每天都更新内容和进度

CanTool项目需求分析：

1. CanTool装置与上位机通过USB串口或蓝牙RFComm实现UART串口通信。
2. 上位机与CanTool装置之间的信息传送方式使用ASCII码格式+ \r（即0x0d）方式进行信息交换。
3. CanTool装置接收CanToolApp发送的CAN信息并显示并发送到CAN总线。
4. 进行单元测试－功能测试－集成测试－场景测试－系统测试－Alpha/Beta Test。
5. 以Arduino微控制器为基础设计CanTool硬件装置，实现CanTool装置信号的发送和接收。

我的任务是实现CanTool装置与上位机（Windows PC）通过USB串口进行UART通信。

1、串口的工作原理

串口通信（Serial Communications）的概念非常简单，串口按位（bit）发送和接收字节，是一种可以将接受来自CPU的并行数据字符转换为连续的串行数据流发送出去，同时可将接受的串行数据流转换为并行的数据字符供给CPU的器件。一般完成这种功能的电路，我们称为串行接口电路。随着发展的需要串口的硬件结构类型多种多样。串行接口按电气标准及协议来分包括RS-232-C、RS-422、RS485等。RS-232-C、RS-422与RS-485标准只对接口的电气特性做出规定，不涉及接插件、电缆或协议。

