雷赛智能
Leadshine

雷赛中型 PLC MC300CS 系列
产品应用手册

- 非常感谢您本次购买雷赛产品
- 使用前请详细阅读此说明书，正确使用产品
- 请妥善保管此说明书

www.leisai.com

202404 V1.0

前　言

感谢您选用深圳市雷赛智能控制股份有限公司中型 PLC 产品。本手册提供了雷赛智能中型 PLC MC300CS 系列产品的编程及应用说明。对于初次使用该产品的用户，请认真阅读本手册。若对产品的功能应用和性能方面有所疑惑，请咨询我公司技术支持人员以获得帮助。

由于产品的改进，手册内容可能持续更新。

技术热线：400-885-5501

版本说明：

(1) 禁止转载本书的部分或者全部内容。

(2) 本书内容有可能变更，恕不另行通知。

目　录

第1章 概述

1.1. 产品简介

MC300CS 中型 PLC 产品是雷赛面向自动化生产线推出的运动控制器。基于 CODESYS 运动控制平台研发，支持 EtherCAT、CANopen 总线；特别适合控制工艺复杂的生产线设备，广泛应用于锂电、包装、物流、3C、电子半导体等行业设备。

MC300CS 中型 PLC 主要特点有：

1） 运动控制能力强，可实现 32 轴 EtherCAT 总线电机、6 路脉冲式电机控制，还可控制多个 CANopen 总线电机、485 总线电机；其中 EtherCAT 总线电机、脉冲电机支持电子齿轮、电子凸轮功能。

2） 通讯接口丰富。有 1 个以太网接口、1 个 EtherCAT 总线接口、1 个 CAN 总线接口，2 个独立的 485 接口、1 个 RS232 接口。

3） 模块化结构，可扩展 32 个本地模块。

4） 自带 16 点输入接口。其中 12 路为高速输入口，且有 6 路与高速计数器复用。

5） 自带 16 点输出接口。其中 12 路为高速输出口，且有 6 路与脉冲式电机控制信号复用。

图 1.1　MC300CS 中型 PLC 照片

MC300CS 系列产品有三个型号：MC308CS、MC316CS 和 MC332CS。MC308CS 可控制 8 个 EtherCAT 总线轴，MC316CS 可控制 16 轴，MC332CS 可控制 32 轴。产品的主要参数如表 1.1 所示。

1.2. MC300CS 的系统应用

MC300CS 中型 PLC 产品提供 Ethernet 接口，EtherNet 口支持 EIP、TCP/UDP（Socket）、ModbusTCP、OPC UA 等协议，支持触摸屏的标签通信连接。

MC300CS 提供 EtherCAT 总线接口，可控制雷赛公司的伺服电机、步进电机，R1/R2 系列远程扩展模块；以及其他具有标准 EtherCAT 总线接口的产品。其典型应用架构如图 1.2 所示。

表 1.1 产品参数表

型号	MC308CS	MC316CS	MC332CS
控制电机数	EtherCAT 8 轴 +脉冲 6 轴	EtherCAT 16 轴 +脉冲 6 轴	EtherCAT 32 轴 +脉冲 6 轴
脉冲轴	本地 6 轴 200K Hz 脉冲输出		
模块扩展	最多可扩展 32 个，支持 R2 系列扩展模块		
以太网	1 个以太网口；支持 MODBUS 主从站、SOCKET		
EtherCAT	EtherCAT 主站；支持最大 128 从站		
串口通信	RS232 一个,RS485 二个；支持自由协议，MODBUS RTU/ASC 主从站		
CAN 通信	CANopen 总线接口 1 个		
编码器接口	6 路，200K Hz		
本体 IO	16 入 16 出		
程序容量	20M Byte		
数据容量	40M + 512K Byte 掉电保持空间		
其他接口	USB(Type C)、SD 卡		
软件功能	点位运动、电子齿轮、电子凸轮等		
编程平台	Leadsys Studio V3.0 及以上版本		
编程语言	ST、LD、CFC、SFC、FBD、IL		
外形尺寸	81.8 × 100 mm		
电源输入	DC 24V		

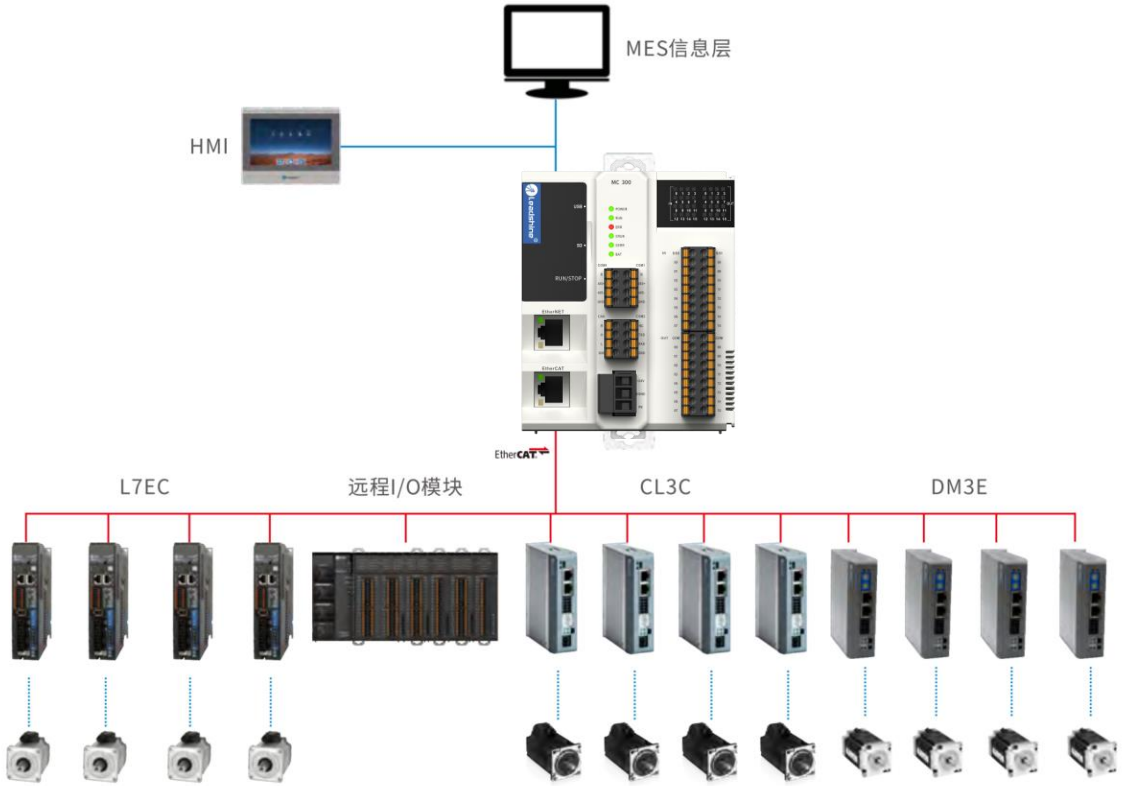


图 1.2　MC300CS 系列 PLC 的应用架构

除了 Ethernet 网口和 EtherCAT 总线接口外，雷赛中型 PLC 还有 RS485、RS232 串口和 CAN 总线等接口，方便连接多种外设。MC300CS 系列 PLC 各种接口的应用如图 1.3 所示。

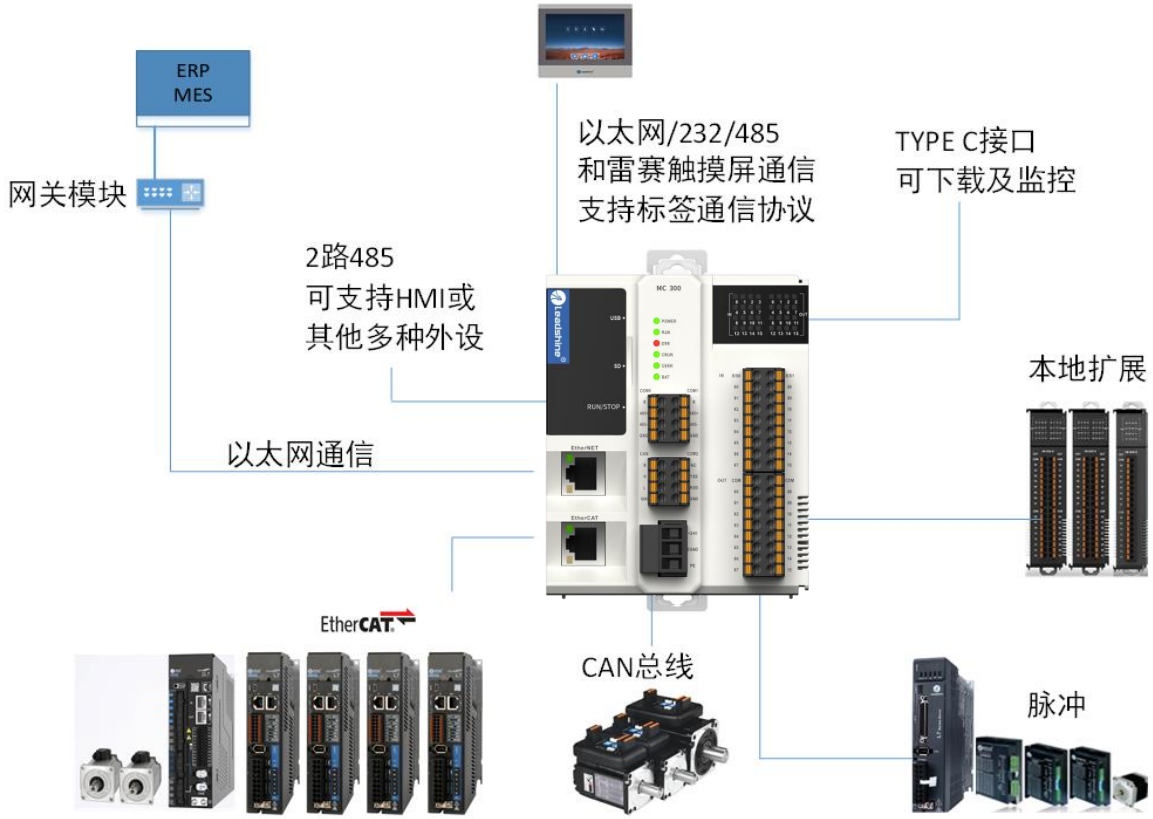


图 1.3　MC300CS 各种接口的应用

雷赛中型 PLC 的基本应用流程如图 1.4 所示。

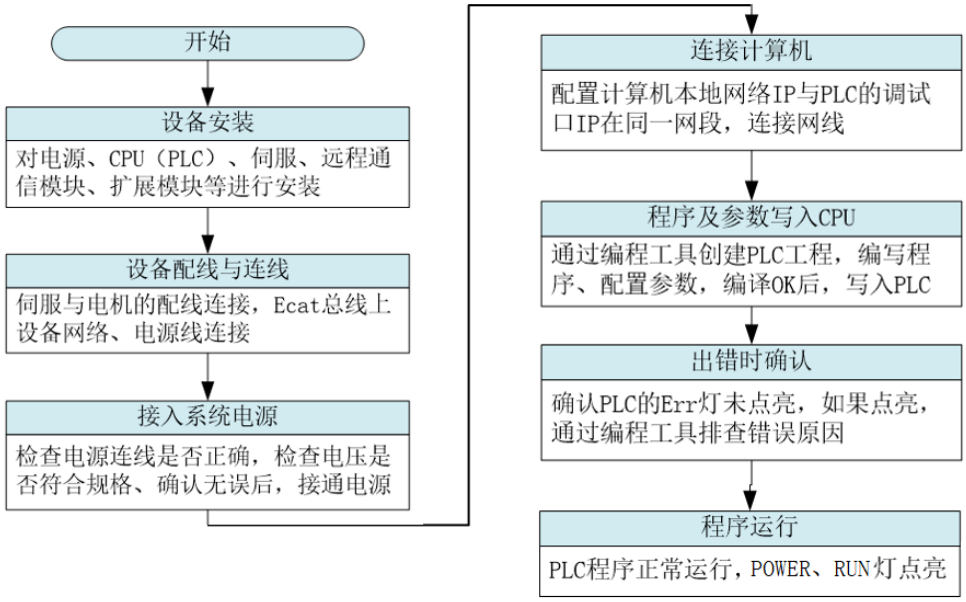


图 1.4　MC300CS 应用流程

1.3. 订货信息

表 1.2　MC300CS 系列 PLC 订货信息

型号	产品描述	物料编码
MC308CS	MC300CS 系列基本型中型 PLC，16 入 16 出，支持 8 轴	83280008
MC316CS	MC300CS 系列基本型中型 PLC，16 入 16 出，支持 16 轴	83280007
MC332CS	MC300CS 系列基本型中型 PLC，16 入 16 出，支持 32 轴	83280006

第2章 MC300CS 硬件接口

2.1. MC300CS 接口与标识

MC300CS 中型 PLC 上的各种接口与标识如图 2.1 所示，接口名称与标识的说明见表 2.1。

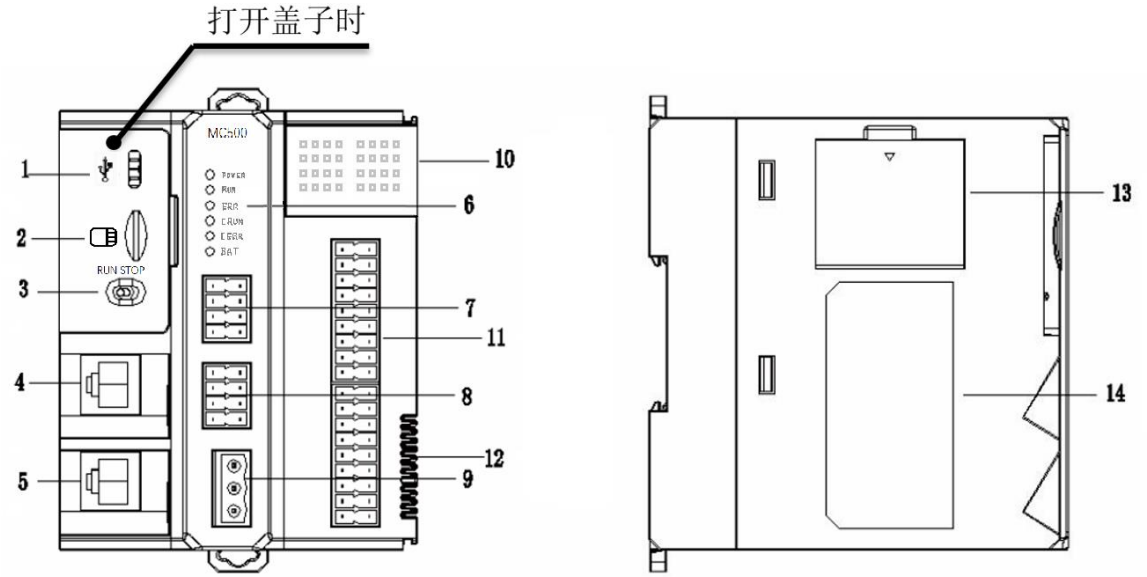


图 2.1　MC300CS 接口与标识分布图

表 2.1　MC300CS 系列 PLC 接口与标识说明

编号	端口类型	接口标识	定义	说明
1	USB 接口		USB 连接	TYPE C 接口，和上位机连接
2	SD 卡插槽	SD	SD 卡座，用于插 SD 卡	用户程序下载，标准 MicroSD 卡 FAT32 格式，支持最大容量 32G
3	RUN/STOP/ RESET 拨码	RUN STOP	RUN-正常运行 STOP-系统停止	RESET 功能通过多次切换 RUN/STOP 状态实现 5 秒内拨动次数超过 5 次则认为触发 RESET 开关。触发 RESET 后，实现恢复出厂设置。
4	以太网口	EtherNET	以太网通信，RJ45 接口	Modbus TCP/IP 协议
5	EtherCAT 接口	EtherCAT	用于 EtherCAT 通信	用于连接 EtherCAT 总线电机和模块。
6	运行状态指示灯	POWER	电源状态提示	上电时灯亮，断电时灯灭
		RUN	系统当前运行状态	运行时灯亮，停机时灯灭
		ERR	PLC 系统故障	系统故障时灯亮，正常时灯灭
		CRUN	CAN 运行	CAN 总线正常运行时灯亮，故障/不使用时灯灭。
		CERR	CAN 报错	CAN 总线故障时灯亮，正常运行/不使用时灯灭。
		BAT	电池错误	电池电量低时报警

7	两个 RS485 接口	左侧 COM0	R	485 终端电阻信号	MODBUS RTU 协议 自由通信协议
			485+	485/A 通信信号正	
			485-	485/B 通信信号负	
			GND	485 通信地	
		右侧 COM1	R	485 终端电阻信号	
			485+	485/A 通信信号正	
			485-	485/B 通信信号负	
			GND	485 通信地	
8	左侧 CAN 总线接口	R		CAN 总线终端电阻信号	CANopen 协议 CAN2.0 协议
		H		CAN 差分对 H 信号	
		L		CAN 差分对 L 信号	
		GND		CAN 通信地	
	右侧 RS232 接口 COM2	TXD		232 发送信号	MODBUS 协议 自由通信协议
		RXD		232 接收信号	
		GND		232 通信地	
9	电源接口	24V		直流 24V +	直流 24V 电源输入
		EGND		直流 24V 地	
		PE		地线	接大地
10	I/O 指示灯	--		16 路输入 16 路输出 指示灯	信号有效时亮灯， 信号无效时灭灯
11	I/O 端子	--		16 路输入，16 路输出	详细定义请参见端子排列
12	模块扩展 接口	--		用于扩展模块连接	最多可扩展 32 个 IO 或其他模块；不支持热插拔
13	电池卡座	Battery		安装电池	实时时钟 RTC 的电源
14	标签	--		PLC 标签	--

2.2. MC300CS 的电源

MC300CS 系列 PLC 需要输入直流 24V 电源，详细规格见表 2.2。若不小心接错电源正负极，PLC 不会启动，但不会损害 PLC。

表 2.2 MC300CS 系列 PLC 的电源规格

项目	规格
输入电压	DC 24V（-15% ~ +20%）
最小电流	105 mA （PLC 空载）
最大电流	2.6 A （PLC 输入输出口满载）
中断时间	10ms 以下的断电时间不影响正常工作
电源保护	支持电源反接保护

2.3. MC300CS 的数字输入输出接口

MC300CS 中型 PLC 本体上的数字输入输出接口如图 2.2 所示，各端子定义如表 2.3 所示。

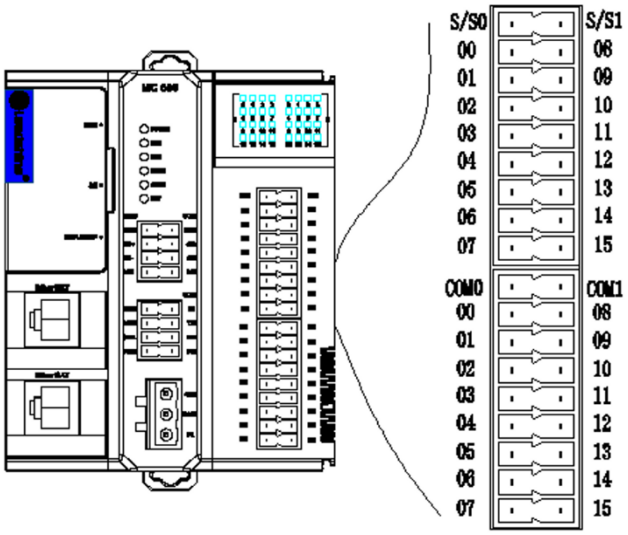


图 2.2　MC300CS 上的数字输入输出接口

表 2.3　MC300CS 上的数字输入输出端子定义

端子	定义	端子	定义
S/S0	输入公共端	S/S1	输入公共端
00	高速输入	08	高速输入
01	高速输入	09	高速输入
02	高速输入	10	高速输入
03	高速输入	11	高速输入
04	高速输入	12	普通输入
05	高速输入	13	普通输入
06	高速输入	14	普通输入
07	高速输入	15	普通输入
COM0	输出公共端	COM1	输出公共端
00	高速输出	08	高速输出
01	高速输出	09	高速输出
02	高速输出	10	高速输出
03	高速输出	11	高速输出
04	高速输出	12	普通输出
05	高速输出	13	普通输出
06	高速输出	14	普通输出
07	高速输出	15	普通输出

