目　　录

7.自控通讯模式开发及注意事项 2
7.1　概述 2
7.2　通讯机制说明 2
7.3　设备驱动开发注意事项 3
7.3.1　实时发送数据 3
7.3.2　发送固定实时请求数据命令 4
7.3.3　优先发送其他数据 4
7.3.4　如何选择 IO 通道发送数据 5
7.3.5　如何以 DeviceCode 分配数据 5
7.3.6　如何改变设备驱动的状态 6
7.4　宿主程序服务实例配置注意事项 6
7.5　自控模式运行效果 8

官方网站：http://www.bmpj.net

7.自控通讯模式开发及注意事项

7.1 概述

自控通讯模式与并发通讯模式类似，唯一的区别是发送请求数据命令，自控通讯模式可以使用定时器，定时发送请求数据命令，不再像并发通讯模式集中发送。

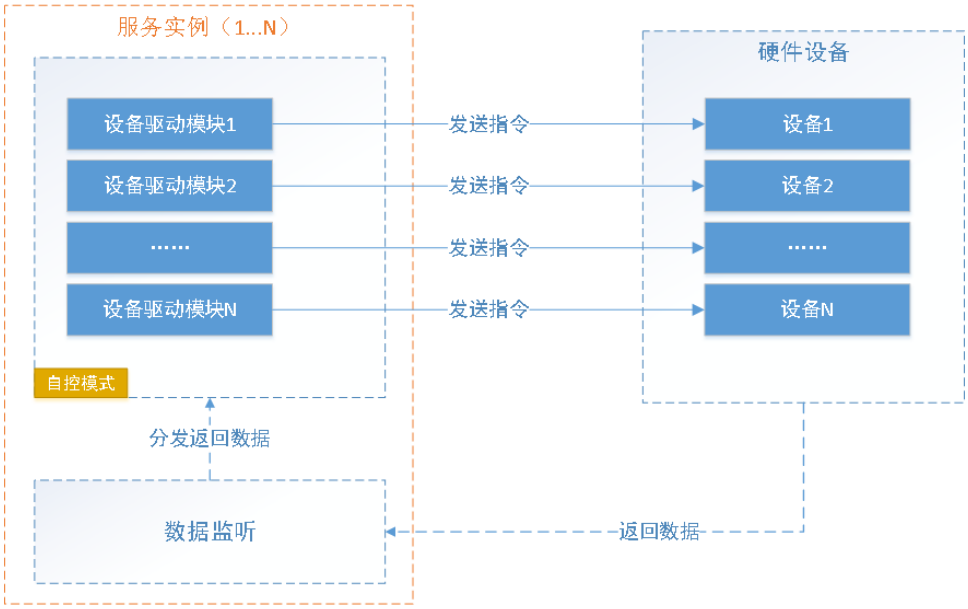
在工业物联网建设中，设备不同、协议不同、场景不同，对于某些不同的设备定时采集数据的频率也不一样，过于高频的数据采集也是对资源的一种浪费，所以就供给二次开发者在开发设备驱动的时候更自主的控制模式。

7.2 通讯机制说明

只有网络通讯时可以使用这种控制模式。自控通讯模式与并发通讯模式类似，区别在于发送指令操作交给设备驱动本身进行控制，或者说交给二次开发者，二次开发者可以通过时钟定时用事件驱动的方式发送指令数据。硬件设 备接收到指令后进行校验，校验成功后返回对应指令的数据，通讯平台异步监听到数据信息后，进行接收操作，然后再进行数据的分发、处理等。

自控通讯模式可以为二次开发者提供精确的定时请求实时数据机制，使通讯机制更灵活、自主，如果多个设备驱动共享使用同一个 IO 通道的话，时间控制会有偏差。

同样涉及到数据的分发，和并发模式一样。通讯结构如下图：



7.3 设备驱动开发注意事项

7.3.1 实时发送数据

ServerSuperIO 框架的 IRunDevice 驱动接口有一个 GetSendBytes 函数，此函数接口会同时协调调用 GetConstantCommand 固定请求数据接口和 SendCache 发送数据的缓存器，并设置设备的优先级别进行调度。

可以继承以前写的设备驱动，在此基础上增加定时发送数据的代码。代码如下：

```
public class DeviceSelfDriver:DeviceDriver
    {
        public DeviceSelfDriver() : base()
        {

        }
        public override void Initialize(string devid)
        {
            base.Initialize(devid);

            this.RunTimerInterval = 5000;
            this.IsRunTimer = true;
        }
```

```
        public override void OnRunTimer()
        {
    5       byte[] data = this.GetSendBytes();
            OnSendData(data);
            base.OnRunTimer();
        }
}
```

