文档名称	YFIOs 技术白皮书		
版本	V1.1.0		
说明	增加 技术特色和优势 章节		
作者	叶帆	日期	2012-12-27
历史	V1.0.0　叶帆 2012-12-20		
文档列表	http://www.sky-walker.com.cn/MFRelease/YF_document_list.pdf		

1 前言

在工控领域，组态软件司空见惯，国外的 iFix、InTouch、WinCC，国内的组态王、力控、MSCG 等等。组态软件的出现彻底解决了软件重复开发的问题，实现模块级复用，好处不仅仅是提高了开发效率，降低了开发周期，更大的优势的是成熟模块的复用，大大提高了系统稳定性和可靠性。

所谓组态（Configuration），就是模块化任意组合（类似积木玩具）。组态软件的主要特点有：

（1）、延展性。所谓延展性，就是系统的延续和易于扩展性，用组态软件开发的系统，当现场或用户需求发生改变时（包括硬件设备或系统结构的改变），用户无需做很多修改，就可以很方便地完成系统的升级和改造；

（2）、易用性。组态软件对底层功能都进行了模块级封装，对于用户，只需掌握简单的编程语言（内嵌的脚本语言，类 Basic 或类 C 语言），甚至不需要编程技术，就能很好地，通过组态配置的方式完成一个复杂系统的开发和集成；

（3）、通用性。不同用户根据系统的不同，利用组态软件提供的 I/O 驱动（如 PLC、仪表、板卡、智能模块、变频器等等驱动）、数据库和图元，就能完成一个具有动画、实时数据处理、历史数据和图表并存，且具有多媒体功能和网络功能的系统工程，不受领域或行业限制。

但是无论是基于 PC 平台的组态软件还是基于 ARM 系统的嵌入式组态软件，其组态粒度都显过大，大部分通过串口、网口、CAN 等通道把个系统模块连接在一起，在一定程度上增加了系统构建的成本和代价。

而以.NET Micro Framework 为依托构建的轻量级嵌入式组态软件（YFIOs）就很好的解决了上述问题，除支持常规的串口、网口、CAN 外，还支持 USB、Wifi、ZigBee、SPI、I2C 等通道，SPI、I2C 片级总线的支持加上强大的托管代码（C#,VB.net）开发能力，使嵌入式硬件系统真正的组态化、模块化成为可能，这项技术的推出，无疑为快速打造形态各异，功能不同的产品提供了最有力的支撑。

2 YFIOs 简介

YFIOs 就是 YFSoft I/O Server 的简称，在物联网、云计算时代，一切以数据为中心，不同的传感器通过不同的方式接入网络，通过云计算的方式为不同的终端用户提供服务。

为了适应这种新形势的发展，加速和降低各种传感器、智能模块的入网代价，以微软成熟的.NET Micro Framework 系统为基础，打造出物联网时代的轻量级嵌入式组态系统 ——

YFIOs。

2.1 技术特色和优势

和传统组态或其他物联网、嵌入式等方案相比，有如下优势：

（1）、组态式搭建系统，自动添加 IO 配置数据，驱动和策略开发接口对外开放；

（2）、支持远程升级，远程调试。

（3）、由于.NET Micro Framework 的跨平台特性，所以基于该框架的 YFIOs 也可以跨平台应用。

（4）、采用 Microsoft Visual Studio 2010 模板进行驱动和策略进行 C#开发，开发门槛较低，和 windows 平台的开发别无二致；

（5）、驱动和策略可以在 Microsoft Visual Studio 2010 开发环境中在线调试；

（6）、策略可以和驱动联动，不仅可以直接调用驱动，还可以和驱动进行关联，事件触发的方式执行策略；

（7）、策略不仅可以调用驱动，彼此之间还可以互相调用；

（8）、驱动和策略可以加密，也可以绑定指定硬件运行，不仅可以保护用户的知识产权，还可以在此基础上为第三方客户提供增值服务。

（9）、运行时小巧轻便，不含 YFHMI 库的运行时仅 19.2K，含 MiniGUI、中文字库、四套图元库的运行时，也仅 355K。

2.2 .NET Micro Framework 简介

Microsoft .NET Micro Framework 将 .NET 的可靠性和效率与 Visual Studio 的高生产率结合起来，以针对价格较低、资源受限的小型设备开发应用程序,可帮助人们使用熟悉的 Visual Studio 工具来构建托管的嵌入式应用程序。2009 年 5 月，.NET Micro Framework 采用 Apache 2.0 license，比 Linux 等开源软件更为彻底的方式实现了源代码完全开放。

2.2.1 哪些领域可以采用.NET Micro Framework 技术？

.NET Micro Framework 技术可以应用到：Sideshow、远程控制、智能家电、教育类机器、医疗电子、零售终端以及汽车电子等行业应用场景；此外由于.NET Micro Framework 集成了各种接口，如串口、网口、Wifi、Zigbee、I2C、SPI、SDIO、USB 等通信接口，加上其应用开发简便，所以在物联网时代，将大有作为。