MC300CS 的数字输入、输出接口电路图如图 2.3、2.4 所示。

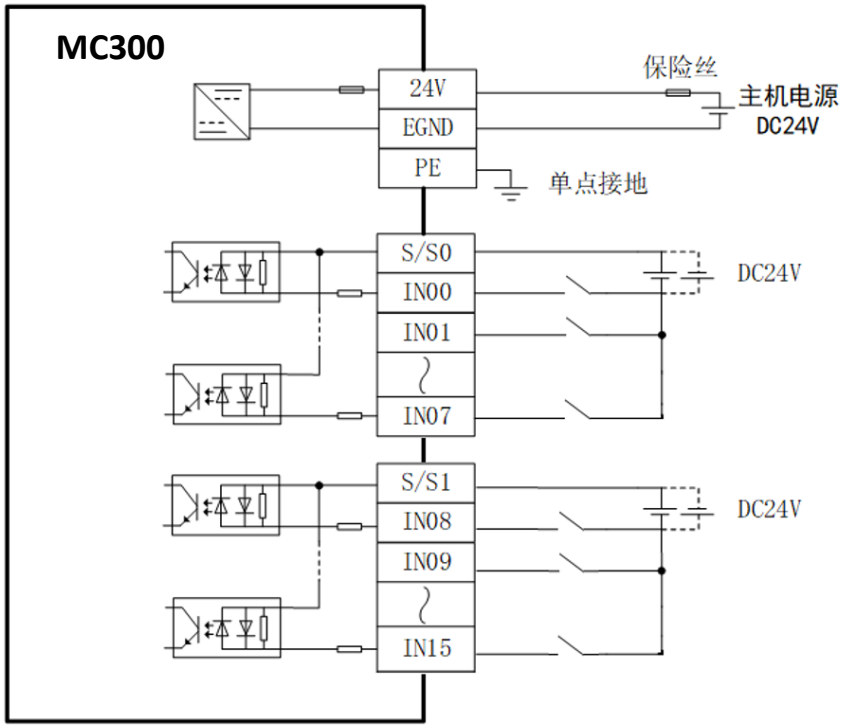


图 2.3　MC300CS 的数字输入接口电路图

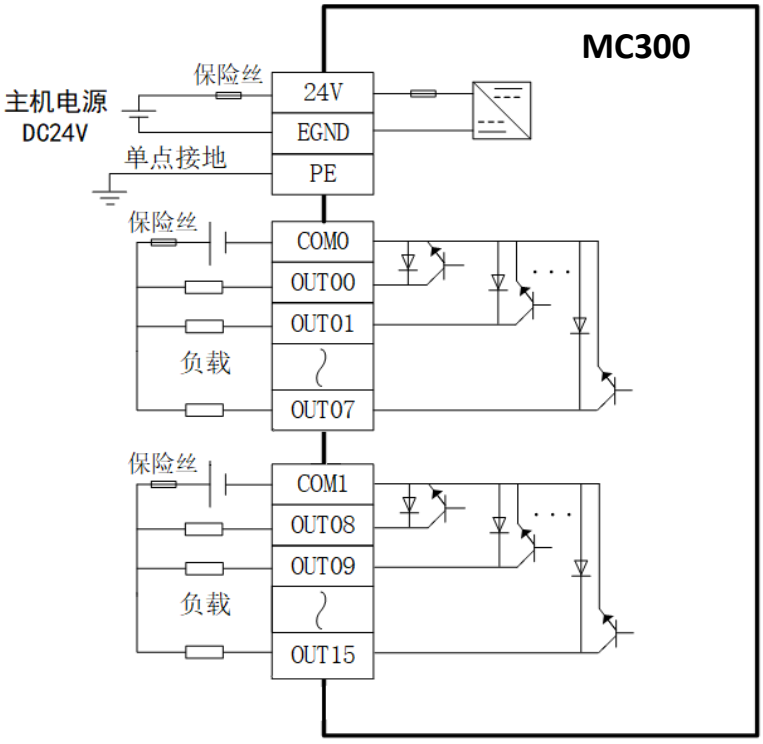


图 2.4　MC300CS 的数字输出接口电路图

MC300CS 的数字输入、输出端口规格参数见表 2.4、表 2.5。

表 2.4 MC300CS 数字输入端口规格参数

项目		规格
输入点数		16 点
输入形式		直流（漏型或源型）
输入电压/电流		DC24V(-15% - +20%)，5mA
动作位准	OFF→ON	高于 DC 15V，电流 4.5mA 以上
	ON→OFF	低于 DC 5V，电流 1mA 以下
输入频率	IN0~IN11	200 kHz 高速输入（单端）
	IN12~IN15	10 kHz 普通输入（单端）
响应时间	OFF→ON	20us
	ON→OFF	50us
输入阻抗		2.7KΩ
输入保护		光电耦合隔离、抗干扰滤波
输入动作显示		输入接通时 LED 灯亮
输入公共端		每 8 点使用一个公共端 S/S

表 2.5 MC300CS 数字输出端口规格参数

项目		规格
输出点数		16 点
输出形式		晶体管/漏型输出
控制回路电压		DC5V~24V
最大负载		0.5A/点，2.4A/COM
动作位准	OFF→ON	高于 DC 15V，电流 3mA 以上
	ON→OFF	低于 DC 5V，电流 1mA 以下
输出频率	OUT0~11	200 kHz 高速输出
	OUT12~15	10 kHz 普通输出
响应时间	OFF→ON	20us
	ON→OFF	50us
ON 时压降		0.2V
开路漏电流		0.1mA 以下
输出口保护		短路保护，过流保护
输出动作显示		输出接通时 LED 灯亮
输出公共端		每 8 点使用一个公共端 COM

2.4. MC300CS 与脉冲式步进电机的连接方法

MC300CS 与脉冲式步进电机驱动器的接线图如图 2.5 所示。图中 M415B 驱动器为单端信号接口；DM442 驱动器为差分信号接口。因为驱动器输入信号的电源电压由 5V

改为 24V，故要串联阻值为 2K 的电阻限流。

若用其他品牌的驱动器，需按其说明书的要求接线。

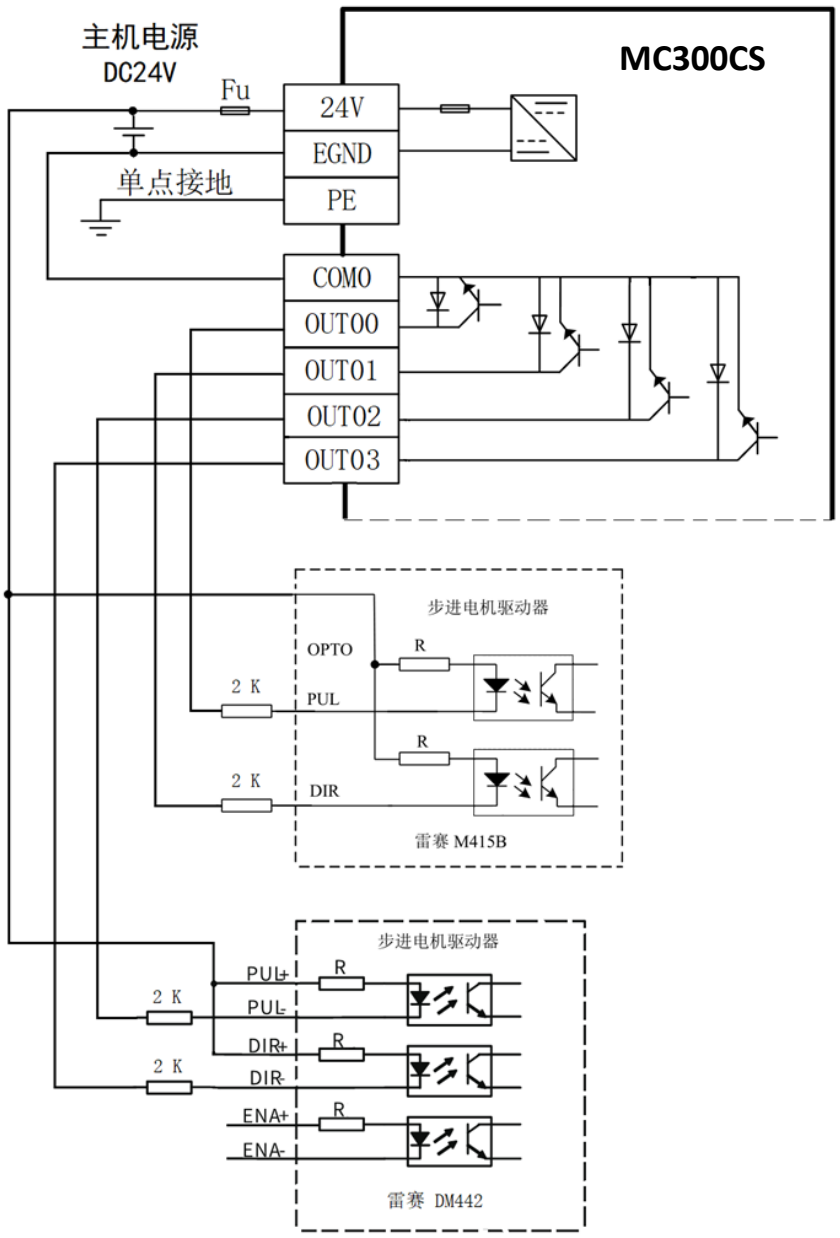


图 2.5　MC300CS 与步进电机驱动器的连接方法

2.5. 线缆选型、制作与安装

MC300CS 的电源线、信号线必须按照表 2.6 的参数选用。

表 2.6　MC300CS 的线缆规格参数

适用信号	配套物料名称	适配线径	
		国标/MM²	美标/AWG
电源线	管型线耳（管型端子）	0.5-1.5	24-16
信号线	管型线耳	0.5-1.5	24-16
接地线	管型线耳	≥2	14-1.5

线缆的制作与安装步骤如下：（参见图 2.6）

1. 剥除电缆绝缘层，露铜部分为 11~14 mm，将线缆穿入线号套管；
2. 将电缆的导体部分穿入线耳圆形孔中，使用线耳厂商推荐的压线钳压紧；
3. 用螺丝刀将电源端子左侧的螺钉松开，将电源线耳插入端子方孔中，然后锁紧螺钉。
4. 用螺丝刀将输入输出口端子孔旁的弹簧按钮压下，将信号线耳插入圆孔中，松开弹簧按钮。

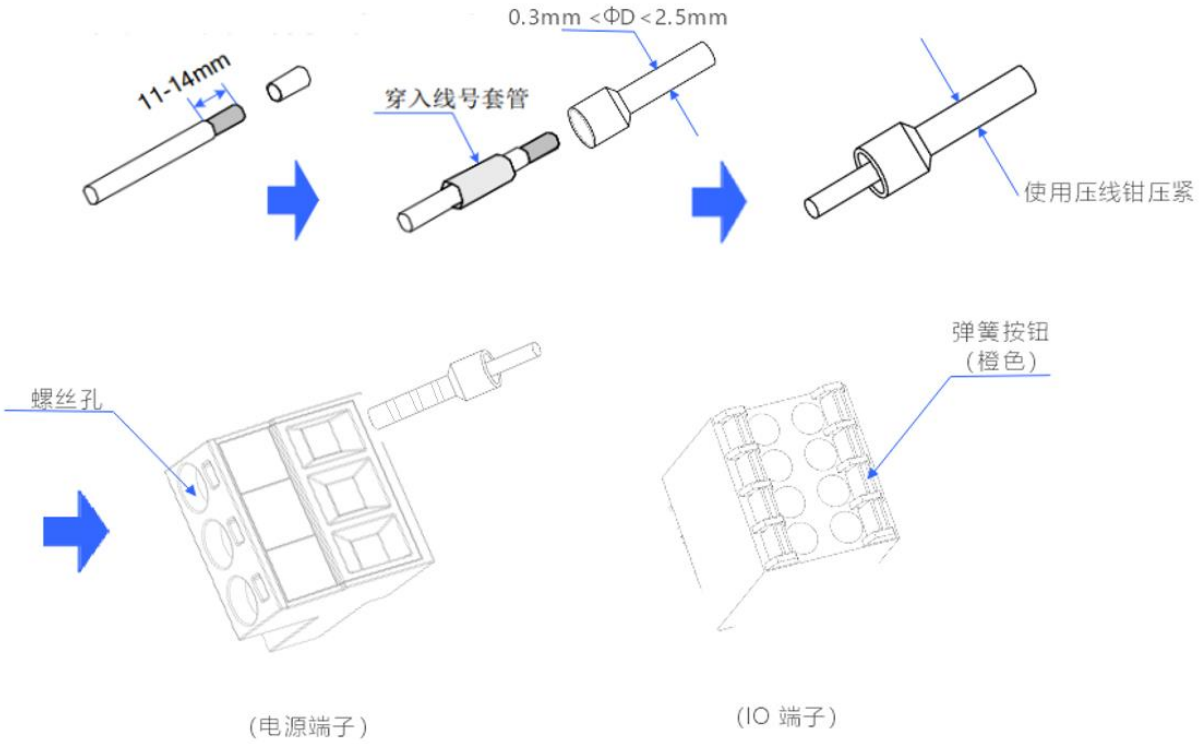


图 2.6　线缆制作、安装方法

接线注意事项：

- I/O信号布线时，避免与动力线等传输强干扰信号的电缆捆在一起，应该分开走线并且避免平行走线。高速I/O口推荐使用屏蔽线缆，以提高抗干扰能力，线长建议3m以内。
- 若采用集电极输出点连接MC主机高速输入点时，建议增加（上拉/下拉）并接电阻于指定IN点与S/S点之间，电阻建议使用1KΩ/2W的电阻。
- 输出口连接继电器、电磁阀等感性负载时，当感性负载突然关断时，会在触点

间产生很大的反向电动势，并产品电弧放电，有可能击穿输出晶体管，用户应根据使用情况，必要时在负载上并联续流二极管，如图2.7所示，延长产品寿命。二极管需满足反向电压是负载电压的5~10倍；正向电流大于负载电流。

- 输出口不允许连接较大容性负载，否则在通道关断时有可能故障。

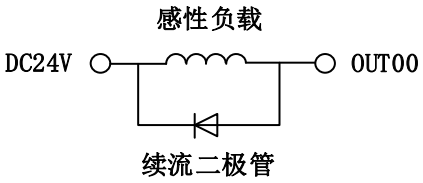


图 2.7　感性负载并联续流二极管

第3章 ST 语言简介

ST 语言即结构化文本编程语言（Structured Text）是 IEC 61131-3 标准支持的五种语言之一，专为可编程逻辑控制器（PLC）设计。它是一种块结构的高级语言，在语法上与 Pascal 语言相同，和 C 语言相似。

注：本手册只使用 ST 语言编程例程。

3.1. ST 语言的数据类型

A. 常用的数据类型有：

BOOL:	逻辑量，取值：TRUE 或 FALSE
SINT:	8 位有符号整数，范围：-128 ~ 127
USINT:	8 位无符号整数，范围：0 ~ 255
INT:	16 位有符号整数，范围：-32768 ~ 32767
UINT、WORD:	16 位无符号整数，范围：0 ~ 65535
DWORD:	32 位无符号整数，范围：0 ~ 4294967295
DINT:	32 位有符号整数， 范围：-2147483648 ~ 2147483647
REAL:	32 位实数，REAL 型的 0 = 0， 正数范围：1.4013×10^{-45} ~ 3.4028×10^{38}， 负数范围：-3.4028×10^{38} ~ -1.4012×10^{-45}
LREAL:	64 位实数，LREAL 型的 0 = 0， 正数范围：2.2251×10^{-308} ~ 1.7977×10^{308}， 负数范围：-1.7977×10^{308} ~ -2.2250×10^{-308}
STRING:	字符串
TIME:	时间值

B. 常量

数字常量可以定义为一个二进制、十六进制和十进制数。二进制和十六进制的数有前缀 2#、16#。

```
例：  2#1001_0011    // 二进制数
      16#A2          // 十六进制
      3.14159        // 十进制
```

字符串常量用单引号定义。例：’enable ’

时间常量：以 t#为前缀

时间单位：d - 天、h - 时、m - 分、s - 秒、ms - 毫秒

```
例：  T#100S12ms
      t#12h34m15s
```

C. 变量

变量名必须以字母开头，可以加入数字。

注意：ST 语言中字母不分大小写！

使用变量必须先定义变量。本地变量只能在一个子程序、或一个函数中使用；全局变量可在所有子程序和函数中使用。定义变量时可赋初值。

本地变量定义的格式：

```
VAR
    i: INT;
    Speed: LREAL := 20;
END_VAR
```

全局变量定义的格式：

```
VAR_GLOBAL
    IN1: BOOL;
    Wcount: INT ;
END_VAR
```

变量类型转换的函数有很多。有两种格式：数据类型 1_TO_数据类型 2（变量）、TO_数据类型 2（变量）。

示例:

```
re: REAL := 1.234;                          // 定义一个 32 位实数
i: INT := TO_INT(re);                       // 将实数转换为整数
temp: STRING := TO_STRING(re);              // 将实数转换为字符串
speed: LREAL := STRING_TO_LREAL(“2.34”); // 将字符串转换为实数
m: DWORD := INT_TO_DWORD(i);          // 将 16 位整数转换为 32 位整数
```

D. 数组

定义一维数组的格式：数组名：ARRAY[0..n] OF 类型；　　//该数组的长度为 n+1

定义二维数组的格式：数组名：ARRAY[0..m,0..n] OF 类型；//(m+1)行×(n+1)列

例：

```
定义：  LineEnd: ARRAY[0..2] OF LREAL := [0,0,0]; //定义时可以赋初值
        answer: ARRAY[0..3,0..3] OF INT;
使用：  LineEnd[1]:= 10;
        answer[i,1]:= hen;
```

E. 结构体

结构体是将多个有关联的变量整合为一体，以便于处理数据。

使用结构体的方法：

1）鼠标右键点击项目树中的 Application，选“添加对象”，添加一个 DUT 对象，然后声明结构体。

2）在变量定义区初始化结构体。

3）与普通变量一样，在程序中给结构体赋值，用结构体传递数据。

例：定义一个结构体 triangleData，用于定义一个三角形的三个顶点。然后初始化结构体，变量名为 trangles，该变量为数组，可存放 9 个三角形的顶点。代码如下。

```
TYPE trangleData :                  // 声明结构体
```

```
    STRUCT
      point1: ARRAY [0..1] OF INT;  // 声明变量 point1，三角形顶点 1 的坐标
      point2: ARRAY [0..1] OF INT;  // 声明变量 point2，三角形顶点 2 的坐标
      point3: ARRAY [0..1] OF INT;  // 声明变量 point3，三角形顶点 3 的坐标
    END_STRUCT
  END_TYPE

  trangles: ARRAY [0..9] OF trangleData;  // 初始化 trangles，可保存 9 个三角形的数据

  // 给第 2 个三角形赋初值
  trangles[1].point1[0] := 10;    // 三角形顶点 1 的坐标
  trangles[1].point1[1] := 10;
  trangles[1].point2[0] := 20;   // 三角形顶点 2 的坐标
  trangles[1].point2[1] := 10;
  trangles[1].point3[0] := 10;   // 三角形顶点 3 的坐标
  trangles[1].point3[1] := 30;
  ……
  // 将第 3 个三角形的起点传递给数组 start
  start[0] := trangles[2].point1[0];
  start[1] := trangles[2].point1[1];
```

