

# EMIO 总线式数字 IO 模块 EIO 系列

## 使用手册

V2.10

修订历史				
修订版本	修订内容	适用硬件	修改时间	修改者
V1.0	第一版发行	EIO2416 EIO4832	201909	XX
V2.8	增加断线及重连回调事件功能	EIO2416 EIO4832	201910	XX
V2.10	增加断电重连回调事件及 Utility	EIO2416 EIO4832	201912	XX

# 目录

第 1 章 EMIO 数字 IO 模块硬件 .....	5
1.1 功能介绍 .....	5
1.2 安装尺寸 .....	7
1.3 端子定义 .....	8
1.4 通用数字输入接口 .....	9
1.5 通用数字输出接口 .....	10
第 2 章 EMIO 数字 IO 模块软件 .....	11
2.1 驱动 API 函数库及例程安装 .....	11
2.2 编程指南 .....	14
2.2.1 Visual C++ MFC 应用程序开发 .....	14
2.2.2 Visual C# Windows 窗体应用程序开发 .....	20
2.2.3 Visual Basic.NET Windows 窗体应用程序开发 .....	24
第 3 章 EMIO 数字 IO 模块 API 函数 .....	26
3.1 模块系统函数 .....	26
3.1.1 EMIO_Get_Connected_Stations() .....	26
3.1.2 EMIO_Get_Connected_Station_Nums() .....	27
3.1.3 EMIO_Get_Connected_Station_Type() .....	27
3.1.4 EMIO_Open() .....	28
3.1.5 EMIO_OpenEx() .....	29
3.1.6 EMIO_Close() .....	30
3.2 模块通用数字输入输出函数 .....	31
3.2.1 EMIO_Get_Input() .....	31

3.2.2 EMIO_Get_Input_Bit().....	32
3.2.3 EMIO_Get_Output().....	33
3.2.4 EMIO_Get_Output_Bit() .....	34
3.2.5 EMIO_Set_Output().....	35
3.2.6 EMIO_Set_Output_Bit().....	36
3.2.7 EMIO_Inv_Output_Bit() .....	37
3.3 模块实用函数 .....	38
3.3.1 EMIO_Get_First_Error_Message().....	38
3.3.2 EMIO_Get_Last_Error_Message() .....	38
3.3.3 EMIO_Get_Error_Message().....	39
3.3.4 EMIO_Get_SN() .....	39
3.3.5 EMIO_Get_SN_Text().....	40
3.3.6 EMIO_Get_Station_Info().....	41
3.3.7 EMIO_Get_Dll_Version().....	41
3.3.8 EMIO_Get_Language() .....	42
3.3.9 EMIO_Set_Language().....	42
3.3.10 EMIO_Get_Process_Type() .....	42
3.3.11 EMIO_Register_Callback_Event() .....	43
3.3.12 EMIO_Get_Disconnect_Counts() .....	44
3.3.13 EMIO_Set_Disconnect_Counts().....	44
3.3.14 EMIO_Get_Interrup_Reconnect_Counts() .....	45
3.3.15 EMIO_Set_Interrup_Reconnect_Counts().....	45
3.3.16 EMIO_Get_PowerOff_Reconnect_Counts().....	46

3.3.17 EMIO_Set_PowerOff_Reconnect_Counts() .....	46
第 4 章 常见问题 FAQ .....	47
4.1 硬件常见 FAQ .....	47
4.2 软件常见 FAQ .....	49

## 前言

尊敬的客户:

欢迎选用本公司数字 IO 模块。

本手册主要对数字 IO 模块的硬件及软件等内容作全面介绍，提供了使用数字 IO 模块所需知识及注意事项，请在熟知本产品的安全注意事项后使用。

由于产品的改进，规格、编写版本的变更等原因，会有适当的改动，恕不另行通知。不承担由于使用本手册或本产品不当，所造成直接的、间接的、特殊的、附带的或相应产生的损失或责任。

# 第 1 章 EMIO 数字 IO 模块硬件

## 1.1 功能介绍



图 1.1.1 总线式系统架构

EMIO 数字 IO 模块系列采用 XILINX FPGA 作为主处理器，以工业以太网总线为载体，采用专用封闭协议保证通讯的稳定性及响应性，总线通讯周期 1ms，自动实现丢包重发机制，支持断线重连中断事件，支持多模块串联，支持与国内外运动控制卡 IO 卡同时使用，支持与 EtherCAT 主站同时使用。软件上提供 SDK API 函数库开发包，驱动及丰富例程，支持 WinXP~Win10 32/64 位系统上使用 VC++ C# VB LabVIEW QT Python 等高级语言的开发。

EMIO 数字 IO 模块系列**不需要**额外的主站系统/主站卡，**只需要**用超 6 类双屏蔽网线与主机串联，上位机用 API 函数库编译应用程序就可以实时控制数字 IO 模块。

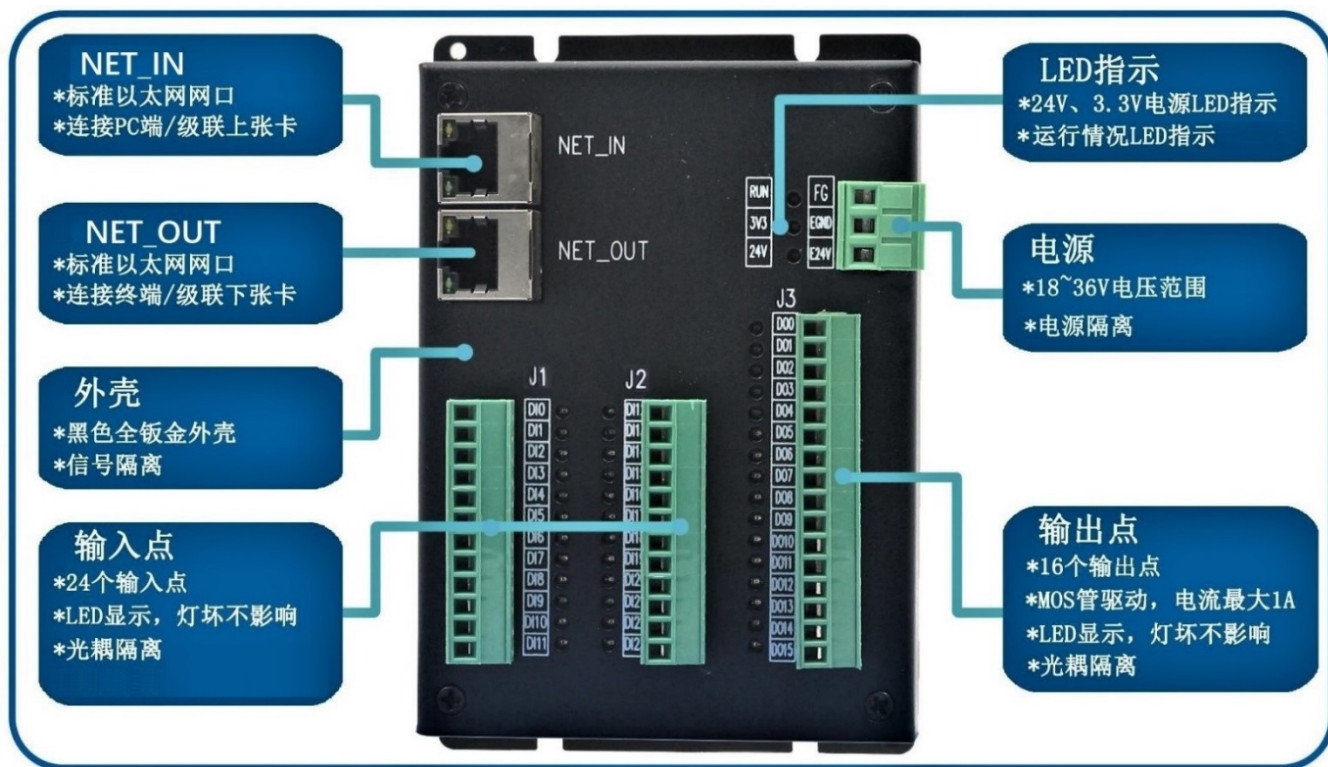


图 1.1.2 EIO2416 功能介绍

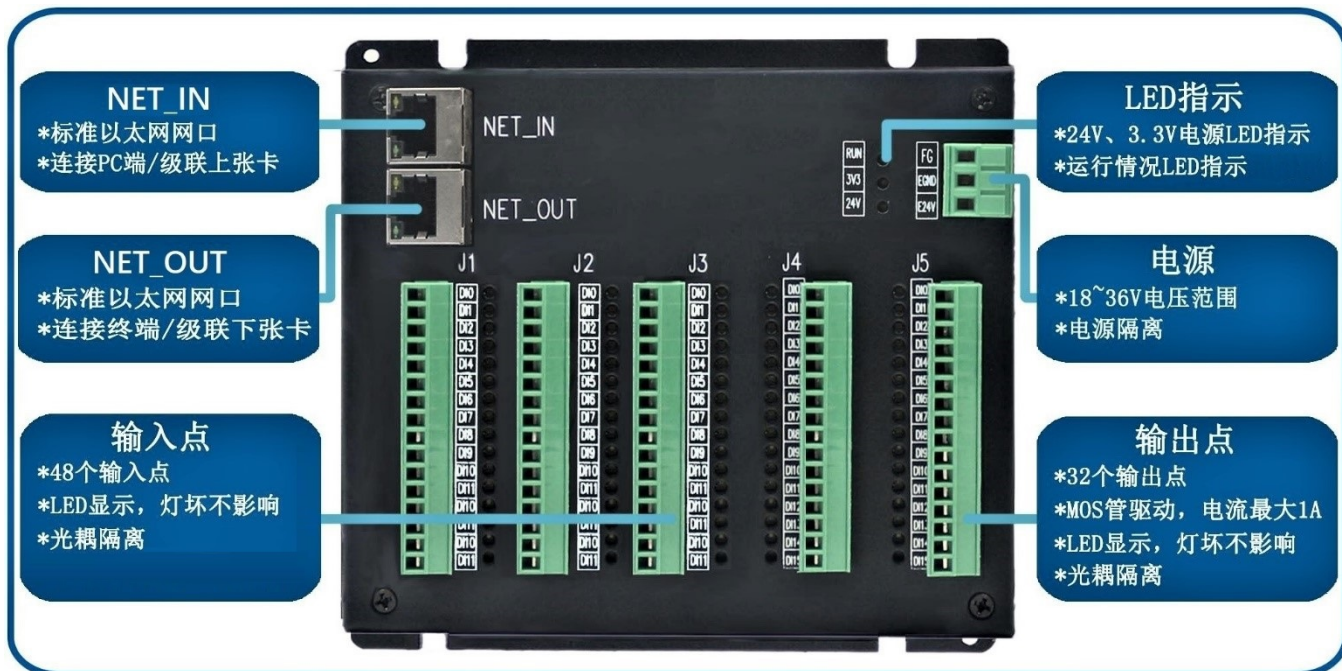


图 1.1.3 EIO4832 功能介绍

## 1.2 安装尺寸

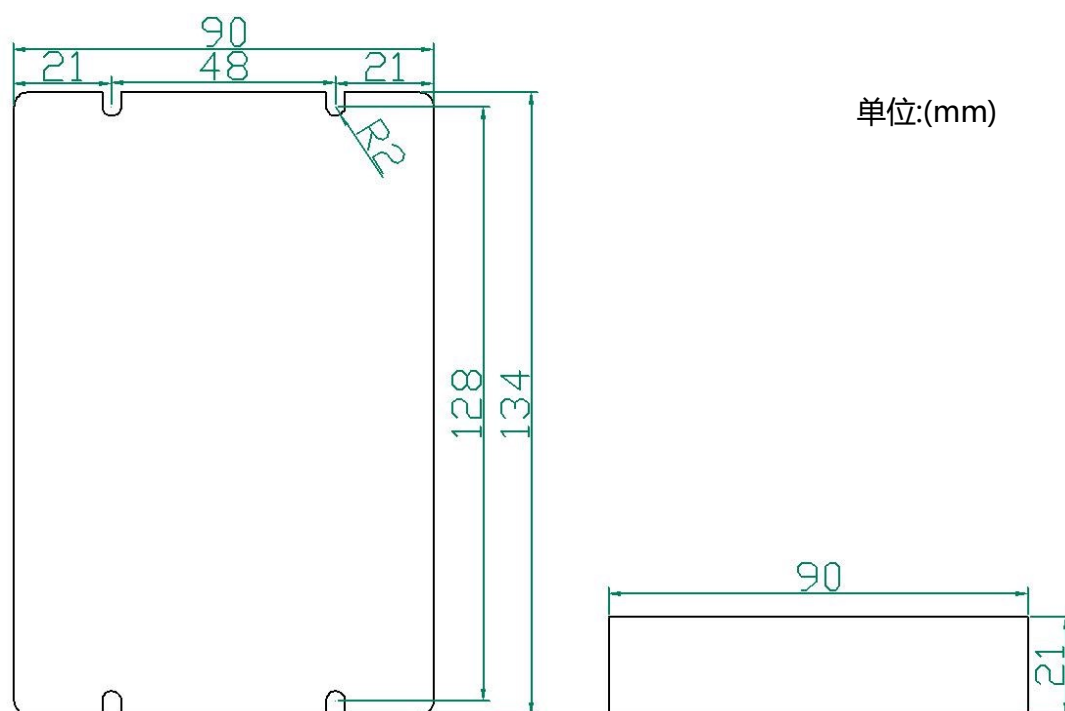


图 1.2.1 EIO2416 安装尺寸

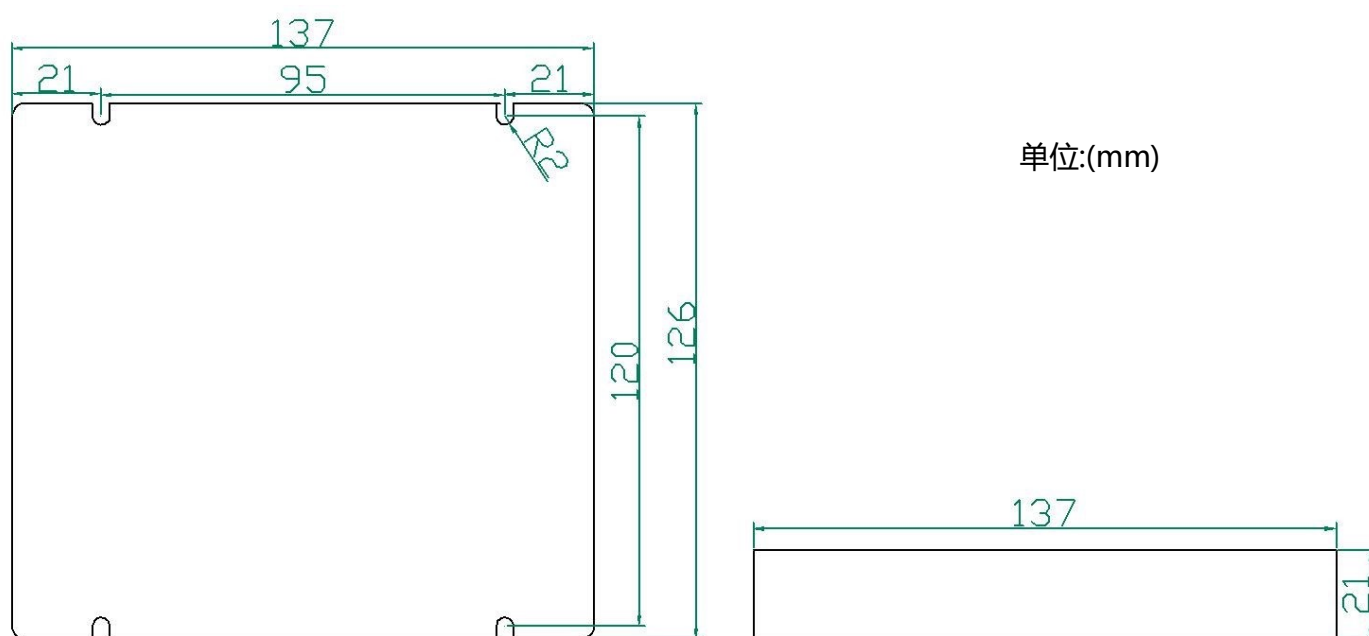


图 1.2.2 EIO4832 安装尺寸

1.3 端子定义

PIN	J0	
1	FG	外壳接地
2	EGND	0V
3	E24V	+24V

表 1.1 电源定义


	PIN	J1		J2		J3	
	1	DI0	输入 0	DI12	输入 12	DO0	输出 0
	2	DI1	输入 1	DI13	输入 13	DO1	输出 1
	3	DI2	输入 2	DI14	输入 14	DO2	输出 2
	4	DI3	输入 3	DI15	输入 15	DO3	输出 3
	5	DI4	输入 4	DI16	输入 16	DO4	输出 4
	6	DI5	输入 5	DI17	输入 17	DO5	输出 5
	7	DI6	输入 6	DI18	输入 18	DO6	输出 6
	8	DI7	输入 7	DI19	输入 19	DO7	输出 7
	9	DI8	输入 8	DI20	输入 20	DO8	输出 8
	10	DI9	输入 9	DI21	输入 21	DO9	输出 9
	11	DI10	输入 10	DI22	输入 22	DO10	输出 10
	12	DI11	输入 11	DI23	输入 23	DO11	输出 11
	13					DO12	输出 12
	14					DO13	输出 13
	15					DO14	输出 14
	16					DO15	输出 15