2.2.2 .NET Micro Framework 与 Window CE 和 Windows XP Embedded 的区别？

.NET Micro Framework 对存储器和处理器的要求更低。开发人员可以在低功耗、低成本的 ARM7、ARM9、Blackfin 和 Cortex-M3 处理器上使用该框架（不需要 MMU 支持），所开发出来的软件仅需要几百 Kbytes 的 RAM 或 Flash/ROM 存储空间。而 Windows Embedded CE 的托管代码环境需要约 10~12Mbytes 的存储空间，基于.NET 的应用编程设备只需要较少的存储

空间，降低了产品成本。

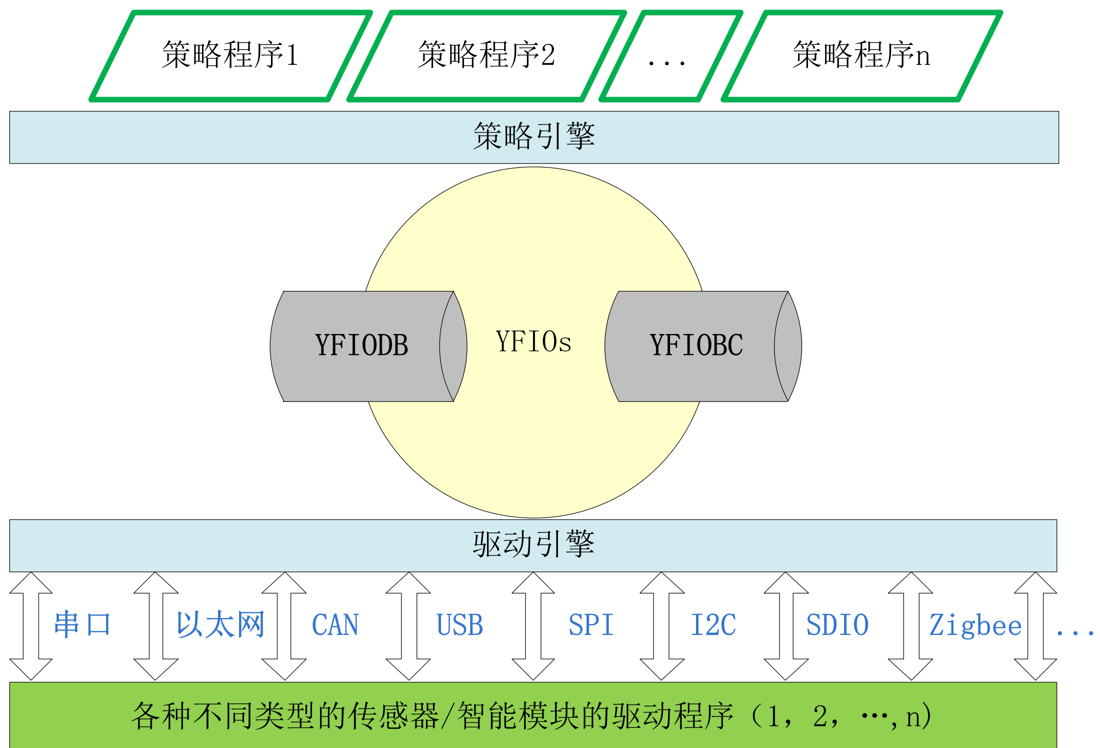
2.2.3 .NET Micro Framework 与其他.NET 平台的区别？

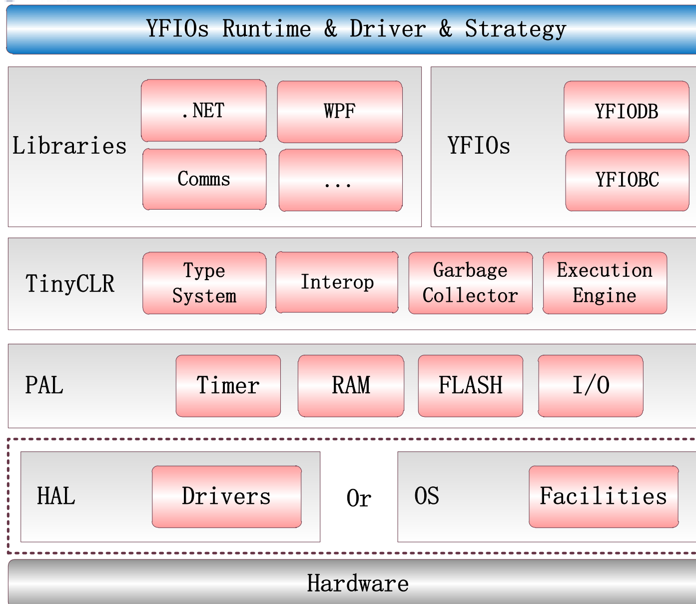
作为.NET 家族的一员，.NET Micro Framework 是微软专门针对超轻量级平台设计的软件架构。与.NET Framework 和.NET Compact Framework 不同的地方是，.NET Micro Framework 具有自启动的特性，并且在 HAL 层，微软将操作系统的必要特性引入，如：启动管理、中断处理、线程调度、内存管理等。.NET Micro Framework 可以单独使用，不需要依托其它操作系统，因此占用空间很小。

