目　　录

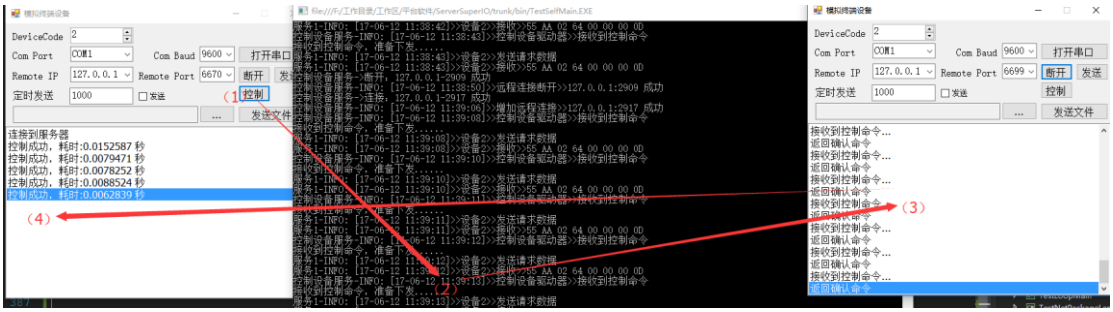
官方网站：http://www.bmpj.net

21. 终端控制传感器或设备，形成回路控制

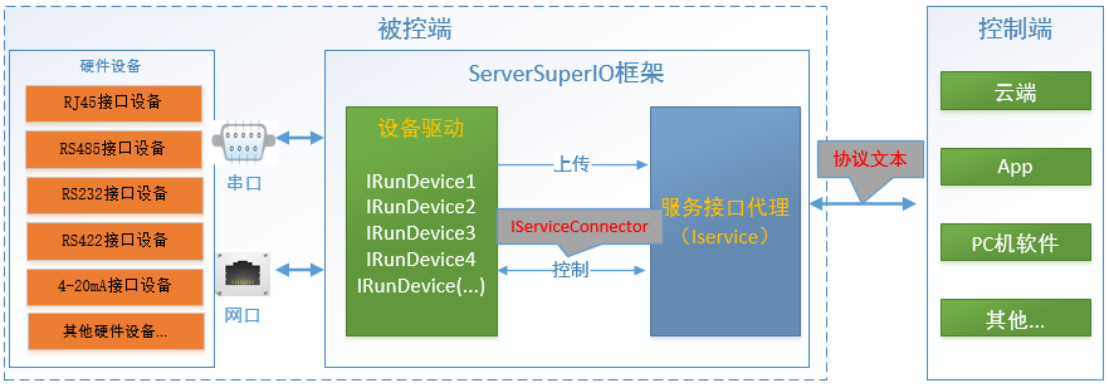
21.1 概述

ServerSuperIO 以前所做的工作逐步为形成回路控制或级联控制打下基础，例如：服务连接器和设备驱动连接器的开发与应用。总之，是通过多种形式下发命令控制设备（驱动）或传感器，云端控制站点或监测点的传感器、App 或者其他终端控制传感器、根据传感器的采集数据控制另一个传感器等。

下面介绍云端、App 或者其他终端如何控制传感器设备（传感器控制传感器类似，请参见：12.服务接口的开发，以及与云端双向交互）。根据通讯协议，结构化方案、不需要太多代码即可完成相应的功能。效果如下图：



21.2 结构示意图



控制端发起控制命令，用 ServerSuperIO 服务接口开发一个简单的代理服务，通过服务连接器 IServiceConnector 接口与设备驱动进行交互，设备驱动接收到控制命令后下发给设备或传感器，等待控制返回的确认消息，再原路返回给控制端。

21.3 通讯协议

有人问为什么不使用 MQTT 协议，那如何兼容不同设备和传感器的协议？以于中国现实情况，显然还不能达到统一标准的水平，在经济不好的情况下，企业也不可能投资替换掉原来的硬件设备。也不符合 ServerSuperIO 设计的原则，就是要搞协议无关性，任何标准或非标准的协议都可以集成进来。如果想过一条河，把桥修好、把索道搭好、把船摆好…具体怎么过河由你自己决定。

有人问 ServerSuperIO 都集成了什么协议？上面已经给出了答案，另外我想说的是没有任何一个框架可以包治百病。从相反的角度来考虑，如果像组态一样把任何协议都加进来，企业又想拿出来多少的价值来对等交换呢，所以协议驱动还是交给大家来自己写吧。