表 1.2 EIO2416 端子定义



PIN	J1		J2		J3		J4		J5	
1	DI0	输入 0	DI16	输入 16	DI32	输入 32	DO0	输出 0	DO16	输出 16
2	DI1	输入 1	DI17	输入 17	DI33	输入 33	DO1	输出 1	DO17	输出 17
3	DI2	输入 2	DI18	输入 18	DI34	输入 34	DO2	输出 2	DO18	输出 18
4	DI3	输入 3	DI19	输入 19	DI35	输入 35	DO3	输出 3	DO19	输出 19
5	DI4	输入 4	DI20	输入 20	DI36	输入 36	DO4	输出 4	DO20	输出 20
6	DI5	输入 5	DI21	输入 21	DI37	输入 37	DO5	输出 5	DO21	输出 21
7	DI6	输入 6	DI22	输入 22	DI38	输入 38	DO6	输出 6	DO22	输出 22
8	DI7	输入 7	DI23	输入 23	DI39	输入 39	DO7	输出 7	DO23	输出 23
9	DI8	输入 8	DI24	输入 24	DI40	输入 40	DO8	输出 8	DO24	输出 24
10	DI9	输入 9	DI25	输入 25	DI41	输入 41	DO9	输出 9	DO25	输出 25
11	DI10	输入 10	DI26	输入 26	DI42	输入 42	DO10	输出 10	DO26	输出 26
12	DI11	输入 11	DI27	输入 27	DI43	输入 43	DO11	输出 11	DO27	输出 27
13	DI12	输入 12	DI28	输入 28	DI44	输入 44	DO12	输出 12	DO28	输出 28
14	DI13	输入 13	DI29	输入 29	DI45	输入 45	DO13	输出 13	DO29	输出 29
15	DI14	输入 14	DI30	输入 30	DI46	输入 46	DO14	输出 14	DO30	输出 30
16	DI15	输入 15	DI31	输入 31	DI47	输入 47	DO15	输出 15	DO31	输出 31

表 1.3 EIO4832 端子定义

## 1.4 通用数字输入接口

EIO2416 模块有 24 路通用数字输入信号(NPN)，EIO4832 模块有 48 路通用数字输入信号(NPN)。所有输入接口均加有光电隔离元件，可以有效隔离外部电路的干扰，以提高系统的可靠性。通用数字输入信号接口原理图如图 1.6 所示。

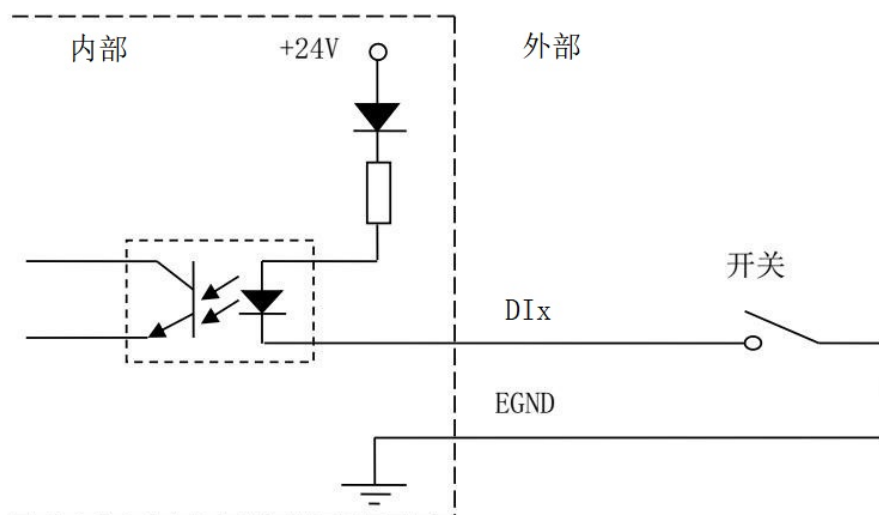


图 1.4.1 通用数字输入电路

## 1.5 通用数字输出接口

EIO2416 模块有 16 路通用数字输出信号(NPN)，EIO4832 模块有 32 路通用数字输入信号(NPN)。由 MOS 管驱动，其最大工作电流为 **1A**，可用于控制继电器、电磁阀、信号灯或其它设备，如图 1.7 所示。

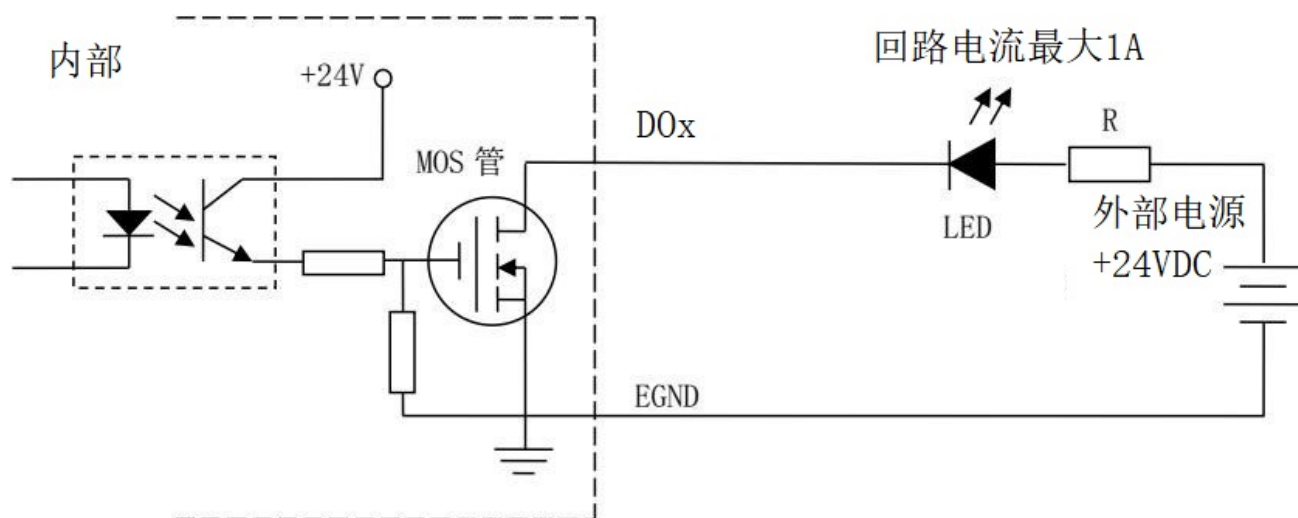


图 1.5.1 通用数字输出电路

## 第 2 章 EMIO 数字 IO 模块软件

### 2.1 驱动 API 函数库及例程安装

目前仅提供 Windows 安装包(支持 Windows XP/Vista/Win7/Win10)。

下载地址:

[https://raw.githubusercontent.com/yunshi-tech/EMIO/master/YSEMIO\\_Setup.exe](https://raw.githubusercontent.com/yunshi-tech/EMIO/master/YSEMIO_Setup.exe)

[https://gitee.com/yunshi-tech/EMIO/raw/master/YSEMIO\\_Setup.exe](https://gitee.com/yunshi-tech/EMIO/raw/master/YSEMIO_Setup.exe)