2.3 YFIOs 系统架构

YFIOs 由三大部分构成，一是 YFIOs 运行时，包含 YFIODB、YFIOBC、驱动引擎和策略引擎四部分；二是应用模块，包含驱动、策略和 IO 数据三部分；三是 YFIOs IDE 环境（YFIOsManager），该工具和 Microsoft Visual Studio 开发工具一起共同完成驱动、策略的开发、配置及部署工作。

系统架构图和 YFIOs 和.NET Micro Framework 关系图（如下图所示）：

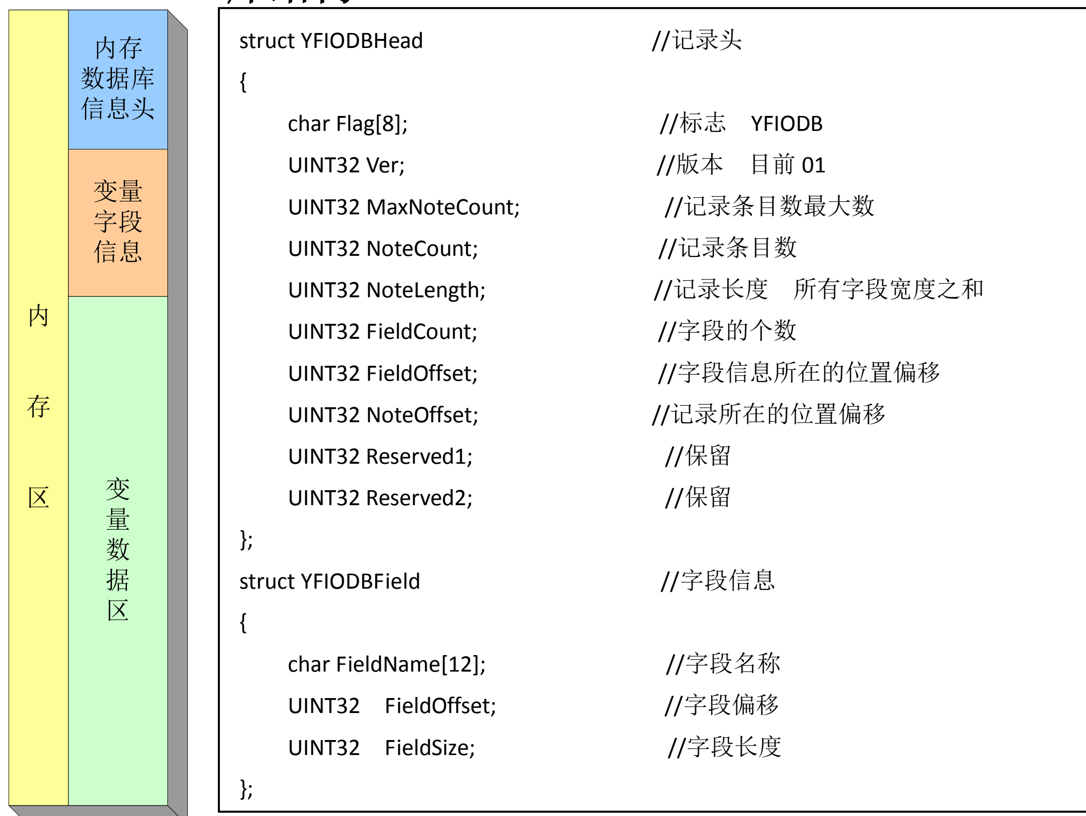




2.4 YFIODB

YFIODB 是一个在内存实现的数据库，主要存放 IO 数据，供驱动程序、策略程序直接访问，从而起到跨模块交换数据的目的。其 IO 数据一般可分两类，一种是内部 IO 数据，该类 IO 数据不绑定任何设备驱动，主要作为中间变量或临时变量来使用；另一种是设备 IO 数据，该类 IO 数据和实际的驱动程序进行绑定，该 IO 数据的值映射驱动所对应的设备参变量的值。

2.4.1 YFIODB 库结构




```
struct YFIODBHead                       //记录头
{
    char Flag[8];                        //标志    YFIODB
    UINT32 Ver;                          //版本    目前 01
    UINT32 MaxNoteCount;                 //记录条目数最大数
    UINT32 NoteCount;                    //记录条目数
    UINT32 NoteLength;                   //记录长度    所有字段宽度之和
    UINT32 FieldCount;                   //字段的个数
    UINT32 FieldOffset;                  //字段信息所在的位置偏移
    UINT32 NoteOffset;                   //记录所在的位置偏移
    UINT32 Reserved1;                    //保留
    UINT32 Reserved2;                    //保留
};
struct YFIODBField                      //字段信息
{
    char FieldName[12];                  //字段名称
    UINT32    FieldOffset;               //字段偏移
    UINT32    FieldSize;                 //字段长度
};
```