LeadSys 已定义好了一些结构体，如：脉冲轴名称：dut_pulse_axis、连续插补数据 MoveSequence 等等，使用这些结构体时，不需要再声明结构体。

3.2. 运算符

常用运算符及其优先级如表 3.1 所示。

表 3.1. 运算符及其优先级

优先级	运算符号	描述
1	（……）	括号
2	函数名（……）;	函数的调用
3	- NOT	负号 非
4	* / MOD	乘 除 求余数
5	+ -	加 减
6	<，<=，>，>=	小于，小于等于，大于，大于等于
7	=，<>	等于，不等于
8	AND XOR OR	与 异或 或

3.3. 常用的数学函数

常用的数学函数如表 3.2 所示。

表 3.2. 常用的数学函数

函数名	描述	说明
ABS（x）	绝对值	
SQRT（x）	平方根	
LN（x）	自然对数	
LOG（x）	10 为底的对数	
EXP（x）	指数	
EXPT（y，x）	幂	计算 y 的 x 次方
SIN（x）	正弦	角度单位为弧度
COS（x）	余弦	
TAN（x）	正切	
ASIN（x）	反正弦	函数值的单位为弧度
ACOS（x）	反余弦	
ATAN（x）	反正切	

3.4. 常用语句

ST 语言的每条语句要由分号“；”结尾。一条语句占用一行或多行。

A. 赋值语句

:=

注：ST 语言中的赋值语句和 C 语言不同，等号前多一个冒号。

例：DelayOn := TRUE;

L := r0 - r0*COS(PI/4);

B. 条件语句

```
IF 条件表达式 THEN
    ……
  ELSE
    ……
END_IF;
```

例：

```
IF sum = 100.0  THEN  // 不要写成 sum := 100 ！否则不报错，但程序不对。
   i:=i+1;
  ELSE
   i:=i-1;
END_IF;
```

C. 循环语句

```
FOR 循环变量 := 初值 TO 终值 BY 步长 DO
……
END_FOR;
```

注：如果步长为 1，BY 1 可省略

D. 条件循环语句

```
WHILE 条件表达式 DO
……
END_WHILE;
```

E. 开关语句

开关语句是一种多分支选择语句。格式如下：

```
CASE <变量> OF
      <数值 1>: <指令块 1>
      <数值 2>: <指令块 2>
      <数值 3, 数值 4, 数值 5>: <指令块 3>
      <数值 6 .. 数值 10>: <指令块 4>
       ……
      <数值 n>: <指令块 n>
    ELSE <其他指令块>
END_CASE;
```

例：

```
CASE INT1 OF
   1, 5: BOOL1 := TRUE;     // 如果 INT1 = 1 或 5，BOOL1 = TRUE，BOOL3 = FALSE
         BOOL3 := FALSE;
      2: BOOL2 := FALSE;  // 如果 INT1 = 2，BOOL2 = FALSE，BOOL3 = TRUE
         BOOL3 := TRUE;
 10..20: BOOL1 := TRUE;    // 如果 INT1 在 10~20 内，BOOL1 = TRUE，BOOL3 = TRUE
         BOOL3:= TRUE;
   ELSE
         BOOL1 := NOT BOOL1;       // 否则，BOOL1 取反，BOOL2 = BOOL1 OR BOOL2
         BOOL2 := BOOL1 OR BOOL2;
END_CASE;
```

F. EXIT 语句

在 FOR、WHILE 或 REPEAT 循环语句中执行 EXIT 语句，循环将立即停止。

F. CONTINUE 语句

在 FOR、WHILE 或 REPEAT 循环语句中执行 CONTINUE 语句，循环将立即开始下一次循环。

例：

```
FOR Counter:=1 TO 5 BY 1 DO
    INT1:=INT1/2;
```

```
        IF INT1=0 THEN
            CONTINUE;     // 避免除数为零
        END_IF
        Var1:=Var1/INT1;  (* 除数不为零时执行 *)
    END_FOR;
```

G. RETURN 语句

退出子程序或函数。

H. 调用功能块、子程序、函数的语句

格式：功能块或子程序名（参数 1，参数 2，……）；

变量 := 函数名（参数 1，参数 2，……）；

I. 注释符号

单行注释符号为：“//”；多行注释符号为“（*” 语句块 “*）”。

J. 查看帮助文件

在雷赛的 LeadSys Studio 软件中，有完整的帮助文件。其中有关 ST 语言比较完整的内容在“目录”下的“CODESYS Development System”→“参考编程”→“编程语言和他们的编辑器”中，如图 3.1 所示。用户有必要仔细阅读。

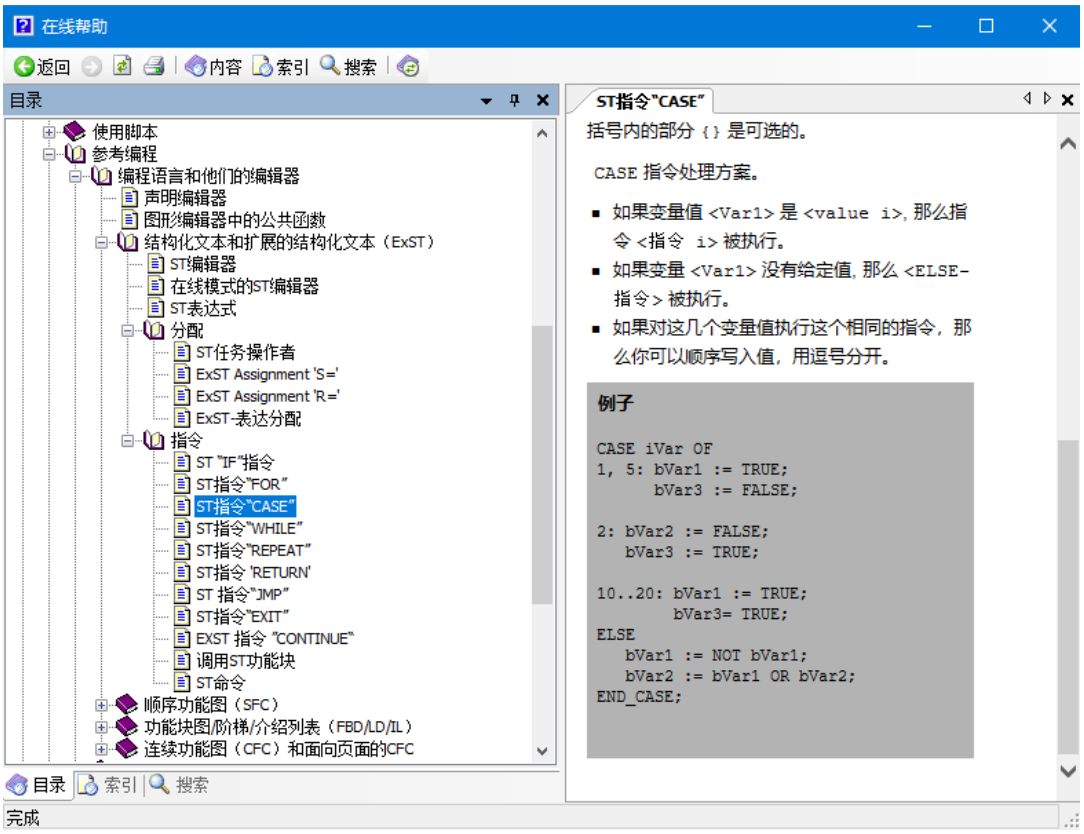


图 3.1　雷赛 LeadSys Studio 软件帮助文件中与 ST 语言相关内容的目录

3.5. 常用的功能模块和指令

A. 通电延时定时器 TON

当定时器的输入端 IN 变为 TRUE 时，经过设定的时间后，定时器的输出端 Q 才变为 TRUE。其代码格式如下：

TON(IN:=(参数), PT:=(参数), Q=>(参数), ET=>(参数));

指令中符号“:=”后为输入参数；符号“=>”后为输入参数。

其中：IN：BOOL 型，其上升沿触发通电延时定时器开始计时。

PT：TIME 型，时间常量，设置通电延时时间。

Q：BOOL 型，当到达延时时间 PT 时，Q 输出 TRUE。

ET：TIME 型，输出从 IN 上升沿开始计时的时间值。

例：按键 IN0 按下后，开启定时器，延时 200ms 后处理其他工作。

```
VAR
     input0: BOOL;      // 按键 IN0
     delay1: TON;       // 声明一个通电延时定时器 delay1
     delay1On: BOOL;    // 定时器开
     delay1Off: BOOL;   // 定时器关
END_VAR

// 程序
delay1(In:=delay1On,pt:=T#200MS,Q=>delay1Off,ET=>); // 定时器 delay1 的延时为 200ms

IF input0=TRUE THEN     // 按键 IN0 闭合
   delay1On:=TRUE;      // 启动定时器 delay1
END_IF
IF delay1Off=TRUE THEN // 定时器 delay1 时间到
   delay1On:=FALSE;     // 定时器 delay1 启动信号置零
   ……                   // 处理其他工作
END_IF
```

B. 上升沿检测触发器 R_TRIG

该功能模块用作检测一个信号的上升沿。其代码格式如下：

R_TRIG(CLK:=（参数）, Q=>（参数）);

其中：CLK：BOOL 型，被测信号；

Q：BOOL 型，触发器的输出信号。如果 CLK 检测到上升沿，则在一个扫描周期内为 TRUE。

例：检测按键 9 的上升沿信号

```
VAR
     R_TRIGin9: R_TRIG ;       // 声明一个上升沿检测触发器
     in9: BOOL;                // 按键 9
      ……
END_VAR
```

```
// 程序
R_TRIGin9 (CLK:= in9);
IF R_TRIGin9.Q THEN     // 如果按键 9 按下
      ……                // 处理相关工作
END_IF
```

C. 移位操作指令

左移指令：SHL(操作数, 左移 1 位的次数)

将操作数按位左移，最低位自动补 0。

右移指令：SHR(操作数, 右移 1 位的次数)

将操作数按位右移，最高低位自动补 0。

循环左移指令：ROL (操作数, 左移 1 位的次数)

最高位移到最低位，其他和左移指令相同。

循环右移指令：ROR (操作数, 右移 1 位的次数)

最低位移到最高位，其他和右移指令相同。

```
例：output:= ROL(output,1);  // output 循环左移 1 位
    input:= SHR(input,2);    // input 右移 2 位
```

D. 取地址指令 ADR（变量）

该指令用于获取变量的地址。

```
例：axisPower.PulAxis_0 := ADR(X); // 将 X 轴的地址赋值给 PulAxis_0
    axisPower.PulAxis_1 := ADR(Y);
    axisPower.PulAxis_2 := ADR(Z);
```

3.6. ST 语言与 C 语言的主要区别

ST 编写的程序执行过程和 PLC 的梯形图一样，是反复扫描运行的。相当于 C 语言程序的最外层有一个死循环语句。

在编写 MC300CS 的程序时，一定要注意程序的运行时间，尤其是使用循环语句的时候要特别注意，程序执行一遍的时间不能超过 2 毫秒，否则，EtherCAT 总线型伺服电机有可能出错。

第4章 LeadSys Studio 软件入门及 IO 控制

雷赛的 LeadSys 软件基于 CODESYS V3.5 开发，包括硬件配置和软件编程，是一个功能丰富的自动化软件。

LeadSys 软件具有的特点：

- 符合 IEC61131-3 国际标准，支持梯形图 LD、指令表 IL、结构化文本 ST、功能块图 FBD、顺序功能流程图 SFC、控制流程框图 CFC 等多种编程语言；
- 强大的软件仿真、在线调试及程序检查能力。不需要连接 PLC 硬件即可运行程序、调试试程序；
- 可以轻松快速地完成 CPU 配置、通信配置、本地 IO 功能配置、EtherCAT 总线配置等；
- 提供丰富的运动控制功能库和行业工艺库，涵盖多种过程控制和运动控制的应用场景。
- LeadSys 软件目前只支持 64 位 Windows 操作系统，不再支持 32 位系统。

4.1. LeadSys Studio 软件安装方法

LeadSys 软件安装步骤如下：

1. 从雷赛智能公司官网下载 LeadSys 软件，网址为：https://www.leisai.com/。
2. 安装之前建议关闭 360 等杀毒软件。
3. 选择以 LeadSys Studio 开头的 exe 文件，点击鼠标右键以管理员的身份运行安装，安装路径下不能有中文。
4. 安装提示界面如图 4.1 所示。用户可选择空间较大的硬盘作为安装路径。在安装中，可选完全安装“Complete”；当遇见“CodeMester 控制中心”界面时，将其关闭即可。
5. 最后，点击“Finish”，表示安装成功，在桌面上自动生成图标为： 

4.2. LeadSys Studio 软件的界面

点击桌面上的 LeadSys Studio 图标，或在 Windows 系统开始菜单中打开 LeadSys Studio 软件。LeadSys Studio 的工程界面及注释如图 4.2 所示。