双击安装包进行驱动，API 及例程的安装。

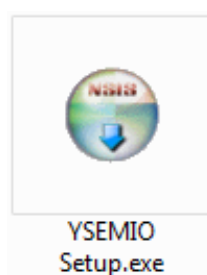


图 2.1.1 驱动安装包程序

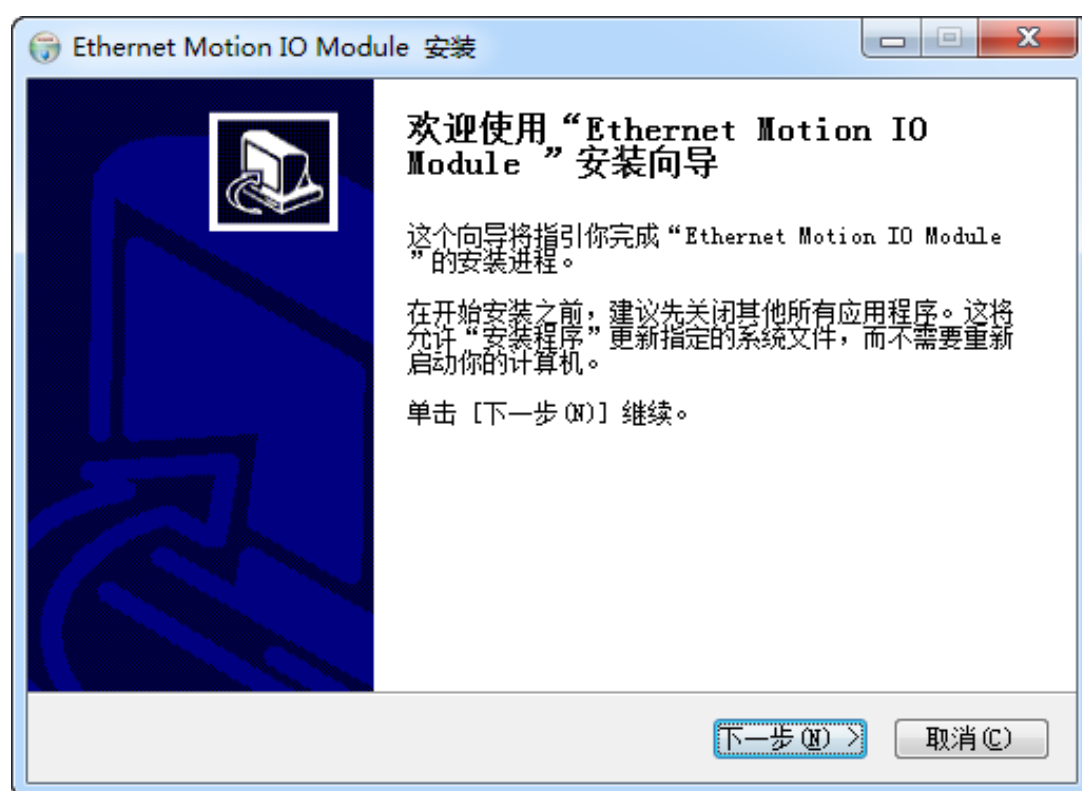


图 2.1.2 驱动安装向导



图 2.1.3 选择安装目录



图 2.1.4 安装完成

安装完成后双击桌面上的 YSEMIO Utility 快捷方式图标运行调试程序。

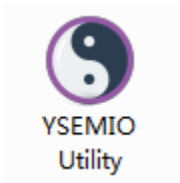


图 2.1.5 调试工具 YSEMIO Utility

双击 DO 图标可以反向操作 DO 等，当有断线及重连时下面输出列表会有提示。

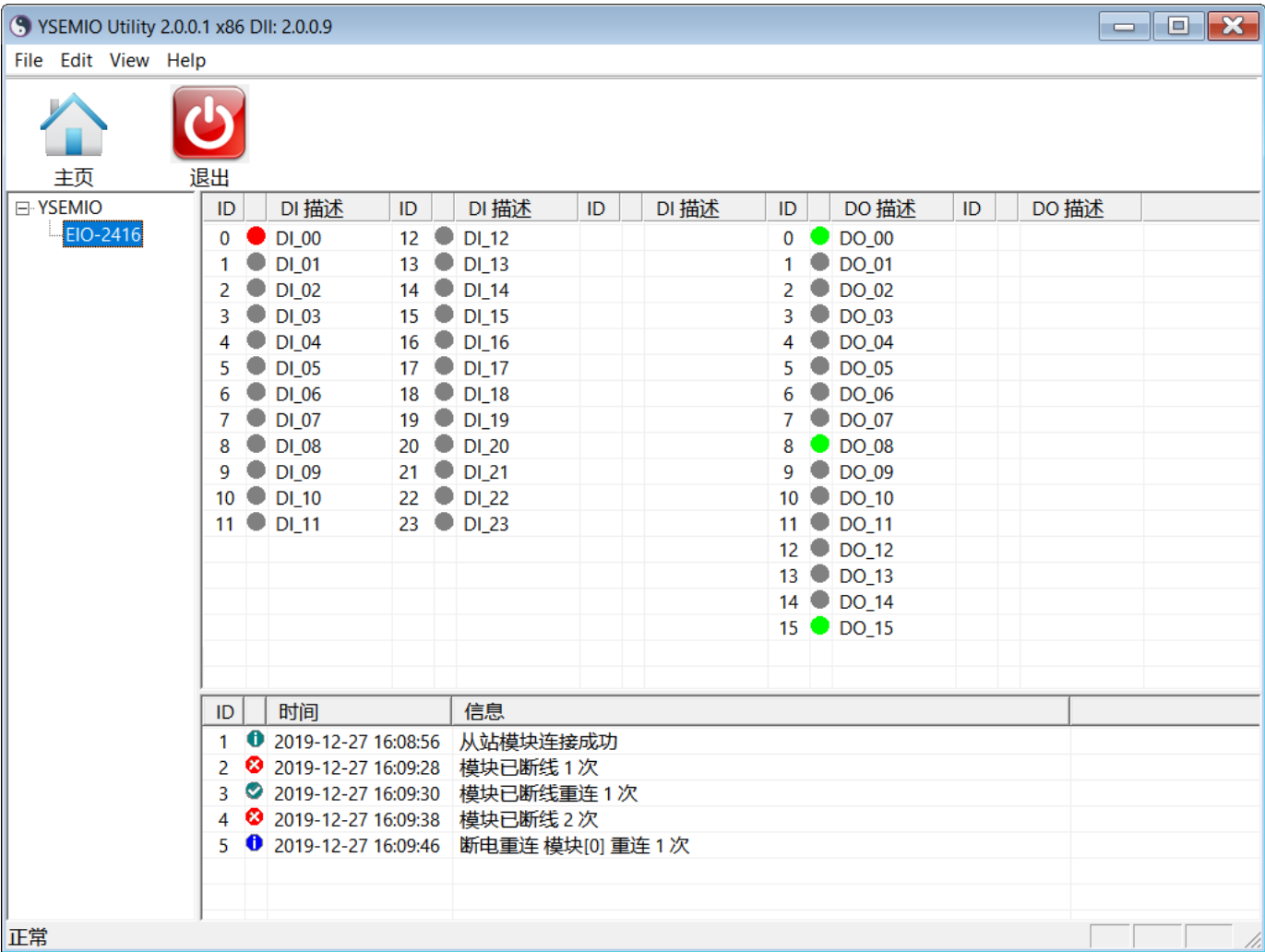


图 2.1.6 YSEMIO Utility 操作

## 2.2 编程指南

### 2.2.1 Visual C++ MFC 应用程序开发

1. 启动 Visual Studio 2010，文件-新建-项目-Visual C++，在 MFC 应用向导下一步中选择 基于对话框类型并完成。



图 2.2.1.1 新建对话框应用程序

2. 设置 项目-属性, 弹出项目属性页对话框(Alt+F7)。

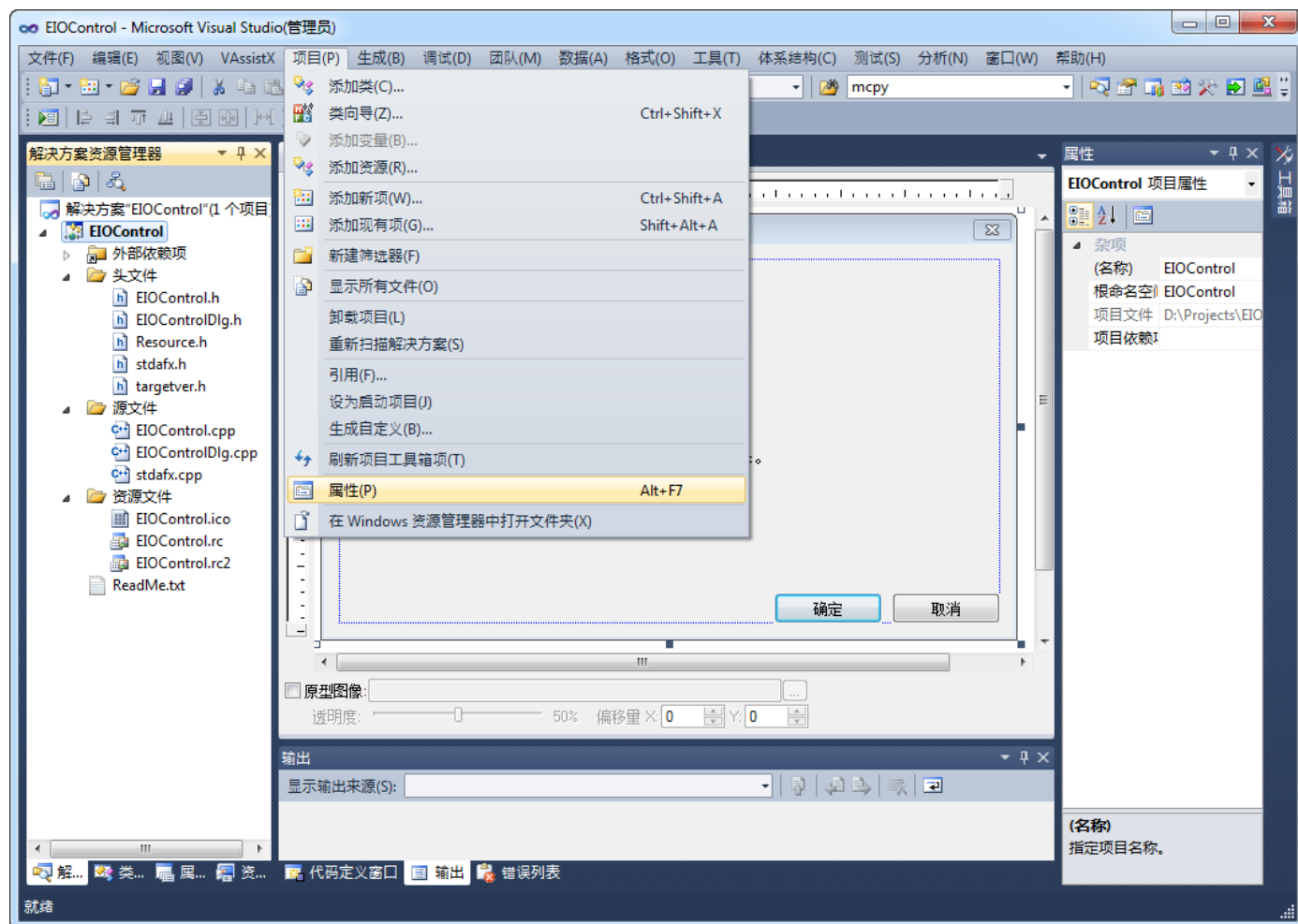


图 2.2.1.2 打开项目属性

3. 在项目属性页对话框, 在 配置属性-VC++ 目录中, 分别设置包含目录和库目录为安装目录位置:

包含目录: C:\Program Files\EMIO\Include

库目录: C:\Program Files\EMIO\Lib\x86 (Win32 程序)

C:\Program Files\EMIO\Lib\x64 (Win64 程序)

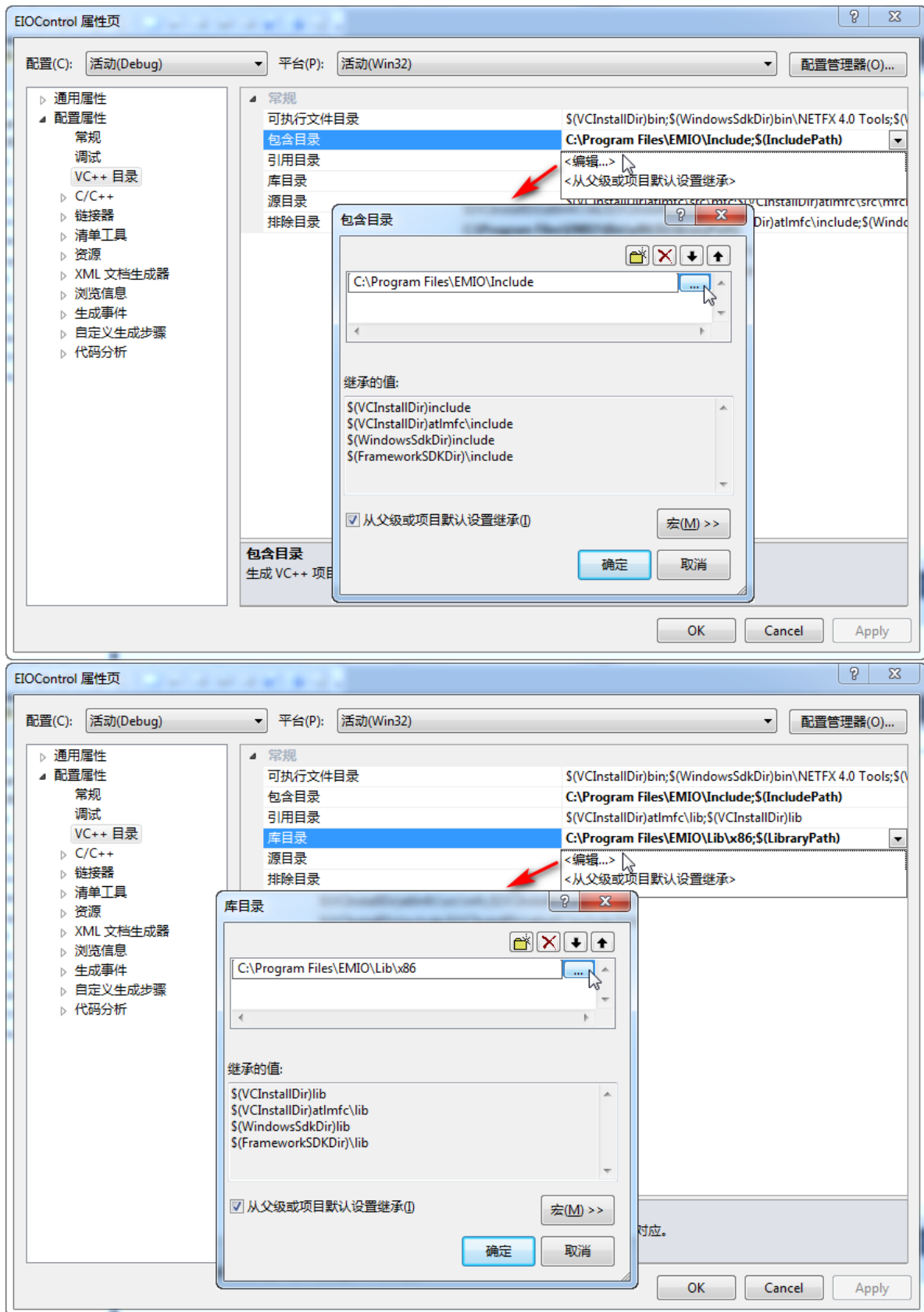


图 2.2.1.3 设置包含目录和库目录



4. 在对话框程序头文件中，加入包括头文件和库文件的代码：

```
#include "YSEMIODef.h"
```

```
#include "YSEMIO.h"
```

```
#pragma comment(lib, "YSEMIO.lib")
```

到此，就可以调用 EMIO API 函数库中的任何函数，开始编写应用程序了。

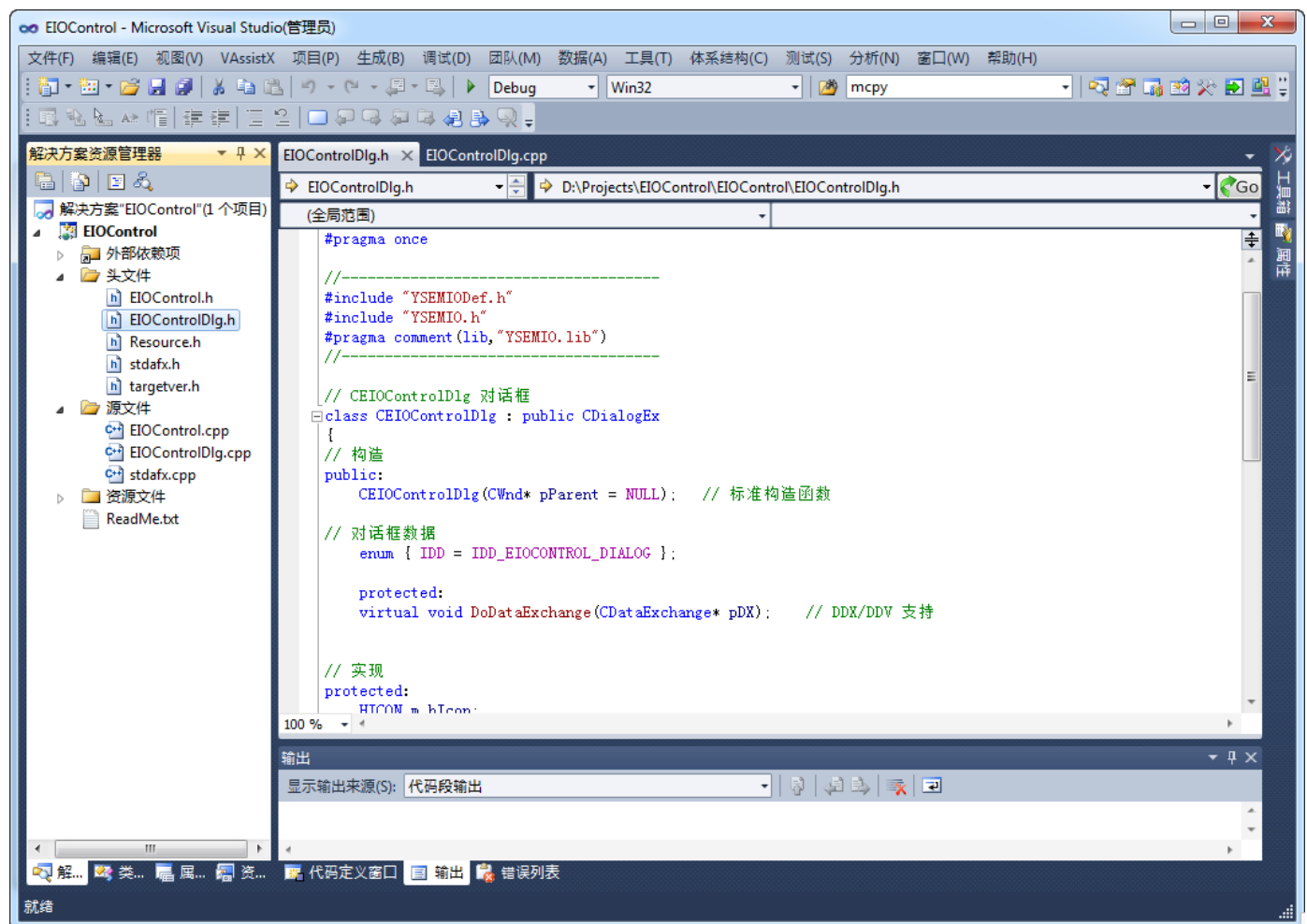


图 2.2.1.4 加入包括头文件和库文件

5. EMIO API 函数库的一般使用流程如下：

- (1). 获取所有已经实际连接上的从站模块的总数及其类型数组。
- (2). 打开并连接指定数目和指定模块类型数组的总线从站模块。
- (3). 操作数字输入输出。
- (4). 关闭总线从站模块。

