目　　录

官方网站：http://www.bmpj.net

1.特点和通讯机制

1.1 特点

1) 轻型高性能通信框架，适用于多种应用场，轮询模式、自控模式、并发模式和单例模式。
2) 设备驱动、IO 通道、控制模式场景协调统一。
3) 设备驱动内轩命令驱动器、命令缓存器、自定义参数和实时数据元素。
4) 框架平台支持按设备命令优先级别进行调度，保证高级别命令及时发送。
5) 一个设备驱动同时支持串口和网络两种通讯方式，可以监视 IO 通道数据。
6) 一个设备驱动，在网络通讯时可以支持 TCP Server 和 TCP Client 两种工作模式。
7) 内置显示视图接口，满足不同显示需求。
8) 内置服务组件接口，可以自定义完成 OPC 服务、4-20mA 输出、LED 大屏显示、短信服务、以及多功能网关服务。
9) 可以创建多服务实例，完成不同业务的拆分。
10) 支持跨平台部署，可以运行在 Linux 和 Windows 系统。

1.2 概述

ServerSuperIO 通信框架的设计思想是在 SuperIO（SIO）基础上发展而来，并没有高大上的技术，主要是工作经验的积累，适合于不同应用场景的物联网的数据采集与交互。ServerSuperIO 和 SuperIO 并不是简单的对 IO 高性能的操作，而是设备驱动、IO 通道、控制模式和实际硬件设备之间的协调机制，各方面之间无缝衔接和运行，也是为了解决现实工作和应用场景的一些痛点。

软硬件之间的数据交互，并且面临着复杂的现场环境：

（1）复杂的、多样的通讯协议。有标准的协议，例如：Modbus 等，也有很多根据标准协议修改的协议格式、以及自定义协议格式，并且千差万别。对于不好的软件架构，疲于应对，增加设备或协议要对整个软件进行梳理，往往在此过程中出现新的问题或 BUG。

（2）针对不同用户对软件界面或功能的要求有很大不同，使之满足不同用户的显示要求，可以自定义数据显示界面。那么就需要提供显示视图接口，与设备驱动进行交互。

（3）既然现场设备的数据被采集上来，那么就需要对其进行处理，不仅仅是保存、查询、报表等，还有：数据转发、数据输出（OPC、模拟量、大屏等）等。那么就需要提供服务性的接口，与设备驱动进行交互。

（4）通讯链路的多种性，对于同一个设备可能要支持 RS232/RS485/RS422、RJ45、3G/4G 等通讯方式，所以对于一个设备要对应多种通讯方式（串口和网络），也给我们的开发造成很大的障碍。

（5）设备驱动、IO 通道和实际的现场硬件终端之间链路复杂，有可能：一个设备驱动对应一个 IO 通道、一个设备驱动对应多个 IO 通道、多个设备驱动对应一个 IO 通道等情况。

（6）既然设备与服务端进行数据交互，那么就应该对设备的通讯状态、IO 状态、以及设备本身的状态进行监控，这样设备才处于可维护状态。

（7）软件各版本、以及软件与硬件之间的兼容性很差，管理起来错综复杂。在框架平台稳定的情况下，只需要更新设备驱动。

为了解决以上诸多问题，开发一个软件框架，支持二次开发。在不对软件框架改动的情况下，能够很方便的接入设备、维护设备、集成设备、处理设备业务数据等。软件框架相对稳定，把容易变化的部分进行灵活设计。

有人问，你这个框架和 SuperSocket、netty.....有什么区别？ServerSuperIO 是通讯框架不？是；ServerSuperIO 支持高并发不？理论上支持；ServerSuperIO 支持跨平台不？在 Ubuntu 上跑过。但是这些并不是 ServerSuperIO 起初设计的初发点，它继承了 SuperIO 的设计思想，后期才逐步的向服务端发展，加强通讯能力、跨平台等等。

ServerSuperIO 是一个物联网框架，首先是以设备（传感器）为核心构建的框架，设备（传感器）的协议无关性，可以随意挂载设备驱动在框架下运行。所以 ServerSuperIO 本质上协调设备驱动（协议）、IO 通道（COM 和 NET）、运行机制（模式）之间的关系，使之无缝结合、运行。