2.4.2 字段组成

序号	变量名称	长度	说明
1	Name	32	变量的名称
2	Type	2	数据类型 B 布尔型 I 整型 F 浮点型 S 字符串
3	Value	32	变量的值
4	Comment	26	注释
5	RWMode	2	读写类型 0 只读 1 只写 2 读写(自动读) 3 读写(手动读) 4-只读(手动)
6	RWFlag	2	R 自动读 W 自动写 r 手动读 n 读不操作 N 写不操作
7	LO	16	下限
8	HO	16	上限
9	DateTime	8	数据更新时 YYYY(2B)MM(1B)DD(1B)HH(1B)mm(1B)SS(1B)
统计		136	1000 个点需要 约 132.8 K Byte 的内存

YFIODB 本身仅仅是一个数据库框架平台，并不包含以上的字段信息，也不包含任何数据。YFIOs 启动后，会根据以上字段定义的信息，创建指定大小的内存数据库表，并且把预先定义好的内部 IO 变量和设备 IO 变量填充到内存数据库中去。

2.4.3 访问接口

YFIODB 访问接口被操作类接口（IOPerate）进一步封装，而操作类接口是驱动和策略标准函数接口的第一个参数，所以任何一个驱动和策略程序都可以操作 YFIODB。

相关操作接口定义如下：

```
//读数据
string IORead(string name);
int IOReadInt(string name);
float IOReadFloat(string name);
//读数据(扩展方式  变量名.字段名)
string IOReadEx(string name);
//写数据（内部写）
int IOWrite(string name, string data);
int IOWrite(string name, int data);
int IOWrite(string name, float data);
//写数据(扩展方式  变量名.字段名)
int IOWriteEx(string name, string data);
//外部写（直接写变量）
int Extern_IOWrite(string name, string data);
//变量读写模式
string IOReadMode(string name);
```

需要说明的是，该接口提供的对 YFIODB 的写操作，并不是直接对 YFIODB 数据库某表某

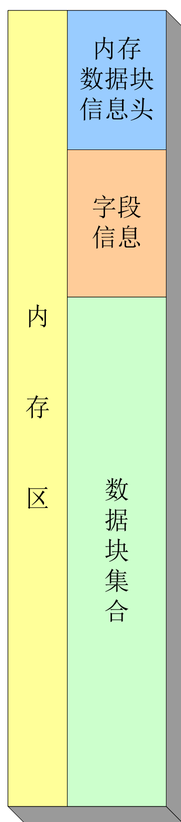
字段，进行写操作，而是根据一定的逻辑算法，对各表项综合操作（注意：扩展方式写操作，是直接对表中具体的项直接进行操作的）。驱动函数要采用内部写模式，执行后，会自动复位“W”标志位，而对策略函数来说，属于用户层面操作，所以要写 YFIODB 的时候，要采用外部写函数，执行后，函数会自动置位“W”标志位。

写 YFIODB 分内外的意义在于：策略函数仅仅是把变量的值写入数据库，而驱动才会真正的把该变量的值写入到实际的设备中去。而通过复位和置位“W”标志可以获知是否要写入到实际设备，或是否写入完成。

2.5 YFIOBC

和 YFIODB 不同，YFIOBC 是用来供驱动程序和策略程序存储和交换大块数据而用的，如摄像头的图像数据。该结构设计的如同文件系统，可新建、删除和读写，其内容大小仅受设备内存的限制。

2.5.1 YFIOBC 库结构




```
struct YFIOBCHead                      //记录头
{
    char Flag[8];                      //标志    YFIOBC
    UINT32 Ver;                        //版本    目前 01
    UINT32 MaxNoteCount;               //记录条目数最大数
    UINT32 NoteCount;                  //记录条目数
    UINT32 NoteLength;                 //记录长度    所有字段宽度之和
    UINT32 Reserved1;                  //保留
    UINT32 Reserved2;                  //保留
};

struct YFIOBCNote                      //字段信息
{
    char IOName[32];                   //IO 名称
    UINT32    MemAddr;                 //内存偏移
    UINT32    MemSeek;                 //读写偏移
    UINT32    MemSize;                 //内存大小
};
```

2.5.2 访问接口

和操作 YFIODB 接口一样，操作 YFIOBC 的接口也封装到操作类接口（IOPerate）中，所以驱动和策略程序都可以操作 YFIOBC。

操作接口定义如下：

```
//删除内存数据条目
```

```
int IOBC_Del(string name);
//size=0 打开，size>0 创建
int IOBC_Create(string name, uint size);
//获取指定条目所分配的内存大小
int IOBC_GetLength(int hander);
//读写偏移设置
int IOBC_Seek(int hander, int offset);
//读内存数据
int IOBC_Read(int hander, byte[] buffer, int offset, int count);
//写内存数据
int IOBC_Write(int hander, byte[] buffer, int offset, int count);
//关闭
int IOBC_Close(int hander);
```

该接口仿照文件操作方式进行操作，其作用类似 Windows 平台上的共享内存操作，读写都在内存中完成。

2.6 驱动开发

一个驱动程序可对应一种设备，也可以对应一类设备，关键在于设备支持的协议是私有的，还是公开的，一般公开的协议，如 Modbus，不同厂家的通信设备都有不同程度的支持（比如支持 3 号或 16 号指令），凡支持该协议的设备，都可以通过同一个设备驱动进行访问，唯一不同的就是设备地址、数据类型、起始地址和数据长度等参变量，我们可以根据实际需要，相应配置即可。

