目　　录

1. 下载地址

iNeuOS 试用地址：http://192.144.173.38:8081/login.html

iNeuOS 默认用户和密码：admin，admin

iNeuView 视图建模使用手册：https://www.cnblogs.com/lsjwq/p/10928843.html

iNeuKernel 物联网框架：https://pan.baidu.com/s/1nxpnC7FazBBVyK9zUFgjyg

2. 系统概述

iNeuKernel 是 iNeuOS 的核心组件之一，支持设备容器的整体运行。iNeuKernel 是一个开放的平台框架，支持设备或传感器的各类协议的二次开发和集成。针对物理设备或传感器定义了一套标准的接口，同时可以自定义属性、行为和事件等。开发好设备驱动后，可以又挂载到 iNeuKernel 运行，实时与物理设备或传感器进行交互。“设备驱动”本身是一个抽象的概念，可能是一个设备或传感器，也可能是一个工厂、楼宇、车辆等，根据实际的应用场景而定。

写这篇文章的目的是希望有更多网友能够参与进来，开发针对领域、行业的设备驱动，网友之间又能够共享，从而形成整体的生态建设。

3. iNeuKernel 部署

1）在 mysql 中创建数据库实例，例如数据库名称为：iNeuKernel。

2）在数据库实例中运行“数据库脚本\iNeuKernel .sql”，初始化数据表。

3）请查看“数据库脚本\upgrade.sql”脚本，在原来数据库基础上更新。

4）在“INeuKernel Designer \INeuKernel\ SourceConfig.cfg”配置 mysql 的数据库实例信息。

5）运行“INeuKernel.Designer.exe“应用程序即可。

4. 二次开发流程

（1）引用 iNeuKernel.dll 程序集。

（2）新建驱动的协议类，并且继承 ProtocolDriver 类。

CheckData：决定了通讯的状态：通讯正常、失败、错误三种情况。

GetCode：获得当前的协议中的设备编码，用于非轮询模式的下分配数据用。还有一个按 IP 地址分配数据。

其他接口视情况，可以不写。

（3）新建驱动的设备类，并且继承 RunDevice。

Protocol：返回新建的协议类，并实例化。

GetConstantCommand：这是下发命令，轮询状态下每次通讯都会调用。

Communicate：接收到的数据，如下 CheckData 验证成功后，返回到这里。

其他参见事例。

（4）目录说明

iNeuKernel Designer-win 或 iNeuKernel Core-linux：正常发布版本，如果缺少引用的 DLL，在这里找。

TestDeviceDriver：事例驱动，与 TestDevice 配合使用。

TestDevice：模拟设备终端。

TestLoopMain：用于测试的主程序。

注：参考事例 DEMO: iNeuKernelDemo,下载地址：

http://www.ineuos.net/index.php/products/ineukernel-15.html

5. 工程目录及引用程序集

参见“iNeuKernelDemo.sln”二次开发工程，工程项目目录，如下表：

序号	项目	说明
1	TestDevice	模拟终端设备或传感器,与 TestDeviceDriver 项目对接，中间通过自定义协议交互。
2	TestDeviceDriver	自定义设备驱动，发送指令给 TestDevice，接收返回的数据，中间通过自定义协议交互。
3	TestLoopMain	轮询通讯模式的宿主程序，在 iNeuKernel 框架下加载和运行设备驱动。
4	TestParallelMain	并发通讯模式的宿主程序，在 iNeuKernel 框架下加载和运行设备驱动。

5	TestSelfMain	自控通讯模式的宿主程序，在 iNeuKernel 框架下加载和运行设备驱动。
6	TestSingletonMain	单例通讯模式的宿主程序，在 iNeuKernel 框架下加载和运行设备驱动。

（1） TestDeviceDriver 设备驱动开发，协议方面内容介绍参见：

如开发一套设备驱动，同时支持串口和网络通讯

https://www.cnblogs.com/lsjwq/p/5986952.html

（2） 通讯模式方面内容介绍参见：

4 种通讯模式机制

https://www.cnblogs.com/lsjwq/p/5890893.html

轮询通讯模式开发及注意事项

https://www.cnblogs.com/lsjwq/p/6033773.html

并发通讯模式开发及注意事项

https://www.cnblogs.com/lsjwq/p/6049173.html

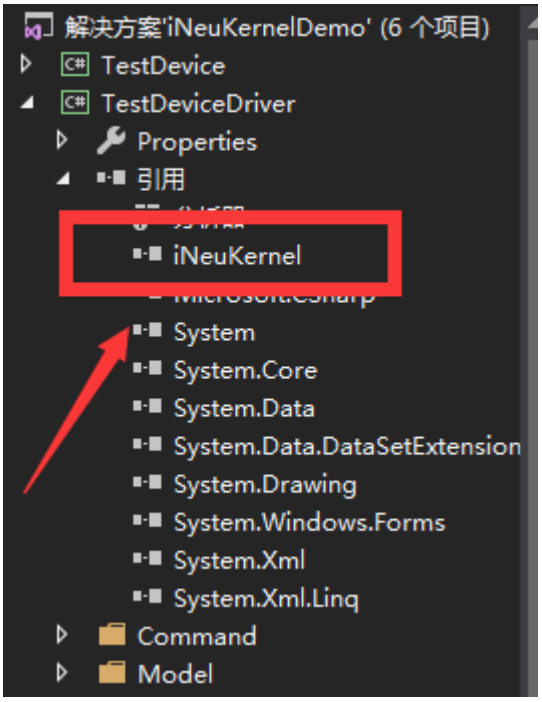
自控通讯模式开发及注意事项

https://www.cnblogs.com/lsjwq/p/6060239.html

单例通讯模式开发及注意事项

https://www.cnblogs.com/lsjwq/p/6079267.html

设备驱动二次开发只需要引用 iNeuKernel.dll 程序集，如下图：



6. 接口介绍

开发一套驱动需要写两个接口：设备接口（RunDevice）和协议接口（ProtocolDriver）,协议接口作为设备接口的一个属性存在。如果想从底层完全开发，可以继承 IRunDevice 和 IProtocolDriver 接口。