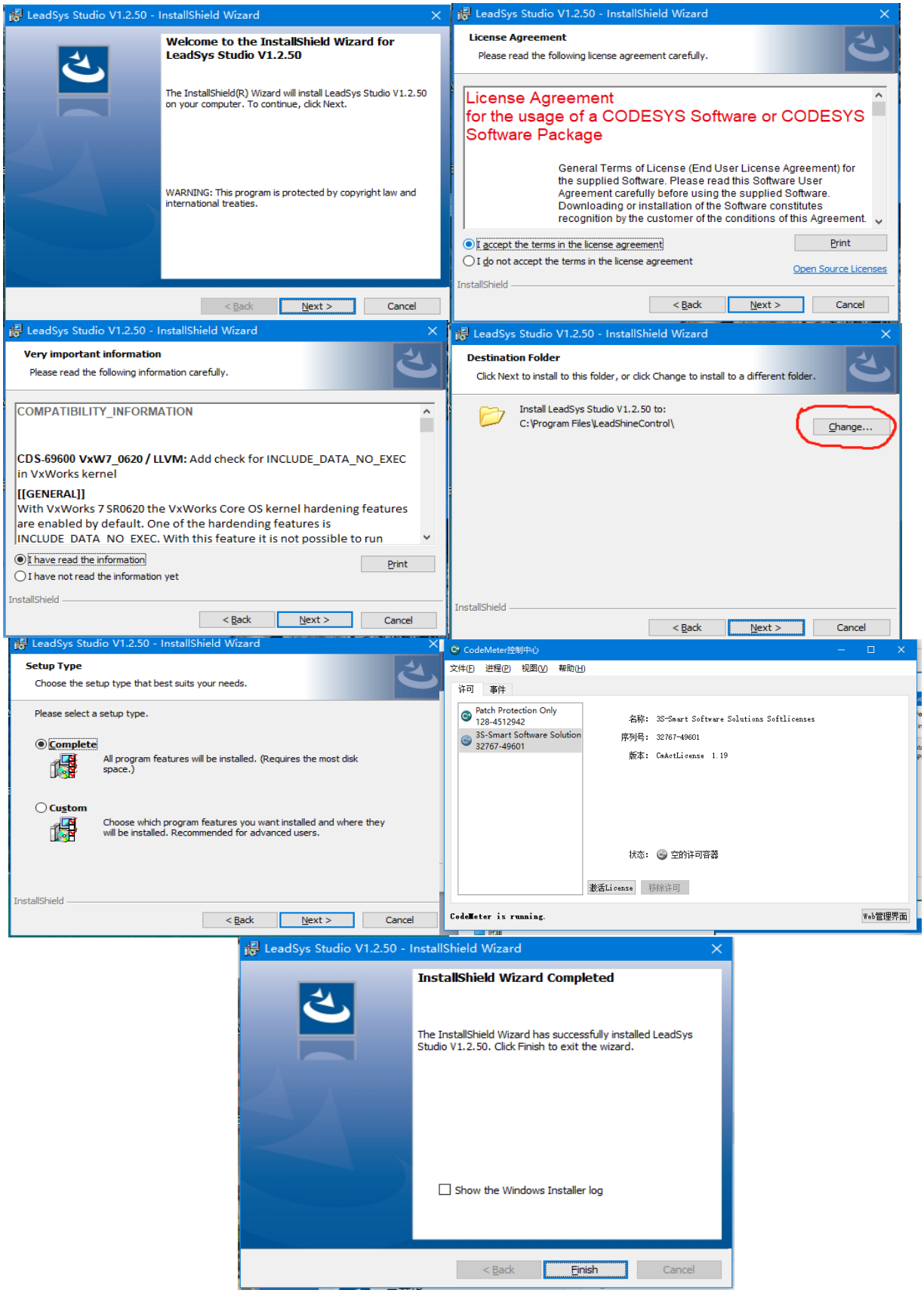


图 4.1　LeadSys 安装提示界面

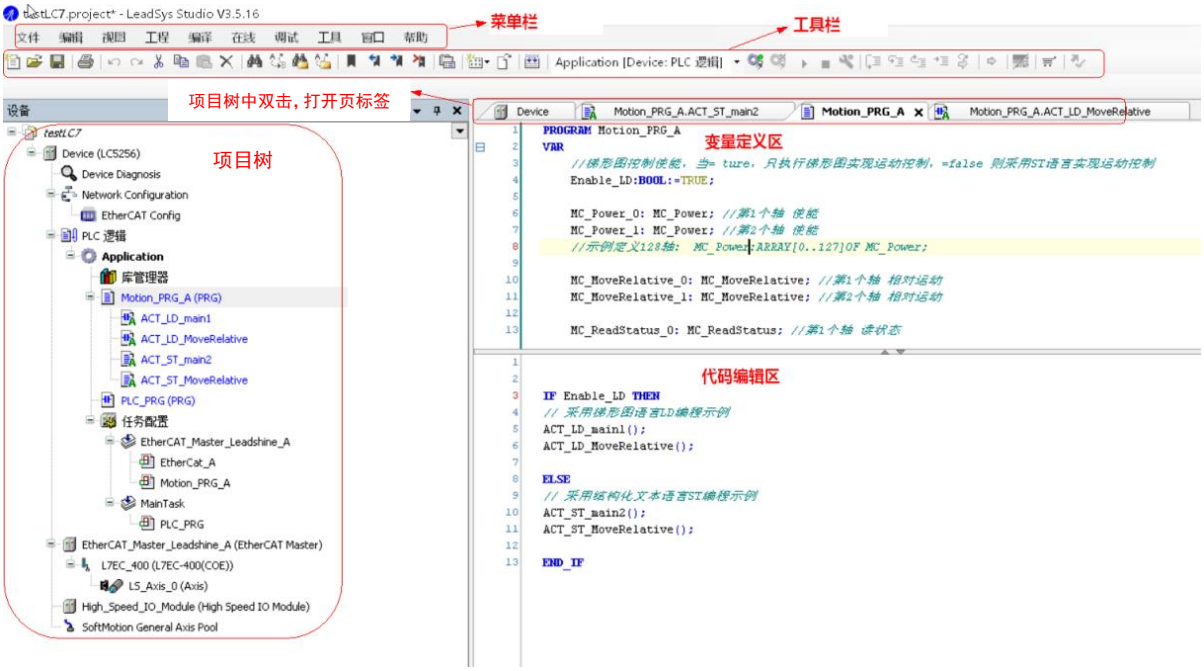


图 4.2　LeadSys Studio 的工程界面

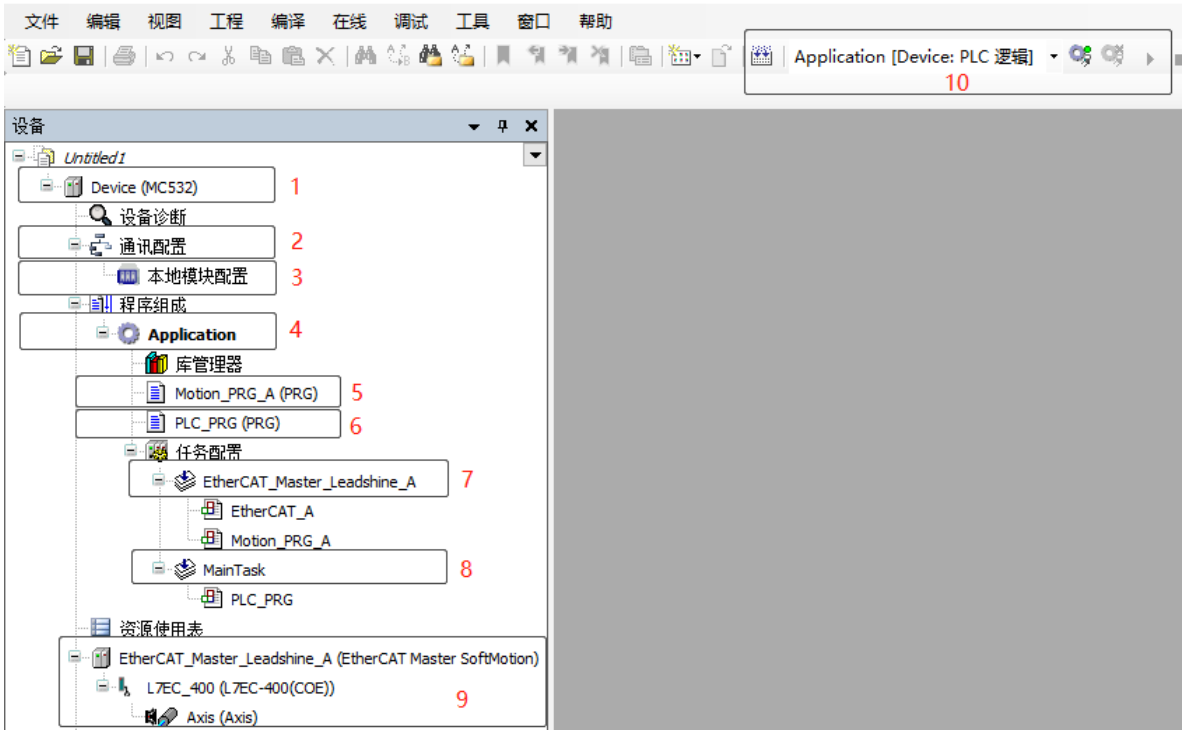


图 4.3　工程界面中关键部件的索引

图 4.3 中标注的各部分名称与注解如下：

① PLC 设备信息。点击鼠标右键可更换 PLC 型号

② EtherCAT 总线配置

③ 本地或远程 IO 模块配置

④ 用户程序管理单元

⑤ EtherCAT 总线 A 运动控制程序 Motion_PRG_A。双总线时，可建程序 Motion_PRG_B。

⑥ PLC_PRG 主程序，编程语言为新建工程时选择的语言。也可删除，重新建 LD 语言的主程序。主程序和运动控制程序的命名可由客户根据习惯或命名规范进行自定义。
⑦ EtherCAT 总线 A 的任务
⑧ 主程序的任务
⑨ EtherCAT 总线从站和 402 轴
⑩ 编译、登录及调试快捷键

4.3. 编写程序的典型步骤

编写调试一个完整的用户程序一般需要以下 6 个步骤：
1）按照 PLC 应用系统的硬件配置和网络架构进行硬件系统配置。
- 将伺服电机驱动器、步进电机驱动器、远程 IO 扩展模块等标准 EtherCAT 总线设备，按照应用系统网络架构配置到 EtherCAT 总线上。
- 若用到远程 IO 耦合器，需要在远程 IO 扩展界面上，按照应用系统配置 IO 模块。

2）将系统架构中的各硬件端口对应的输入端口变量（I）、输出变量（Q）或数值（M）与用户程序中的变量进行关联。
3）配置网络通信的同步周期（如 EtherCAT 总线）。根据各任务的实时性要求，配置用户程序单元的执行周期。
4）根据应用系统的控制工艺编写用户程序。
5）将编写好的程序单元 POU 按照需求放置到对应的任务中。
6）登录 PLC，下载用户程序，仿真调试、排错；再连接设备调试，直到程序正确运行。

4.4. IO 控制方法及例程

IO 控制的方法很简单。先定义输入口、输出口信号的变量，然后让输入口、输出口变量与输入、输出口的地址建立映射关系。如：变量 input0、output1 与输入口 IN0、输出口 OUT1 建立映射关系。

之后，当输入口的电平发生变化，对应的输入口变量也随着变化。如：输入口 IN0 由低电平变为高电平，则变量 input0 的值由 FALSE 变为 TRUE；当输出口变量的值发生变化，对应的输出口电平也随之变化。如：输出口变量 output1 由 TRUE 变为 FALSE，则输出口 OUT1 的电平由高电平变为低电平。

4.4.1. 例程：IO 口控制“跑马灯”

编写一个程序，当 MC300CS 上的输入口 IN0 闭合，MC300CS 上的 15 个输出口上的 LED 灯以跑马灯的方式闪烁，即：OUT0 上的 LED 灯亮 0.3 秒后熄灭，OUT1 上的 LED 灯亮 0.3 秒后熄灭，直到 OUT15 上的 LED 灯亮 0.3 秒后熄灭；再变为 OUT0 上的 LED 灯亮

0.3 秒后熄灭；这样一直循环。

1. 硬件系统的搭建与设置

参考图 2.1、2.7，给 MC300CS 接上 24V 直流电源。

参考图 2.2、2.3、2.4、2.7，在 MC300CS 的 IN0 端口上接好一个开关，在输出口上接 LED 灯；也可以不接 LED 灯，通过输出口指示灯看输出口的状态。

将 PC 机的网线接入 MC300CS 的 EtherNET 端口；将 PC 机网口的 IP 地址改为 192.168.1.n，n 不等于 3，即不要与控制器的 IP 一样。

2. 编写软件

1） 编写主程序

打开 LeadSys 软件，鼠标点击菜单中的“文件”→“新建工程”，界面如图 4.4 所示；选择“标准工程”，选择本地电脑硬盘上的存放 PLC 工程的位置，填写本工程的名称，如：“movingLED”，然后点击“确定”。

所谓“工程”，就是一个完整的 PLC 应用程序包，它包含了控制器网络参数的设置、电机驱动器、IO 模块的参数配置、程序模块和代码等内容。

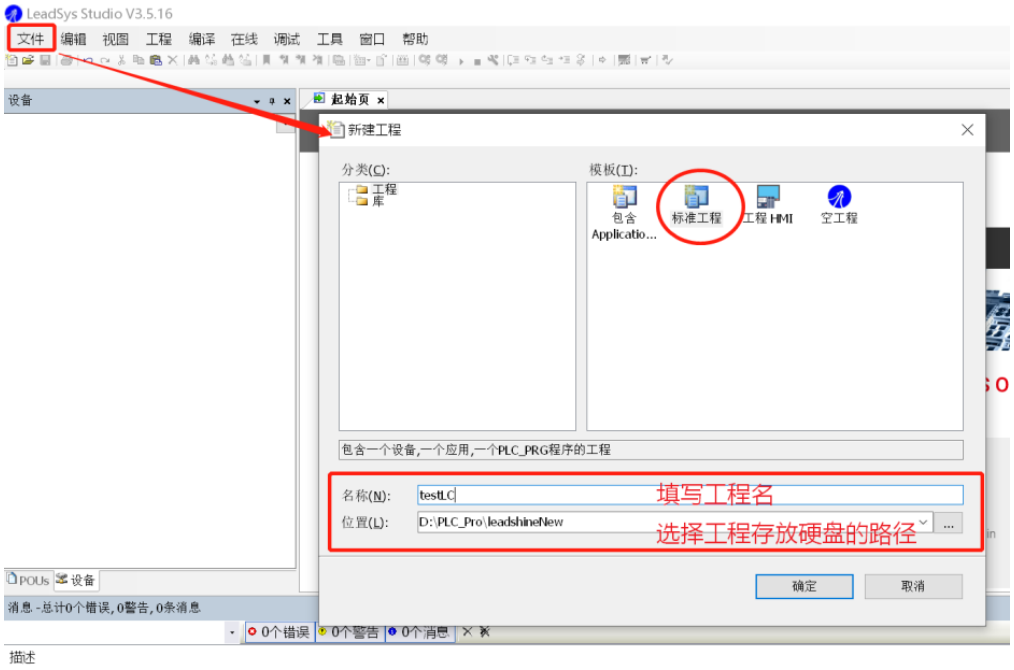


图 4.4 新建工程界面

然后选择 PLC 类型，如图 4.5 所示，设备选择 MC332(请根据 PLC 实际类型选择)。再选择主程序 PLC_PRG 的编程语言（默认为 ST），点击“确定”后，进入编程界面。

在左边的设备栏中用鼠标双击项目树上的 PLC_PRG(用户程序管理单元)，打开编程界面，在变量定义区定义主程序的变量，在代码编辑区编写主程序。如图 4.6 所示。

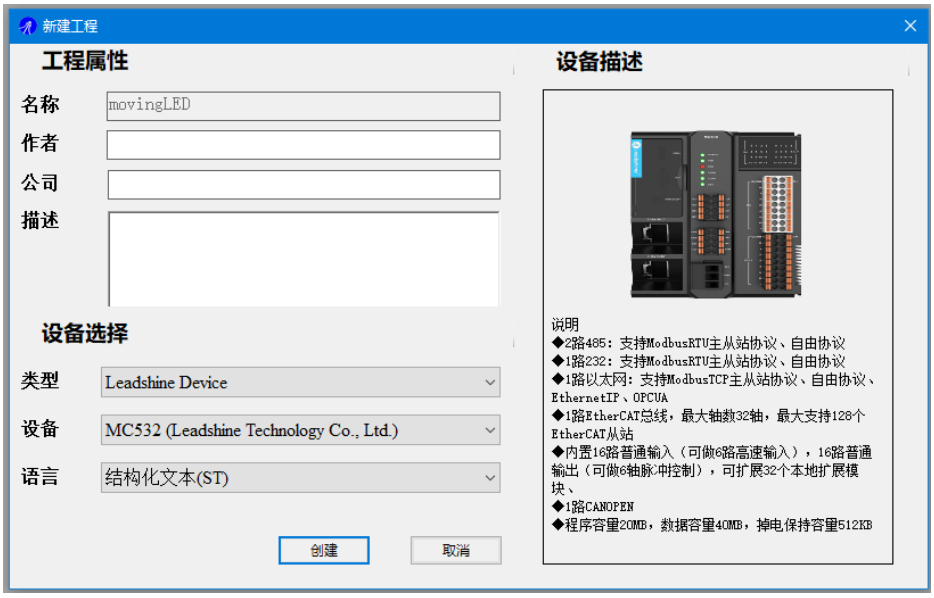


图 4.5　选择 PLC 类型、编程语言

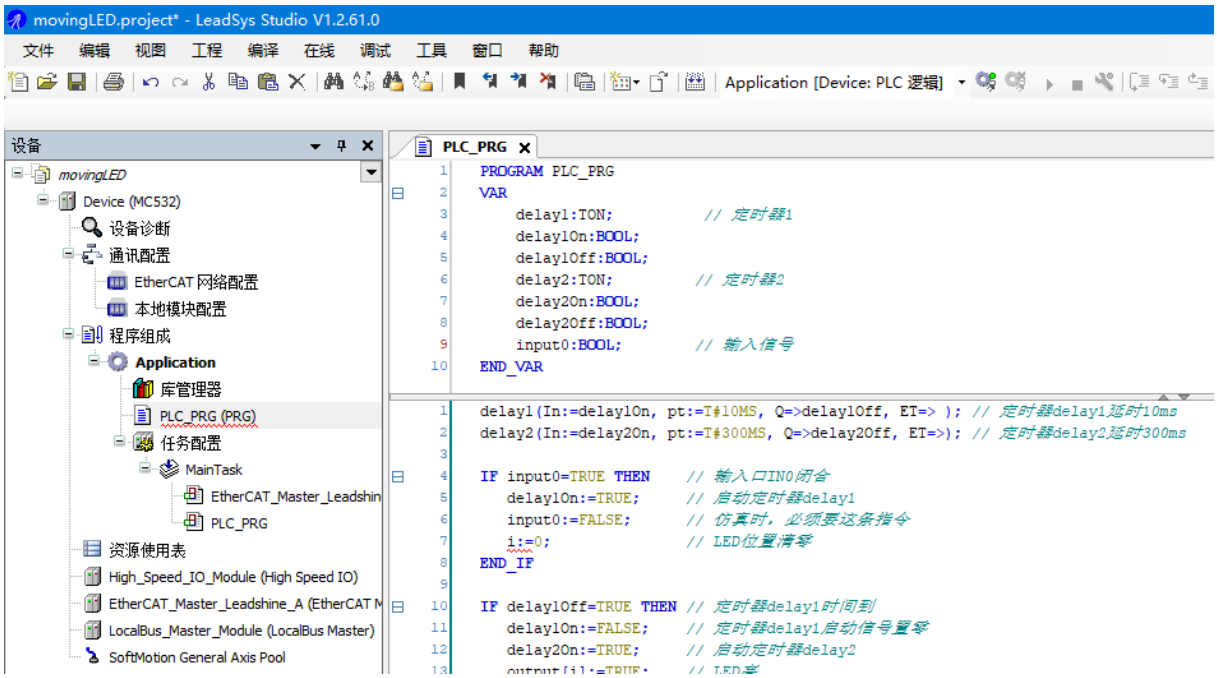


图 4.6　编程界面

详细代码如下。

```
PROGRAM PLC_PRG
VAR
    delay1:TON;          // 定时器 1
    delay1On:BOOL;
    delay1Off:BOOL;
    delay2:TON;          // 定时器 2
    delay2On:BOOL;
    delay2Off:BOOL;
```

```
    input0:BOOL;          // 输入信号
END_VAR


delay1(In:=delay10n, pt:=T#10MS, Q=>delay10ff, ET=> ); // 定时器 delay1 延时 10ms
delay2(In:=delay20n, pt:=T#300MS, Q=>delay20ff, ET=>); // 定时器 delay2 延时 300ms

IF input0=TRUE THEN     // 如果输入口 IN0 闭合
   delay10n:=TRUE;      // 启动定时器 delay1
   input0:=FALSE;       // 仿真时，必须要这条指令
   i:=0;                // LED 位置清零
END_IF

IF delay10ff=TRUE THEN // 如果定时器 delay1 时间到
   delay10n:=FALSE;     // 定时器 delay1 启动信号置零
   delay20n:=TRUE;      // 启动定时器 delay2
   output[i]:=TRUE;     // LED 亮
   Act0utput();         // 调用子程序写输出口
END_IF

IF delay20ff=TRUE THEN // 如果定时器 delay2 时间到
   delay20n:=FALSE;     // 定时器 delay2 启动信号置零
   delay10n:=TRUE;      // 启动定时器 delay1
   output[i]:=FALSE;    // LED 灭
   Act0utput();         // 调用子程序写输出口
   i:=i+1;              // LED 下移一位
   IF i=16 THEN         // 如果超出最后一个 LED，移至第一位
       i:=0;
   END_IF
END_IF
```

注：因全局变量 i 和 output[i]、子程序 Act0utput 还未定义，故程序有几处报错。

2） 定义全局变量

鼠标右键点击项目树中的“Application”，选择“全局变量列表”，如图 4.7 所示。给“全局变量列表”设一个名称为：GVL。

然后，在项目树中鼠标双击“GVL”打开全局变量列表的编辑界面，添加代码如下。

```
VAR_GLOBAL
    output: ARRAY[0..15] OF BOOL;  // 输出口的状态
    i:INT;                         // 输出口编号
END_VAR
```

注意：VAR_GLOBAL 前的{attribute 'qualified_only'}要删除！

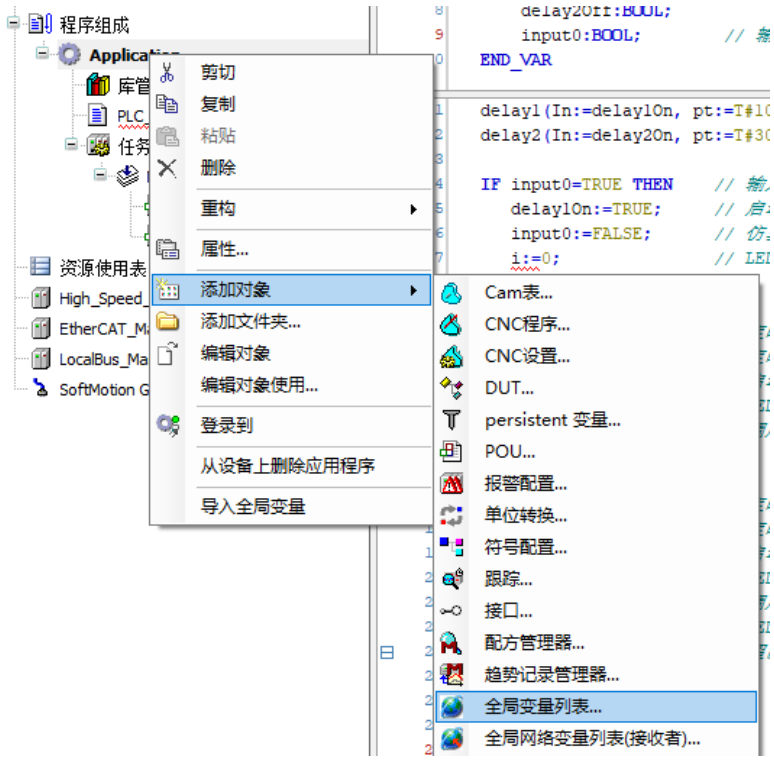


图 4.7　添加全局变量

3） 编写子程序

鼠标右键点击项目树中的“Application”，选择“POU”，如图 4.8 所示。

在弹出的对话框中选择“程序”、填写子程序的名称：ActOutput，选子程序的语言为“结构化文本”，然后点“打开”。在项目树中鼠标双击“ActOutput”，可打开子程序的编辑界面。子程序代码如下。

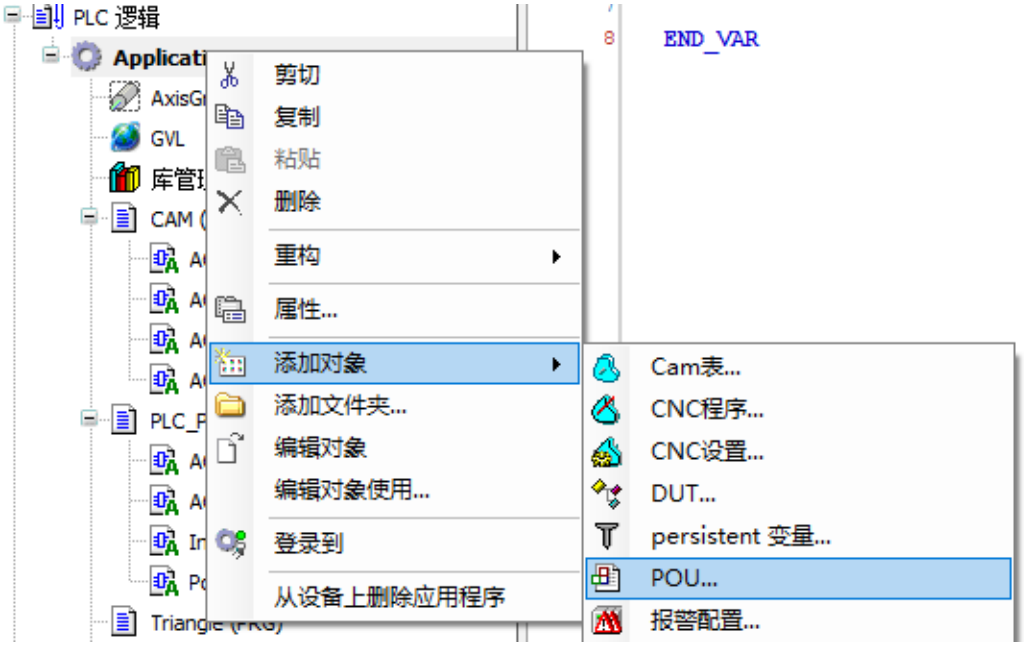


图 4.8　添加子程序

```
PROGRAM ActOutput
VAR
    out0: BOOL ;   // 输出口变量
    out1: BOOL ;
    out2: BOOL ;
    out3: BOOL ;
    out4: BOOL ;
    out5: BOOL ;
    out6: BOOL ;
    out7: BOOL ;
    out8: BOOL ;
    out9: BOOL ;
    out10: BOOL ;
    out11: BOOL ;
    out12: BOOL ;
    out13: BOOL ;
    out14: BOOL ;
    out15: BOOL ;
END_VAR

CASE i OF                    // 写输出口
    0: out0:=output[0];
    1: out1:=output[1];
    2: out2:=output[2];
    3: out3:=output[3];
    4: out4:=output[4];
    5: out5:=output[5];
    6: out6:=output[6];
    7: out7:=output[7];
    8: out8:=output[8];
    9: out9:=output[9];
    10: out10:=output[10];
    11: out11:=output[11];
    12: out12:=output[12];
    13: out13:=output[13];
    14: out14:=output[14];
    15: out15:=output[15];
END_CASE;
```