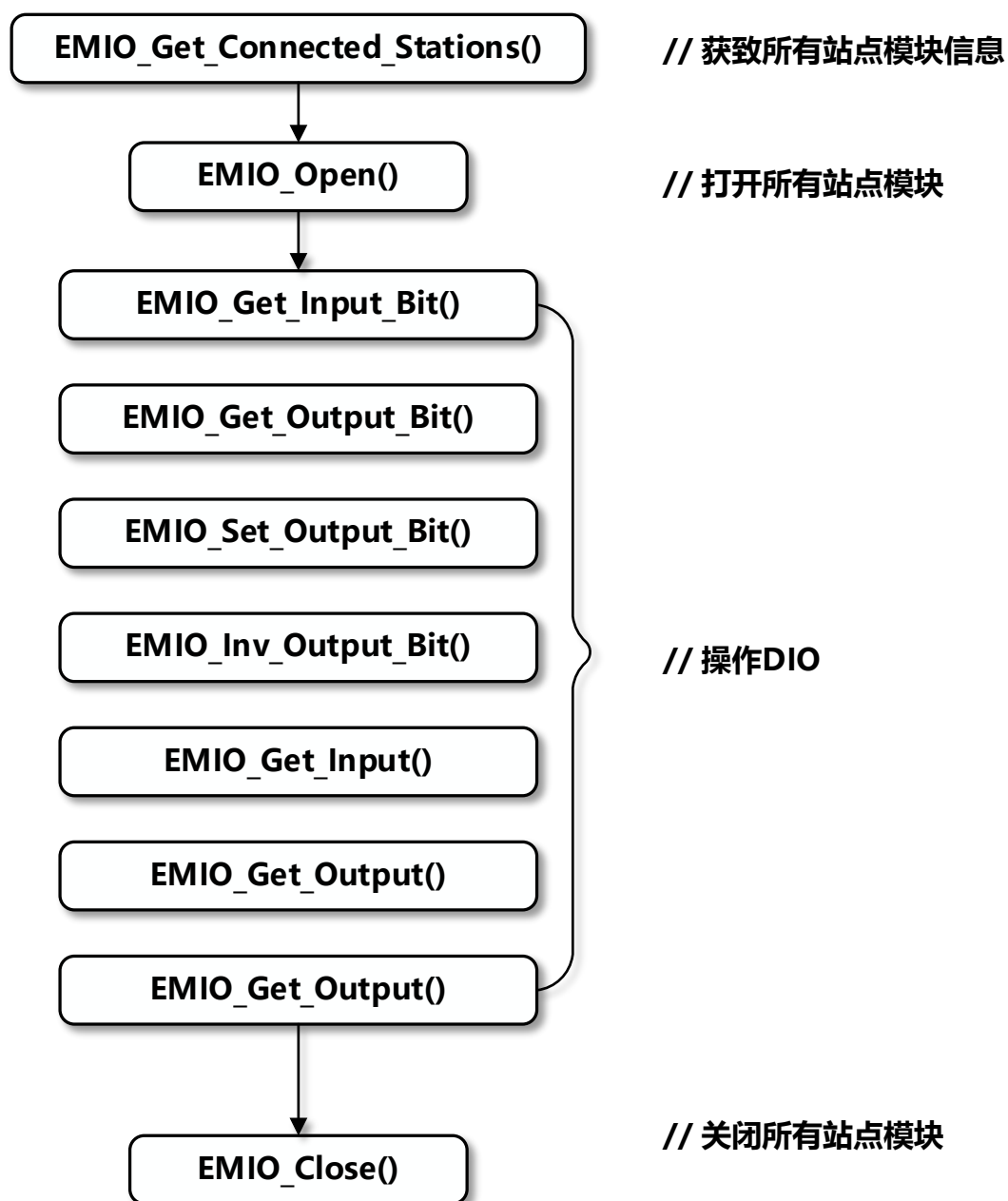


图 2.2.1.5 VC++ 函数库使用流程

6. EMIO API 函数库的断线/重连事件使用流程如下：

- (1). 自定义断线/重连事件处理函数。
- (2). 在打开模块后注册已定义的断线/重连事件函数。
- (3). 网络断线/重连后自动执行断线/重连事件处理函数一次。

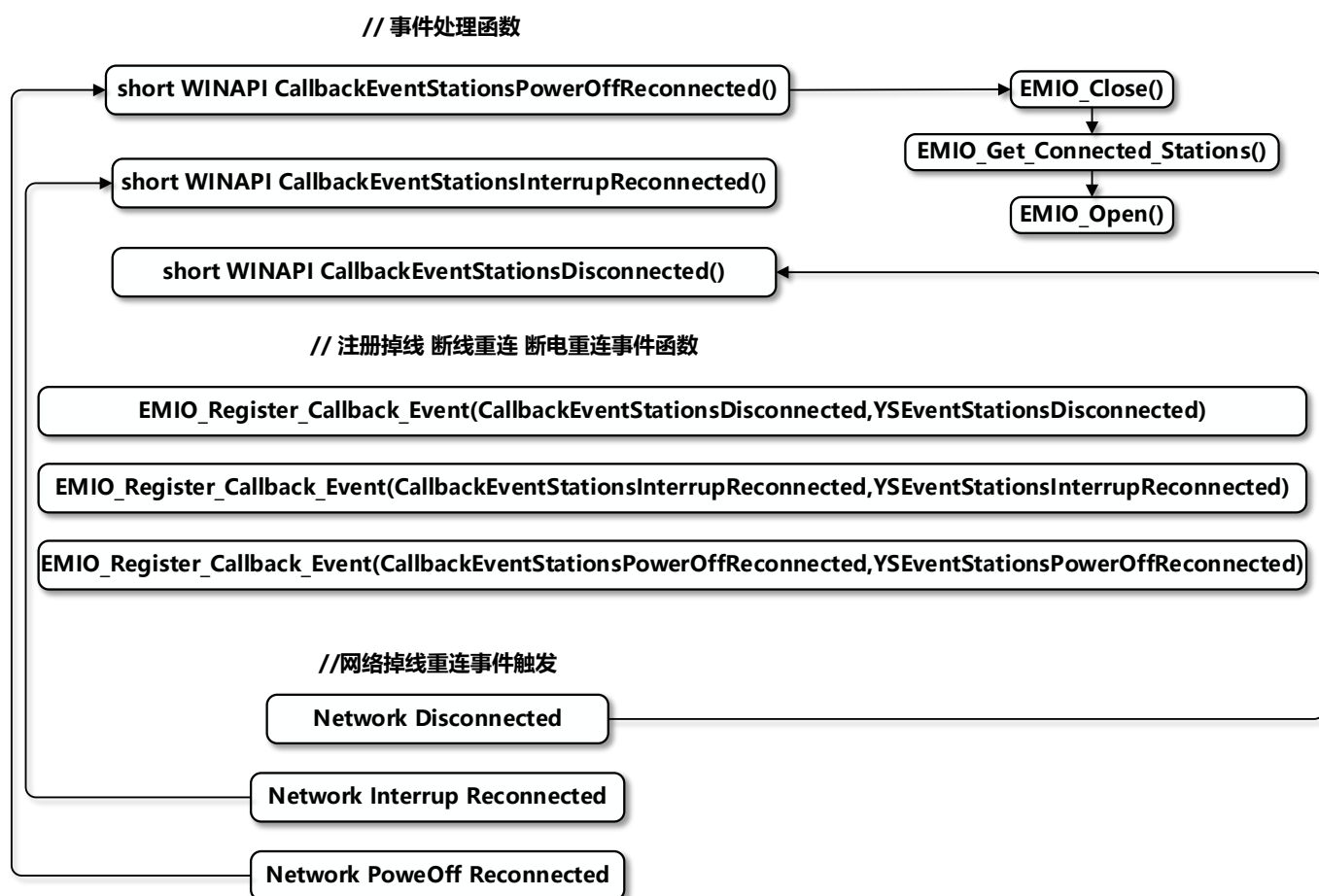


图 2.2.1.6 VC++断线/重连事件使用流程

EMIO API 函数库详细具体使用方法，请参考例程。

## 2.2.2 Visual C# Windows 窗体应用程序开发

1. 启动 Visual Studio 2010，文件-新建-项目-Visual C#，选择 Windows 窗体应用程序。

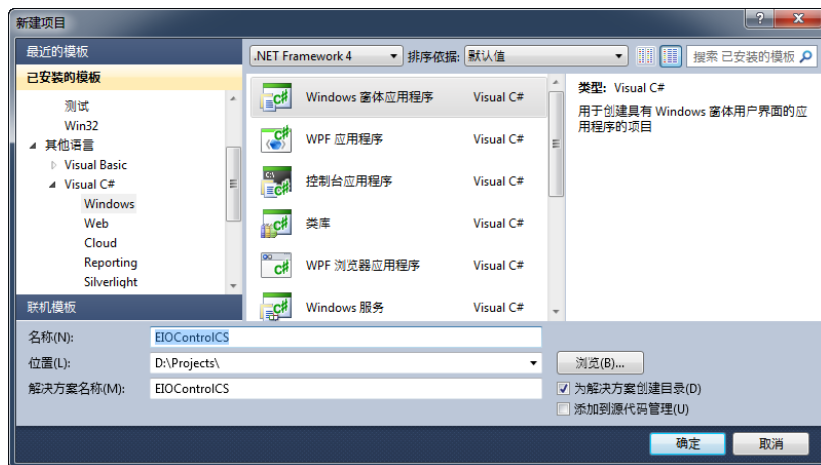


图 2.2.2.1 新建 Windows 窗体应用程序

2. 项目-添加现有项，选择 C:\Program Files\EMIO\Include\YSEMIO.cs 文件。

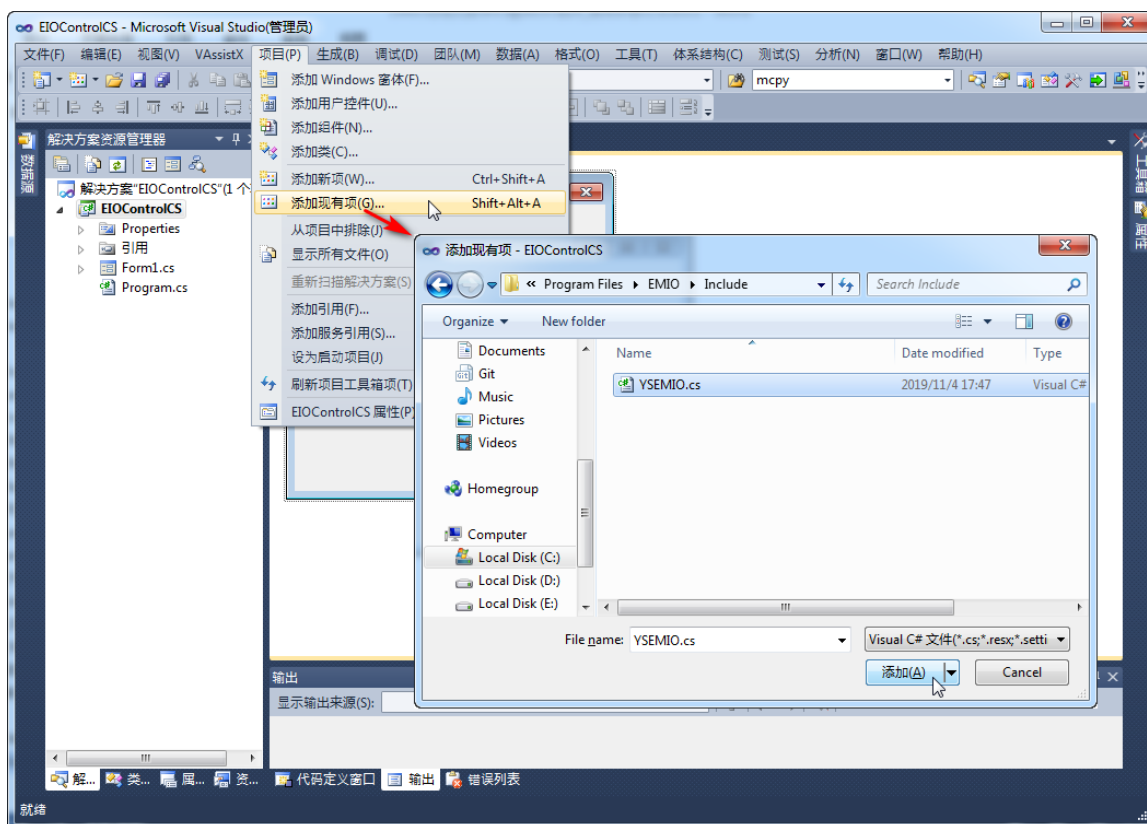


图 2.2.2.2 添加现有项 YSEMIO.cs

### 3. 在 Form1.cs 中添加使用命名空间代码 using YSEMIO;

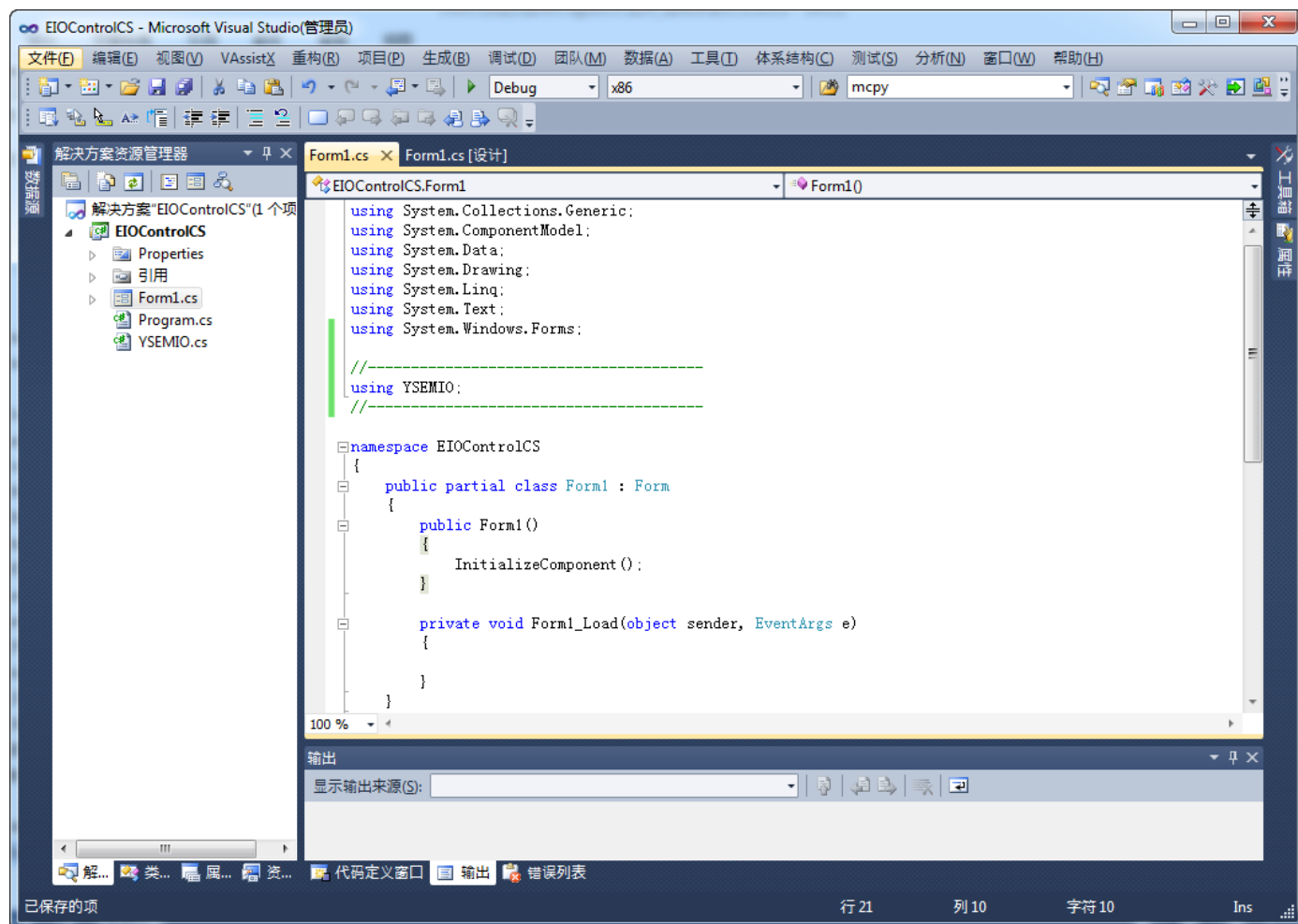


图 2.2.2.3 使用命名空间 YSEMIO

到此，就可以调用 EMIO API 函数库中的任何函数，开始编写应用程序了。

4. EMIO API 函数库的一般使用流程如下：

- (1). 获取所有已经实际连接上的从站模块的总数及其类型数组。
- (2). 打开并连接指定数目和指定模块类型数组的总线从站模块。
- (3). 操作数字输入输出。
- (4). 关闭总线从站模块。

