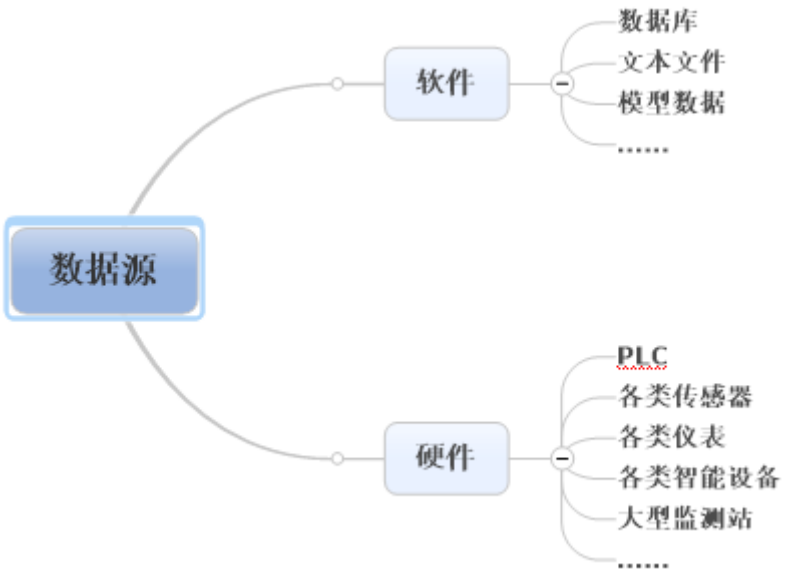
目　　录

官方网站：http://www.bmpj.net

10．持续传输大块数据流的两种方式（如：文件）

10.1 概述

以现在物联网的现状或是对物联网的认知，特别是工业物联网，必须具备集成多种数据源的能力。数据源大体分两类：硬件产生和软件产生。如下图：

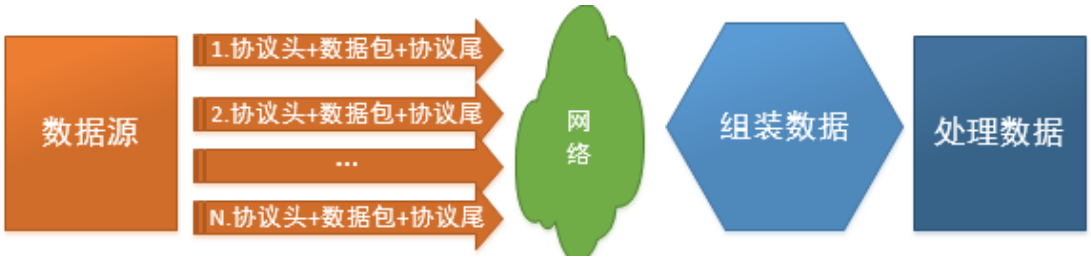


基于现实情况，作为物联网框架必须具备各类数据的集成能力，以及各种应用场景。以数据大小为例，小到一次接收缓存承载能力范围内的数据，大到超出一次接收缓存承载能力范围的数据，只要网络允许，都有可能。以前的连载文章都是以小的数据包为基础介绍的，这篇文章介绍大块数据流的传输方式。

10.2 大块数据流的两种传输方式

10.2.1 协议数据包的方式

这种方式是规定好数据包协议的开头和结尾，把大块数据分解成一定长度的小数据包，以协议头+小数据包+协议尾的组合方式分批次进行数据传输。接收到每个批次的数据后，再进行数据校验，拼装数据，还原出完整的数据。示意图如下：