3. 程序仿真运行

在菜单中选“在线”，点击“仿真”，“仿真”前多出一个符号“√”。

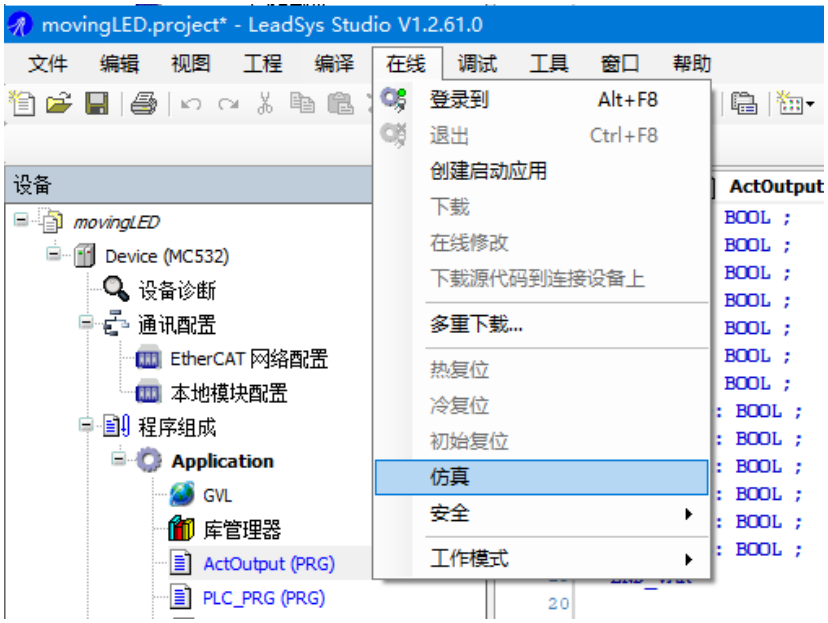


图 4.9　选择仿真模式

鼠标点击工具栏中的快捷键“”编译 PLC 程序，再点击登录按钮“”，如图 4.9 所示，可登录 PLC 并下载程序；再点击按钮“”，可运行程序。

仿真过程中首次下载程序时，有如图 4.10 所示的提示界面。选择“是”即可。

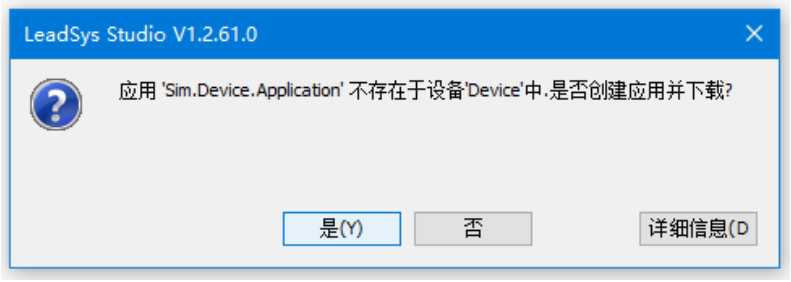


图 4.10　首次仿真下载程序的提示界面

在变量栏中找到变量 input0，点击其“准备值”栏，显示“TRUE”，再右键点击“TRUE”，选“写入“Device.Application”的所有值”，或按 Ctrl+F7 键，即将 TRUE 赋值给了变量 input0。“跑马灯”启动，可看到变量 i 和定时器的信号在不停地变化。如图 4.11 所示。

其他类型的变量可以在“准备值”栏中用键盘输入给定值。

打开子程序 ActOutput 的界面，可以看到输出口变量 out0~out15 的状态像跑马灯一样在变。如图 4.12 所示。

点击菜单栏上的快捷键“停止”可使程序停止；再点击快捷键“退出”，则程序退到编辑状态。如图 4.13 所示。

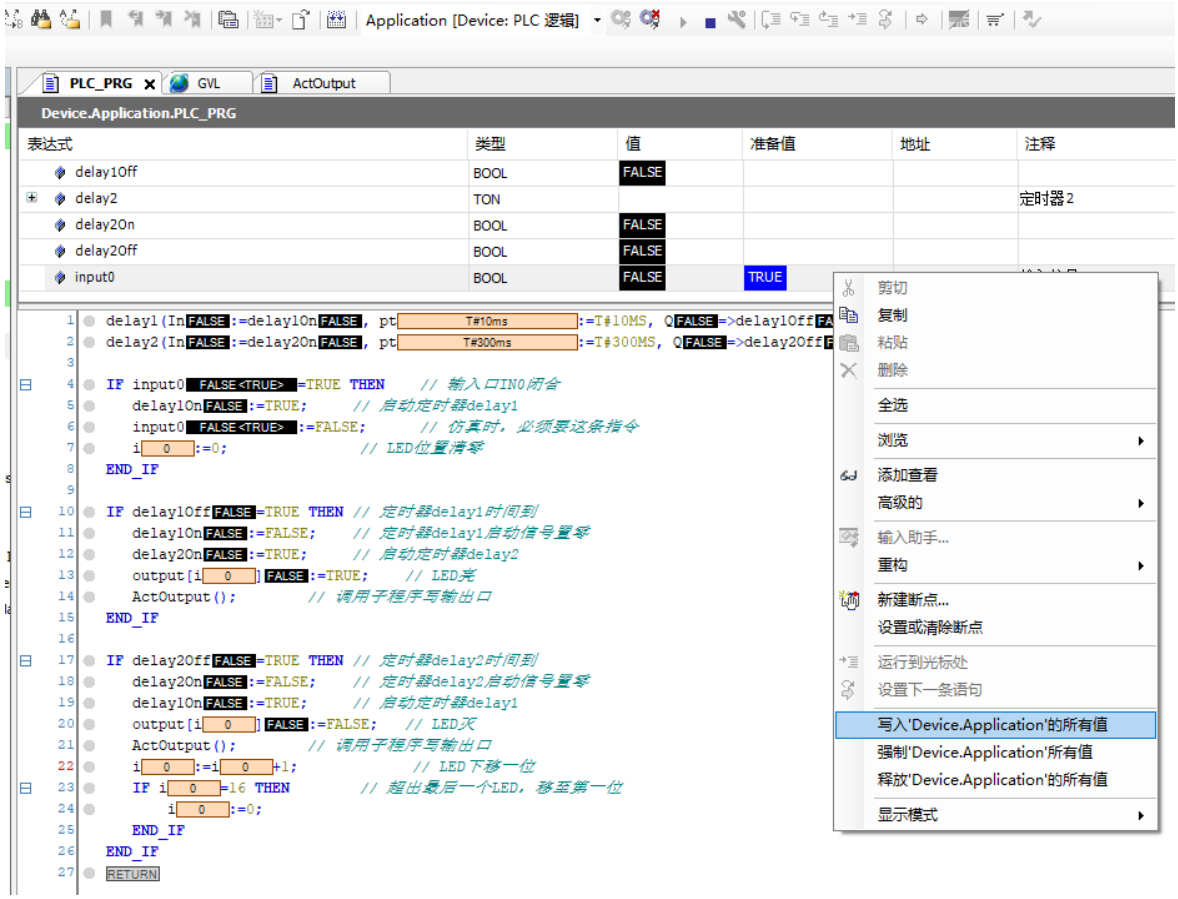


图 4.11　向变量写入数值

图 4.12　看跑马灯仿真结果

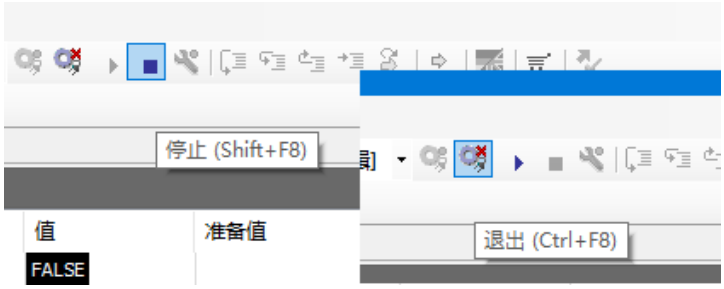


图 4.13　程序停止，退回至编辑状态

4. 程序实际运行

1）PC 机连接 PLC

点击菜单“在线”，点击“仿真”，让其前面的打勾符号消除。

打开 24V 电源，给 MC300CS 上电。

鼠标双击项目树中的“Device”，打开“Device”界面；再点击其界面上的“扫描网络”，软件自动扫描到 PLC，选中 MC-532，点击“确定”，PC 机和 PLC 即连接成功。如图 4.14 所示。

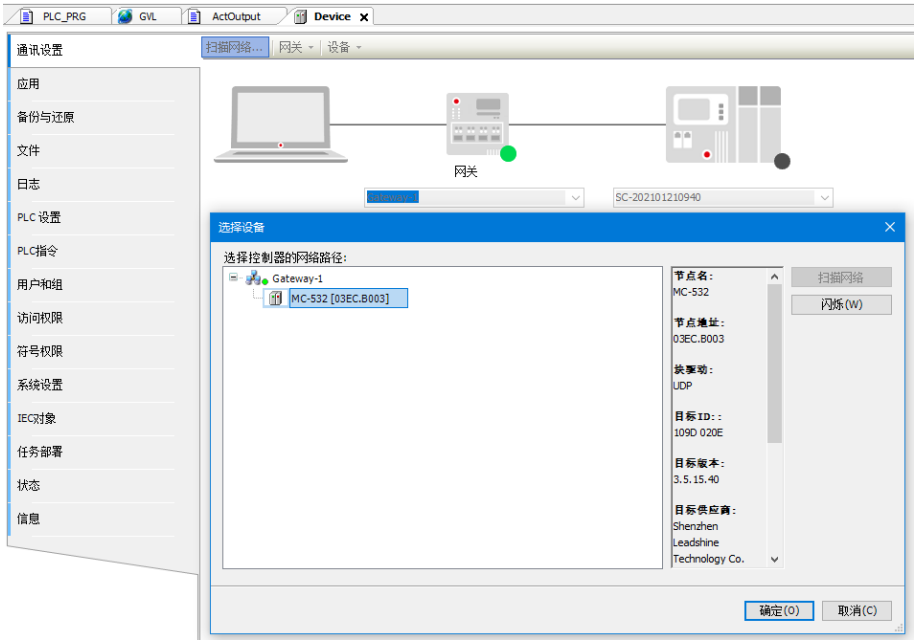


图 4.14　连接 PLC

2）映射 IO 变量

双击项目树中的“High_Speed_IO_Module”，进入 IO 设置界面，如图 4.15 所示；点击“Internal I/O 映射”栏，点击“变量”下的加号，显示 Genral_IO_IN 的所有变量。

在“通道”栏找到 Bit0，将鼠标移至该行“变量”栏中双击，出现符号“…”，如图 4.14 所示；点击该符号，出现“输入助手”界面；依次点击“Application”和“PLC _PRG”前的加号，显示已定义的变量；选择“input0”后，点击“确定”键，完成输入口与其控制信号的映射。

以同样的方法，可完成输出口与其控制变量 out0～out15 的映射，参见图 4.16。

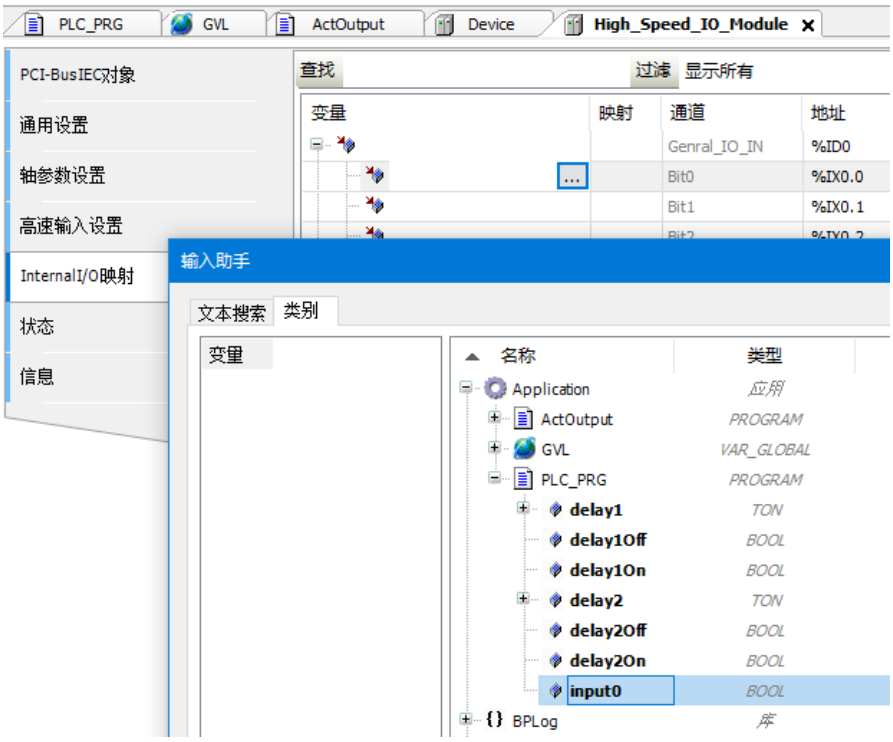


图 4.15　映射输入口变量

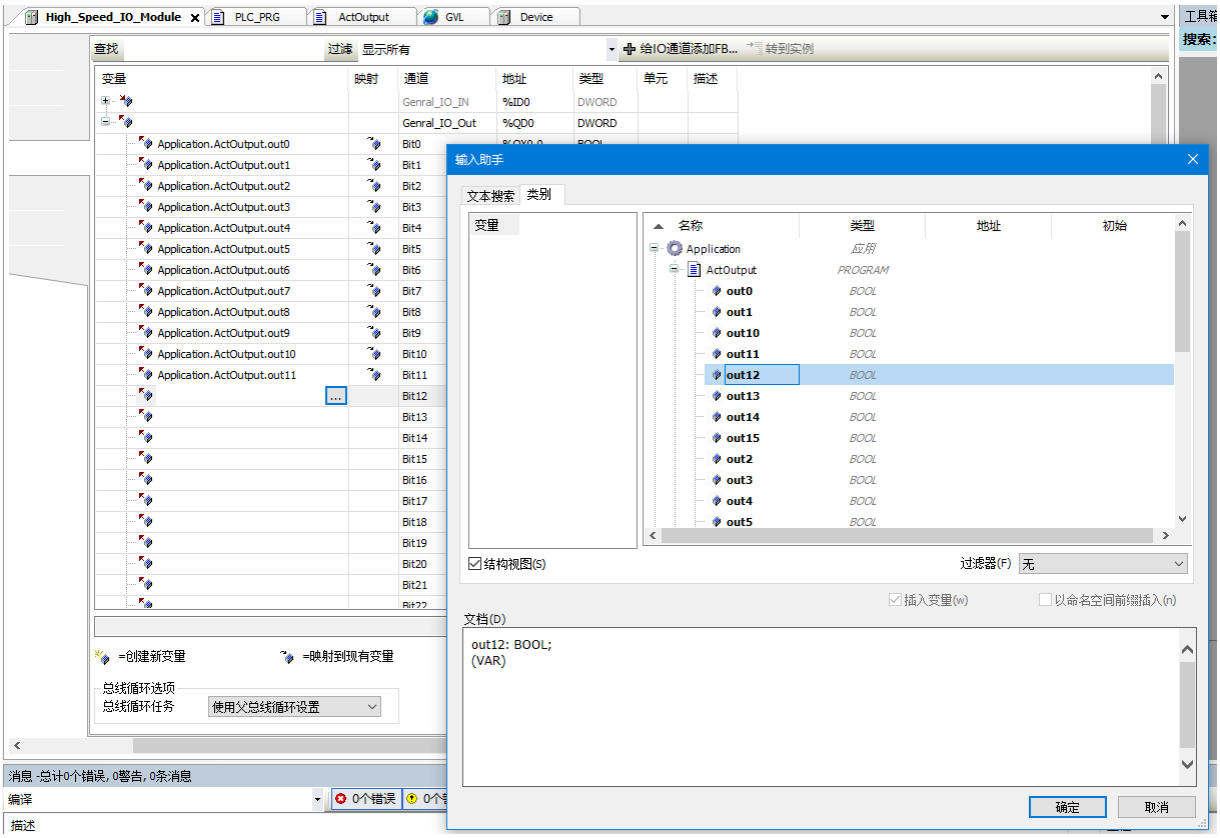


图 4.16　映射输出口变量

程序编译、下载的过程与仿真过程的相同。第一次下载时，提示界面如图 4.17 所示。选“是”即可。

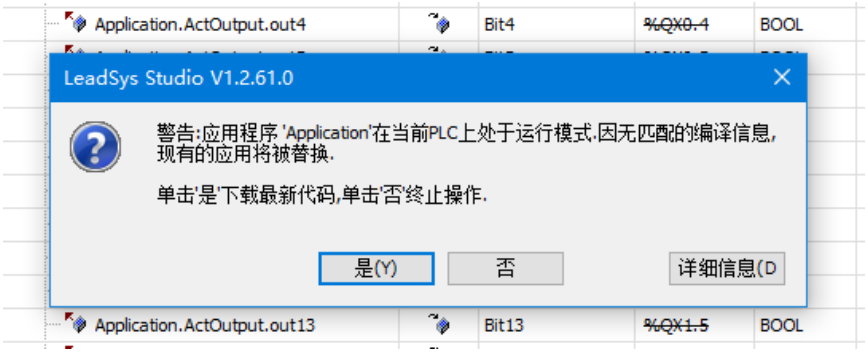


图 4.17 第一次下载程序时的提示

修改程序后再次编译、下载程序，提示界面如图 4.18 所示。一般选择“登录并下载”，这样程序开始运行时，各变量自动复位。

如果选“登录-在线修改”，程序中的变量保持上次运行时的值。

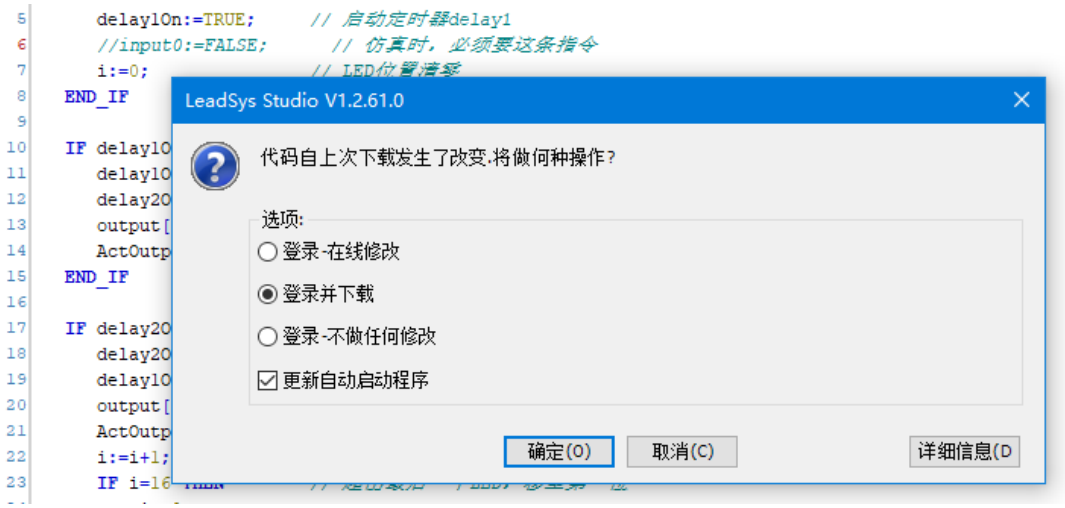


图 4.18 再次下载程序时的提示

点击按钮“ ”，运行程序；闭合 IN0 上的开关，输出口上的 LED 灯、输出口的指示灯按“跑马灯”方式轮流闪烁。

同样，点击菜单栏上的快捷键“停止”可使程序停止；再点击快捷键“退出”，则程序退到编辑状态。

为简化程序，输出口也可以直接用数组来控制。一个数组元素对应于一个输出口，且数组后的索引号及方括号要手动输入。如图 4.19 所示。

4.4.2. 在程序中直接完成 IO 映射的方法

如图 4.20 所示，MC300CS 系列 PLC 的 16 路输入口 Bit0~Bit15 对应的地址为%IX0.0~%IX0.15；也可以用一个 WORD 型地址表示为%IW0。