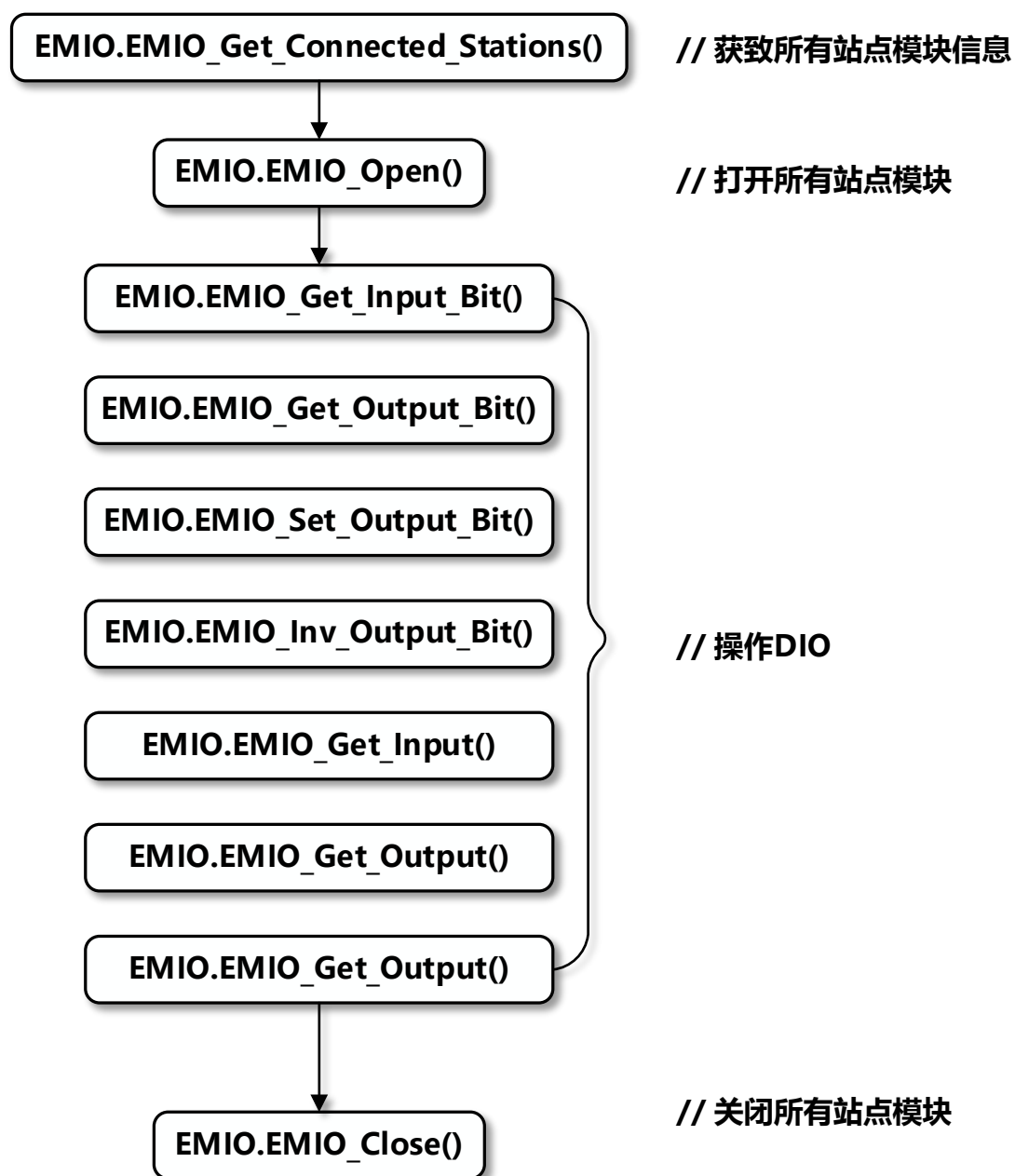


图 2.2.2.4 VC#函数库使用流程

5. EMIO API 函数库的断线/重连事件使用流程如下：

- (1). 自定义断线/重连事件处理函数。
- (2). 定义并新建断线/重连事件委托。
- (3). 在打开模块后注册已定义的断线/重连事件委托。
- (4). 网络断线/重连后自动执行断线/重连事件处理函数一次。

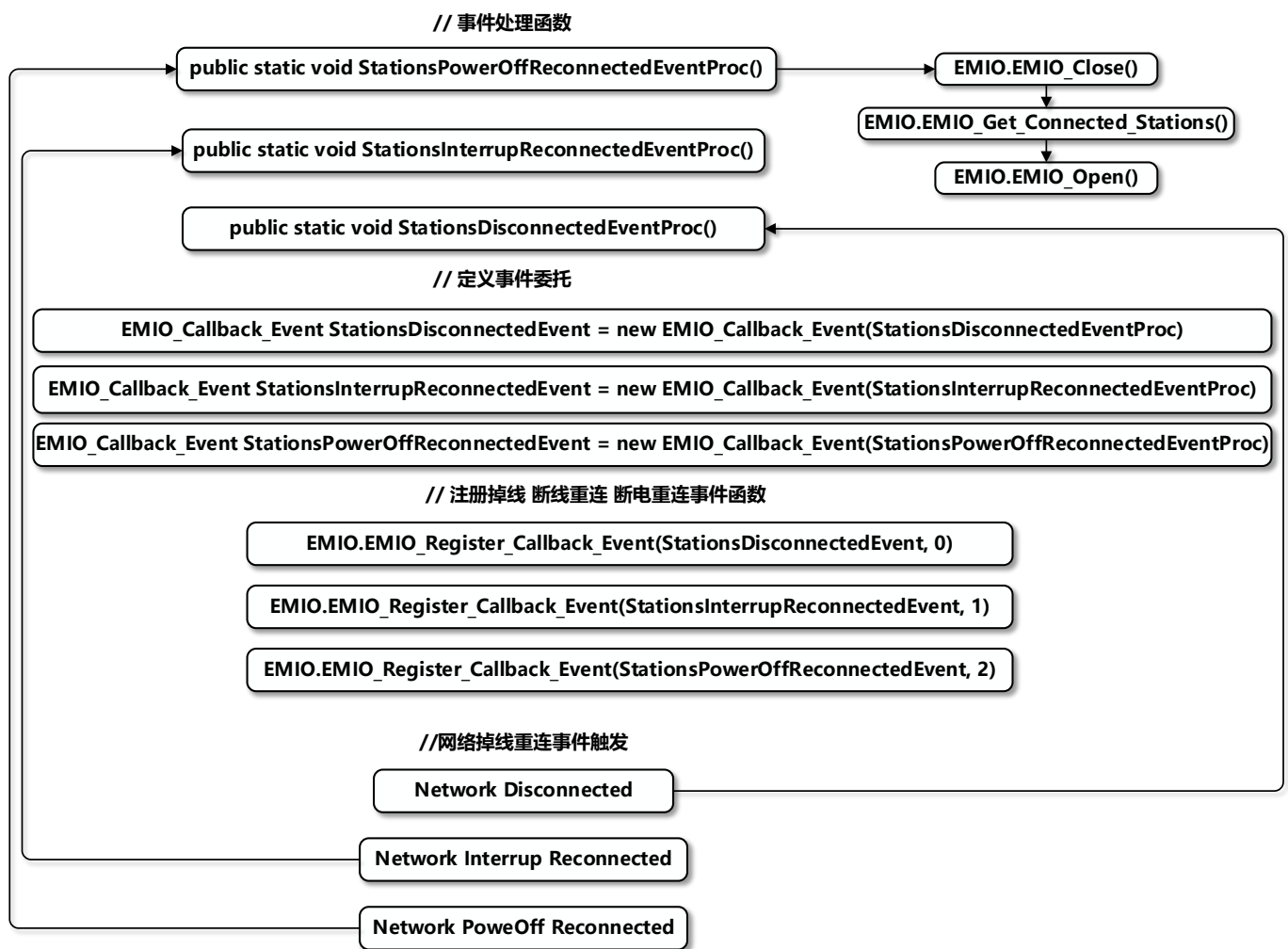


图 2.2.2.5 VC# 断线/重连事件使用流程

## 2.2.3 Visual Basic.NET Windows 窗体应用程序开发

1. 启动 Visual Studio 2010，文件-新建-项目-Visual Basic，选择 Windows 窗体应用程序。

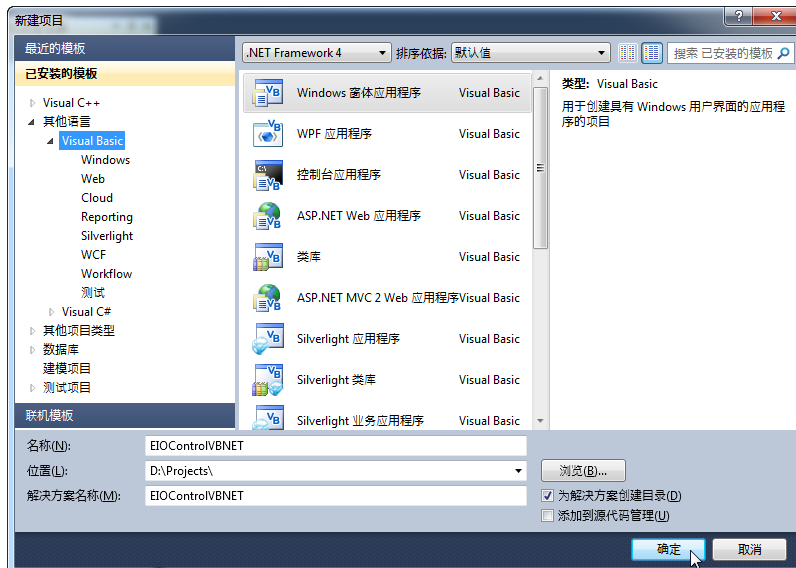


图 2.2.3.1 新建 Windows 窗体应用程序

2. 项目-添加现有项，选择 C:\Program Files\EMIO\Include\YSEMIO.vb 文件。

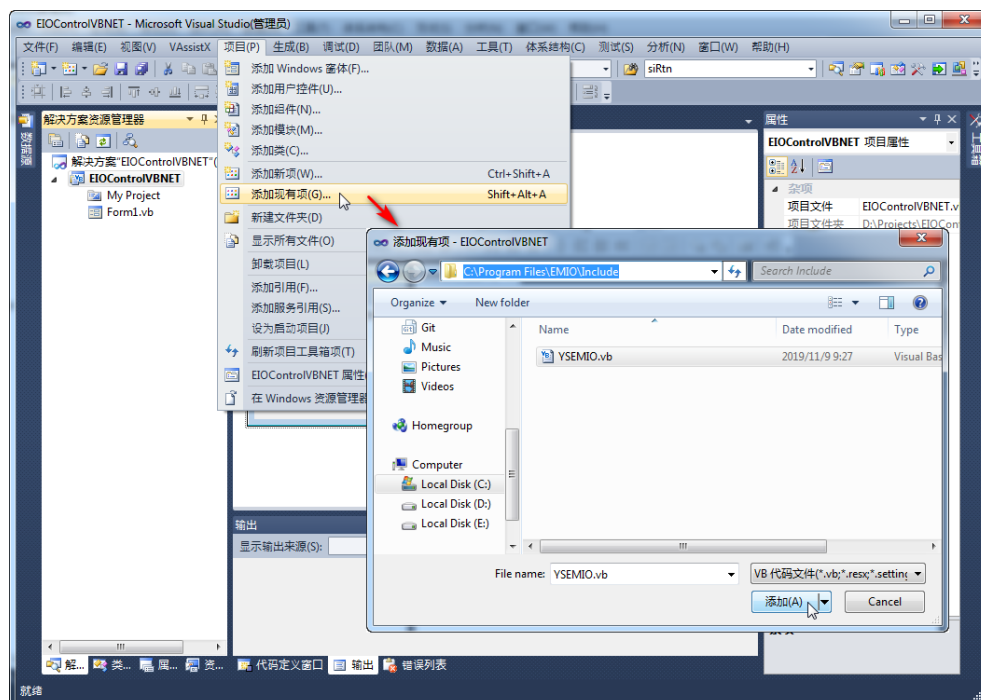


图 2.2.3.2 添加现有项 YSEMIO.vb



到此，就可以调用 EMIO API 函数库中的任何函数，开始编写应用程序了。

3. EMIO API 函数库的一般使用流程与图 2.2.1.5 VC++ 函数库使用流程一致。

4. EMIO API 函数库的断线/重连事件使用流程如下：

- (1). 自定义断线/重连事件处理函数。
- (2). 定义并新建断线/重连事件委托。
- (3). 在打开模块后注册已定义的断线/重连事件委托。
- (4). 网络断线/重连后自动执行断线/重连事件处理函数一次。