2.6.1 驱动接口类

```
public interface IDriver
{
    DriverInfo GetDriverInfo();
    int OnLoad(Device dv, IOperate op, object arg);
    int OnRun(Device dv, IOperate op, object arg);
    int OnUnload(Device dv, IOperate op, object arg);
}
```

驱动程序必须要实现这四个函数接口，其中GetDriverInfo仅供上位机配置程序调用。

（1）、GetDriverInfo – 返回驱动相关信息（请参见2.5.3）。

（2）、OnLoad – 驱动被加载时，将自动调用OnLoad方法。用户可以在该函数内，完成一些初始化操作。

（3）、OnRun – 根据配置不同，该函数按指定的时间间隔连续被系统调用（如果时间间隔配置为0，则系统不会自动调用OnRun方法）。同一个接口配置的驱动，将共享一个线程，系统将依次调用该方法。

（4）、OnUnload – 驱动被卸载时，系统将调用OnUnload。（目前YFIOs系统不支持驱动卸载）。

2.6.2 通信接口

```
public enum DeviceConnMode
{
    SerialPort = 0,
    Ethernet,
    CAN,
    USB,
    SPI,
    I2C,
    SDIO,
    Zigbee,
    AD,
    DA,
    I,
    Q,
    PWM,
    Other,
}
```

2.6.3 驱动配置信息类

```
public class DriverInfo
{
    //32byte,驱动名称（要保证唯一）
    public string Name;
    //16byte,版本信息
    public string Ver;
    //64byte,说明
     public string Explain;
    //16byte,开发者
    public string Developer;
    //16byte,日期
    public string Date;
    //自动化标志
    //0 bit 0 - 系统为你初始化通信接口  1 - 由驱动程序本身完成通信接口初始化
    //1 bit 0 - 无操作                  1 - 由驱动程序本身完成IO变量添加
    //2~31 bit 备用
    public int AutoFlag;
    //通信方式
    public DeviceConnMode ConnMode;
    //64byte,设备制造商
    public string Manufacturer;
```

```
    //32byte,设备类型
    public string DeviceType;
    //设备参数
    //硬件端口名称    空为无效项
    public string PortAddrExplain;
    //硬件端口默认值 项选择（如果有的话）用"|" 分隔开，默认项为第一个
    public string PortAddrValue;
    //端口参数名称    空为无效项
    public string PortConfigExplain;
    //端口参数默认值 项选择（如果有的话）用"|" 分隔开，默认项为第一个
    public string PortConfigValue;
    //设备地址名称    空为无效项
    public string DeviceAddrExplain;
    //设备地址默认值 项选择（如果有的话）用"|" 分隔开，默认项为第一个
    public string DeviceAddrValue;
    //设备参数名称    空为无效项
    public string DeviceConfigExplain;
    //设备参数默认值 项选择（如果有的话）用"|" 分隔开，默认项为第一个
    public string DeviceConfigValue;
    //项参数
    //8*32 byte,连接项名称
    public string[] ItemExplain;
    //8*4 byte 默认值 项选择（如果有的话）用"|" 分隔开，默认项为第一个
    public string[] ItemValue;
     //扩展配置信息的长度 如果为0，则表示没有（上位机管理程序使用）
     public int ConfigSize;
}
```

2.6.4 扩展配置接口

如果驱动程序提供的标准配置项，不足以配置驱动，则可以自行定制驱动配置页，自行生成配置数据，驱动自行解析。

DriverInfo 信息类中的最后一项 ConfigSize，就是定义该配置信息的大小。驱动的实例类中会含有一个 Config 字节数组，存放上位机管理程序配置的信息。

（【接口说明】选项就是选择该驱动后，自动出现的页面，不过该页面并没有配置任何数据，仅仅起到提示说明的作用）

```
public interface IConfig
{
    //建议面板大小319*203
    Panel[] GetPanel(byte[] InitConfig,ConfigParameter parameter);
    byte[] GetConfig();
}

public class ConfigParameter
{
    public string[] IODataNames;
    public string[] DeviceNames;
    public string[] StrategyNames;
    public object Sender;
}
```

上位机管理程序会向驱动配置面板提供当前所有 IO 内存变量名称，驱动名称和策略名称等等信息。

2.6.5 驱动的执行

驱动程序除了按设定的扫描时间周期执行外，还可以把扫描时间设置为 0，表示不会自动运行。设置为该模式的驱动，一般被策略程序直接调用而得以执行。

另外驱动还可以设置为 Disabled，这样该驱动任何方式的调用将被禁止，如该驱动不存在一样。

2.7 策略开发

可以把 YFIOs 运行时想象成一个支持多任务的操作系统，这样每个策略的 OnRun 接口，都可以当成一个进程的 Main 函数，唯一不同的是，这个 Main 函数被调用的机制多种多样（参见策略执行模式）。

策略就是一段代码，一段标准的.NET Micro Framework 程序，可以根据项目的需求充分访问.NET Micro Framework 已有的开发资源（如各类库函数），编写任意功能的代码模块。