1.1 设备驱动接口

设备接口（RunDevice）作为物理设备的抽象层，定义了属性、函数和事件，下表只列出来简单开发设备驱动的接口，复杂应用后续介绍。具体介绍如下表：

序号	分类	接口名称	说明
1	函数	Initialize(object obj)	初始化设备，加载设备驱动第一个需要做的事，可以在这初始化数据和自定义加载配置信息。
2		IList<IRequestInfo> GetConstantCommand()	如果 ProtocolDriver.SendCache 中没缓存命令，则调用该函数返回要发送的数据,一般返回获得设备的实时数据命令。除自控模式外，需要自主调用 OnSendData 函数发送数据，自定义发送

			周期。
3		Communicate(IResponseInfo info)	通讯正常，符合通讯协议，这个函数负责接收和处理数据。ProtocolDriver.CheckData 决定返回的数据是通讯正常、通讯干扰、通讯中断。
4		CommunicateInterrupt(IResponseInfo info)	通讯中断，未接收到数据。ProtocolDriver.CheckData 决定返回的数据是通讯正常、通讯干扰、通讯中断。
5		CommunicateError(IResponseInfo info)	通讯错误，有数据接收到，但是不符合协议，有可能丢包或通讯受到干扰。ProtocolDriver.CheckData 决定返回的数据是通讯正常、通讯干扰、通讯中断。
6		CommunicateNone()	通讯未知，默认状态，一般不用。
7		UnknownIO()	未知通讯接口，串口未打开或网络未找到连接，调用该函数。
8		CommunicateStateChanged(CommunicateState comState)	通讯状态改变调用该函数，包括：通讯正常、通讯中断和通讯错误。
9		ChannelStateChanged(ChannelState channelState)	通道状态改变调用该函数，包括：IO 通道打开和关闭。
10		GetObject()	自定义设备返回对象，扩展接口，暂时不用。
11		ShowContextMenu()	显示上下文菜单，当不同的设备驱动需要不同的窗体来完成业务操作的时候调用，与宿主程序配合使用。
12		Exit()	当宿主程序关闭时,响应设备退出操作。
13		Delete()	当通过宿主程序删除设备时，响应删除操作。
14		Receive(IChannel io, IReceiveFilter receiveFilter)	接收数据接口，可以重写这个接口，自定义接收数据规则。
15		Send(IChannel io, IRequestInfo request, WebSocketFrameType frameType=WebSocketFrameType.Binary)	发送数据接口，可能重写这个接口，自定义发送数据规则，例如：发送两个命令才返回一个条数据。
14	属于	Protocol	协议驱动，定义了发送和接收数据的规则，参见继承 ProtocolDriver 的实体类。
15		DeviceType	设备的类型，包括：普通设备和虚拟设备。一般不使用虚拟设备。
16		ModelNumber	设备型号或编号。
17		DevicePriority	设备调试优先级设置，包括：普通和优先。设置优先级别，则最先调度执行设备驱动，例如下发控制指令。
18	事件	OnSendData(IRequestInfo request, WebSocketFrameType frameType = WebSocketFrameType.Binary);	发送数据事件，一般在自控模式下使用该函数，自定义周期下发命令。
19		IDeviceConnectorCallbackResult RunDeviceConnector(IFromDevice fromDevice, IDeviceToDevice	设备连接器，可以与其他设备驱动进行交互，发送数据和异步接收返回数据（DeviceConnectorCallback）。

		toDevice, AsyncDeviceConnectorCallback callback)	
20		DeviceConnectorCallback(object obj)	异步接收其他设备驱动返回的数据，一般调用 RunDeviceConnector 之后执行。
21		DeviceConnectorCallbackError(Exception ex)	异常接收其他设备驱动执行过程返回的异常信息，一般调用 RunDeviceConnector 之后执行。
22		IServiceConnectorCallbackResult RunServiceConnector(IFromService fromService, IServiceToDevice toDevice,AsyncServiceConnectorCallback callback)	服务连接器，用于设备驱动与服务之间进行数据交互。

1.2 设备协议接口

协议接口（ProtocolDriver）定义了协议的一些常规操作，如下表：

序号	分类	接口名称	说明
1		CheckData(byte[] data)	校验数据，决定了当前通讯状态：通讯正常、通讯错误、通讯中断。
2		GetCommand(byte[] data)	获得返回数据的命令。
3		GetAddress(byte[] data)	获得返回数据的地址。
4		GetCheckData(byte[] data)	获得返回数据的校验部分。
5		GetCode(byte[] data)	获得返回数据的编码，如果设备分配数据设置为 ServerConfig.DeliveryMode=DeliveryMode.DeviceCode，在非轮询模式下调用这个接口，决定当前数据是否是这个设备驱动处理。
6		GetPackageLength(byte[] data, IChannel channel, ref int readTimeout)	获得应该接收的数据长度，如果当前接收的数据小于这个返回值，那么继续接收数据，直到大于等于这个返回长度。如果接收数据超时，则直接返回当前已经接收的数据。如果 ServerConfig.CheckPackageLength = true 则起作用。
7		GetHead(byte[] data)	获得返回数据的头。
8		GetEnd(byte[] data)	获得返回数据的结尾。
9		DriverCommand	调用继承 ProtocolCommand 接口的实体类的 ExcuteCommand 方法，用于执行操作。
10		DriverAnalysis	调用继承 ProtocolCommand 接口的实体类的 Analysis 方法，用于解析数据。
11		DriverPackage	调用继承 ProtocolCommand 接口的实体类的 Package 方法，用于打包发送数据。
12		GetProcotolCommand(string cmdName)	获得继承 ProtocolCommand 接口的实体类对象。
13		SendCache	发送数据缓存。

