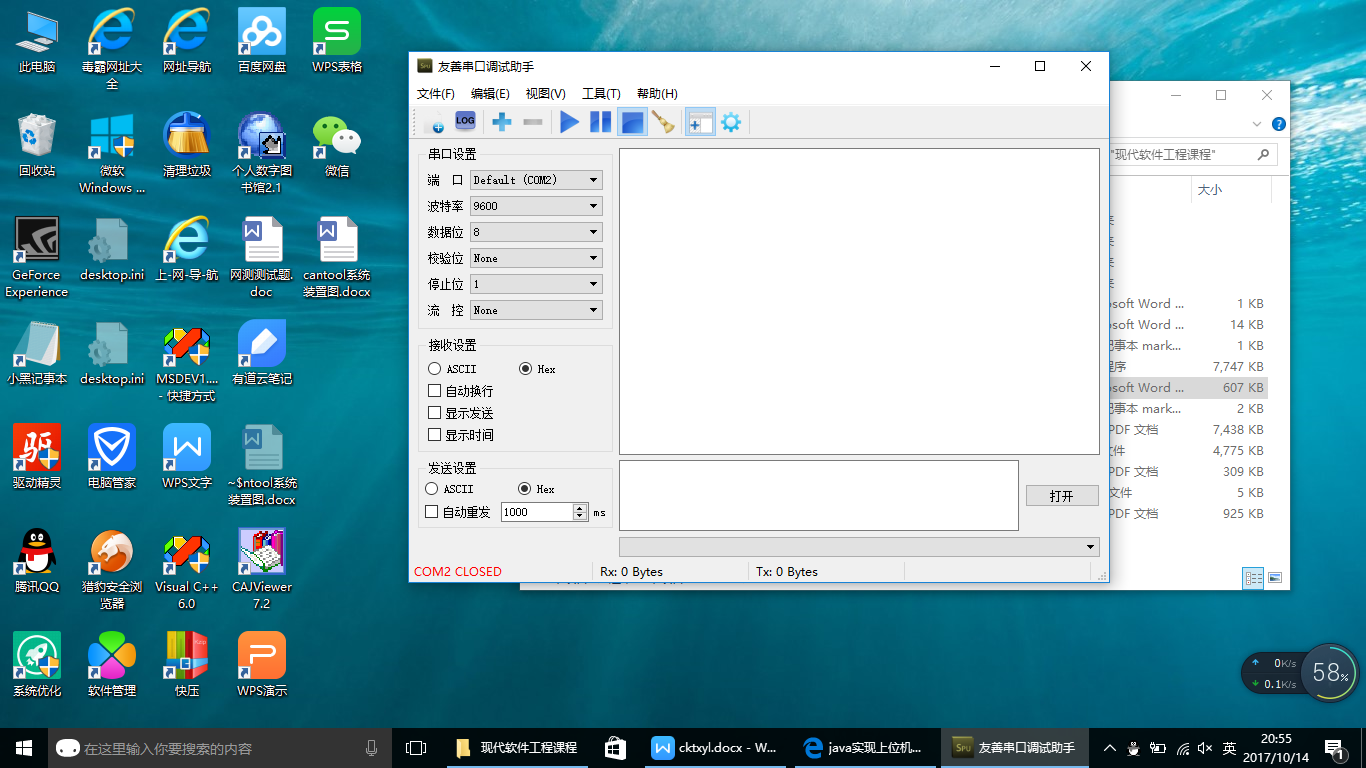
System.out.println("该COM端口名称:"+arraylist.get(index));

//测试串口配置的相关方法

}

}

通过计算机对串口的自检后，可以对串口参数进行简单的配置。常见的配置可以从常见的串口助手中得到启发。以下是一个串口助手的人机交换界面：



以下是对串口设置主要代码：

/\*

\* 串口常见设置

\* 1)打开串口

\* 2)设置波特率 根据单板机的需求可以设置为57600 ...

\* 3)判断端口设备是否为串口设备

\* 4)端口是否占用

\* 5)对以上条件进行check以后返回一个串口设置对象new UARTParameterSetup()

\* 6)return:返回一个SerialPort一个实例对象，若判定该com口是串口则进行参数配置

\* 若不是则返回SerialPort对象为null

\*/

public static final SerialPort portParameterOpen(String portName,int baudrate)

{

SerialPort serialPort=null;

try

{ //通过端口名识别串口

CommPortIdentifier portIdentifier = CommPortIdentifier.getPortIdentifier(portName);

//打开端口并设置端口名字 serialPort和超时时间 2000ms

CommPort commPort=portIdentifier.open(portName,1000);

//进一步判断comm端口是否是串口 instanceof

if(commPort instanceof SerialPort)

{

System.out.println("该COM端口是串口！");

//进一步强制类型转换

serialPort=(SerialPort)commPort;

//设置baudrate 此处需要注意:波特率只能允许是int型 对于57600足够

//8位数据位

//1位停止位

//无奇偶校验

serialPort.setSerialPortParams(baudrate, SerialPort.DATABITS\_8,SerialPort.STOPBITS\_1, SerialPort.PARITY\_NONE);

//串口配制完成 log

System.out.println("串口参数设置已完成，波特率为"+baudrate+",数据位8bits,停止位1位,无奇偶校验");

}

//不是串口

else

{

System.out.println("该com端口不是串口,请检查设备!");

//将com端口设置为null 默认是null不需要操作

}

}

catch (NoSuchPortException e)

{

e.printStackTrace();

}

catch (PortInUseException e)

{

e.printStackTrace();

}

catch (UnsupportedCommOperationException e)

{

e.printStackTrace();

}

return serialPort;

}

 接下来就是上位机与下位机之间的双向通信的功能实现了。该部分主要是利用java的输入输出流来实现。以下是主要代码：

/\*

\* 串口数据发送以及数据传输作为一个类

\* 该类做主要实现对数据包的传输至下单板机

\*/

class DataTransimit

{

/\*

\* 上位机往单板机通过串口发送数据

\* 串口对象 seriesPort

\* 数据帧:dataPackage

\* 发送的标志:数据未发送成功抛出一个异常

\*/

public static void uartSendDatatoSerialPort(SerialPort serialPort,byte[] dataPackage)

{

OutputStream out=null;

try

{

out=serialPort.getOutputStream();

out.write(dataPackage);

out.flush();

} catch (IOException e)

{

e.printStackTrace();

}finally

{

//关闭输出流

if(out!=null)

{

try

{

out.close();

out=null;

System.out.println("数据已发送完毕!");

} catch (IOException e)

{

e.printStackTrace();

}

}

}

}

/\*

\* 上位机接收数据

\* 串口对象seriesPort

\* 接收数据buffer

\* 返回一个byte数组

\*/

public static byte[] uartReceiveDatafromSingleChipMachine(SerialPort serialPort)

{

byte[] receiveDataPackage=null;

InputStream in=null;

try

{

in=serialPort.getInputStream();

//获取data buffer数据长度

int bufferLength=in.available();

while(bufferLength!=0)

{

receiveDataPackage=new byte[bufferLength];

in.read(receiveDataPackage);

bufferLength=in.available();

}

}

catch (IOException e)

{

e.printStackTrace();

}

return receiveDataPackage;

}

 通过以上关于Uart两个基本类实现对底层Uart的功能封装，其中一个类主要负责Uart串口自检和基本设置，另外一个类主要has数据传输的两个方法。