7.3.2 发送固定实时请求数据命令

自控通讯模式定时发送请求数据命令，同样是以呼叫应答的方式向设备发送请求实时数据命令，对于同一个设备的请求实时数据命令一般相对固定。在调度某一具体设备驱动的时候，会调用固定的调用 IRunDevice 驱动接口的 GetConstantCommand 函数，以获得请求实时数据的命令。代码如下：

```
        public override byte[] GetConstantCommand()
        {
            byte[] data = this.Protocol.DriverPackage<String>("0", "61", null);
            string hexs = BinaryUtil.ByteToHex(data);
            OnDeviceRuningLog("发送>>"+hexs);
            return data;
        }
```

this.Protocol.DriverPackage 驱动调用 61 命令获得要发送的命令，并返回 byte[]数组，ServerSuperIO 获得数据后会自动通过 IO 接口下发命令数据。如果返回 null 类型，系统不进行下发操作。

7.3.3 优先发送其他数据

对于一个设备不可能只有一个读实时数据的命令，可能还存在其他命令进行交互，例如：读参数、实时校准等，这时就需要进行优先级调度发送数据信息。可以通过两种方式让 ServerSuperIO 框架优先调度该设备驱动。

1. 把命令增加发送数据缓存中，框架从缓存中获得数据后会自动删除，代码如下：

```
this.Protocol.SendCache.Add("读参数",readParaBytes);
```

2. 设置设备的优先级别属性，代码如下：

```
this.DevicePriority=DevicePriority.Priority;
```

7.3.4 如何选择 IO 通道发送数据

集中发送数据时，涉及到如何关联设备驱动与 IO 通道，框架会以 DeviceParameter.NET.RemoteIP 设置的终端 IP 参数进行选择 IO 通道发送数据。但是如果终端设备是动态 IP 地址的话，那么 RemoteIP 参数也应该是变动的。这时就需要设置服务实例是以 DeviceCode 的方式分布数据到设备驱动，终端设备先发送简单的验证数据，保证发送的 DeviceCode 与设备驱动的相对应，设备驱动接收到验证数据后需要保存临时的 RemoteIP 信息，这样保证在发送数据的时候参数准确找到要请求数据的 IO 通道到终端设备。

例如下面代码：

```
public override void Communicate(ServerSuperIO.Communicate.IRequestInfo info)
{
            this.DeviceParameter.NET.RemoteIP = info.Channel.Key;
            this.DeviceParameter.Save(this.DeviceParameter);
            ......
}
```

7.3.5 如何以 DeviceCode 分配数据

如果服务实例设置以 DeliveryMode.DeviceCode 模式分配数据，那么就需要在通讯协议接口里实现过滤 DeviceCode 编码的接口。

例如下面的代码：

```
internal class DeviceProtocol:ProtocolDriver
 {
     public override string GetCode(byte[] data)
     {
         byte[] head = new byte[] {0x55, 0xaa};
         int codeIndex = data.Mark(0, data.Length, head);
         if (codeIndex == -1)
         {
             return String.Empty;
         }
         else
         {
```

```
                return data[codeIndex + head.Length].ToString();
            }
        }
}
```

7.3.6 如何改变设备驱动的状态

不像轮询通讯模式，发送数据、接收数据是一个轮回，在接收数据的过程后驱动设备驱动，设备执行整个生命周期的流程，根据接收到的数据，会自动改变设备驱动的状态。

自控通讯模式和并发通讯模式更多强调请求数据的方式不同，那么不能一直发送请求数据命令，而设备状态一直不改变，例如：通讯正常变成了通讯中断、通讯中断变成了通讯正常。这两种通讯模式的发送与接收过程有一个协调机制，发送 3 次请求数据命令，而没有接收到任何数据，会自动调用设备驱动的接口，以驱动设备驱动的整个执行的流程，这样设备的状态会自动发生改变，而不需要二次开发写相应的代码。