这种方式存在以下几个问题：

（1） 每个包的数据出现问题后，要进行数据补发。要设计好协议，完成补发机制。

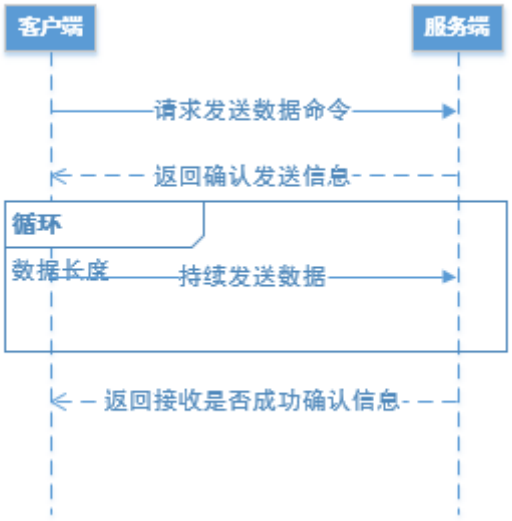
（2） 数据源是多种多样的，例如：压缩文件、序列化的文件、加密的文件等等，那么就存在每个小数据包的数据有可能会和协议头或协议尾一致，甚至和 CRC 校验一致的情况，从而导致数据无法正常校验和解析，这时进行补发数据，可能出现同类情况是大概率事件。

选择这种传输大块数据流的方式，要根据现场的实际情况进行选择，规避可能出现的风险，提高项目、产品整体的稳定性。

如果选择这种方式，那么根据前面介绍的文章，就可以实现，网友可以自己动手实现。这篇文章主要介绍下面这种方式。

10.2.2 请求长度、确认的方式

客户端先发送请求发送数据的命令，并且在命令标识本次要发送数据的长度。如果服务端接收到该请求命令后，根据判断请求数据长度是否在允许范围内，然后返回相同命令数据或其他确认数据给客户端，标识是否允许发送该长度的数据信息。如果可以发送，那么客户端则持续发送数据流，服务端也进行持续接收阶段。示意图如下：



针对这种数据传输的方式，ServerSuperIO 专门提供了接口。下面进行详细的介绍。

10.3 实现持续传输大块数据

10.3.1 设计请求发送数据协议

请求发送 0x62 指令，共 10 个字节，校验和为从“从机地址”开始的累加和，不包括“数据报头”、“校验和”和“协议结束”。

请求指令数据帧如下：

帧结构	数据报头		从机地址	令代码指	数据长度（整型）	校验和	协议结束
	0x55	0xAA		0x61			0x0D
字节数	1	1	1	1	4	1	1

服务端接收到该请求命令后，返回同样的命令信息给客户端，客户端则进入持续发送数据的状态。

10.3.2 客户端代码实现

先发送请求数据命令，代码如下：

```
private void btSendFile_Click(object sender, EventArgs e)
{
            try
            {
                if (this._tcpClient == null)
                {
                    return;
                }

                if (!File.Exists(this.txtFilePath.Text))
                {
                    WriteLog("请选择文件");
                    return;
                }

                byte[] backData = new byte[10];
                backData[0] = 0x55;
                backData[1] = 0xaa;//协议头
                backData[2] = byte.Parse(this.numericUpDown1.Value.ToString());//从机地
址
                backData[3] = 0x62;//命令

                int count=(int)(new FileInfo(this.txtFilePath.Text)).Length;
                byte[] countBytes = BitConverter.GetBytes(count);

                backData[4] = countBytes[0];
                backData[5] = countBytes[1];
                backData[6] = countBytes[2];
                backData[7] = countBytes[3];

                byte[] checkSum = new byte[6];
                Buffer.BlockCopy(backData, 2, checkSum, 0, checkSum.Length);

                backData[8] = (byte)checkSum.Sum(b => b);//计算校验和

                backData[9] = 0x0d;

                this._tcpClient.Client.Send(backData, 0, backData.Length,
SocketFlags.None);
            }
            catch (SocketException ex)
            {
                Disconnect();
```

```
                WriteLog(ex.Message);
            }
}
```