一直在工业领域混，做集成系统、远程监测监控等等，所以 ServerSuperIO 不仅仅是一个通讯框架，更多的是结合了工作实践经验，本着能够解决实质问题。

1.3 ServeSuperIO 与 SuperIO 的区别

序号	属性	SSIO	SIO
1	应用场景	适用于高频的数据采集与控制，可以部署在服务器端。	适用于一般性的上位机数据采集，例如：局域网内的厂级服务端应用。
2	控制模式	轮询模式、自控模式、并发模式、单例模式	轮询模式、自控模式、并发模式、
3	性能	高性能	性能不如SSIO
4	服务实例	一个进程可以创建多个服务实例	一个进程只能创建一个服务实例
5	跨平台	支持Linux和Windows	只支持Windows各版本操作系统
6	二次开发	方便（不包括界面）	只需要继承就可以创建一个完整的应用程序
7	代码结构	更合适	使用的单例模式较多
8	串口组件	SerialPort	PCOMM
9	网络组件	SocketAsyncEventArgs	Socket
10	开源	开源	没有开源
11	OPC	不支持	支持
12	模拟量	不支持	支持
13	插件	需要自己二次开发	完全支持插件化部署

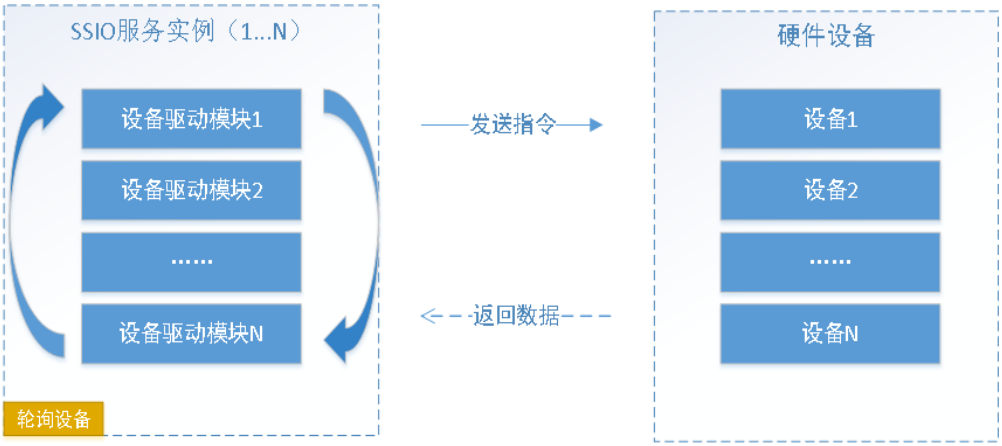
1.4 运行和通讯机制

1.4.1 轮询模式

这是框架最早的运行模式，串口和网络通讯时都可以使用这种控制模式。当有多个设备 连接到通讯平台时，通讯平台会轮询调度设备进行通讯任务。某一时刻只能有一个设备发送请求命令、等待接收返回数据，这个设备完成发送、接收（如果遇到超时 情况，则自动返回）后，下一个设备才进行通讯任务，依次轮询设备。

应用场景是这样的，服务端与设备进行通讯遵循呼叫应答的方式，也就是 IO 可用的情况下，服务端先发起通讯命令请求，设备根据命令信息，检验通过后返回数据给服务端。这种通讯模式很好理解，每个设备的通讯都遵循排队的原则。但是如果某个设备的命令需要及时发送，怎么办？ServerSuperIO 框架是支持设备优先级别调度的，例如：对某个设备要进行实时的检测，需要连续发送命令，那么就需要对设备进行高级别设置，发送请求数据命令。