16 路输出口 Bit0~Bit15 对应的地址为%QX0.0~%QX0.15；也可以用一个 WORD 型地址表示为%QW0。

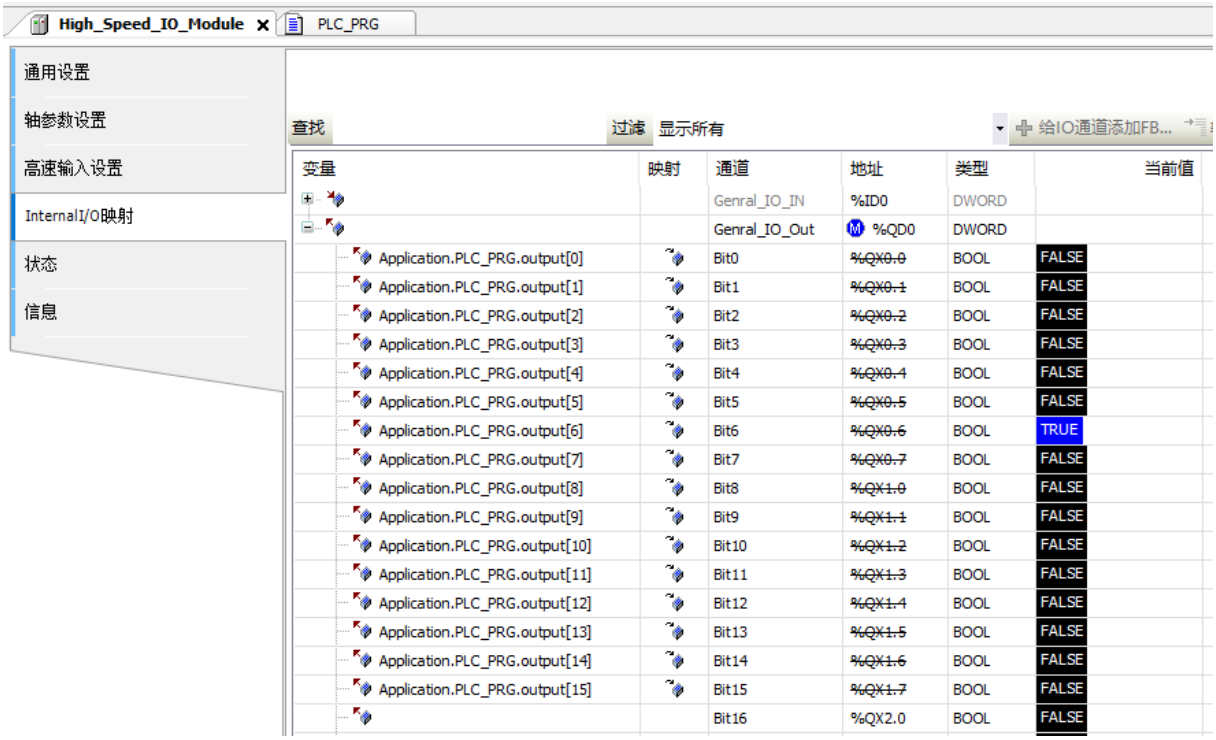


图 4.19　数组映射至输出口

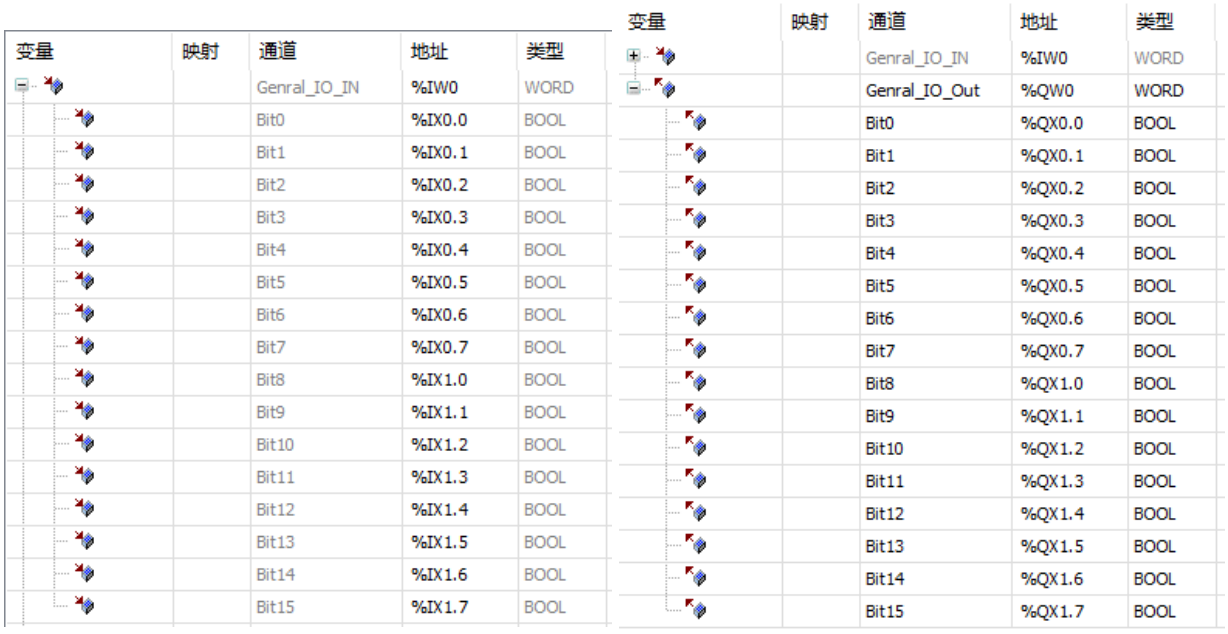


图 4.20　输入、输出口对应的地址

可以在定义变量时，直接用输入输出口的地址与变量建立映射关系；方法是在变量后加“AT 地址”。变量可以是 BOOL 型，也可以是 WORD 型。详见下面的 2 个例程。

注意：不能同时使用 BOOL 型和 WORD 型变量与输入口或输出口映射，否则，会发生变量定义冲突。

例程 1：用输入口 IN0 控制输出口 OUT1

```
PROGRAM PLC_PRG
VAR
    input0 AT %IX0.0: BOOL;   // 变量 input0 与输入口 IN0 建立映射关系
    output1 AT %QX0.1: BOOL;  // 变量 output1 与输出口 OUT1 建立映射关系
END_VAR
IF input0 = TRUE THEN
    output1 := TRUE;
    ELSE
    output1 := FALSE ;
END_IF
```

例程 2：将 16 路输出口定义为一个 WORD 型变量，并完成跑马灯控制。

```
PROGRAM PLC_PRG
VAR
    input1 AT %IX0.1: BOOL;    // 输入口 IN1 的信号
    output AT %QW0: WORD:=1;   // output 与输出口 0~15 建立映射关系
    delay1:TON;                // 定时器 1
    delay1On:BOOL;
    delay1Off:BOOL;
    delay2:TON;                // 定时器 2
    delay2On:BOOL;
    delay2Off:BOOL;
    R_TRIG1: R_TRIG ;          // 声明一个上升沿检测触发器
END_VAR

delay1(In:=delay1On, pt:=T#500MS, Q=>delay1Off, ET=> ); // 定时器 delay1 延时 500ms
delay2(In:=delay2On, pt:=T#500MS, Q=>delay2Off, ET=> ); // 定时器 delay2 延时 500ms
IF input1=TRUE THEN    // 如果输入口 IN1 闭合
  delay1On:=TRUE;     // 启动定时器 delay1
END_IF
IF delay1Off=TRUE THEN // 如果定时器 delay1 时间到
   delay1On:=FALSE;    // 定时器 delay1 启动信号置零
   delay2On:=TRUE;     // 启动定时器 delay2
END_IF
IF delay2Off=TRUE THEN // 如果定时器 delay2 时间到
   delay2On:=FALSE;    // 定时器 delay2 启动信号置零
   delay1On:=TRUE;     // 启动定时器 delay1
END_IF
R_TRIG1 (CLK:= delay1On);    // 将定时器 1 的启动信号给触发器
IF R_TRIG1.Q THEN            // 如果有上升沿
    output:= ROL(output,1);  // 输出口变量循环左移 1 位，实现跑马灯控制
END_IF
```

第5章 脉冲轴和总线轴的参数设置

在编写 MC300CS 系列 PLC 的电机控制程序之前，必须完成轴的参数设置。脉冲轴、EtherCAT 总线和 CANopen 总线轴的参数设置方法不尽相同。

5.1. 脉冲轴的参数设置

MC300CS 系列 PLC 的脉冲轴参数主要有：脉冲输出模式、脉冲当量、回零参数等。本节只介绍脉冲输出模式、脉冲当量的设置方法；回零参数在第 6 章第 3 节回原点运动中介绍。

5.1.1 脉冲模式与脉冲当量的设置

脉冲输出模式即控制脉冲型电机的信号类型，主要有三种：脉冲+方向模式、CW/CCW 输出模式、AB 相输出模式。

脉冲+方向模式：即脉冲信号 PUL 和方向信号 DIR，如图 5.1 所示。PUL 信号的脉冲数对应于电机的运行距离，其脉冲频率对应于电机运行速度。DIR 信号不同的电平决定电机运行方向。

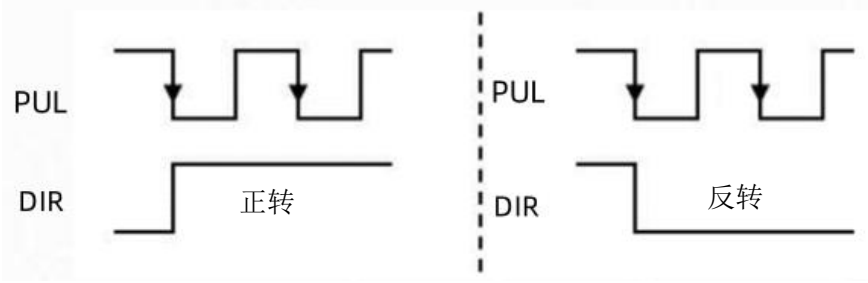


图 5.1　PUL+DIR 信号

CW/CCW 输出模式：为双脉冲输出模式，CW 为正转脉冲信号时，CCW 信号为高电平；CCW 为反转脉冲信号时，CW 信号为高电平。如图 5.2 所示。

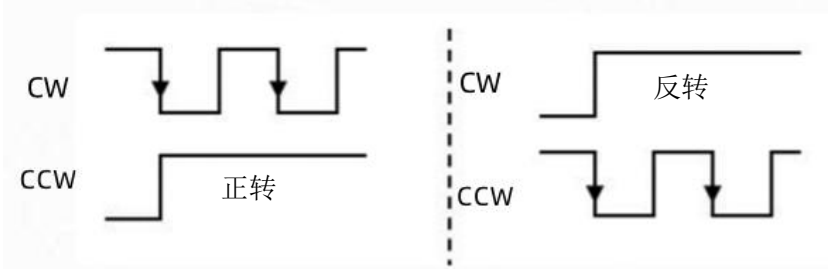


图 5.2　CW/CCW 信号

AB 相输出模式：该模式是 2 个相位相差为 90 度的方波信号。当 A 相信号相位超前，电机正转；当 A 相信号相位滞后，电机反转。如图 5.3 所示。

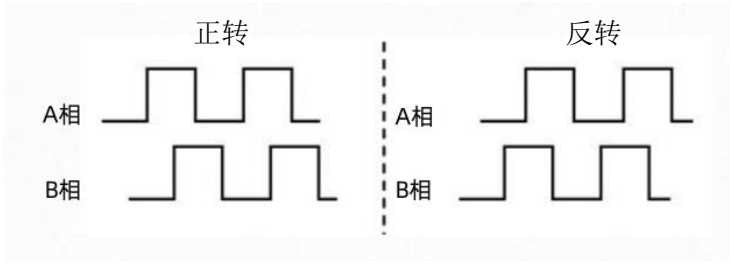


图 5.3　AB 相信号

为了便于控制平台运动，通常运动距离以 mm 为单位。控制运动平台移动 1mm，电机所需的脉冲数即为脉冲当量。

用户也可以根据实际情况将运动距离的单位设为厘米、旋转角度等；并要清楚机械传动机构的参数，如：齿轮的减速比、丝杆导程、同步带节圆周长，并计算输出平台移动 1 个用户单位需要的脉冲数。

例如：设以 mm 为用户单位，轴的传动机构为滚珠丝杆，其导程为 5mm，即电机转动一圈，平台移动的距离为 5mm；如图 5.4 所示。若伺服电机每转脉冲数为 10000，则脉冲当量计算方法如下。

$$\text{脉冲当量} = \frac{\text{伺服电机每转脉冲数}}{\text{滚珠丝杆的导程}} = \frac{10000}{5} = 2000 \ \text{脉冲/mm}$$

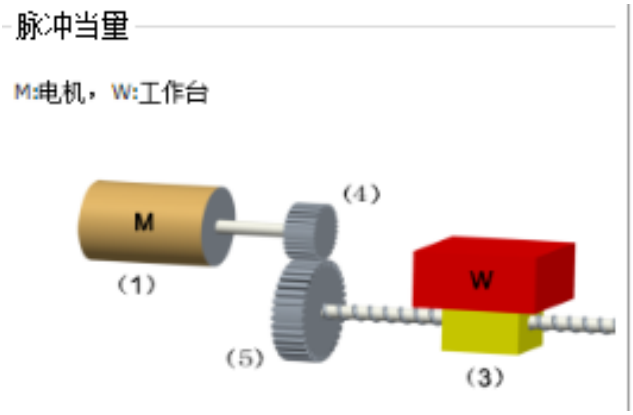


图 5.4　脉冲当量示意图

注意： MC300CS 系列 PLC 的高速脉冲口的最大输出频率为 200kHz。所以当脉冲当量设置为 10000 时，指令中设置的最大速度不能超过 20 unit/s。否则调用运动控制指令，速度超过 20 unit/s 时，指令会出错。

5.1.2 脉冲轴初始化与使能指令

在程序中首先要调用“LS_MotionControl_P”指令进行脉冲轴初始化。注意：“LS_MotionControl_P”是一个函数，故不需要对其进行实例化声明。

然后用“MC_Power”指令使能脉冲轴。之后才能执行电机的运动控制指令。

脉冲轴初始化指令的格式如下。

```
LS_MotionControl_P( stAxis:=脉冲轴结构体, xClearErr:=清除函数内部报错,
                    fLimtAxisSpeedJump:=速度跳变, xDone=>指令完成, xError=>指令出错,
                    eErrorID=>错误代码, xLimitAxisMoveFlag=>限制轴运动标志);
```

指令中的重要参数有：

stAxis：6 个脉冲规划轴结构体，调用“LS_MotionControl_P”之前需要用 ADR 指令为轴结构体指定具体脉冲轴轴号的地址。

xClearErr：用于清除函数内部报错状态。使用之前需要调用“MC_Reset”指令清除轴报错状态，再使用 xClearErr 进行函数内部错误清除。

fLimtAxisSpeedJump：限制单周期内速度跳变最大值。当速度跳变大于设置值时，模块报错并限制脉冲输出。

轴使能指令格式如下。

```
MC_Power( Axis:=轴号, Enable:=TRUE 时启动本指令, bRegulatorOn:=TRUE 时使能电机,
          bDriveStart:=TRUE 时关闭本指令紧急停止， Status=>运行状态 ，
          bRegulatorRealState=>轴使能信号状态 ， bDriveStartRealState=>允许驱动状态,
          Busy=>指令执行中, Error=>指令出错 ， ErrorID=>错误代码 );
```

注意：指令中 Enable、bRegulatorOn 和 bDriveStart 都要设为 TRUE。

5.1.3 脉冲轴点位运动例程

本例程使用 MC300CS PLC 控制 L7RS-400 伺服驱动器及电机执行点位运动指令 MC_MoveRelative（第 6 章有点位运动指令的详细介绍）。

L7RS-400 伺服驱动器设置的电机控制信号模式为脉冲+方向；设置了电机每转为 1000 个脉冲。本例将用户单位设为电机旋转圈数。即脉冲当量为 1000。

当输入口 IN0 上的开关闭合后，程序执行点位运动指令，运动距离 Distance 为 10 圈，速度 Velocity 为 20 圈/秒，加速度 Acceleration 为 50 圈/秒^2，减速度 Deceleration 为 100 圈/秒^2。

1、搭建硬件系统

参考图 2.1、2.7，给 MC300CS 接上 24v 直流电源。

参考图 2.2、2.3、2.4、2.7 分别在 MC300CS 的 IN0 端口接好开关，输出口 OUT0 接驱动器 PUL-端口、输出口 OUT1 接驱动器 DIR-端口、OUT7 接至驱动器使能输入，如图 5.5 所示。