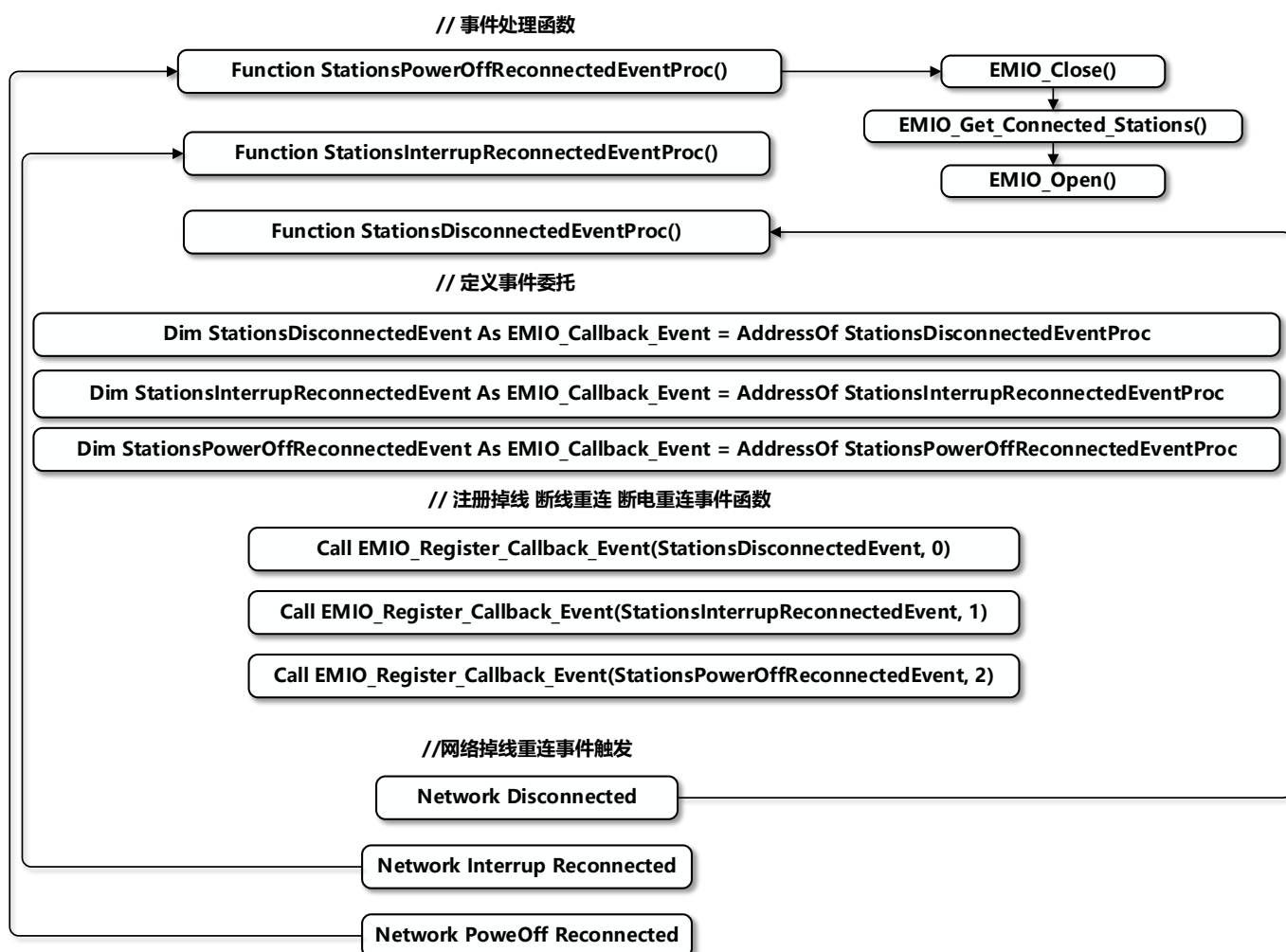


图 2.2.3.3 VB.NET 断线/重连事件使用流程

## 第 3 章 EMIO 数字 IO 模块 API 函数

### 3.1 模块系统函数

#### 3.1.1 EMIO\_Get\_Connected\_Stations()

获取所有已经实际连接上的从站模块的总数及其类型数组。

EMIO_Get_Connected_Stations			
函数原型	short EMIO_Get_Connected_Stations( unsigned short *Connected_Numbers, unsigned short *Connected_Station_Types, unsigned short Unicode = 1 );		
说明	获取所有已经实际连接上的从站模块的总数目 Connected_Numbers 及其类型 Connected_Station_Types 数组指针。类型定义如下： ( 0 : StationTypeIO2416 ) ( 1 : StationTypeIO4832 )		
原型参数			
类型	名称	输入/输出	说明
unsigned short*	Connected_Numbers	输出	返回实际连接的从站模块总数目
unsigned short*	Connected_Station_Types	输出	返回实际连接的所有从站模块的类型数组指针
unsigned short	Unicode	输入	程序是否使用 Unicode 字符集
C++ 示例	unsigned short uiStationNums = 0; unsigned short uiSationTypes[56] = {0}; //还不知道实际连接的从站模块数目 //获取所有已经实际连接上的从站模块的总数及其类型数组 short siRtn = EMIO_Get_Connected_Stations(&uiStationNums, uiSationTypes);		
C# 示例	ushort Station_Nums = 0; ushort[] Station_Types = new ushort[56]; short siRtn = EMIO.EMIO_Get_Connected_Stations(ref Station_Nums, ref Station_Types[0]);		

### 3.1.2 EMIO\_Get\_Connected\_Station\_Nums()

获取所有已经实际连接上的从站模块的总数目。

EMIO_Get_Connected_Station_Nums	
函数原型	short EMIO_Get_Connected_Station_Nums();
说明	获取并返回所有已经实际连接上的从站模块的总数目
C++ 示例	short iStationNums = EMIO_Get_Connected_Station_Nums();
C# 示例	short iStationNums = EMIO.EMIO_Get_Connected_Station_Nums();

### 3.1.3 EMIO\_Get\_Connected\_Station\_Type()

获取指定站点 ID 序号从站模块的类型数组。

EMIO_Get_Connected_Station_Type			
函数原型	short EMIO_Get_Connected_Station_Type(unsigned short Station_Number);		
说明	根据从站模块的站点 ID 序号 Station_Number 获取并返回该模块的类型。类型定义如下：（ 0 : StationTypeIO2416 ）（ 1 : StationTypeIO4832 ）		
原型参数			
类型	名称	输入/输出	说明
unsigned short	Station_Number	输入	指定从站模块的站点 ID 序号
C++ 示例	<pre>//首先获取已经实际连接上的从站模块的总数目 unsigned short uiStationNums = EMIO_Get_Connected_Station_Nums(); //根据从站模块的总数目分配存储所有从站类型的数组 unsigned short* puiStationType = new unsigned short[uiStationNums]; for(unsigned short idx=0; idx&lt;uiStationNums; idx++) {     //根据从站模块的站点 ID 序号 Station_Number 获取并返回该模块的类型。     puiStationType[idx] = EMIO_Get_Connected_Station_Type (idx); } delete puiStationType;</pre>		
C# 示例	<pre>ushort uiStationNums = EMIO.EMIO_Get_Connected_Station_Nums(); ushort[] puiStationType = new ushort[uiStationNums];</pre>		

	<pre> for(ushort idx=0; idx&lt;uiStationNums; idx++) {     puiStationType[idx] = EMIO.EMIO_Get_Connected_Station_Type(idx); } </pre>
--	--

### 3.1.4 EMIO\_Open()

打开总线从站模块。

EMIO_Open			
函数原型	short EMIO_Open(unsigned short Connection_Numbers, unsigned short *Station_Types);		
说明	打开并连接指定数目 Connection_Numbers 和指定模块类型 Station_Types 数组的总线从站模块		
原型参数			
类型	名称	输入/输出	说明
unsigned short	Connection_Numbers	输入	指定需要打开的所有从站模块的总数目。 注意这个总数目不一定与实际连接的所有从站模块的总数目相同，最好由 EMIO_Get_Connected_Station_Nums()获取实际连接上的从站模块的总数目
unsigned short*	Station_Types	输入	指定需要打开的所有从站模块的类型数组指针
C++ 示例	unsigned short iStationNums = 0; //这时还不知道实际连接的从站模块数目，预定义 56 个 unsigned short iSationTypes[56] = {0}; //获取所有已经实际连接上的从站模块的总数目及其类型数组 short siRtn = EMIO_Get_Connected_Stations(&iStationNums, iSationTypes); //打开所有从站模块 siRtn = EMIO_Open(iStationNums, iSationTypes);		
C# 示例	ushort Station_Nums = 0; ushort[] Station_Types = new ushort[56]; //获取所有已经实际连接上的从站模块的总数目及其类型数组		

	<pre>short siRtn = EMIO.EMIO_Get_Connected_Stations(ref Station_Nums, ref Station_Types[0]); //打开所有从站模块 siRtn = EMIO.EMIO_Open(Station_Nums, ref Station_Types[0]);</pre>
--	---

### 3.1.5 EMIO\_OpenEx()

打开总线从站模块扩展。

EMIO_OpenEx			
函数原型	short EMIO_OpenEx(unsigned short Connection_Numbers , unsigned short *Station_Numbers, unsigned short *Station_Types);		
说明	打开并连接指定数目 Connection_Numbers，指定从站模块的站点 ID 序号 Station_Numbers 数组，指定模块类型 Station_Types 数组的总线从站模块。		
原型参数			
类型	名称	输入/输出	说明
unsigned short	Connection_Numbers	输入	指定需要打开的所有从站模块的总数目。 注意这个总数目不一定与实际连接的所有从站模块的总数目相同，最好由 EMIO_Get_Connected_Station_Nums()获取实际连接上的从站模块的总数目。
unsigned short*	Station_Numbers	输入	所有从站模块的站点 ID 序号数组指针
unsigned short*	Station_Types	输入	所有从站模块的类型数组指针
C++ 示例	unsigned short iStationNums = 2; //指定需要打开 2 个从站模块 unsigned short iSationNumbers[2] = {0, 1}; //指定从站站点 ID 序号 unsigned short iSationTypes[2] = {StationTypeIO2416, StationTypeIO4832}; //打开 2 个站点号分别为 {0, 1},类型分别为 //{StationTypeIO2416,StationTypeIO4832}的从站模块 short siRtn = EMIO_OpenEx(iStationNums, iSationNumbers, iSationTypes);		
C# 示例	ushort Station_Nums = 2; //指定需要打开 2 个从站模块 ushort[] Station_Numbers = new ushort[]{0,1}; //指定从站站点 ID 序号 //指定从站站点类型		

```

ushort[] Station_Types  = new ushort[]
{ (ushort)StationType.StationTypeIO2416,
  (ushort)StationType.StationTypeIO4832 };
ushort siRtn = EMIO.EMIO_OpenEx(Station_Nums, ref Station_Numbers[0],
ref Station_Types[0]); // 打开总线从站模块

```

### 3.1.6 EMIO\_Close()

关闭总线从站模块。

EMIO_Close	
函数原型	short EMIO_Close();
说明	关闭总线上所有从站模块。
C++示例	short siRtn = EMIO_Close();
C# 示例	short siRtn = EMIO.EMIO_Close();

### 3.2 模块通用数字输入输出函数

#### 3.2.1 EMIO\_Get\_Input()

获取指定站点的数字输入逻辑状态。

EMIO_Get_Input			
函数原型	short EMIO_Get_Input(unsigned long *All_Input_Logic1, unsigned long *All_Input_Logic2, unsigned short Station_Number);		
说明	获取指定站点 ID 序号 Station_Number 从站模块的所有数字输入状态并返回到 All_Input_Logic1(Input31~ Input0)和 All_Input_Logic2(Input47~ Input32)。注意：返回值是按位对应的，如返回 65536 即 0xffff（二进制 1111 1111 1111 1111）即 16 个 DI 均为 1，16 个 LED 灯均为熄灭状态。		
原型参数			
类型	名称	输入/输出	说明
unsigned long*	All_Input_Logic1	输出	按位返回数字输入状态(0 ~0xffffffff)。
unsigned long*	All_Input_Logic2	输出	按位返回数字输入状态(0 ~0xffffffff)。
unsigned short	Station_Number	输入	从站模块的站点 ID 序号。
C++示例	unsigned long All_Input_Logic1 = All_Input_Logic2 = 0; short siRtn = 0; //获取站点号 gStation_Number[0]的 EIO2416 所有数字输入 DI 状态 //All_Input_Logic1(DI31 ~ DI0)和 All_Input_Logic2( DI47~ DI32)按位对应。 siRtn = EMIO_Get_Input ( & All_Input_Logic1 ,& All_Input_Logic2 , gStation_Number[0] );		
C# 示例	uint AllOutputLogic1=0, AllOutputLogic2 = 0; short siRtn = 0; //获取站点号 gStation_Number[0]的 EIO2416 所有数字输入 DI 状态 //All_Input_Logic1(DI31 ~ DI0)和 All_Input_Logic2( DI47~ DI32)按位对应。 siRtn = EMIO.EMIO_Get_Input(ref AllOutputLogic1, ref AllOutputLogic2, gStation_Numbers[0]);		

3.2.2 EMIO\_Get\_Input\_Bit()

获取指定输入位和指定站点的数字输入位逻辑状态。

EMIO_Get_Input_Bit			
函数原型	short EMIO_Get_Input_Bit ( unsigned short Bit_Input_Number, unsigned short *Bit_Input_Logic, unsigned short Station_Number );		
说明	获取指定输入位 Bit_Input_Number 和指定站点 ID 序号 Station_Number 从站模块,的输入逻辑状态 Bit_Input_Logic。注意：输入位 Bit_Input_Number 是按值对应设输入的，如值为 15 即获取 DI 15 操作。		
原型参数			
类型	名称	输入/输出	说明
unsigned short	Bit_Input_Number,	输入	按值指定数字输入位(0 ~ 31)。
unsigned short*	Bit_Input_Logic	输出	0: LED 灯亮状态 1: LED 灯熄状态
unsigned short	Station_Number	输入	从站模块的站点 ID 序号。
C++ 示例	unsigned short Bit_Logic = 0; short siRtn = 0; //获取站点号 gStation_Number[0]的 EIO2416 的 DI15 的输入状态 siRtn = EMIO_Get_Input_Bit (15, &Bit_Logic, gStation_Number[0]); //设置站点号 gStation_Number[0]的 EIO2416 的 DI0 的输入状态 siRtn = EMIO_Get_Input_Bit (0, &Bit_Logic,, gStation_Number[0]); //设置站点号 gStation_Number[1]的 EIO4832 的 DI31 的输入状态 siRtn = EMIO_Get_Input_Bit ( 31, &Bit_Logic, gStation_Number[1] ); //设置站点号 gStation_Number[1]的 EIO4832 的 DI1 的输入状态 siRtn = EMIO_Get_Input_Bit ( 1, &Bit_Logic, gStation_Number[1] );		
C# 示例	ushort Bit_Logic = 0; short siRtn = 0; //获取站点号 gStation_Number[0]的 EIO2416 的 DI15 的输入状态 siRtn = EMIO.EMIO_Get_Input_Bit(15, ref Bit_Logic ,gStation_Numbers[0]); //设置站点号 gStation_Number[1]的 EIO4832 的 DI31 的输入状态 siRtn = EMIO.EMIO_Get_Input_Bit(31, ref Bit_Logic ,gStation_Numbers[1]);		



### 3.2.3 EMIO\_Get\_Output()

获取指定站点的数字输出逻辑状态。

EMIO_Get_Output			
函数原型	short EMIO_Get_Output(unsigned long* All_Output_Logic, unsigned short Station_Number);		
说明	获取指定站点 ID 序号 Station_Number 从站模块的数字输出状态并返回到 All_Output_Logic。注意：返回值是按位对应的，如返回 65536 即 0xffff（二进制 1111 1111 1111 1111）即 16 个 DO 均为 1，16 个 LED 灯均为熄灭状态。		
原型参数			
类型	名称	输入/输出	说明
unsigned long	All_Output_Logic	输出	返回数字输出状态(0 ~0xffffffff)。
unsigned short	Station_Number	输入	从站模块的站点 ID 序号。
C++ 示例	unsigned long All_Output_Logic = 0; short siRtn = 0; //获取站点号 gStation_Numbers[0]的 EIO2416 输出状态到 All_Output_Logic //如 All_Output_Logic = 0xfffe EIO2416 DO0 状态为 0, DO1-DO15 状态为 1 siRtn = EMIO_Get_Output( &All_Output_Logic , gStation_Numbers[0] ); //获取站点号 gStation_Numbers[1]的 EIO4832 输出状态到 All_Output_Logic //如 All_Output_Logic=0xffffffffe EIO4832 DO0 状态为 0, DO1-DO31 状态为 1 siRtn = EMIO_Get_Output( &All_Output_Logic , gStation_Numbers[1] );		
C# 示例	uint AllOutputLogic = 0; short siRtn = 0; //获取站点号 gStation_Numbers[0]的 EIO2416 输出状态到 All_Output_Logic //如 All_Output_Logic = 0xfffe EIO2416 DO0 状态为 0, DO1-DO15 状态为 1 siRtn = EMIO.EMIO_Get_Output(ref AllOutputLogic, gStation_Numbers[0]); //获取站点号 gStation_Numbers[1]的 EIO4832 输出状态到 All_Output_Logic //如 All_Output_Logic=0xffffffffe EIO4832 DO0 状态为 0, DO1-DO31 状态为 1 siRtn = EMIO.EMIO_Get_Output(ref AllOutputLogic, gStation_Numbers[1]);		