7.4 宿主程序服务实例配置注意事项

在宿主程序中创建服务实例的时候，需要把服务实例的配置参数设置为自控通讯模式，并启动服务实例，把实例化的设备驱动增加到该服务实例中。代码如下：

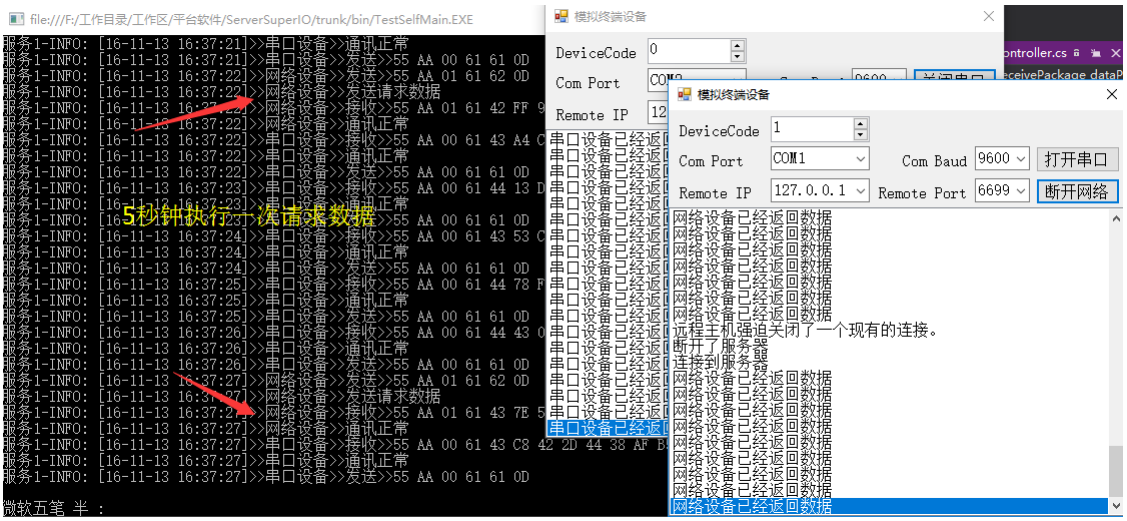
```
static void Main(string[] args)
      {
          DeviceDriver dev1 = new DeviceDriver();
          dev1.DeviceParameter.DeviceName = "串口设备";
          dev1.DeviceParameter.DeviceAddr = 0;
          dev1.DeviceParameter.DeviceID = "0";
          dev1.DeviceDynamic.DeviceID = "0";
          dev1.DeviceParameter.DeviceCode = "0";
          dev1.DeviceParameter.COM.Port = 1;
          dev1.DeviceParameter.COM.Baud = 9600;
          dev1.CommunicateType = CommunicateType.COM;
          dev1.Initialize("0");

          DeviceSelfDriver dev2 = new DeviceSelfDriver();
```

```
            dev2.DeviceParameter.DeviceName = "网络设备";
            dev2.DeviceParameter.DeviceAddr = 1;
            dev2.DeviceParameter.DeviceID = "1";
            dev2.DeviceDynamic.DeviceID = "1";
            dev2.DeviceParameter.DeviceCode = "1";
            dev2.DeviceParameter.NET.RemoteIP = "127.0.0.1";
            dev2.DeviceParameter.NET.RemotePort = 9600;
            dev2.CommunicateType = CommunicateType.NET;
            dev2.Initialize("1");

            IServer server = new ServerManager().CreateServer(new ServerConfig()
            {
                ServerName = "服务1",
                ComReadTimeout = 1000,
                ComWriteTimeout = 1000,
                NetReceiveTimeout = 1000,
                NetSendTimeout = 1000,
                ControlMode = ControlMode.Self,
                SocketMode = SocketMode.Tcp,
                StartReceiveDataFliter = false,
                ClearSocketSession = false,
                StartCheckPackageLength = false,
                CheckSameSocketSession = false,
                DeliveryMode = DeliveryMode.DeviceCode,
            });

            server.AddDeviceCompleted += server_AddDeviceCompleted;
            server.DeleteDeviceCompleted+=server_DeleteDeviceCompleted;
            server.Start();

            //server.AddDevice(dev1);
            server.AddDevice(dev2);

            while ("exit" == Console.ReadLine())
            {
                server.Stop();
            }
        }
```

ControlMode = ControlMode. Self 代码是设置服务实例调度设备为并发控制模式；以 DeliveryMode = DeliveryMode.DeviceCode 方式进行数据分发，当然我现在模拟的是固定的终端 IP。

7.5 自控模式运行效果

1.图片



2.视频

http://v.qq.com/x/page/i0345hhgfrn.html