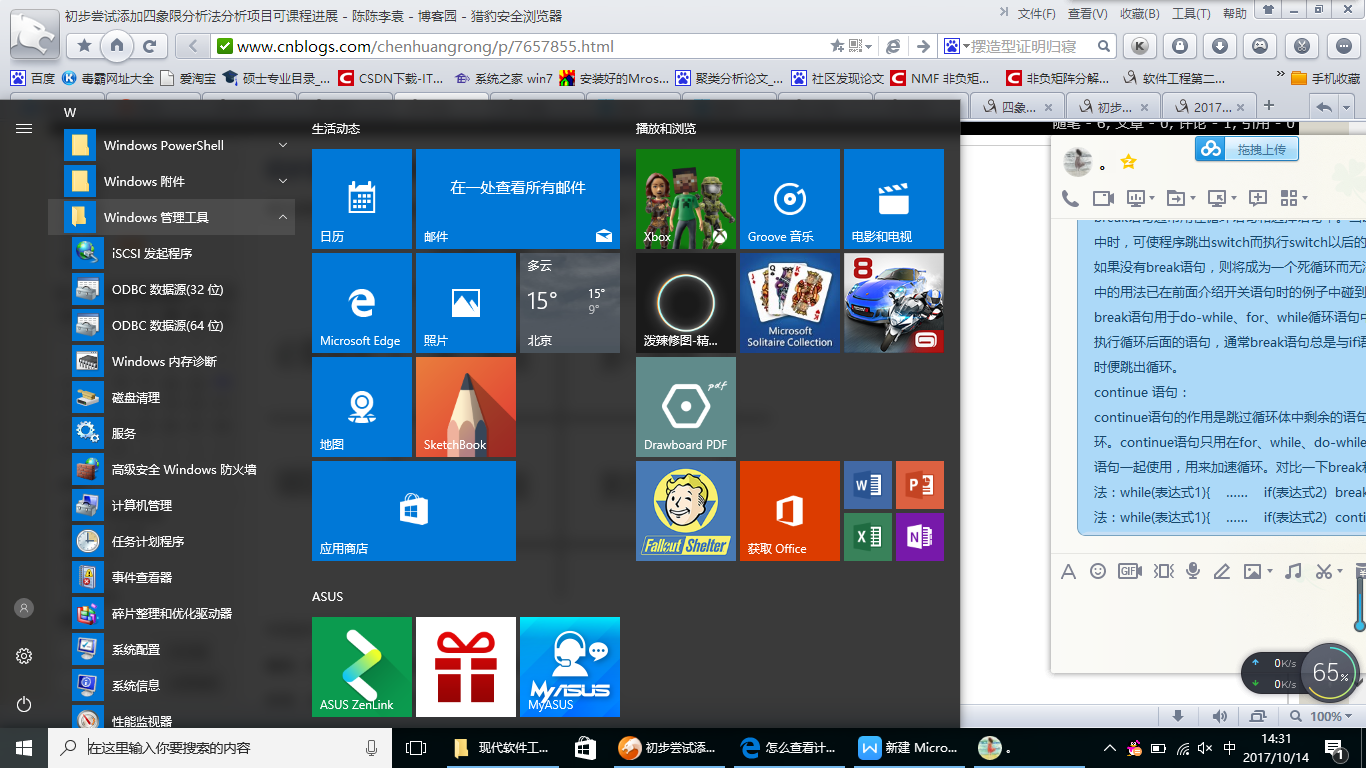
2、基于USB接口的设计分析

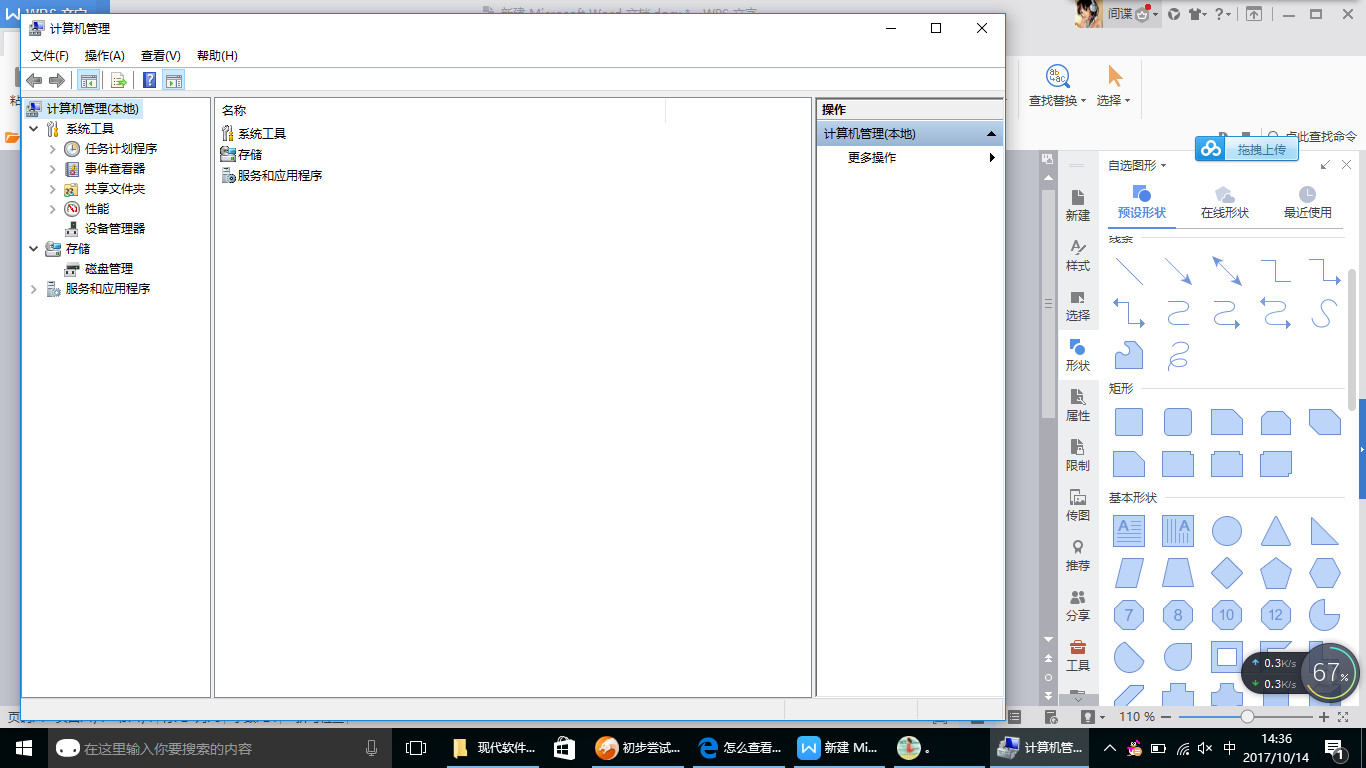
USB 2.0的工作原理

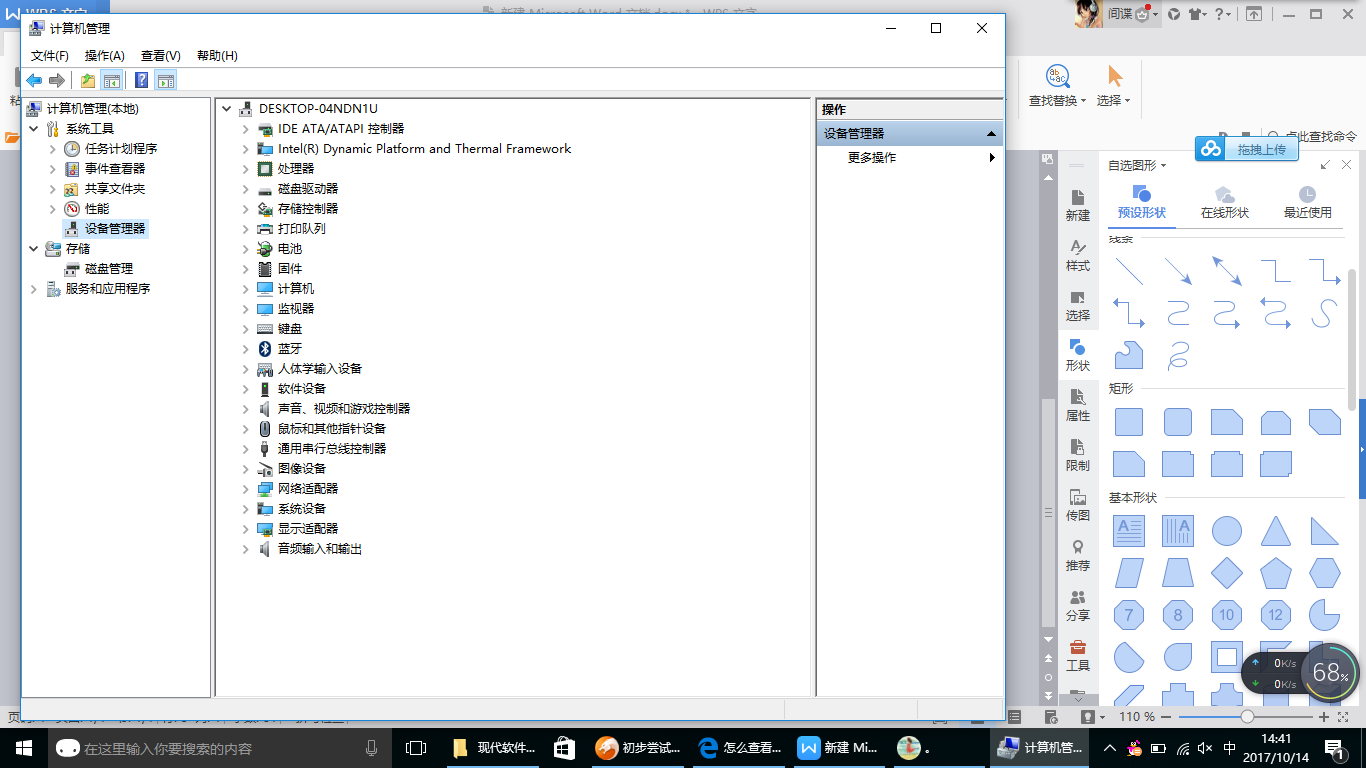
USB设备插到USB接口，PC的总线驱动（ms提供）枚举设备，给USB设备分配一个地址，并通过USB标准协议来获取USB的设备描述符，这个描述符包括USB设备程序中的一个描述表，它描述了USB的配置数，每个配置的接口数及可选接口，已及接口的端点（endpoint）个数及端点的功能（传输方式bulk，interrupt，iso，端点的大小，方向等），并且可以发出选择配置、接口来选择一种配置进行传输。PC通过获取USB的vid，pid来加载USB设备的相应驱动程序。然后就可以通过驱动程序发送ioctl交给总线驱动，总线驱动封装成urb（USB请求块）与USB设备的程序进行通讯，以达到控制USB设备。