### 3.2.4 EMIO\_Get\_Output\_Bit()

获取指定输出位和指定站点的数字输出位逻辑状态。

EMIO_Get_Output_Bit			
函数原型	short EMIO_Get_Output_Bit (unsigned short Bit_Output_Number, unsigned short* Bit_Output_Logic, unsigned short Station_Number );		
说明	获取指定输出位 Bit_Output_Number 和指定站点 ID 序号 Station_Number 的从站模块的输出逻辑状态 Bit_Output_Logic 。注意：指定输出位 Bit_Output_Number 是按值对应的，如值为 15 即获取 DO15 状态。		
原型参数			
类型	名称	输入/输出	说明
unsigned short	Bit_Output_Number,	输入	按值指定数字输出位(0 ~ 31)。
unsigned short*	Bit_Output_Logic	输出	0: LED 灯亮状态 1: LED 灯熄状态
unsigned short	Station_Number	输入	从站模块的站点 ID 序号。
C++ 示例	unsigned short Bit_Logic = 0; short siRtn = 0; //获取站点号 gStation_Number[0]的 EIO2416 的 DO15 的输入状态 siRtn = EMIO_Get_Output_Bit (15, &Bit_Logic, gStation_Numbers[0]); //获取站点号 gStation_Number[0]的 EIO2416 的 DO0 的输入状态 siRtn = EMIO_Get_Output_Bit (0, &Bit_Logic,, gStation_Numbers[0]); //获取站点号 gStation_Number[1]的 EIO4832 的 DO31 的输入状态 siRtn = EMIO_Get_Output_Bit ( 31, &Bit_Logic, gStation_Numbers[1] ); //获取站点号 gStation_Number[1]的 EIO4832 的 DO1 的输入状态 siRtn = EMIO_Get_Output_Bit ( 1, &Bit_Logic, gStation_Numbers[1] );		
C# 示例	ushort Bit_Logic = 0; short siRtn = 0; //获取站点号 gStation_Number[0]的 EIO2416 的 DO15 的输入状态 EMIO.EMIO_Get_Output_Bit(15, ref Bit_Logic, gStation_Numbers[0]); //获取站点号 gStation_Number[1]的 EIO4832 的 DO31 的输入状态 EMIO.EMIO_Get_Output_Bit(31, ref Bit_Logic, gStation_Numbers[1]);		

3.2.5 EMIO\_Set\_Output()

设置指定站点的数字输出逻辑。

EMIO_Set_Output			
函数原型	short EMIO_Set_Output(unsigned long All_Output_Logic, unsigned short Station_Number);		
说明	设置指定站点 ID 序号 Station_Number 从站模块的数字 IO 输出状态 All_Output_Logic。注意： All_Output_Logic 是按位对应设输出的，如 16 个 DO 均为 1 即 0xffff（二进制 1111 1111 1111 1111） 其值为 65536，熄灭 16 个 LED 灯。		
原型参数			
类型	名称	输入/输出	说明
unsigned long	All_Output_Logic	输入	按位设置的数字 IO 输出状态(0 ~0xffffffff)。
unsigned short	Station_Number	输入	从站模块的站点 ID 序号。
C++示例	short siRtn = 0; //设置站点号 gStation_Number[0]的 EIO2416 的 16DO 输出为 0 点亮 LED 灯 siRtn = EMIO_Set_Output( 0x0, gStation_Numbers[0] ); //设置站点号 gStation_Number[0]的 EIO2416 的 16DO 输出为 1 熄灭 LED 灯 siRtn = EMIO_Set_Output( 0xffff, gStation_Numbers[0] ); //设置站点号 gStation_Number[1]的 EIO4832 DO0 输出为 0 DO1-DO31 输出为 1 siRtn = EMIO_Set_Output( 0xffffffe, gStation_Numbers[1] ); //设置站点号 gStation_Number[1]的 EIO4832 32DO 输出为 1 熄灭 LED 灯 siRtn = EMIO_Set_Output( 0xffffffff, gStation_Numbers[1] );		
C# 示例	short siRtn = 0; //设置站点号 gStation_Number[0]的 EIO2416 的 16DO 输出为 0 点亮 LED 灯 siRtn = EMIO.EMIO_Set_Output(0x0, gStation_Numbers[0]); //设置站点号 gStation_Number[0]的 EIO2416 的 16DO 输出为 1 熄灭 LED 灯 siRtn = EMIO.EMIO_Set_Output(0xffff, gStation_Numbers[0]); //设置站点号 gStation_Number[1]的 EIO4832 DO0 输出为 0, DO1-DO31 输出为 1 siRtn = EMIO.EMIO_Set_Output(0xffffffe, gStation_Numbers[1]); //设置站点号 gStation_Number[1]的 EIO4832 32DO 输出为 1 熄灭 LED 灯 siRtn = EMIO.EMIO_Set_Output(0xffffffff, gStation_Numbers[1]);		

### 3.2.6 EMIO\_Set\_Output\_Bit()

设置指定输出位和指定站点的数字输出位逻辑。

EMIO_Set_Output_Bit			
函数原型	short EMIO_Set_Output_Bit (unsigned short Bit_Output_Number, unsigned short Bit_Output_Logic, unsigned short Station_Number );		
说明	设置指定输出位 Bit_Output_Number 和指定站点 ID 序号 Station_Number 的从站模块的输出逻辑 Bit_Output_Logic 。注意：指定输出位 Bit_Output_Number 是按值对应设输出的，比如值为 15 即对 DO15 操作。		
原型参数			
类型	名称	输入/输出	说明
unsigned short	Bit_Output_Number,	输入	按值指定数字输出位(0 ~ 31)。
unsigned short	Bit_Output_Logic	输入	0: 点亮 LED 灯 1: 熄灭 LED 灯
unsigned short	Station_Number	输入	从站模块的站点 ID 序号。
C++示例	short siRtn = 0; //设置站点号 gStation_Numbers[0]的 EIO2416 的 DO15 输出为 0 点亮 LED 灯 siRtn = EMIO_Set_Output_Bit (15, 0, gStation_Numbers[0]); //设置站点号 gStation_Numbers[0]的 EIO2416 的 DO0 输出为 1 熄灭 LED 灯 siRtn = EMIO_Set_Output_Bit (0, 1, gStation_Numbers[0]); //设置站点号 gStation_Numbers[1]的 EIO4832 的 DO31 输出为 0 点亮 LED 灯 siRtn = EMIO_Set_Output_Bit (31, 0, gStation_Numbers[1] ); //设置站点号 gStation_Numbers[1]的 EIO4832 的 DO1 输出为 1 熄灭 LED 灯 siRtn = EMIO_Set_Output_Bit (1, 1, gStation_Numbers[1] );		
C# 示例	short siRtn = 0; //设置站点号 gStation_Numbers[0]的 EIO2416 的 DO15 输出为 0 点亮 LED 灯 siRtn = EMIO.EMIO_Set_Output_Bit(15, 0, gStation_Numbers[0] ); //设置站点号 gStation_Numbers[1]的 EIO4832 的 DO31 输出为 0 点亮 LED 灯 siRtn = EMIO.EMIO_Set_Output_Bit (31, 0, gStation_Numbers[1] );		

3.2.7 EMIO\_Inv\_Output\_Bit()

反向设置指定输出位和指定站点的数字输出位逻辑。

EMIO_Inv_Output_Bit			
函数原型	short EMIO_Inv_Output_Bit (unsigned short Bit_Output_Number, unsigned short Station_Number );		
说明	反向设置指定输出位 Bit_Output_Number 和指定站点 ID 序号 Station_Number 的从站模块的数字输出状态。注意：反向输出位 Bit_Output_Number 是按值对应设输出的，如值为 15 即对 DO15 操作。		
原型参数			
类型	名称	输入/输出	说明
unsigned short	Bit_Output_Number,	输入	按值指定反向数字输出位(0 ~ 31)。。
unsigned short	Station_Number	输入	从站模块的站点 ID 序号。
C++示例	short siRtn = 0; //反向站点号 gStation_Numbers[0]的 EIO2416 的 DO15 的输出 原 LED 灯亮变灭 siRtn = EMIO_Inv_Output_Bit (15, gStation_Numbers[0]); //反向站点号 gStation_Numbers[1]的 EIO4832 的 DO31 的输出 原 LED 灯灭变亮 siRtn = EMIO_Inv_Output_Bit (31, gStation_Numbers[1]);		
C# 示例	short siRtn = 0; //反向站点号 gStation_Numbers[0]的 EIO2416 的 DO15 的输出 原 LED 灯亮变灭 siRtn = EMIO.EMIO_Inv_Output_Bit (15, gStation_Numbers[0]); //反向站点号 gStation_Numbers[1]的 EIO4832 的 DO31 的输出 原 LED 灯灭变亮 siRtn = EMIO.EMIO_Inv_Output_Bit (31, gStation_Numbers[1]);		

## 3.3 模块实用函数

### 3.3.1 EMIO\_Get\_First\_Error\_Message()

获取系统出现的第一个错误信息。

EMIO_Get_First_Error_Message			
函数原型	short  EMIO_Get_First_Error_Message( LPTSTR Error_Message, unsigned short Unicode = 1 );		
说明	根据指定字符集类型获取系统出现的第一个错误信息字符数组。Unicode 类型定义如下：( 0 : 非 Unicode ) ( 1 : Unicode )		
原型参数			
类型	名称	输入/输出	说明
LPTSTR	Error_Message	输出	返回系统出现的第一个错误信息
unsigned short	Unicode	输入	程序是否使用 Unicode 字符集
C++ 示例	TCHAR cErrMsg[256]; short siRtn = EMIO_Get_First_Error_Message ( cErrMsg , 0 );		
C# 示例	StringBuilder ErrorMsg = new StringBuilder("", 100); short siRtn = EMIO. EMIO_Get_First_Error_Message ( ErrorMsg, 0 );		

### 3.3.2 EMIO\_Get\_Last\_Error\_Message()

获取系统出现的最后一个错误信息。

EMIO_Get_Last_Error_Message			
函数原型	short   EMIO_Get_Last_Error_Message( LPTSTR Error_Message, unsigned short Unicode = 1 );		
说明	根据指定字符集类型获取系统出现的最后一个错误信息字符数组。Unicode 类型定义如下：（ 0 : 非 Unicode ）（ 1 : Unicode ）		
原型参数			
类型	名称	输入/输出	说明

LPTSTR	Error_Message	输出	返回系统出现的最后一个错误信息
unsigned short	Unicode	输入	程序是否使用 Unicode 字符集
<b>C++ 示例</b>	TCHAR cErrMsg[256]; short siRtn = EMIO_Get_Last_Error_Message ( cErrMsg , 0 );		
<b>C# 示例</b>	StringBuilder ErrorMsg = new StringBuilder("", 100); short siRtn = EMIO.EMIO_Get_Last_Error_Message ( ErrorMsg, 0 );		

### 3.3.3 EMIO\_Get\_Error\_Message()

根据指定错误代码和指定字符集获取具体错误信息。

EMIO_Get_Error_Message			
函数原型	short  EMIO_Get_Error_Message ( LPTSTR Error_Message, short Error_Code, unsigned short Unicode = 1 );		
说明	根据错误代码 Error_Code 和字符集 Unicode 获取具体错误信息字符数组。 Unicode 类型定义如下：（ 0：非 Unicode ）（ 1：Unicode ）		
原型参数			
类型	名称	输入/输出	说明
LPTSTR	Error_Message	输出	返回具体的错误信息
short	Error_Code	输入	具体的错误代码
unsigned short	Unicode	输入	程序是否使用 Unicode 字符集
C++ 示例	TCHAR cErrMsg[256]; short siRtn = EMIO_Get_Error_Message( cErrMsg , -18, 0 );		
C# 示例	StringBuilder ErrorMsg = new StringBuilder("", 100); short siRtn = EMIO.EMIO_Get_Error_Message( ErrorMsg, -18, 0 );		

### 3.3.4 EMIO\_Get\_SN()

获取指定站点的硬件序列号信息。

EMIO_Get_SN			
函数原型	short EMIO_Get_SN ( INT64 *Serial_Number, unsigned short Station_Number);		
说明	获取指定站点 Station_Number 的硬件序列号信息返回到 Serial_Number 变量中。		
原型参数			
类型	名称	输入/输出	说明
INT64 *	Serial_Number	输出	返回硬件序列号，维一的 64 位硬件码
unsigned short	Station_Number	输入	从站模块的站点 ID 序号。
C++ 示例	INT64 ISN =0; short siRtn = EMIO_Get_SN(&ISN, Station_Numbers[0]);		
C# 示例	Int64 ISN = 0; short siRtn = EMIO.EMIO_Get_SN(ref ISN, gStation_Numbers[0]);		

### 3.3.5 EMIO\_Get\_SN\_Text()

获取指定站点的硬件序列号信息字符数组。

EMIO_Get_SN_Text			
函数原型	short EMIO_Get_SN_Text (LPTSTR Serial_Number, unsigned short Station_Number, unsigned short Unicode = 1);		
说明	获取指定站点 Station_Number 的硬件序列号返回到 Serial_Number 字符数组中。		
原型参数			
类型	名称	输入/输出	说明
LPTSTR	Serial_Number	输出	返回硬件序列号，维一的 64 位硬件码
unsigned short	Station_Number	输入	从站模块的站点 ID 序号。
unsigned short	Unicode	输入	程序是否使用 Unicode 字符集
C++ 示例	TCHAR pszStationSN[MAX_PATH] = {0}; short siRtn = EMIO_Get_SN_Text ( pszStationSN , Station_Numbers[0] );		
C# 示例	StringBuilder SN = new StringBuilder("", 100); short siRtn = EMIO.EMIO_Get_SN_Text(SN, 0);		



### 3.3.6 EMIO\_Get\_Station\_Info()

获取指定站点的站点信息。

EMIO_Get_Station_Info			
函数原型	short EMIO_Get_Station_Info (unsigned long * Station_Info , unsigned short Station_Number);		
说明	获取指定站点 Station_Number 的站点信息返回到 Station_Info 数组中。		
原型参数			
类型	名称	输入/输出	说明
unsigned long *	Station_Info	输出	返回站点信息(类型 固件版本)
unsigned short	Station_Number	输入	从站模块的站点 ID 序号。
C++ 示例	unsigned long ulStationInfo = 0; short siRtn = EMIO_Get_Station_Info( ulStationInfo , Station_Numbers[0] );		
C# 示例	uint ulStationInfo = 0; short siRtn = EMIO.EMIO_Get_Station_Info(ref ulStationInfo, gStation_Numbers[0]);		

### 3.3.7 EMIO\_Get\_Dll\_Version()

获取软件动态库版本信息。

EMIO_Get_Dll_Version			
函数原型	short EMIO_Get_Dll_Version (LPTSTR DllVersion);		
说明	获取软件动态库版本信息返回到 DllVersion 字符数组中。		
原型参数			
类型	名称	输入/输出	说明
LPTSTR	DllVersion	输出	返回软件动态库版本信息
C++示例	TCHAR pszFileVersion[MAX_PATH] = {0}; short siRtn = EMIO_Get_Dll_Version(pszFileVersion);		

<b>C# 示例</b>	<pre>StringBuilder Version = new StringBuilder("", 100); short siRtn = EMIO.EMIO_Get_Dll_Version(Version);</pre>
--------------	--