将 PC 机的网线接入 MC300CS 的 EtherNET 端口；将 PC 机网口的 IP 地址改为 192.168.1.n，n 不等于 3 即可。

2、设置脉冲轴参数

打开 LeadSys studio 软件，新建工程后，在左侧设备项目树处鼠标选中

“High_Speed_IO_Module”后，双击进入配置页。

首先在“轴参数设置”下选择“轴 0”；再点击“启用”，使方框内出现符号“√”；若未点击“启用”，不能设置该界面参数。如图 5.6 所示。

设置脉冲模式：鼠标点击“脉冲高+方向高”右侧，出现如图 5.7 所示的下拉菜单，在其中选择所需脉冲模式。本例选择“脉冲高+方向高”模式。

再设置脉冲当量：分子为 1000，分母为 1。

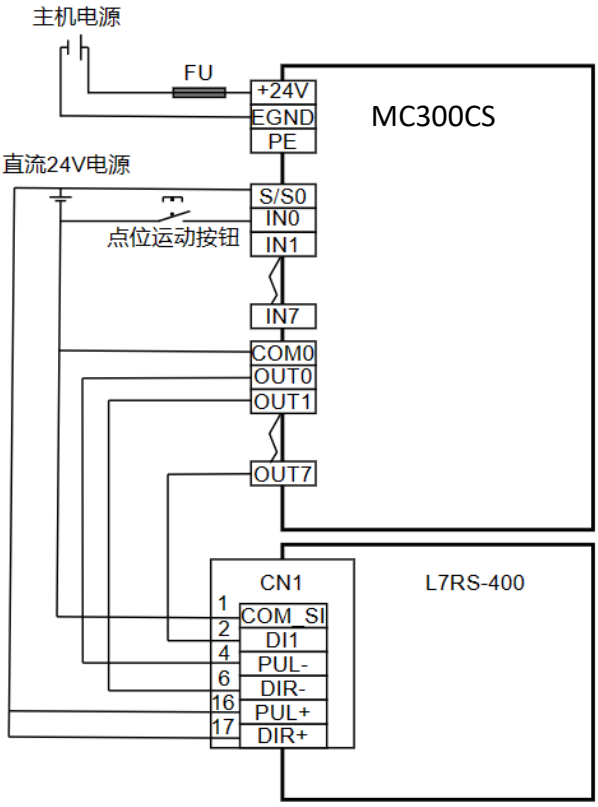


图 5.5　MC300 与 L7RS-400 驱动器的接线图

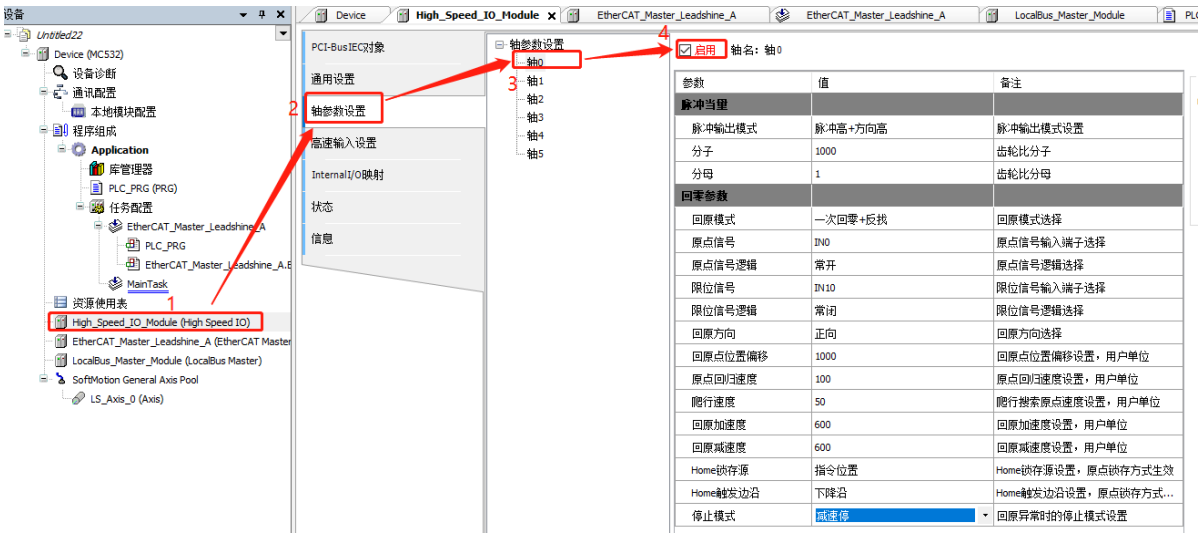


图 5.6　脉冲轴启动及参数设置

图 5.7　脉冲轴输出模式设置

3、编写程序

程序代码如下。

```
PROGRAM PLC_PRG
VAR
    axes:dut_pulse_axis;                    // 定义脉冲轴结构体
    MC_Power_0: MC_Power;                   // 实例化脉冲轴使能指令
    MC_MoveRelative_0: MC_MoveRelative;     // 实例化脉冲轴相对运动指令
    StartX AT %IX0.0: BOOL;                 // 脉冲轴点位运动启动按钮
END_VAR

axes.pulaxis_0:=ADR(LS_Axis_0); // 生成轴 LS_Axis_0 的参数地址
// 脉冲轴初始化：
LS_MotionControl_P(stAxis:=axes, xClearErr:=, fLimtAxisSpeedJump:=, xDone=>,
                   xError=>,eErrorID=>, xLimitAxisMoveFlag=> );
// 脉冲轴使能：
MC_Power_0 (Axis:=LS_Axis_0, Enable:=1, bRegulatorOn:=1, bDriveStart:=1,
            Status=>,bRegulatorRealState=>, bDriveStartRealState=>,
            Busy=>,Error=>, ErrorID=> );
// 点位运动：
MC_MoveRelative_0 ( Axis:=LS_Axis_0, Execute:=StartX, Distance:=10, Velocity:=20,
                    Acceleration:=50, Deceleration:=100, Jerk:=1000 , BufferMode:= ,
                    Done=>, Busy=> ,Active=> , CommandAborted=> , Error=> , ErrorID=> );
```

注意：程序运行前，需要将项目树中的 EtherCAT 总线主站删除；或在 EtherCAT 总线任务中调用 PLC_PRG，即用鼠标将 PLC_PRG 拖至 EtherCAT 总线任务之下。

4、添加变量的跟踪曲线

在程序调试过程中，LeadSys Studio 软件可显示变量随时间变化的曲线，为程序测试提供了非常直观的信息。使用方法如下。

- 鼠标右键点击项目树中的“Application”，选择“跟踪”，如图 5.8 所示。
- 在弹出的对话框中填写一个跟踪曲线的名称，默认名为 Trace。
- 在项目树中用鼠标双击跟踪曲线名，打开跟踪曲线的界面。

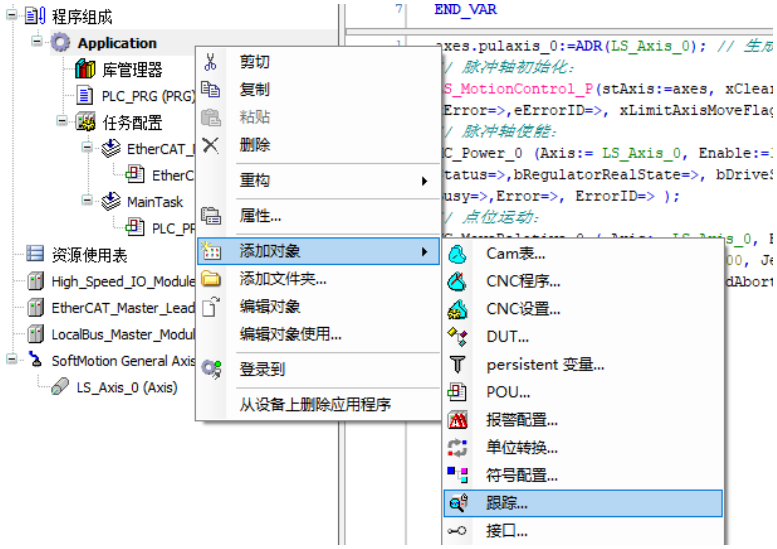


图 5.8　添加跟踪曲线

- 在跟踪曲线界面的右上角，点击“配置”，弹出对话框如图 5.9 所示。将其中的“任务”选“Main Task”，分辨率选微秒；再点击“高级”，根据需要修改“每个变量的追踪编辑器缓冲大小”，即采样总时间。

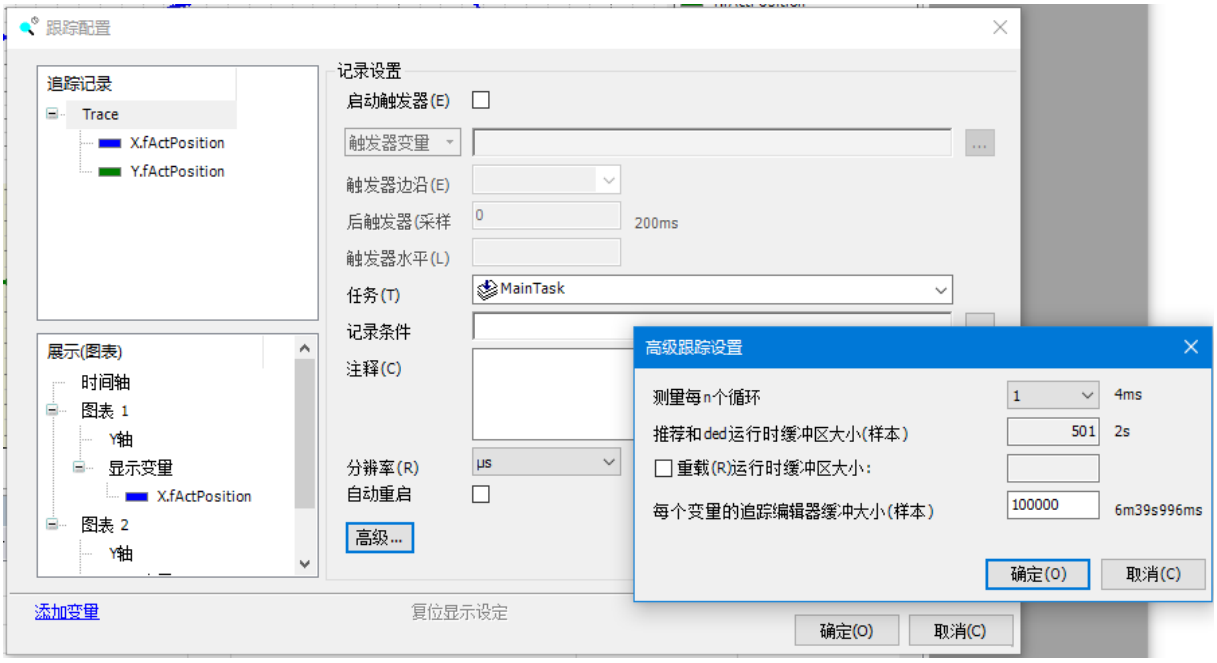


图 5.9　配置跟踪曲线的参数

- 再点击界面右上角的“添加变量”，弹出如图 5.10 所示的对话框。在“跟踪变量”中选“IOConfig_Globals”下的轴号 LS_Axis_0 中，分别选中、添加实际位置 fActPosition 和实际速度 fActVelocity 这 2 个变量。
  程序中的变量可以在 Application 下的 PLC_PRG 中选取。
- 运行程序后，在变量跟踪界面上点击鼠标右键，选“下载跟踪”，如图 5.11 所示，即可开始记录所添加变量的变化曲线。

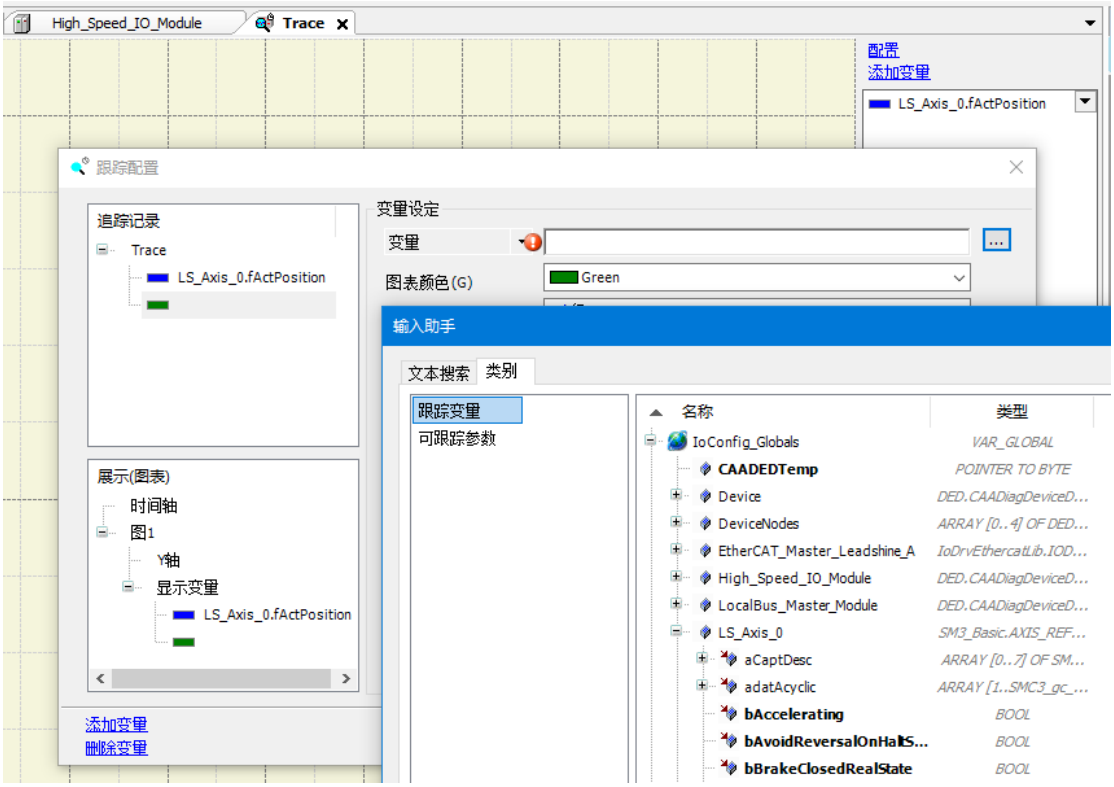


图 5.10　选择跟踪变量

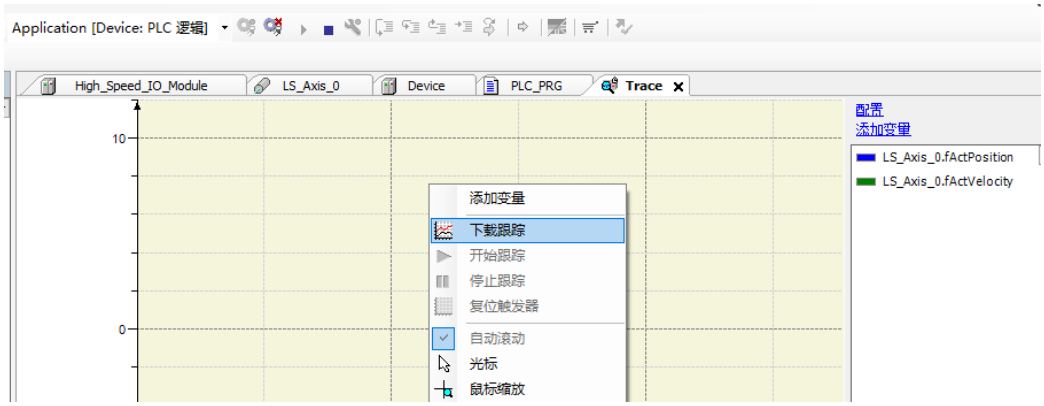


图 5.11　下载跟踪曲线

5、运行程序

编译、下载程序，运行程序。按下点位运动按钮后，程序执行点位运动指令。程序运行情况如图 5.12 所示、相对点位运动的位移曲线和速度曲线如图 5.13 所示。

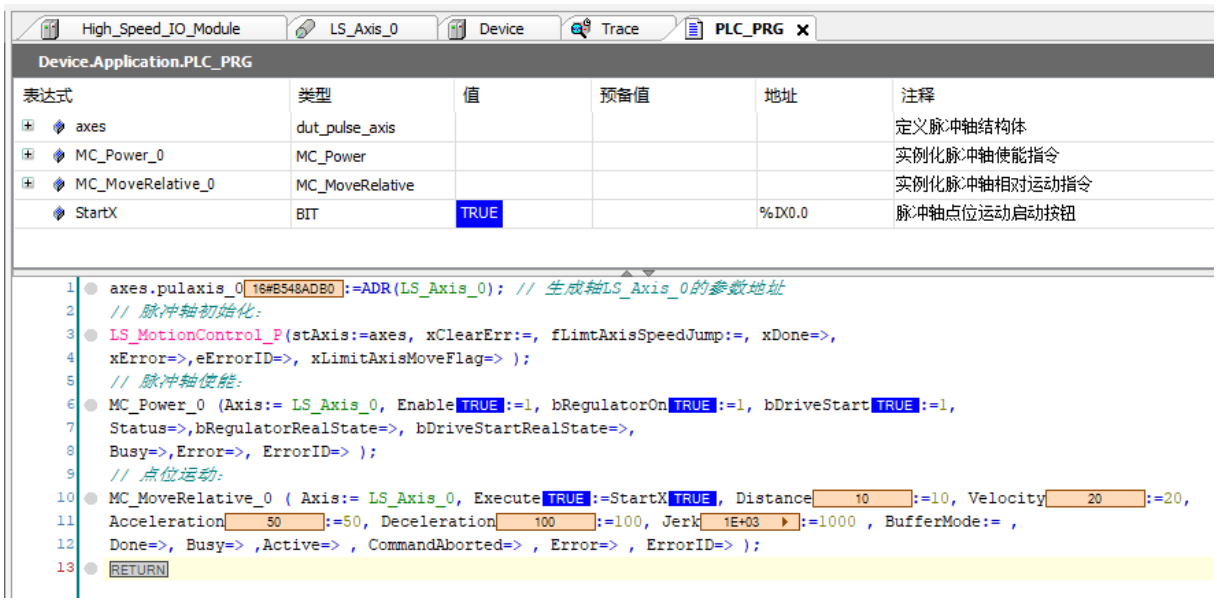


图 5.12　程序运行情况

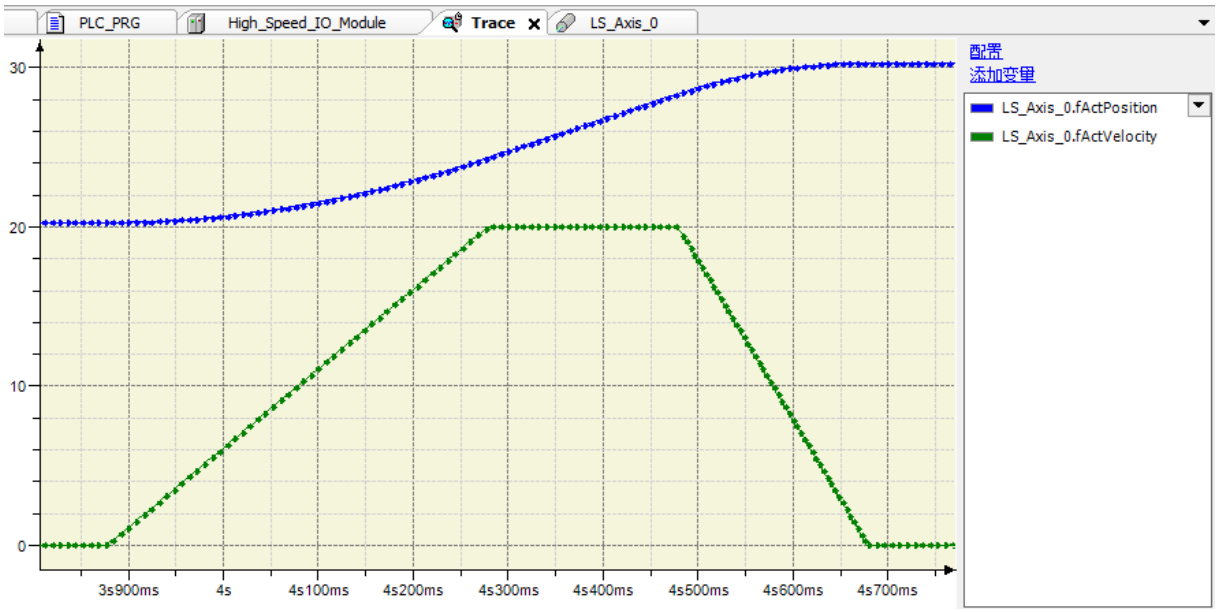


图 5.13　相对点位运动的位移曲线和速度曲线

5.2.　EtherCAT 总线轴参数设置

EtherCAT 总线轴的参数主要有：脉冲当量、回零模式等参数。

5.2.1 EtherCAT 轴添加方式

在使用 EtherCAT 总线轴之前，要将总线轴添加至项目树中。添加方式有以下两种。

方式一：

在 Leadsys studio 软件中，新建工程时会自动添加 EtherCAT 主站，不需要配置 EtherCAT 主站，鼠标选中“EtherCAT_Master_Leadshine_A”并右键单击，再选择添加设备，弹出“添加设备”窗口，选择“Servo Drives”，添加对应的驱动器设备，如图 5.14 所示。