2.7.1 策略接口类

```
public interface IStrategy
{
```

```
    StrategyInfo GetStrategyInfo();
    int OnLoad(IOperate op, object arg);
    int OnRun(IOperate op, StrategyMode mode, object arg);
    int OnUnload(IOperate op, object arg);
}
```

策略程序必须要实现这四个函数接口，其中GetStrategyInfo仅供上位机配置程序调用。

（1）、GetDriverInfo – 返回策略相关信息（请参见2.5.3）。

（2）、OnLoad – 策略被加载时，将自动调用OnLoad方法。用户可以在该函数内，完成一些初始化操作。

（3）、OnRun – 根据配置不同，该函数以事件、循环等等方式被系统自动调用。

（4）、OnUnload – 策略被卸载时，系统将调用OnUnload。（目前YFIOs系统不支持策略卸载）。

2.7.2 策略执行模式

```
public enum StrategyRunMode
{
    None = 0,          //无动作
    Loop,              //循环执行
    System_Loop,       //系统循环执行
    //事件驱动
    Event_System_Launch_Before,
    Event_System_Launch_After,
    Event_System_Error_Process,
    Event_Driver_Run_Before,
    Event_Driver_Run_After,
}
```

和最初的定义的执行模式不同，新版策略执行模块简化了许多。

（1） None – 策略定义为该模式，意味着需要其它策略来调用才能被执行。系统本身只负责加载策略和调用策略的初始化接口，

（2） Loop – 系统自动为策略创建一个线程，然后按指定的间隔，连续调用策略的OnRun的接口。

（3） System_Loop – 系统不会另外为策略创建线程，而是在主线程里（也就是Main函数中的while循环里）不断调用策略的OnRun接口，如果多个策略配置了该模式，则这些策略的OnRun接口将依次执行。建议包含界面的策略配置成这种执行模式，并且仅且只有一个这样的策略配置成这种模式。

（4） Event_System_Launch_Before – 配置为该模式，策略将在YFIOs执行Launch函数之前执行该策略。Launch函数执行的功能主要是初始化驱动、挂载驱动事件策略、创建线程执行驱动、初始化策略和创建线程执行策略。

（5） Event_System_Launch_After – 策略将在YFIOs执行Launch函数之后执行。

（6） Event_System_Error_Process – 当系统出现异常和错误的时候，将会自动调用配置为该模式的策略。

（7） Event_Driver_Run_Before – 该策略执行模式需要指定关联触发的驱动，在系统调用驱动OnRun接口之前，会自动执行配置该模式的策略。注意，当策略调用

DriverRun接口来执行驱动的OnRun函数时，该事件也会被触发。

（8） Event_Driver_Run_After – 和Event_Driver_Run_Before执行模式类似，只是在调用驱动的OnRun接口之后，触发该事件。

注意：策略并不仅支持一种策略执行模式，同一个策略可以配置多个执行模式，只要符合条件，该策略将会被调用。

2.7.3 策略的执行

策略除了按策略执行模式执行外，策略之间还可以互相调用，并且还可以直接调用指定名称的驱动程序的接口函数。

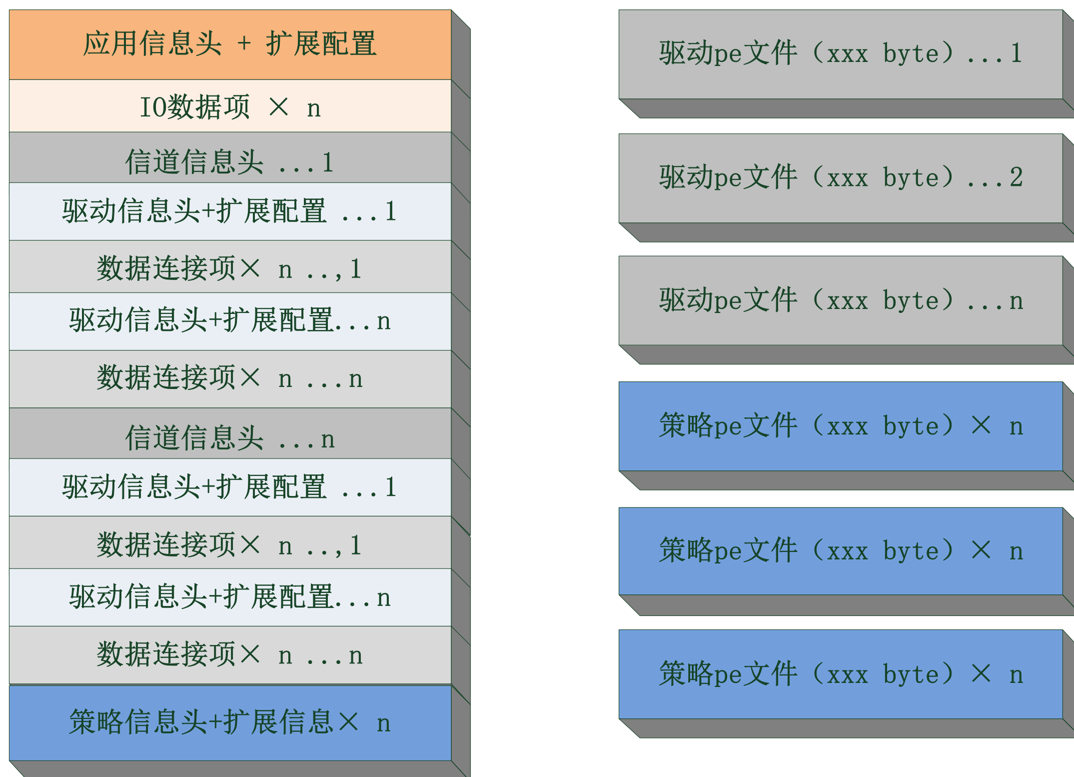
策略在配置的时候，也可以设置为Disabled，这样该策略的所有接口将无法访问，和该策略不存在一样。