### 3.3.8 EMIO\_Get\_Language()

获取软件动态库错误信息语言类型。

EMIO_Get_Language	
函数原型	short EMIO_Get_Language ();
说明	获取并返回软件动态库错误信息语言类型。( 0 : 简体中文 ) ( 1 : 英文 )
C++ 示例	short siLanguage = EMIO_Get_Language();
C# 示例	short siLanguage = EMIO.EMIO_Get_Language();

### 3.3.9 EMIO\_Set\_Language()

设置软件动态库错误信息语言类型。

EMIO_Set_Language			
函数原型	short EMIO_Set_Language ( short Language );		
说明	设置软件动态库错误信息语言类型。( 0 : 简体中文 ) ( 1 : 英文 )		
原型参数			
类型	名称	输入/输出	说明
short	Language	输入	错误信息语言类型
C++ 示例	short siRtn = EMIO_Set_Language(0);		
C# 示例	short siRtn = EMIO.EMIO_Set_Language(1);		

### 3.3.10 EMIO\_Get\_Process\_Type()

获取进程的类型。

EMIO_Get_Process_Type	
函数原型	short EMIO_Get_Process_Type ();
说明	获取并返回进程的类型。(0 : 32 位进程) ( 1 : 64 位进程 )
C++ 示例	short siProcessType = EMIO_Get_Process_Type();
C# 示例	short siProcessType = EMIO.EMIO_Get_Process_Type();

### 3.3.11 EMIO\_Register\_Callback\_Event()

注册断线，重连或断电回调事件。

EMIO_Register_Callback_Event			
函数原型	short EMIO_Register_Callback_Event ( PFEMIO_Callback_Event CallbackFunc, unsigned short EventType );		
说明	注册断线或重连回调事件。		
原型参数			
类型	名称	输入/输出	说明
PFEMIO_Callback_Event	CallbackFunc	输入	回调事件函数指针。
unsigned short	EventType	输入	回调事件类型。 0: YSEventStationsDisconnected 1: YSEventStationsInterrupReconnected 2: YSEventStationsPowerOffReconnected
C++示例	//1.自定义一个被回调函数 short WINAPI CallbackEventStationsDisconnected() { CString sDisconnected= (CTime::GetCurrentTime()).Format(_T("模块已断线 %Y%m%d %H%M%S")); return 0; }  //2.在打开从站模块后 注册被回调函数及其类型		

	EMIO_Register_Callback_Event(CallbackEventStationsDisconnected, YSEventStationsDisconnected);
<b>C# 示例</b>	<pre>//1.自定义一个事件处理函数 public static void StationsDisconnectedEventProc() {     String sDisconnected = "模块已断线"; } //2.声明并新建一个事件委托及其处理函数 EMIO_Callback_Event StationsDisconnectedEvent = new EMIO. EMIO_Callback_Event ( StationsDisconnectedEventProc ); //3.在打开从站模块后 注册事件委托及其类型 EMIO.EMIO_Register_Callback_Event( StationsDisconnectedEvent , 0);</pre>

### 3.3.12 EMIO\_Get\_Disconnect\_Counts()

获取网络掉线的次数。

EMIO_Get_Disconnect_Counts	
函数原型	short EMIO_Get_Disconnect_Counts ();
说明	获取网络掉线的次数。
C++示例	short uiDisCnts = EMIO_Get_Disconnect_Counts();
C# 示例	short uiDisCnts = EMIO.EMIO_Get_Disconnect_Counts();

### 3.3.13 EMIO\_Set\_Disconnect\_Counts()

设置网络掉线的次数。

EMIO_Set_Disconnect_Counts	
函数原型	short EMIO_Set_Disconnect_Counts (short Disconnect_Counts );
说明	设置网络掉线的次数。

原型参数			
类型	名称	输入/输出	说明
short	Disconnect_Counts	输入	设置网络掉线的次数，清零
<b>C++示例</b>	short siRtn = EMIO_Set_Disconnect_Counts(0);		
<b>C# 示例</b>	short siRtn = EMIO.EMIO_Set_Disconnect_Counts(0);		

### 3.3.14 EMIO\_Get\_Interrup\_Reconnect\_Counts()

获取网络中断掉线后重新连接的次数。

EMIO_Get_Interrup_Reconnect_Counts	
函数原型	short EMIO_Get_Interrup_Reconnect_Counts ();
说明	获取网络中断后重新连接的次数。
<b>C++示例</b>	short uiInterrupReCnts = EMIO_Get_Interrup_Reconnect_Counts();
<b>C# 示例</b>	short uiInterrupReCnts = EMIO.EMIO_Get_Interrup_Reconnect_Counts();

### 3.3.15 EMIO\_Set\_Interrup\_Reconnect\_Counts()

设置网络中断掉线后重新连接的次数。

EMIO_Set_Interrup_Reconnect_Counts			
函数原型	short EMIO_Set_Interrup_Reconnect_Counts ( short Interrup_Reconnect_Counts);		
说明	设置网络中断掉线后重新连接的次数。		
原型参数			
类型	名称	输入/输出	说明
short	Interrup_Reconnect_Counts	输入	设置网络中断掉线后重连的次数，清零
C++示例	short siRtn = EMIO_Set_Interrup_Reconnect_Counts(0);		
C# 示例	short siRtn = EMIO.EMIO_Set_Interrup_Reconnect_Counts(0);		

### 3.3.16 EMIO\_Get\_PowerOff\_Reconnect\_Counts()

获取指定站点模块断电后重新连接的次数。

EMIO_Get_PowerOff_Reconnect_Counts			
函数原型	short EMIO_Get_PowerOff_Reconnect_Counts (unsigned short Station_Number);		
说明	获取指定站点模块断电后重新连接的次数。		
原型参数			
类型	名称	输入/输出	说明
unsigned short	Station_Number	输入	从站模块的站点 ID 序号。
C++ 示例	short uiReCnts = EMIO_Get_PowerOff_Reconnect_Counts(0);		
C# 示例	short uiReCnts = EMIO.EMIO_Get_PowerOff_Reconnect_Counts(1);		

### 3.3.17 EMIO\_Set\_PowerOff\_Reconnect\_Counts()

设置指定站点模块断电后重新连接的次数。

EMIO_Set_PowerOff_Reconnect_Counts			
函数原型	short EMIO_Set_PowerOff_Reconnect_Counts (short PowerOff_Reconnect_Counts , unsigned short Station_Number);		
说明	设置指定站点模块断电后重新连接的次数。		
原型参数			
类型	名称	输入/输出	说明
short	PowerOff_Reconnect_Counts	输入	设置站点模块断电后重新连接的次数
unsigned short	Station_Number	输入	从站模块的站点 ID 序号。
C++ 示例	short siRtn = EMIO_Set_PowerOff_Reconnect_Counts(0,0);		
C# 示例	short siRtn = EMIO.EMIO_Set_PowerOff_Reconnect_Counts(0,1);		

## 第 4 章 常见问题 FAQ

### 4.1 硬件常见 FAQ

Q: 总线式系统架构是否可以同时用两个或多个网口连接并同时打开所有从站模块？

A: **不可以！** 为保证通讯的实时及稳定性，同一时间段内只能打开一个网口串联的所有从站模块。

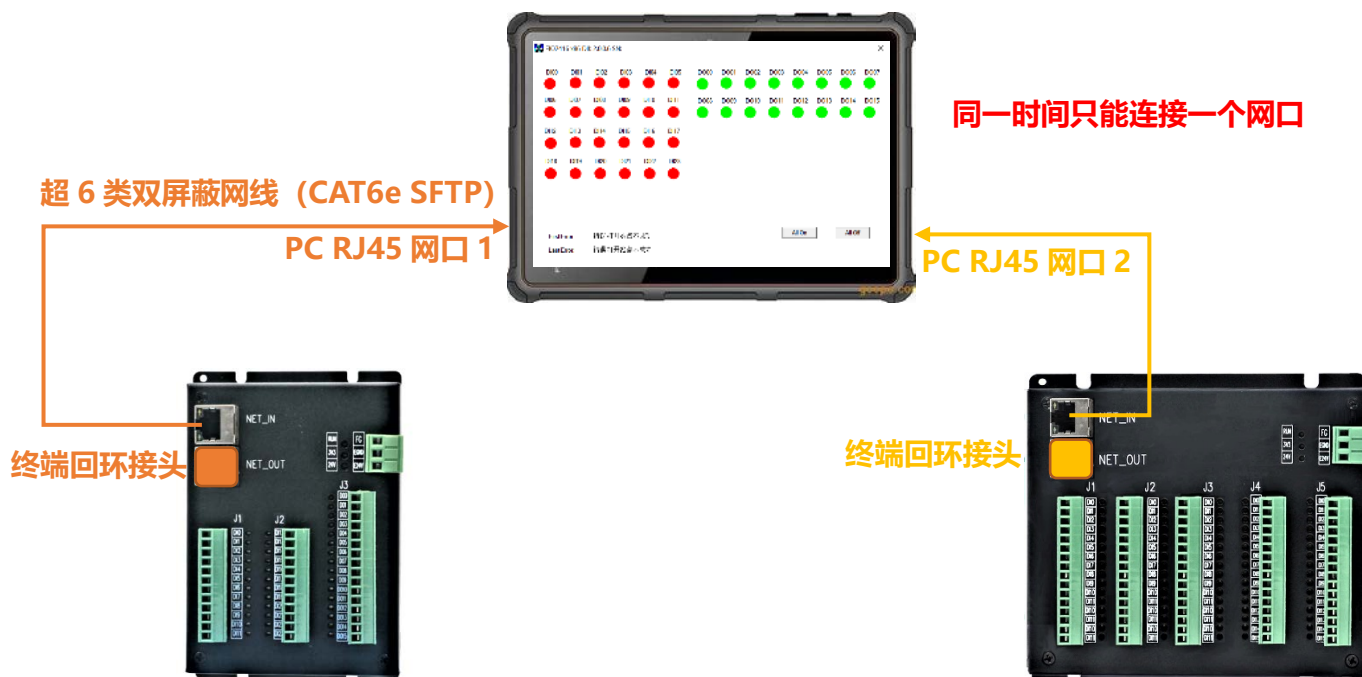


图 4.1.1 总线式系统架构

Q: NET\_IN 和 NET\_OUT 口接入顺序可不可以接反？

A: **不可以！** 如果 API 一直返回打开站点不成功，请先确认网线的接入顺序，必须严格按照从 NET\_IN 口接入，串接的话从 NET\_OUT 接出，如不串接，端回环接头必须接 NET\_OUT 口。

Q: 可以不接终端回环接头么？

A: **不可以！** 必须接终端回环接头。

(1). 不接终端回环接头, 返回打开站点不成功

(2). 接了终端回环接头, 打开站点成功后, 拔掉回环头, 报网络链接中断; 重新插上后, 恢复正常。

---

Q: 总线式系统架构最多可以串联多少个模块?

A: 为保证通讯周期 1ms, 总线式系统架构最多可以级联 56 个模块。以 48DI32DO 来算, 最多总共可以控制  $80 \times 56 = 4480$  点(2784DO/896DI)。

---

Q: 如果级联少于 56 个模块, 总线通讯周期会少于 1ms 么?

A: 不会, 总线通讯周期依然是 1ms, 如果需要更改总线通讯周期, 请联系厂商客制化。

---

Q: 可以使用普通的非屏蔽网线么?

A: **不可以!** 网络线缆的质量对通讯的稳定性有极其重要的作用, 非屏蔽/普通/劣质网线容易引起网络断线及通讯不稳定。请使用厂商经过测试的配套超 6 类双屏蔽网线 (CAT6e SFTP)。

---

Q: API 函数是否支持 Linux 和 Mac 系统?

A: 目前暂时不支持, 如果需要其他操作系统支持, 请联系厂商客制化。

---

Q: 电脑主站网口是否要设置固定 IP 地址?

A: **需要!** 设置与模块连接的电脑主站网口为固定 IP 地址, 由于 Windows 的网络接口特性, 固定 IP 地址不仅能使通讯更加稳定, 而且能加快网络断线与重连的速度。



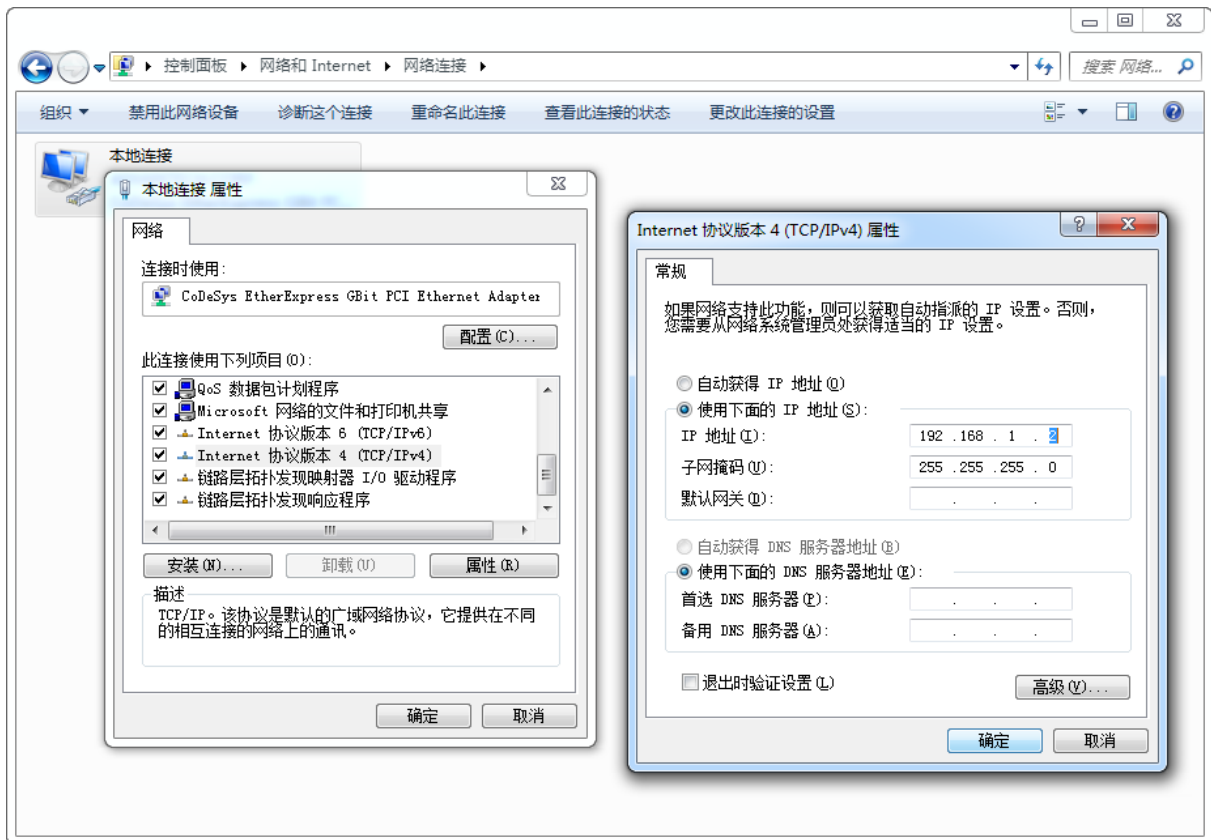


图 4.1.2 设置网络连接固定 IP 地址

## 4.2 软件常见 FAQ