我们演示的协议如下图：

```
发送控制命令：0x55 0xaa addr 0x63 para crc 0x0d
返回控制确认：0x55 0xaa addr 0x64 0x00 crc 0x0d

state：0 成功；1 超时；2 失败；
```

21.4 控制端

控制端包括很多种：云端向下级发送控制命令、App 或 Pc 机软件连接服务发送控制命令等等。发送控制命令如下图：

```
private void button2_Click(object sender, EventArgs e)
{
    _stopwatch.Reset();
    if (_stopwatch.IsRunning)
    {
        _stopwatch.Restart();
    }
    else
    {
        _stopwatch.Start();
    }
    byte[] ctrlCmd = new byte[] { 0x55, 0xaa, byte.Parse(this.numericUpDown1.Value.ToString()), 0x63, 0x00,
      0x00, 0x00, 0x0d }; //CRC没有计算
    _tcpClient.Client.Send(ctrlCmd);
}
```

21.5 代理服务（SSIO 服务接口）

代理服务是通过 ServerSuperIO 的 IService 接口实现，在继承类中使用

ServerSuperIO 框架本身的单例模式开发代理服务，代码如下：

```
public override void StartService()
        {
            string devId = "ControlDeviceService";
            Driver dev = new Driver();
            dev.ReceiveRequestInfos += Dev_ReceiveRequestInfos;
            dev.DeviceParameter.DeviceName = "控制设备驱动器";
            dev.DeviceParameter.DeviceAddr = 0;
            dev.DeviceParameter.DeviceID = devId;
            dev.DeviceParameter.DeviceCode = "";
            dev.DeviceDynamic.DeviceID = devId;
            dev.DeviceParameter.NET.RemoteIP = "127.0.0.1";
            dev.DeviceParameter.NET.RemotePort = 9600;
            dev.DeviceParameter.NET.ControllerGroup = "LocalGroup";
            dev.CommunicateType = CommunicateType.NET;
            dev.Initialize(devId);

            IServer server = new ServerManager().CreateServer(new ServerConfig()
            {
                ServerName = "控制设备服务",
                ListenPort=6670,
                ComReadTimeout = 1000,
                ComWriteTimeout = 1000,
                NetReceiveTimeout = 1000,
                NetSendTimeout = 1000,
                ControlMode = ControlMode.Singleton,
                SocketMode = SocketMode.Tcp,
                StartReceiveDataFliter = false,
                ClearSocketSession = false,
                StartCheckPackageLength = false,
                CheckSameSocketSession = false,
            });

            server.AddDeviceCompleted += server_AddDeviceCompleted;
            server.DeleteDeviceCompleted += server_DeleteDeviceCompleted;
            server.SocketConnected += server_SocketConnected;
            server.SocketClosed += server_SocketClosed;
            server.Start();

            server.AddDevice(dev);
        }
```

dev.ReceiveRequestInfos 事件是控制驱动继承 ServerSuperIO 框架中 RunDevice 驱动类扩展的事件接口，ServerSuperIO 单例模式接收到数据信息，如果符合协议标准会把数据信息反馈给驱动程序的 Communicate 接口，ReceiveRequestInfos 事件把数据信息传递给代理服务订阅该事件的 Dev_ReceiveRequestInfos 函数。代码如下图：

```
public override void Communicate(IRequestInfo info)
{
    OnDeviceRuningLog("接收到控制命令");
    if(ReceiveRequestInfos!=null)
    {
        ReceiveRequestInfos.BeginInvoke(info, null, null);
    }
}
```

代理服务中的 Dev_ReceiveRequestInfos 函数，通过服务连接器接口 IServiceConnector，根据 DeviceCode（addr）把信息传递给相应的设备驱动。代码如下图：

```
private void Dev_ReceiveRequestInfos(IRequestInfo obj)
{
    if(obj!=null)
    {
        byte[] data = obj.Data;

        string devCode = data[2].ToString();
        ISocketSession session = (ISocketSession)obj.Channel;
        string key = String.Format("{0}:{1}", session.RemoteIP, session.RemotePort);

        _channelCache.Add(new ControlChannel(devCode,key,session));

        OnServiceConnector(new FromService(this.ServiceName, ServiceKey, this), new ServiceToDevice(devCode,
          key, data,null));
    }
}
```