14		ReceiveFilter	解析数据协议过滤器。

如果高级应用，请参见：

协议过滤器，解决一包多发、粘包、冗余数据

https://www.cnblogs.com/lsjwq/p/6083633.html

持续传输大块数据流的两种方式（如：文件）

https://www.cnblogs.com/lsjwq/p/6087544.html

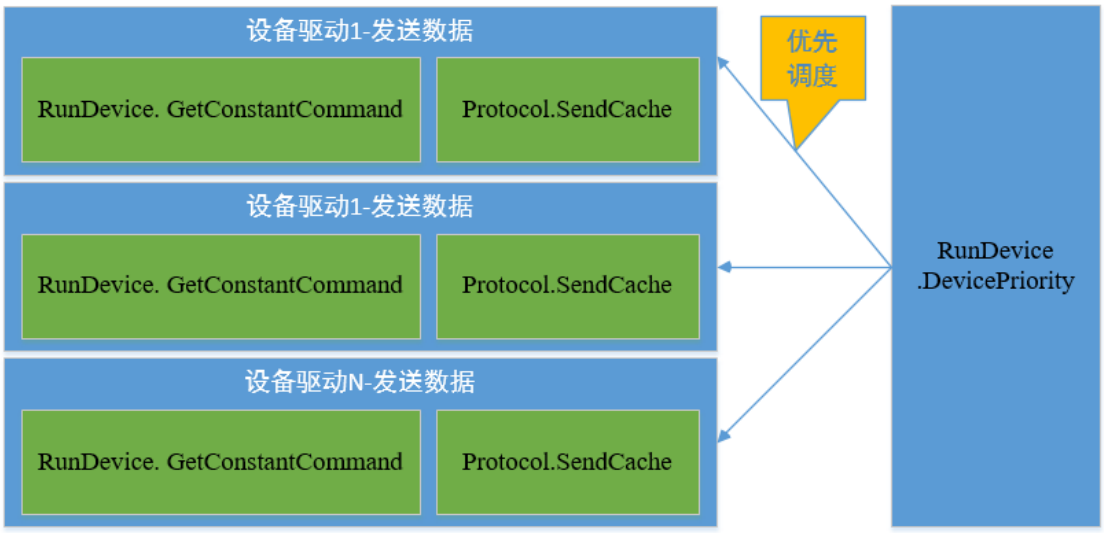
7. 数据发送和接收流程

1.3 数据发送方式及流程

发送数据有两种方式：RunDevice. GetConstantCommand 函数和 ProtocolDriver.SendCache 属性。如果 SendCache 中有命令数据，则优化发送，如果 SendCache 中没有命令数据，则调用 GetConstantCommand 函数返回要发送的数据。

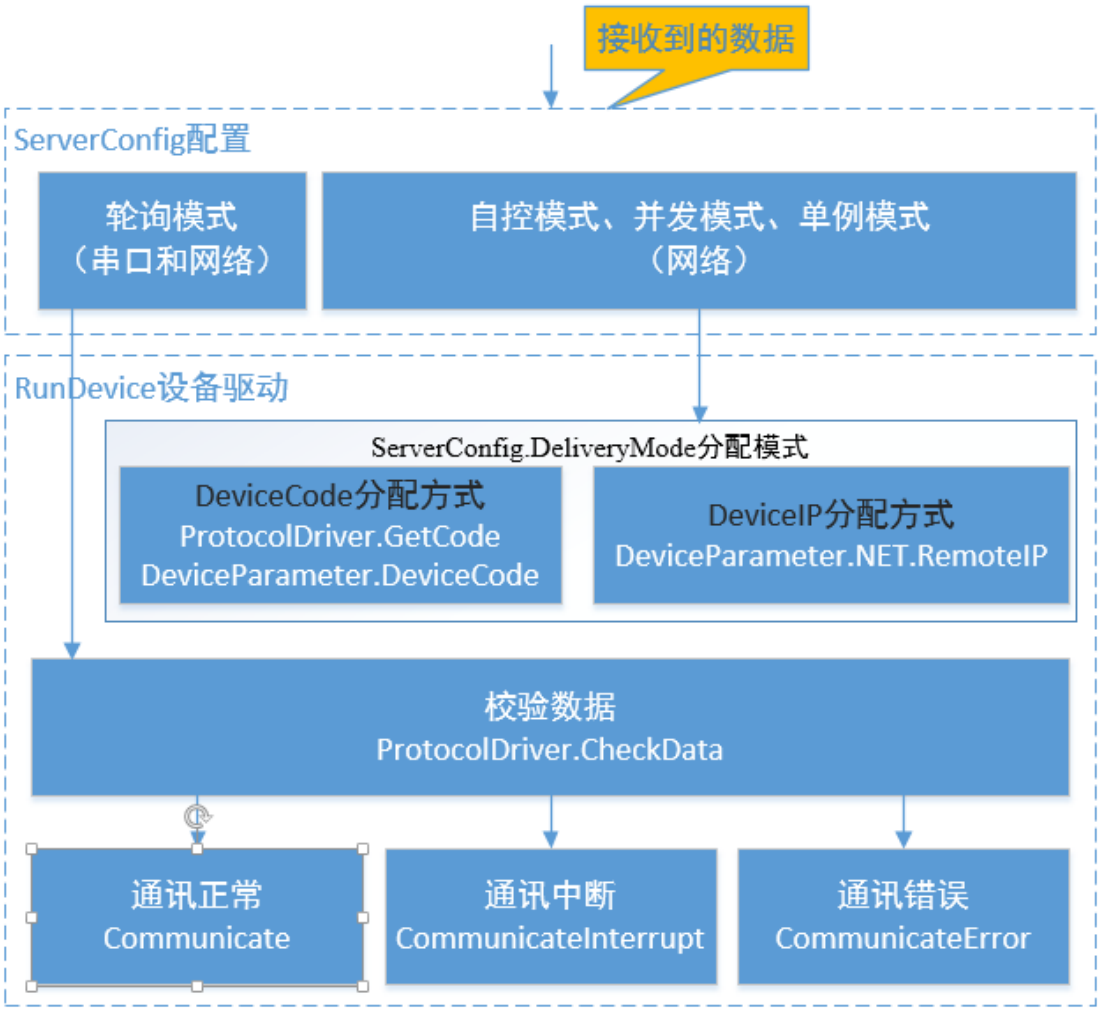
可以设置 RunDevice .DevicePriority 优化级别，设置“优先”模式，则在所有设备驱动中最先调用该设备，发送数据并等待返回数据。