2.7.4 扩展配置接口

和驱动程序的扩展配置接口相同，请参见 2.5.4 项的介绍。

2.8 YFIOs 项目存储

2.8.1 项目存储映像图



2.8.2 项目信息头

```
public class ApplicationHead
{
    public string Flag = "YFIOs";          //标志  YFIOs 8 byte
    public uint Ver;                       //版本
    public uint ApplicationHeadSize;       //ApplicationHead大小
    public uint ChannelHeadSize;           //ChannelHead大小
    public uint DeviceHeadSize;            //DeviceHead大小
    public uint StrategyHeadSize;          //StrategyHeadSize大小
    public uint StrategyModeSize;          //StrategyModeSize大小
    public uint IOItemSize;                //IOItem大小
    public uint IODataSize;                //IOData大小
    public string Name="";                 //应用名称   32 byte
    //服务器URL，128byte 既可以是IP+端口模式 192.168.0.1:80 ,也可以是标准url格式
    public string Server = "http://192.168.1.100";
    public int MaxDBCount = 256;           //IO数据最大条目数
    public int MaxBCCount = 8;             //IO数据块最大个数
    public uint IODataCount;               //IO数据个数
    public uint ChannelCount;              //信道个数
    public uint StrategyCount;             //策略个数
    //总调试模式开关，驱动中的debugmode为单个控制
    public uint DebugMode = 0xEC;
    public uint ConfigSize;                //扩展配置信息
}
```

2.8.3 信道信息头

```
public class ChannelHead
{
    //禁止执行 0 - 允许执行  1 - 禁止执行
    public int Disabled;
    //通道模式 0 - 分别打开端口  1 - 统一打开端口
    public int ChannelMode = 0;
    //通信方式
    public DeviceConnMode ConnMode;
    //端口地址  串口：1.串口：1...n  网络：端口号 ...
    public int PortAddr;
    //32byte,端口参数   如串口：波特率,数据位,校验方式,停止位  如9600,N,8,1
    public string PortConfig;
    //设备个数
    public uint DeviceCount;
}
```

2.8.4 驱动信息头

```
public class DeviceHead
{
    public int Disabled;                        //0 - 执行 1 - 禁止执行 不调用相关函数
    public string Name;                         //32byte,设备名称
    public int AutoFlag;                        //自动化标志
    public DeviceConnMode ConnMode;             //通信方式
    public int PortAddr;                        //端口地址  1.串口:1...n  网络:端口号 ...
    public string PortConfig;                   //32byte,端口参数
    public int DeviceAddr;                      //设备地址
    public string DeviceConfig;                 //32byte,设备参数
    public int Scantime;                        //扫描周期(ms)  如果为0, 则禁止扫描
    public int Overtime;                        //超时时间(ms)
    public int ErrorScantime;                   //故障扫描周期(s
    public int ErrorMaxScantime;                //最长故障扫描周期(s)
    public int AcceptBufferLength;              //接收缓冲区大小
    public int SendBufferLength;                //发送缓冲区大小
    public int ReadDataBufferLength;            //一次从端口读取的字符数
    public int DebugMode;                       //debug模式, 控制驱动是否显示一些调试信息
    public uint IOItemCount;                    //设备连接项个数
    public uint ConfigSize;                     //扩展配置信息
    public uint PeSize;                         //pe文件的大小
    public uint PeAddr;                         //pe文件存放的地址（相对地址）
}
```

2.8.5 策略信息头

```
public class StrategyHead
{
    public int Disabled;                        //禁止执行, 不调用相关函数
    public string Name;                         //32byte,策略名称
    public uint ModeCount;                      //策略运行模式个数
    public uint ConfigSize;                     //扩展配置信息
    public uint PeSize;                         //pe文件的大小
    public uint PeAddr;                         //pe文件存放的地址（相对地址）
}
```