接收到服务端的确认信息后，持久发送数据的代码如下：

```
private void SendFile()
        {
            FileStream fs = null;
            try
            {
                if (this._tcpClient == null)
                {
                    return;
                }

                string fileName = this.txtFilePath.Text;
                if (!File.Exists(fileName))
                {
                    WriteLog("请选择文件");
                    return;
                }
                WriteLog("开始传输>>");
                byte[] sendBuffer = new byte[1024];
                fs = new FileStream(fileName, FileMode.Open, FileAccess.Read,
FileShare.Read);

                long length = fs.Length;
                int count = 0;
                Stopwatch watch = new Stopwatch();
                watch.Start();
                while (length > 0)
                {
                    int sendNum = fs.Read(sendBuffer, 0, sendBuffer.Length);

                    sendNum = _tcpClient.Client.Send(sendBuffer, 0, sendNum,
SocketFlags.None);

                    length -= sendNum;
                    count += sendNum;

                    float percent = ((fs.Length - length) / (float)fs.Length) * 100.0f;
                    WriteLog("已传:" + percent.ToString("0.00") + "%");
                }
```

```
                watch.Stop();

                WriteLog("传输完毕!总数:" + count.ToString() + ",耗时:" +
watch.Elapsed.TotalSeconds.ToString(CultureInfo.InvariantCulture));
            }
            catch (SocketException ex)
            {
                this.Disconnect();
                WriteLog(ex.Message);
            }
            catch (Exception ex)
            {
                WriteLog(ex.Message);
            }
            finally
            {
                if (fs != null)
                {
                    fs.Close();
                    fs.Dispose();
                }
            }
        }
```

10.3.3 ServerSuperIO 框架的实现及注意事项

客户端的代码实现基本上没有什么好讲的，主要是介绍基于 ServerSuperIO 框架，以设备驱动的方式是怎么实现的。注：以下自控模式实现。

1. 协议接口的实现

DeviceProtocol:ProtocolDriver 接口有一个 GetPackageLength(byte[] data, IChannel channel, ref int readTimeout)函数接口，data 参数是请求发送数据的命令，channel 参数是当前 IO 通道的实例，readTimeout 是自定义返回接收数据长度所要使用的时间，如果返回值为 0 的话，则认为不进入持续接收数据任务。可以通过 channel 参数直接返回确认信息，具体代码如下：

```
public override int GetPackageLength(byte[] data, IChannel channel, ref int readTimeout)
        {
            if (data == null || data.Length <= 0)
                return 0;
```

```
            readTimeout = 2000;

            if (CheckData(data))
            {
                try
                {
                    if (data[3] == 0x62) //发送文件请求
                    {
                        int length = BitConverter.ToInt32(new byte[] {data[4], data[5],
data[6], data[7]}, 0);

                        if (length <= 1024*1024) //1M
                        {
                            int num = channel.Write(data);
                            if (num > 0)
                            {
                                Console.WriteLine("返回文件请求确认数据");
                                return length;
                            }
                            else
                            {
                                return 0;
                            }
                        }
                        else
                        {
                            return 0;
                        }
                    }
                    else
                    {
                        return 0;
                    }
                }
                catch (Exception)
                {

                    return 0;
                }
            }
            else
            {
```

```
                Console.WriteLine("校验错误");
                return 0;
            }
        }
```

2. 协议命令的实现

为了实现对大块数据的处理，专门增加一个协议命令，用于解析、保存数据。代码如下：

```
internal class DeviceFileCommand:ProtocolCommand
    {
        public override string Name
        {
            get { return CommandArray.FileData.ToString(); }
        }

        public override dynamic Analysis<T>(byte[] data, T t)
        {
            if (t != null)
            {
                string path = AppDomain.CurrentDomain.BaseDirectory +
DateTime.Now.ToString("yyyyMMddHHmmss") + ".txt";
                File.WriteAllBytes(path, t as byte[]);
                return path;
            }
            else
            {
                return null;
            }
        }

    }
```

3. 设备驱动调用协议，并驱动协议命令

在接收大块数据流的时候，会把所有数据信息返回到设备驱动的 Communicate 接口，其中 info 参数的 Data 是当前请求数据的命令，BigData 就是持续接收数据的信息，通过调用 this.Protocol.DriverAnalysis 协议接口驱动协议命令 DeviceFileCommand。具体代码如下：

```
public override void Communicate(ServerSuperIO.Communicate.IRequestInfo info)
        {
            string hexs = BinaryUtil.ByteToHex(info.Data);
```

```
            OnDeviceRuningLog("接收>>" + hexs);

            byte[] cmds = this.Protocol.GetCommand(info.Data);
            CommandArray cr = (CommandArray)cmds[0];

            dynamic obj = this.Protocol.DriverAnalysis<byte[]>(cr.ToString(), info.Data,
info.BigData);
            if (obj != null)
            {
                if (cr == CommandArray.RealTimeData)
                {
                    _deviceDyn.Dyn = (Dyn)obj;
                }
                else if (cr == CommandArray.FileData)
                {
                    OnDeviceRuningLog("文件存储路径：" + obj.ToString());
                }
            }

            OnDeviceRuningLog("通讯正常");
        }
```