示意，如下图：



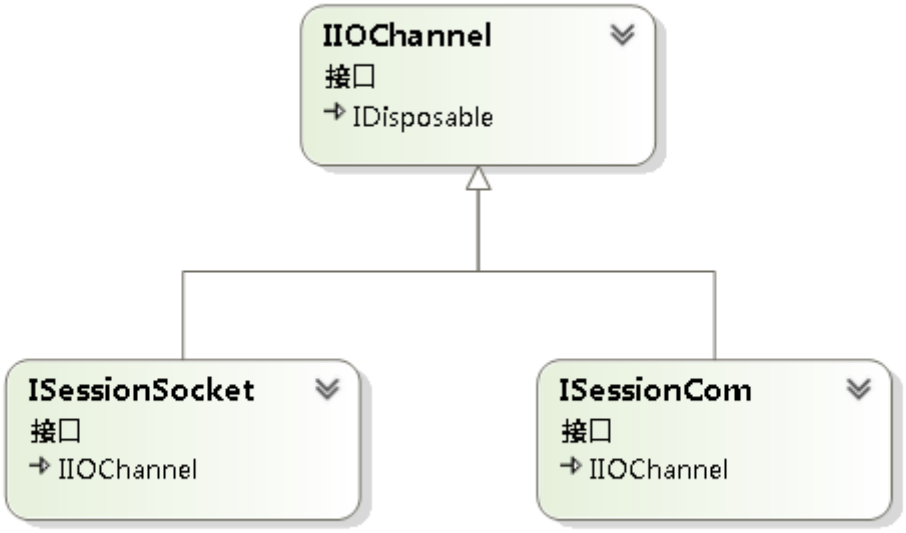
1.4 数据接收方式及流程

ServerConfig.ControlMode 决定了通讯模式，在轮询模式下一个设备驱动发送和接收数据是一个完整的过程，依次再调度下一个设备驱动，ProtocolDriver.CheckData 决定了当前通讯状态，分别调用 RunDevice 接口中的 Communicate、CommunicateInterrupt 和 CommunicateError 函数；在其他通讯模式下，会根据 ServerConfig.DeliveryMode 设置的参数（DeviceIP 和 DeviceCode）分配数据到相应的设备驱动处理，DeviceIP 分配模式是根据设置的 RunDevice.DeviceParameter.NET.RemoteIP 参数决定分配给哪个驱动处理，DeviceCode 分配模式是根据 ProtocolDriver.GetCode 与 RunDevice.DeviceParameter.DeviceCode 是否一致决定分配给哪个驱动处理。示意，如下图：



1.5 重写发送和接收接口

开发一套设备驱动同时具备串口和网络通讯能力，通讯接口在逻辑上是统一的，在此基础上串口和网络也有自己的 IO 通讯特点，根据不同的通讯方式，可以把 IChannel 实例转换成 ISessionSocket 或 ISessionCom 实例。如下图：



一个请求命令分两次发送，每次发送数据时的串口校验位不同。先发送地址信息，这时串口的配置为 Baud,m,8,1；再发送请求命令信息，这时的串口配置为 Baud,s,8,1。这样完成一次请求数据的命令。如下图：

1. 通信命令格式：

要实现与仪表通信，主机需向仪表分两步发送控制命令字。然后通过通信中断处理仪表响应数据。

(1) 先把通信协议设置为：Baud_Rate，m,8,1
通过串口发送 Address -> 仪表
延时(0.01s)

(2) 接着把通信协议设置为：Baud_Rate，S,8,1
通过串口发送以下数据到仪表：
CommandCode FunCode Data0 Data1 Data2 Data3 Crc_Code ->仪表

可以重写发送数据接口函数，完成特殊的发送数据要求。如下图：

```
public override void Send(IIOChannel io, byte[] sendbytes)
{
    if (this.CommunicationType == CommunicationType.COM)
    {
        byte[] addr = new byte[1];
        byte[] cmd = new byte[7];
        Buffer.BlockCopy(sendbytes, 0, addr, 0, 1);
        Buffer.BlockCopy(sendbytes, 1, cmd, 0, 7);

        ISessionCom comIO = (ISessionCom)io;//把io实例转换成串口实例
        //配置串口参数
        comIO.IOSettings(this._DeviceParameter.COM.Port, this._DeviceParameter.COM.Baud, DataBits.BIT_8, StopBits.STOP_1, Parity.P_MRK);
        comIO.WriteIO(addr);//发送数据

        System.Threading.Thread.Sleep(10);//延时

        comIO.IOSettings(this._DeviceParameter.COM.Port, this._DeviceParameter.COM.Baud, DataBits.BIT_8,StopBits.STOP_1, Parity.P_SPC);
        comIO.WriteIO(cmd);

    }
    else
    {
        this.OnDeviceRuningLogHandler("不支持网络模式发送数据");
    }
}
```