检查自己的电脑是否有可以用的端口：

打开管理，点击“计算机管理”，然后点击“设备管理器”，如下图，可以看到并没有出现端口，这是因为没有连接设备，如果连接了就会出现。







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++)

{

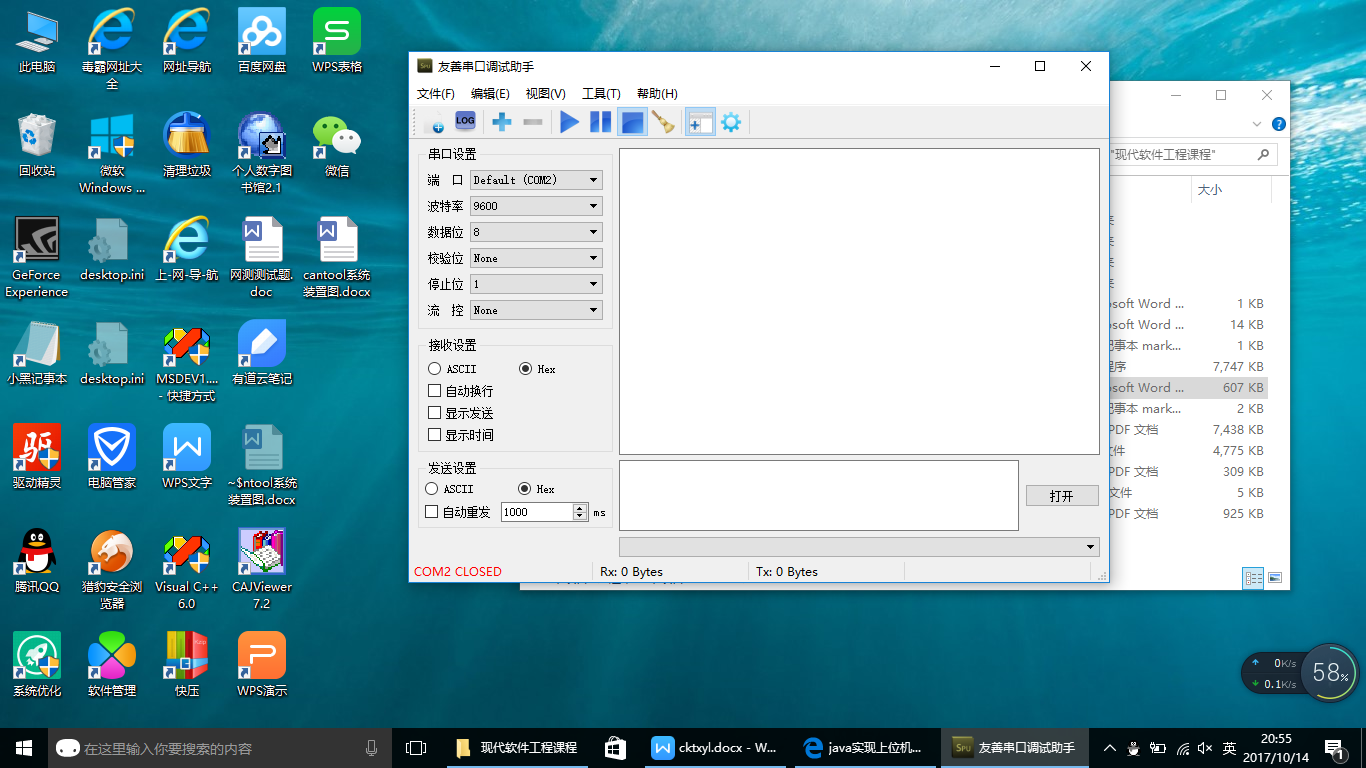
System.out.println("该COM端口名称:"+arraylist.get(index));

//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。