2.8.6 数据连接项

```
public class IOItem
{
```

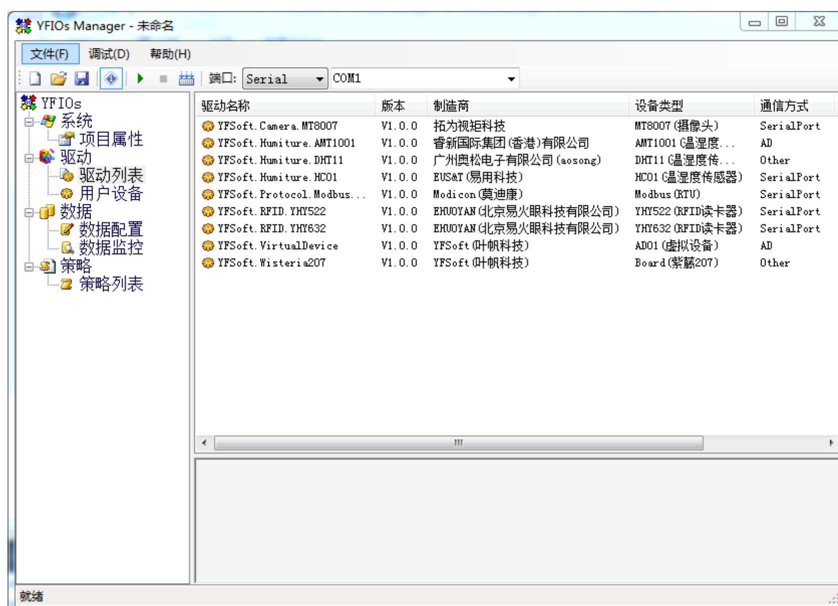
```
        public string Name;                //32byte,内存变量名称（对数组，仅指数组名）
        public int[] Param = new int[8];       //中间传递变量，由驱动程序自己设定，自己解释
    }
```

2.8.7 IO 数据

```
    public class IOData
    {
        public string Name = "";         //32数据名称
        public string Type = "";         //2 数据类型 B布尔型 I整型 F浮点型 S 字符串
        public string Value = "";         //32    变量的值
        public string Comment = "";       //26    注释
        //2  读写模式 0 只读 1 只写 2 读写(自动读) 3 读
        public string RWMode = "";         写(手动读) 4-只读(手动)
        //2  R 自动读 W 自动写  r 手动读 n读不操作 N 写不操作
        public string RWFlag = "";
        public string LO = "";            //16    下限
        public string HO = "";            //16    上限
        public string DateTime = "";      //8  数据更新时间
    }
```

3 YFIOs 应用开发

3.1 YFIOsManager 简介

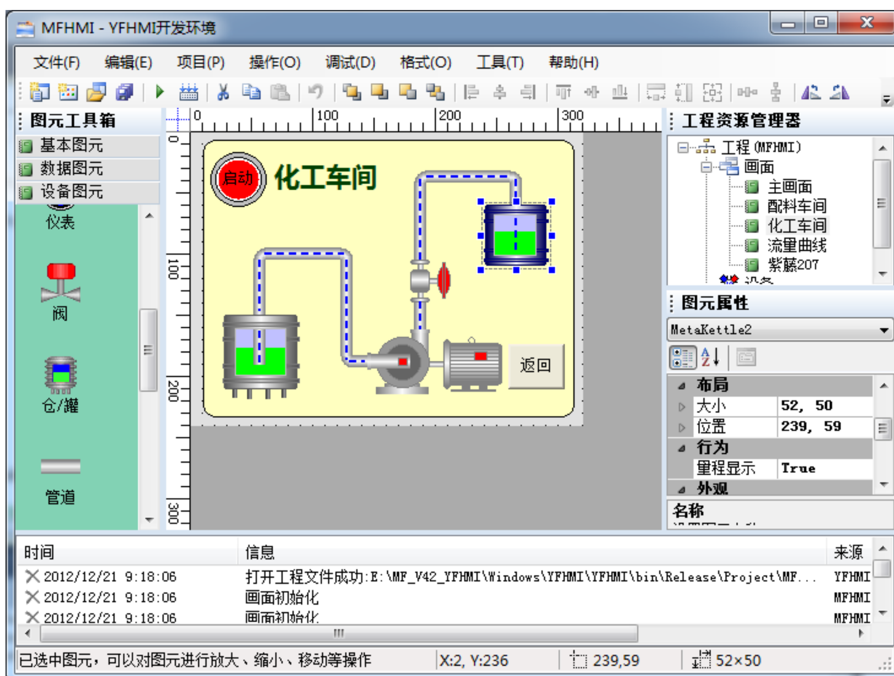


操作视频演示：http://v.youku.com/v_show/id_XNDkxMzgyNTgw.html

3.2 YFIOs 应用实例

3.2.1 农村个人医疗远程助理

3.2.2 YFHMI 物联网画面组态系统



操作演示视频：http://v.youku.com/v_show/id_XNDg2MjMxODI4.html

设备运行视频：http://v.youku.com/v_show/id_XNDg2MjM4MTQw.html

4 相关资源

1、.NET Micro Framework 官方网址
http://www.microsoft.com/netmf/default.mspx
2、.NET Micro Framework 官方博客
http://blogs.msdn.com/netmfteam/
3、中文博客
http://blog.csdn.net/yefanqiu
http://www.cnblogs.com/yefanqiu
4、叶帆科技
http://www.sky-walker.com.cn/
5、物联网中间件技术开发论坛
http://www.yfios.net