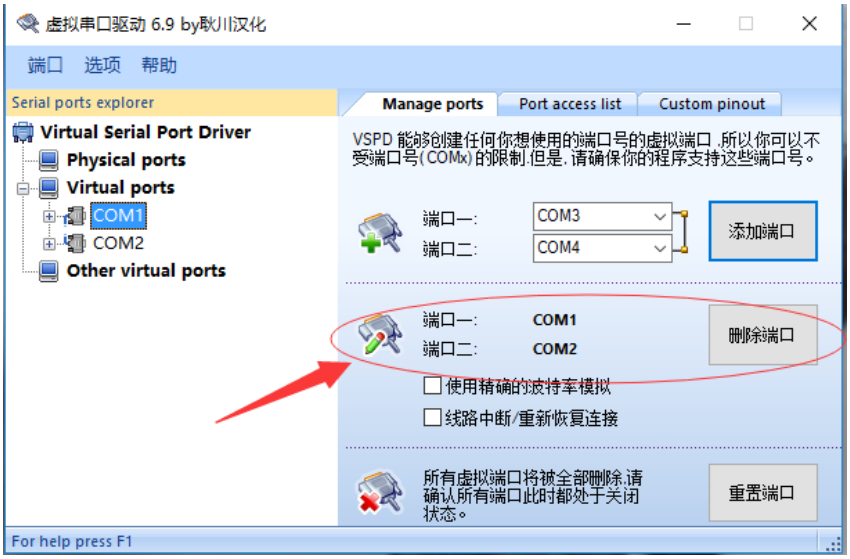
重写接收数据接口，需要把串口设置修改成默认的配置，避免影响其他设备驱动的通讯，如下图：

```
public override byte[] Receive(IIOChannel io)
{
    byte[] data = base.Receive(io);
    if (this.CommunicationType == CommunicationType.COM)
    {
        ISessionCom comIO = (ISessionCom) io; //把io实例转换成串口实例
        //把串口参数配置成默认值，避免影响其他设备驱动的通讯
        comIO.IOSettings(this._DeviceParameter.COM.Port, this._DeviceParameter.COM.Baud, DataBits.BIT_8,StopBits.STOP_1, Parity.P_NONE);
    }
    return data;
}
```

以上只是举一下简单的例子，网通通讯模式下也可以重写发送和接收接口实现特殊场景的通讯要求。

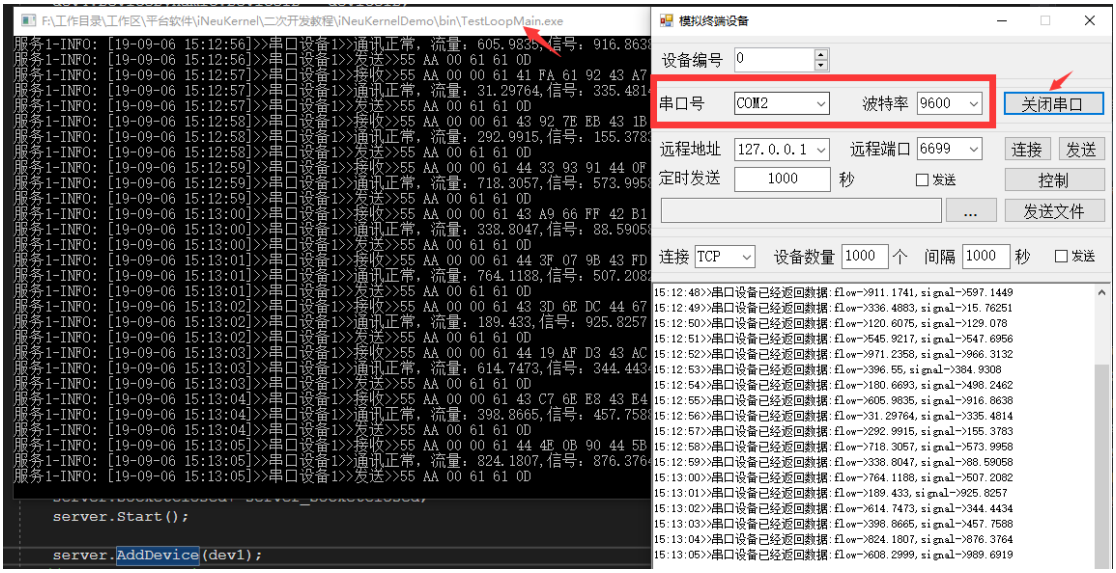
8. 测试驱动

测试前需要安装 Virtual Serial Port Driver 虚拟串口软件，虚拟出来 COM1 和 COM2 两个串口，之间进行数据交互。如下图：



以“iNeuKernelDemo.sln”工程为例，“TestLoopMain”项目作为 iNeuKernel 宿主程序，承载设备驱动的加载、运行；“TestDeviceDriver”项目是基于 iNeuKernel 二次开发的设备驱动，负责与物理设备进行数据交互；“TestDevice”项目模拟现实中的物理设备与 TestDeviceDriver 驱动进行交互。