代理服务通过 ServiceConnectorCallback 和 ServiceConnectorCallbackError 函数接口接收设备驱动反馈的结果信息，如果中间出现异常会调用 ServiceConnectorCallbackError，如果正常会调用 ServiceConnectorCallback 函数，ServiceConnectorCallback 函数接口根据记录的命令与 IO 通道的对应关系，再把结果发送给控制端。ServiceConnectorCallback 代码如下图：

```
public override void ServiceConnectorCallback(object obj)
{
    string[] arr = (string[])obj;

    ControlChannel channel = _channelCache.FirstOrDefault(c => c.DeviceCode == arr[0] && c.Key == arr[1]);
    if(channel!=null)
    {
        byte[] successCmd = new byte[] { 0x55, 0xaa, byte.Parse(channel.DeviceCode), 0x64, 0x00, 0x00, 0x00,
          0x0d }; //CRC没有计算

        channel.Channel.Write(successCmd);

        _channelCache.Remove(channel);
    }
}
```

在这里边有一个注意的地方，就是设备驱动在规定的时间内没有反馈控制命令的确认信息，也就是传感器没有反馈相应的信息。这种情况要增加一个定时检测服务，如果超时没有反馈信息，发送给控制端相应的消息。代码如下图：

```
private void timer_Elapsed(object sender, System.Timers.ElapsedEventArgs e)
{
    DateTime now = DateTime.Now;
    IEnumerable<ControlChannel> timeoutChannels = _channelCache.Where(c => (now - ((ControlChannel)
      c).ActiveTime).Seconds > 5000);

    if (timeoutChannels.Any())
    {
        System.Threading.Tasks.Parallel.ForEach(timeoutChannels, c =>
        {
            byte[] timeoutCmd = new byte[] { 0x55, 0xaa, byte.Parse(c.DeviceCode), 0x64, 0x01, 0x00, 0x00,
              0x0d }; //CRC没有计算

            c.Channel.Write(timeoutCmd);
        });
    }
}
```

21.6 设备驱动

这个设备驱动与传感器相对应，之间相互过行数据交互。设备驱动的 RunServiceConnector 接口负责接收代理服务 Dev_ReceiveRequestInfos（OnServiceConnector）函数传递过来的命令信息。代码如下图：

```
public override IServiceConnectorCallbackResult RunServiceConnector(IFromService fromService, IServiceToDevice
   toDevice,AsyncServiceConnectorCallback callback)
{
    _serviceCallback = callback;
    _channelKey = toDevice.Text;

    Console.WriteLine("接收到控制命令，准备下发......");

    //this.Protocol.SendCache.Add("0x63", toDevice.DataBytes);
    //this.DevicePriority = DevicePriority.Priority;

    OnSendData(toDevice.DataBytes);

    return new ServiceConnectorCallbackResult(true,this.DeviceParameter.DeviceName+",执行完成");
}
```

有两点说明：1.接收到命令数据后可以通过 OnSendData 函数立即下发数据信息，以设置的 IP 查找相应的 IO 通道，适用于自控模式。2. 接收到命令数据后放到 this.Protocol.SendCache 协议缓存中，等待下发命令，适用于轮询、并发模式。

针对于返回的结果对象 ServiceConnectorCallbackResult 的 isAsyn 参数，如果为 true，说明通过 AsyncServiceConnectorCallback callback 返回结果信息，也就是说要等待传感器返回确认信息，并且设备驱动接收后再反馈到代理服务；如果为 false，说明会立即反馈到代理服务，适用于传递数据信息而不管与传感器是否交互成功。

可以在这个函数中把 callback 参数进行临时保存，等待传感器返回确认信息后在 Communicate 函数中触发异步回调到代理服务。代码如下图：

```
else if(cr==CommandArray.BackControlCommand)
{
    if(_serviceCallback!=null)
    {
        _serviceCallback.BeginInvoke(new string[] { this.DeviceParameter.DeviceCode, _channelKey },
          null, null);
    }
}
```

21.7 Demo 说明

打开两个 TestDevice 程序，一个作为设备传感器，一个作为控制端，DeviceCode 要以应；TestDeviceDriver 是设备驱动，在服务实例中加载，我用的是自控模式，使用 TestSelfMain 项目；ControlDeviceService 是代理服务，在 TestSelfMain 中加载。具体参见工程代码：**http://pan.baidu.com/s/1c1ZZLOO**。

备注：将来我们的大数据平台，也可以通这种模式下发控制命令到站点。