4. 宿主程序服务实例配置注意事项

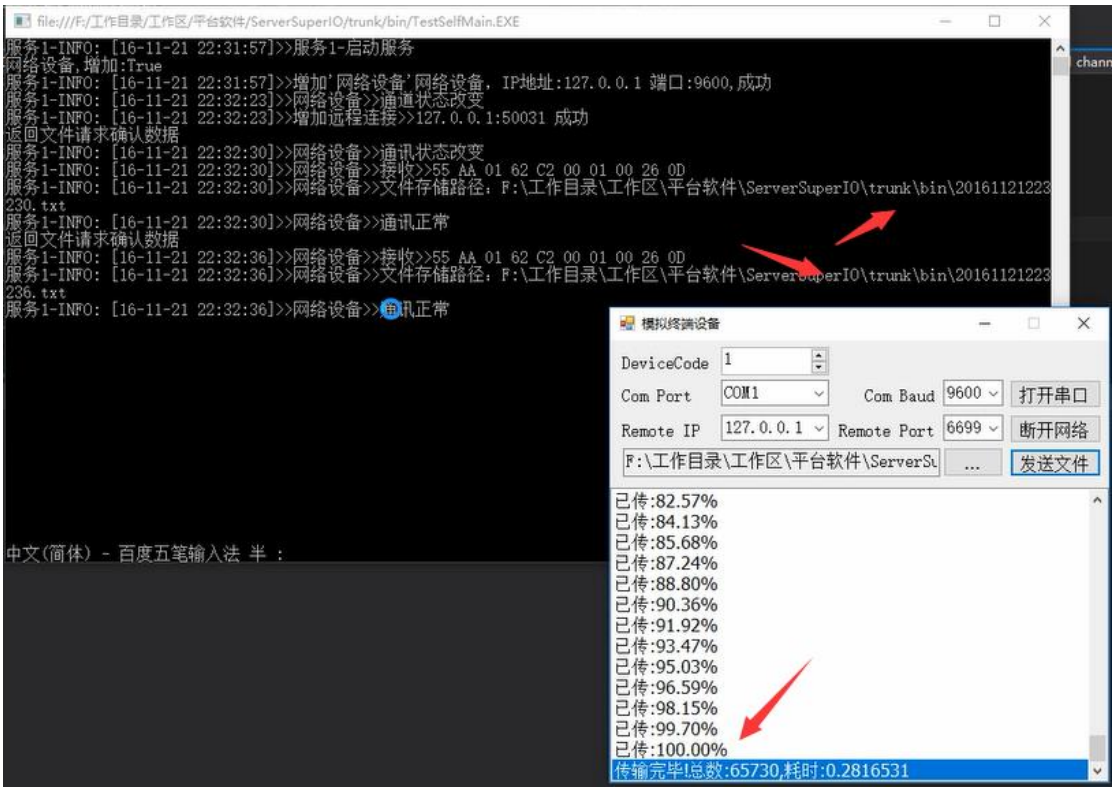
主要在配置参数中配置 StartCheckPackageLength = true,在接数据的过程中会检测相应设备驱动的协议接口 GetPackageLength。

```
static void Main(string[] args)
        {
            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 = true,
            ClearSocketSession = false,
            StartCheckPackageLength = true,
            CheckSameSocketSession = false,
            DeliveryMode = DeliveryMode.DeviceIP,
        });

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

        server.AddDevice(dev2);

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

10.4 运行效果

1. 图片



2. 视频

https://imgcache.qq.com/tencentvideo_v1/playerv3/TPout.swf?max_age=86400&v=20161117&vid=t0348kct9a4&auto=0