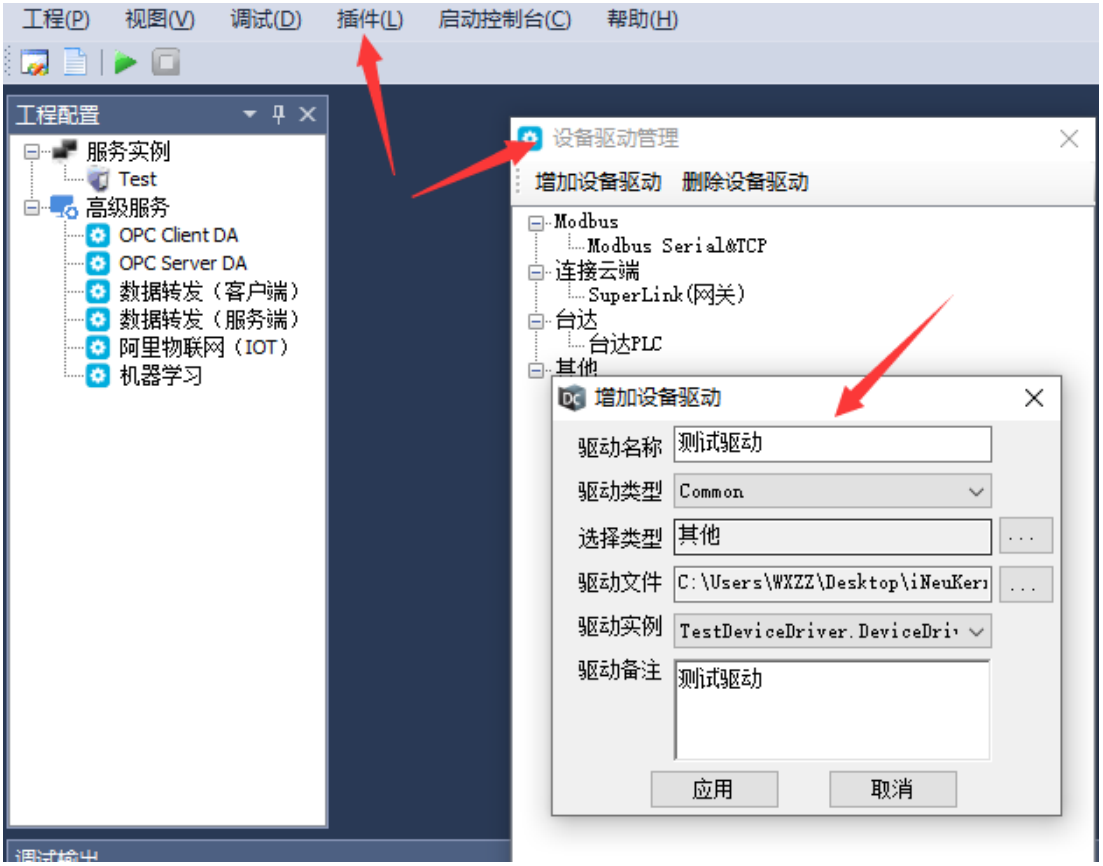
运行“TestLoopMain”项目和“TestDevice”项目，可以增加断点进行调试，设置好参数可以看到通讯效果。如下图：