通讯结构如下图：



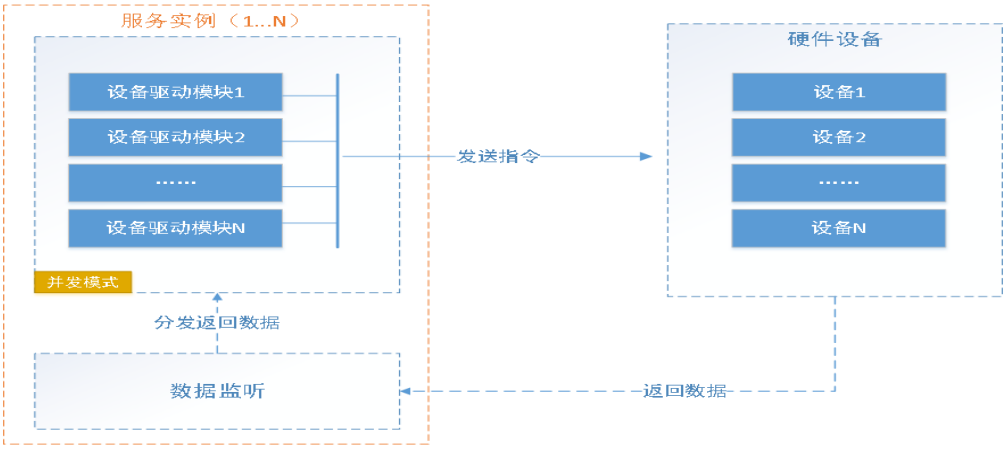
1.4.2 并发模式

网络通讯的情况下，轮询模式显然效率比较低，那么可以采用并发模式。并发通讯模式是集中发送给所有设备请求指令，框架是采用循环同步方式发送请求命令给每个 IO 通道对应的设备，当然也可以采用并行异步方式集中发送请求命令。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。

那么这里就涉及到 IO 通道接收到的数据是异步接收的，如何才能和设备驱动匹配上（把数据分发到设备驱动上），这是能过 DeviceCode 和 DeviceIP 两种方式来实现的。DeviceCode 可以是设备地址或是设备编码，DeviceIP 是预先设置好的参数，要求终端设备的 IP 地址是固定的。