9. 挂载驱动

运行“iNeuKernel.Designer.exe”工具，将来会通过 iNeuOS 的设备容器进

行统一管理，【插件】->【设备驱动管理】可以挂载刚才开发好的设备驱动，之后就可以在服务实例下运行了。如下图：



10. 运行驱动

详细过程参见：

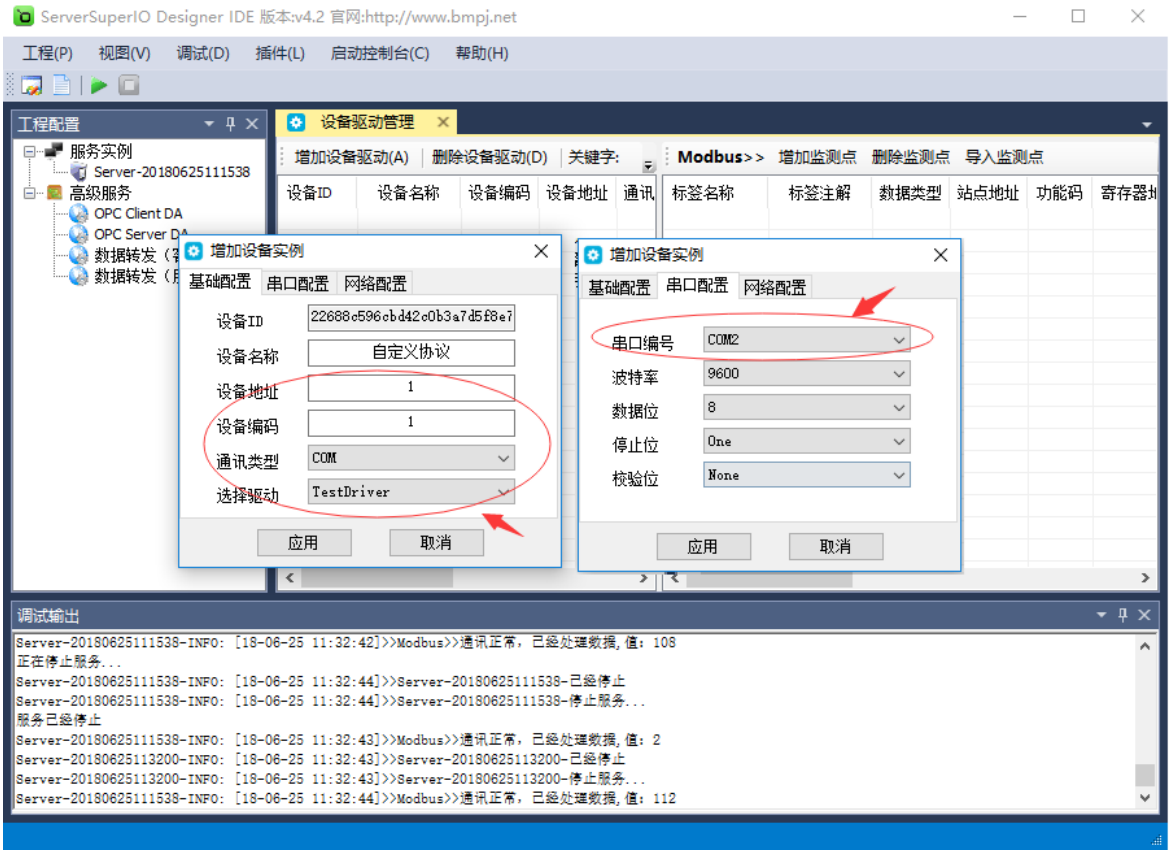
标准 Modbus 和非标准协议的使用、测试以及驱动开发

https://www.cnblogs.com/lsjwq/p/9225566.html

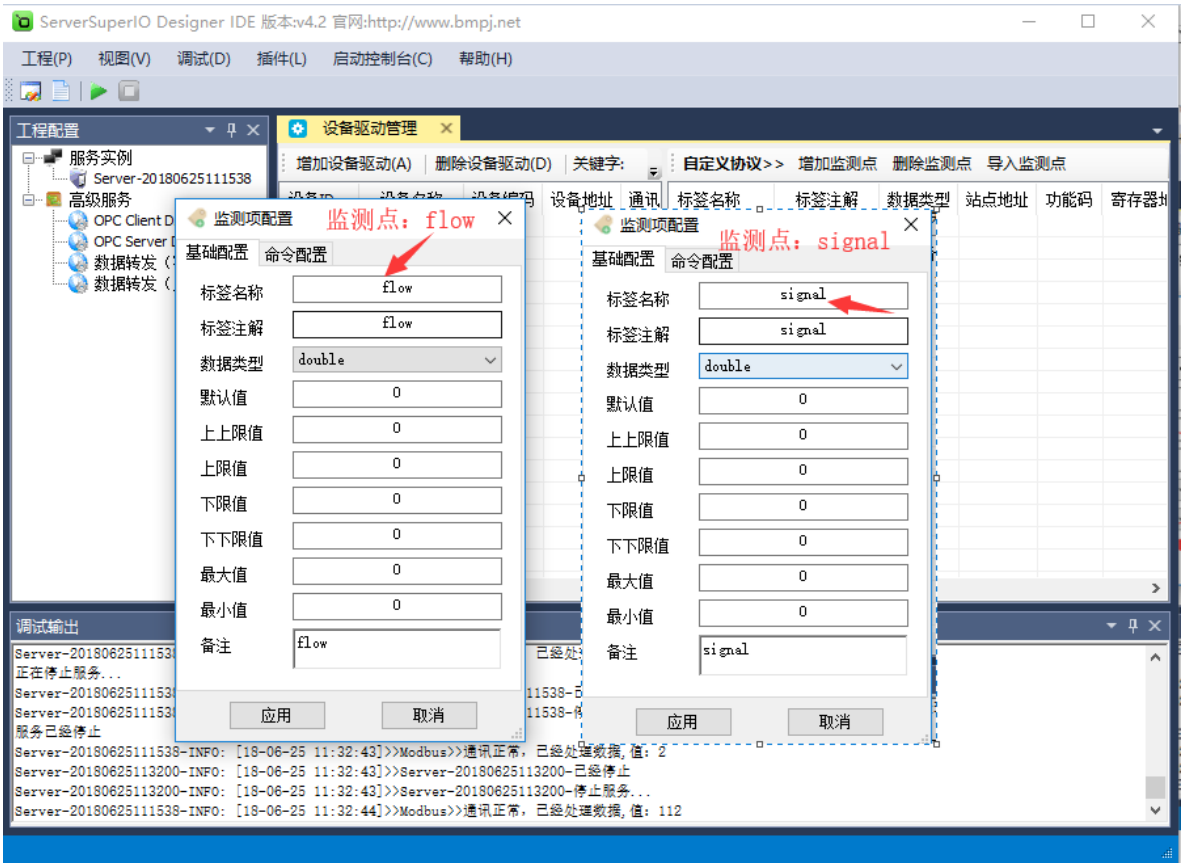
非标准协议的使用和测试与 Modbus 的操作过程一致，但是选择增加设备驱动和配置增加监测点要根据实际情况而定。