通讯结构如下图：



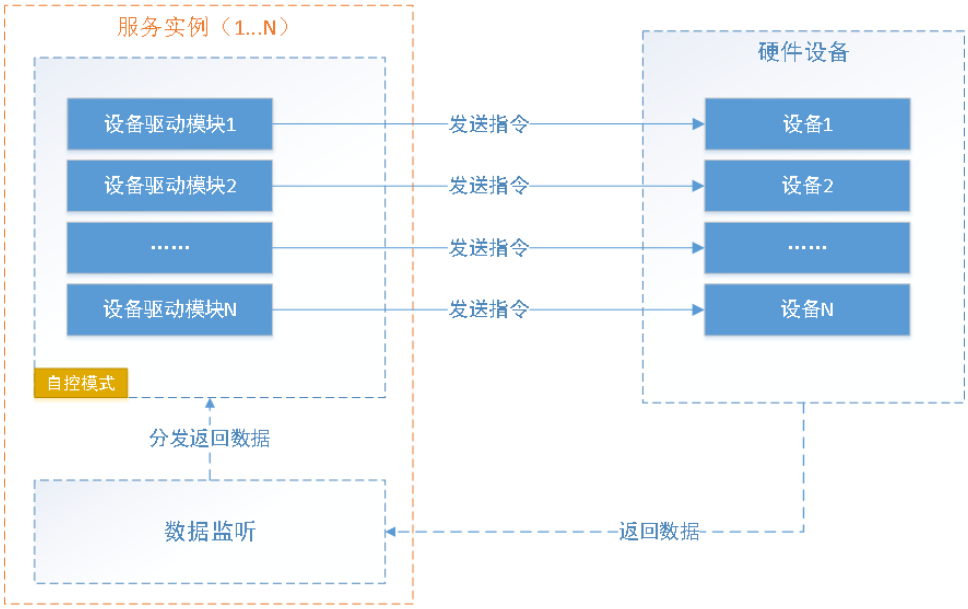
1.4.3 自控模式

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。

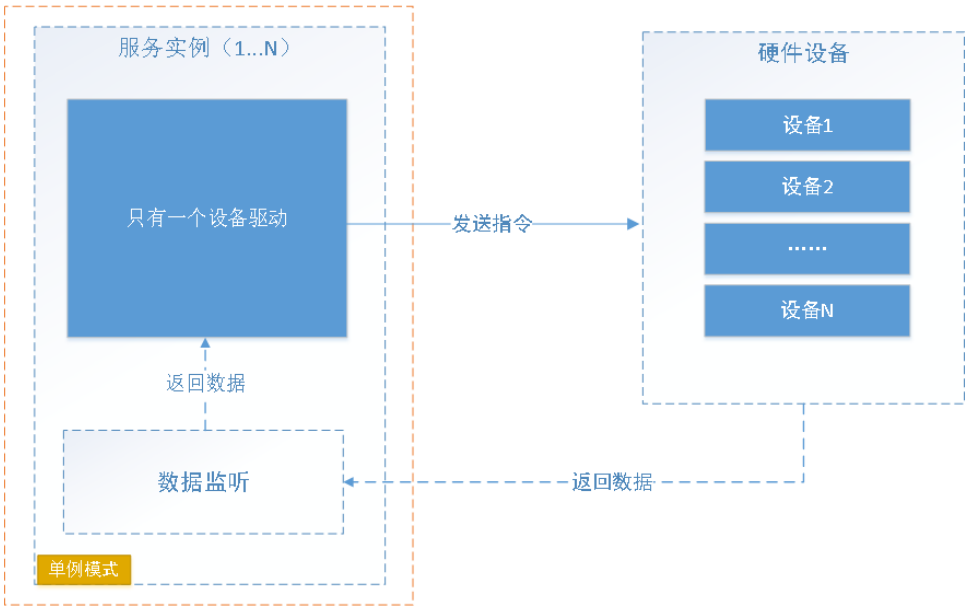
通讯结构如下图：



1.4.4 单例模式

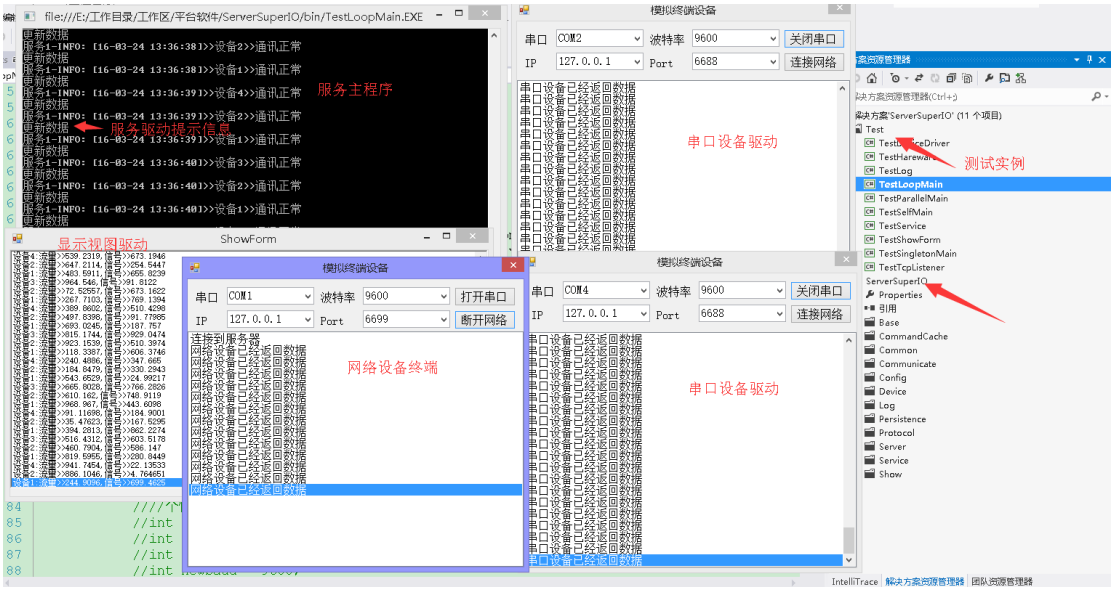
只有网络通讯时可以使用这种控制模式。在一个服务实例内只能有一个设备驱动，相当于一个设备驱动对应着 N 多个硬件设备终端。更适合通讯的数据协议有固定的标准，以命令关键字处理不同的数据。适用于高并发的硬件终端设备主动上传数据，服务器端根据数据信息进行处理和返回相应的数据。

通讯结构如下图：



1.5 跨平台运行

1.5.1 Windows 运行效果



1.5.2 Linux 运行效果