（1）配置设备驱动和监测点

步骤一：增加设备驱动，如下图：

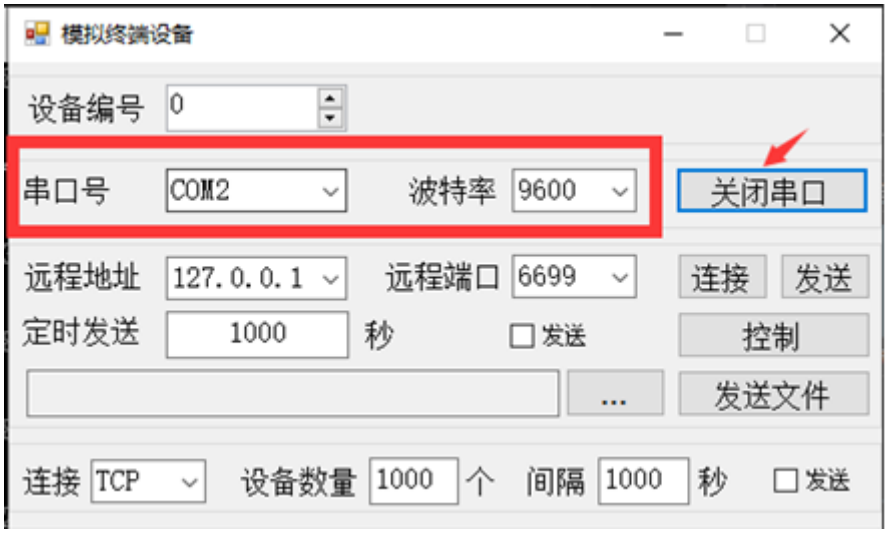


步骤二：增加监测点，如下图：

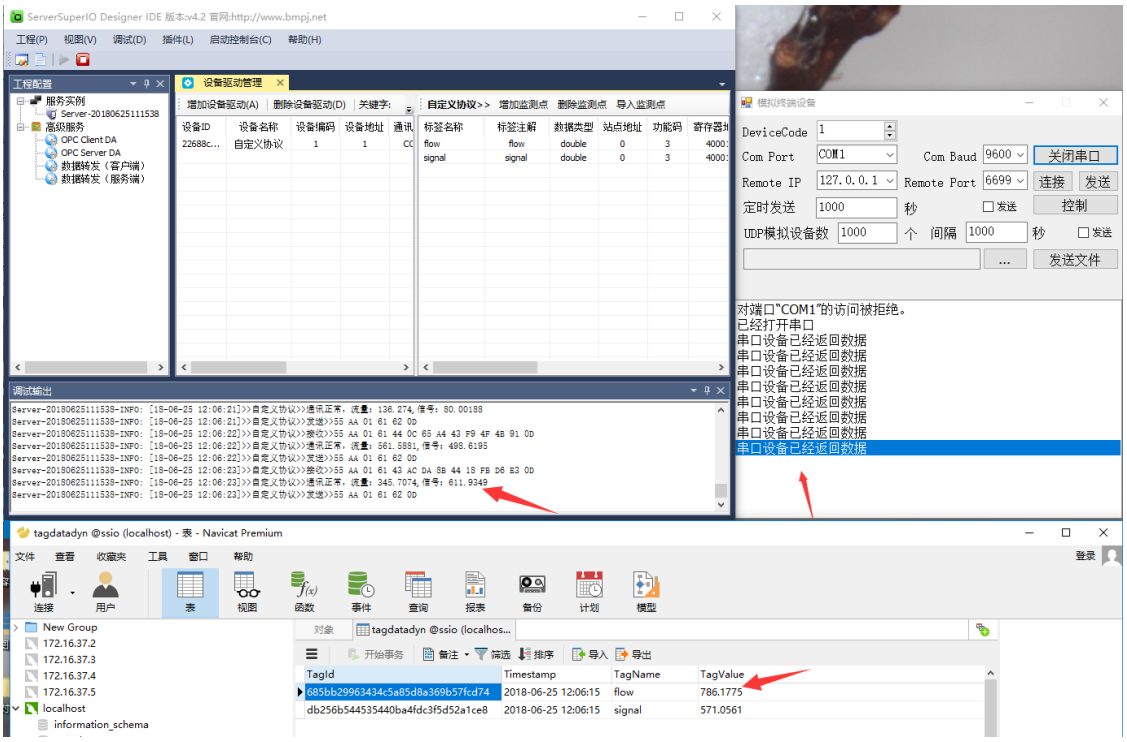


（2）启动模拟终端

在测试工具目录中运行【TestDevice（模拟终端设备）.exe】程序，并且配置串口参数，如下图：



（3）选择【调试】->【运行】，实现运行效果，如下图：


ServerSuperIO Designer IDE 版本v4.2 官网:http://www.bmpj.net
工程(P) 视图(V) 调试(D) 插件(L) 启动控制台(C) 帮助(H)
工程配置
服务实例
Server-20180625111538
高级服务
OPC Client DA
OPC Server DA
数据转发（客户端）
数据转发（服务端）
设备驱动管理
增加设备驱动(A) 删除设备驱动(D) 关键字:
设备ID 设备名称 设备编码 设备地址 通讯
22688c... 自定义协议 1 1 CC
自定义协议>> 增加监测点 删除监测点 导入监测点
标签名称 标签注解 数据类型 站点地址 功能码 寄存器
flow flow double 0 3 4000
signal signal double 0 3 4000
调试输出
Server-20180625111538-INFO: [18-06-25 12:06:21]>>自定义协议>>通讯正常，流量：136.274，信号：80.00188
Server-20180625111538-INFO: [18-06-25 12:06:21]>>自定义协议>>发送>>55 AA 01 61 62 0D
Server-20180625111538-INFO: [18-06-25 12:06:22]>>自定义协议>>接收>>55 AA 01 61 44 0C 65 A4 43 F9 4F 4B 91 0D
Server-20180625111538-INFO: [18-06-25 12:06:22]>>自定义协议>>通讯正常，流量：561.5881，信号：498.6195
Server-20180625111538-INFO: [18-06-25 12:06:22]>>自定义协议>>发送>>55 AA 01 61 62 0D
Server-20180625111538-INFO: [18-06-25 12:06:23]>>自定义协议>>接收>>55 AA 01 61 43 AC DA 8B 44 18 FB D6 B3 0D
Server-20180625111538-INFO: [18-06-25 12:06:23]>>自定义协议>>通讯正常，流量：345.7074，信号：611.9349
Server-20180625111538-INFO: [18-06-25 12:06:23]>>自定义协议>>发送>>55 AA 01 61 62 0D
模拟终端设备
DeviceCode 1
Com Port COM1
Com Baud 9600
关闭串口
Remote IP 127.0.0.1
Remote Port 6699
连接
发送
定时发送 1000 秒
发送
控制
UDP模拟设备数 1000 个 间隔 1000 秒
发送
...
发送文件
对端口"COM1"的访问被拒绝。
已经打开串口
串口设备已经返回数据
串口设备已经返回数据
串口设备已经返回数据
串口设备已经返回数据
串口设备已经返回数据
串口设备已经返回数据
串口设备已经返回数据
串口设备已经返回数据
tagdatadyn @ssio (localhost) - 表 - Navicat Premium
文件 查看 收藏夹 工具 窗口 帮助
登录
连接 用户 表 视图 函数 事件 查询 报表 备份 计划 模型
New Group
172.16.37.2
172.16.37.3
172.16.37.4
172.16.37.5
localhost
information_schema
对象
tagdatadyn @ssio (localhos...
开始事务 备注 筛选 排序 导入 导出
TagId Timestamp TagName TagValue
685bb29963434c5a85d8a369b57fcd74 2018-06-25 12:06:15 flow 786.1775
db256b544535440ba4fdc3f5d52a1ce8 2018-06-25 12:06:15 signal 571.0561