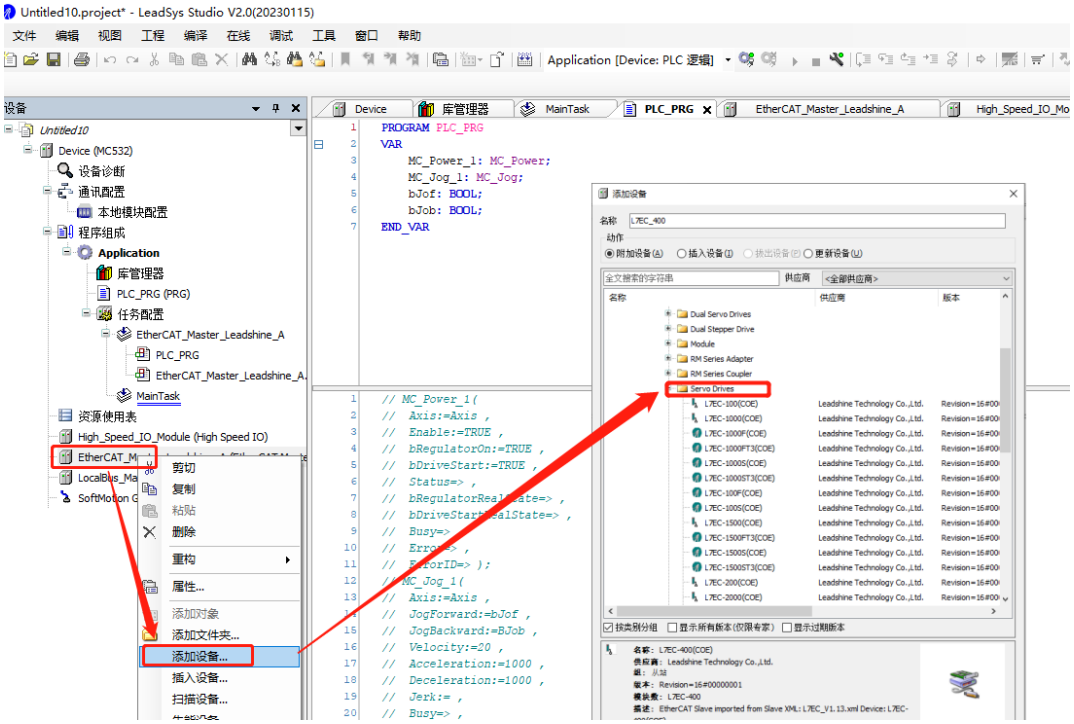


图 5.14　添加 EtherCAT 伺服驱动器

也可以选中“通讯配置”后，双击进入“通讯配置”界面。选中右侧工具箱设备进行拖动添加。如图 5.15 所示。

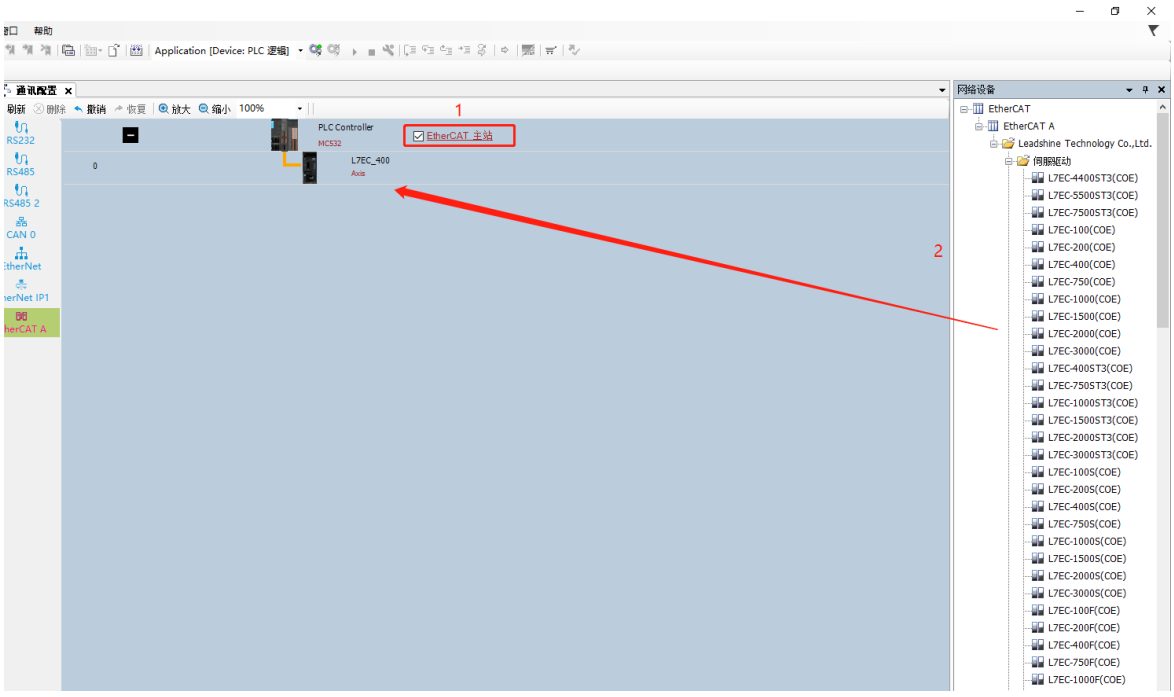


图 5.15　拖拽设备进行添加

添加后的设备会在“EtherCAT_Master_Leadshine_A”下显示，如图 5.16 所示。选中“L7EC-400(COE)”设备，点击鼠标右键，选择添加“SoftmotionCia402 轴”。

该方式通常在硬件系统没有接入总线设备时使用。

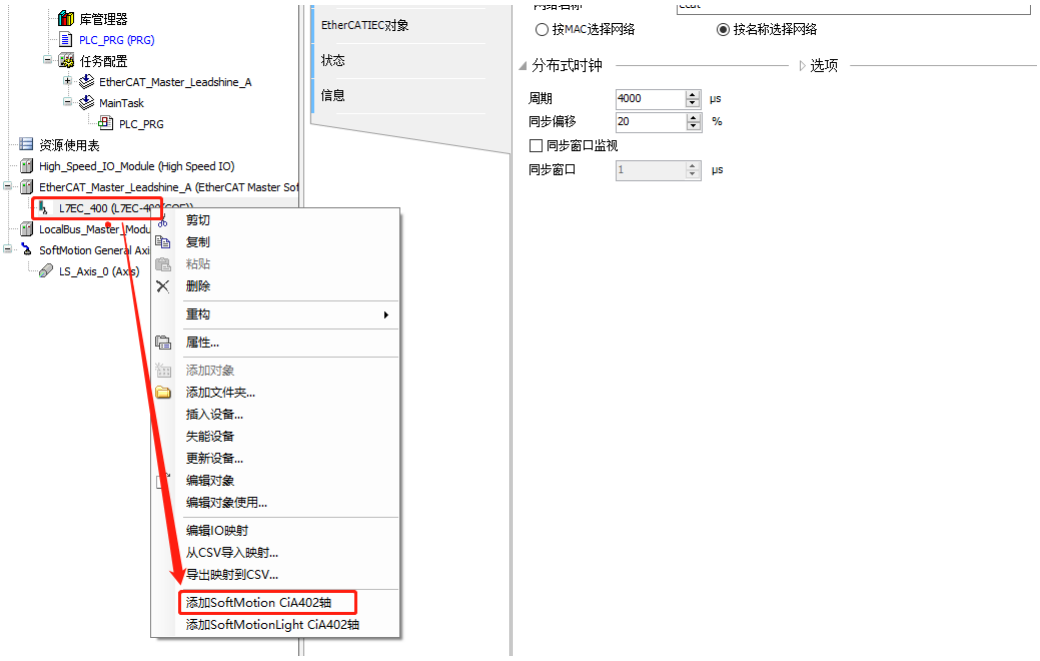


图 5.16　添加 CIA402 轴

方式二：

当硬件已接好时，通常采用该方法。

软件处于在线状态时，鼠标右键点击“EtherCAT_Master_Leadshine_A”。选择“扫描设备”，弹出“设备扫描”窗口。选择设备进行添加。如图 5.17 所示。

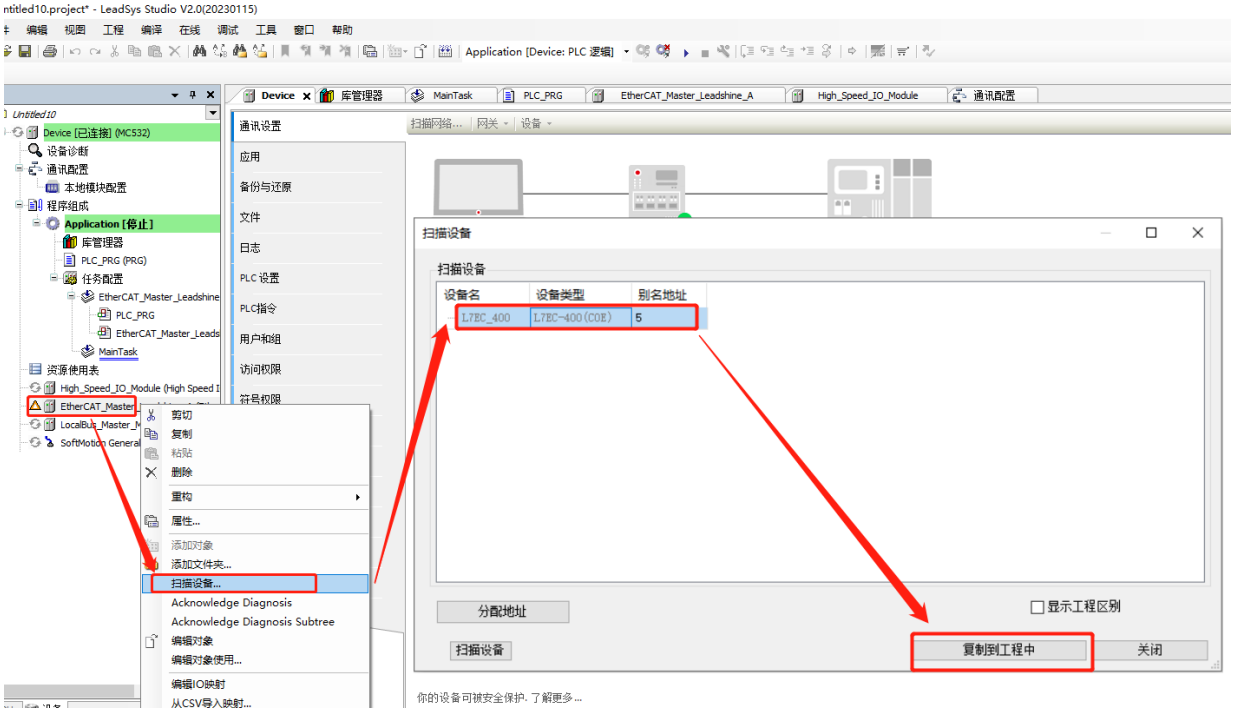


图 5.17　扫描设备进行添加

5.2.2 EtherCAT 总线轴参数设置方法

用鼠标在项目树中点击“L7EC-400(COE)”下的“AXis”，打开 EtherCAT 总线轴参数设置页面。选择“脉冲当量换算”后设置参数，步骤如图 5.18 所示。

图 5.18　单位换算配置图

“反向”：勾选后，电机与默认的运动方向相反运行。

“电机旋转一圈的指令脉冲数”：电机旋转一圈时需要的脉冲数，若电机旋转一圈时需要 10000 个脉冲，“电机旋转一圈的指令脉冲数”左侧方框应当设置为 10000。

“工作台旋转一圈的工作行程”：电机旋转一圈，工作平台运动的距离。

使用变速装置：根据设备的实际情况，对工作台行程进行设定。

不使用变速装置情况下，电机直接驱动丝杆。以电机转动一圈丝杆移动 10mm 为例，设置参数如图 5.19 所示。

图 5.19　不使用变速装置

使用变速装置时，伺服电机通过减速机构驱动丝杆。例如：减速机构的减速比为 2：1，丝杆导程为 10mm，设置参数如图 5.20 所示。

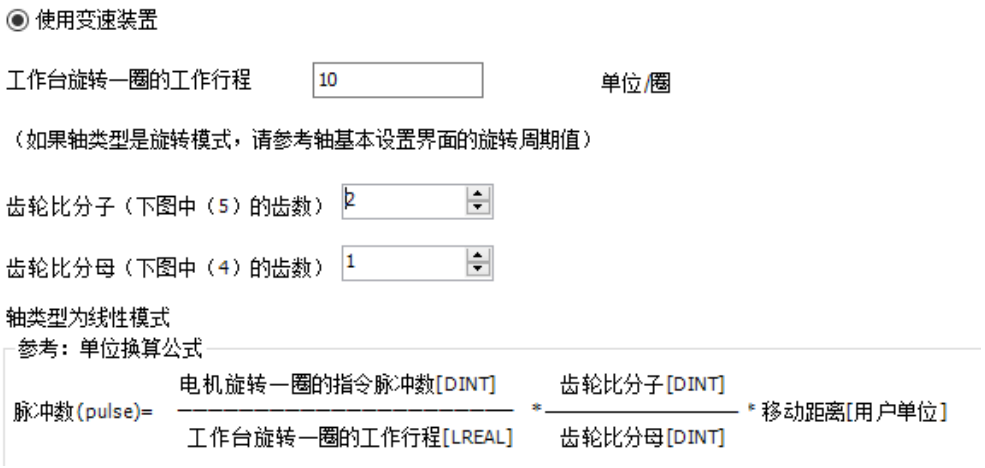


图 5.20　使用变速装置

EtherCAT 总线轴运行时需要设置运行模式、工作模式、以及一些运动参数设置。如图 5.21 所示。

图 5.21　EtherCAT 常用参数设置

EtherCAT 总线轴运行模式有虚拟模式和实轴模式。

虚拟模式：即可以用于模拟运行调试，不需要接入实际的伺服电机，通过模拟运行，可以获得调试参数。勾选“虚拟模式”前面的方框，即进入虚拟模式。

实轴模式用于控制实际的伺服电机。有些电机参数必须在实轴模式下才可以读写，如：伺服功能码、在线 COE。

轴类型分为有限型和模数型。

“有限”：即直线运动模式，轴在的反馈位置增大时表示轴正在进行正向运动，反之则是反向运动。

“模数”：即旋转运动模式，电机无限旋转，不限制移动范围。指令位置在 0~模数值 u 的区间内循环，模数值无法设置负值，且允许设置的范围为 0~360。

“软件限位”：勾选使能后，可限制轴运动的范围；并要设置正、负方向的极限位置。如果轴运动到软件限位位置时，PLC 将迅速停止运动，停止时间与“软件错误

反应”中的参数有关。

“软件错误反应”：设置运动过程中发生软件限位时停止运动的减速度和最大运动距离。该设置在软件限位使能状态下才有意义。

“动态限制”：用于设置运动指令中的参数的最大值。速度、加速度、减速度不能设置为 0；设置为 0 时将产生警告。

“速度斜坡类型”：设置轴的速度曲线类型（第 6 章中有详细介绍）。

“标识符”：轴的 ID 号。

“位置滞后监视”：设置位置滞后限制，监控轴的运行状态。

在执行定位指令和速度指令期间，伺服电机驱动器实际工作在 CSP（周期性同步位置）模式，位置曲线的规划在控制器侧完成。控制器通过 0X607A 向伺服驱动器发送目标位置，伺服驱动器驱动伺服电机运动，电机编码器的位置通过 0X6064 反馈到控制器，由于伺服驱动器和电机本身的原因，0x607A 和 0X6064 之间存在偏差值。

该差值即为位置滞后误差，如果轴的位置滞后误差的绝对值超过设置轴限制值时，则轴报告跟随误差过大故障并进入停止状态。

5.2.3 Ethercat 总线轴例程

本例程使用 MC300CS 控制 L7EC-400 伺服驱动器及电机执行点位运动。

1、搭建硬件系统

用网线将 L7-400EC 伺服驱动器与 MC300CS 通过 EtherCAT 口连接。其他和上例相同。

2、编写软件

打开 LeadSys Studio 软件，新建工程后，连接 PLC。选中左侧项目树中“EtherCAT_Master_Leadshine_A”，点击鼠标右键。选择“扫描设备”，添加 L7-400EC 伺服驱动器。参见图 5.17。

选中左侧项目树中“Axis”，进入配置页后选择“脉冲当量换算”选项卡，将“电机旋转一圈的指令脉冲数”设置为 10000，“工作台旋转一圈的工作行程”设置为 1。如图 5.22 所示。

图 5.22 设置脉冲当量

选中“SoftMotion 驱动通用”选项卡，轴类型设置为“有限”，“动态限制”中“速

度”设置为 30000，加减速度均设置为 100000，加加速度设置为 100000。如图 5.23 所示。

图 5.23 轴类型、软件限位、动态参数设置

将任务配置中“MaskTask”下的“PLC_PRG”移至“EtherCAT_Master_Leadshine_A”。如图 5.24 所示。

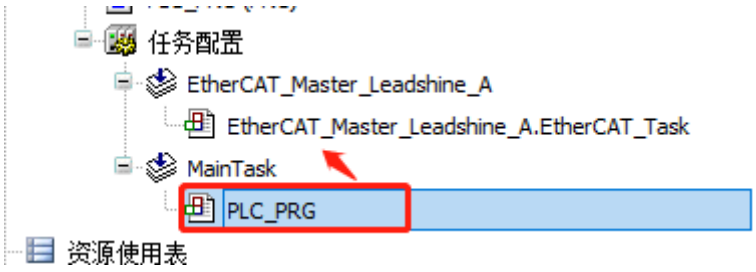


图 5.24 在“EtherCAT_Master_Leadshine_A”中调用“PLC_PRG”

程序代码如下。和脉冲轴不同的是：总线轴不需要进行初始化。

```
PROGRAM PLC_PRG
VAR
    MC_Power_1: MC_Power;                  // 实例化使能指令
    StartX AT %IX0.0: BOOL;                // 点位运动启动按钮
    MC_MoveRelative_1: MC_MoveRelative; // 实例化相对运动指令
END_VAR

MC_Power_1( Axis:=Axis, Enable:=TRUE , bRegulatorOn:=TRUE , bDriveStart:=TRUE,
            Status=> ,bRegulatorRealState=> , bDriveStartRealState=> , Busy=> ,
            Error=> , ErrorID=> ); // 使能 EtherCAT 总线轴

//进行相对运动：
MC_MoveRelative_1(Axis:=Axis , Execute:=StartX , Distance:=10 , Velocity:=10 ,
                  Acceleration:=10000 , Deceleration:=10000 , Jerk:= , BufferMode:= ,
                  Done=> , Busy=> , Active=> , CommandAborted=> , Error=> , ErrorID=> );
```

5.3 CAN 总线轴参数设置

将 CAN 总线电机及驱动器与 MC300CS PLC 连接好后，进行 CAN 总线轴设备添加、参数设置，方法如下。

5.3.1 添加描述文件和库文件

在设置 CAN 轴的参数之前，需要添加 CAN 轴的描述文件和 CANopen 函数库。

添加描述文件

新建工程后，单击“菜单栏-工具”选项，选择“设备存储库”，点击安装，选择对应的描述文件进行添加，如图 5.25 所示。

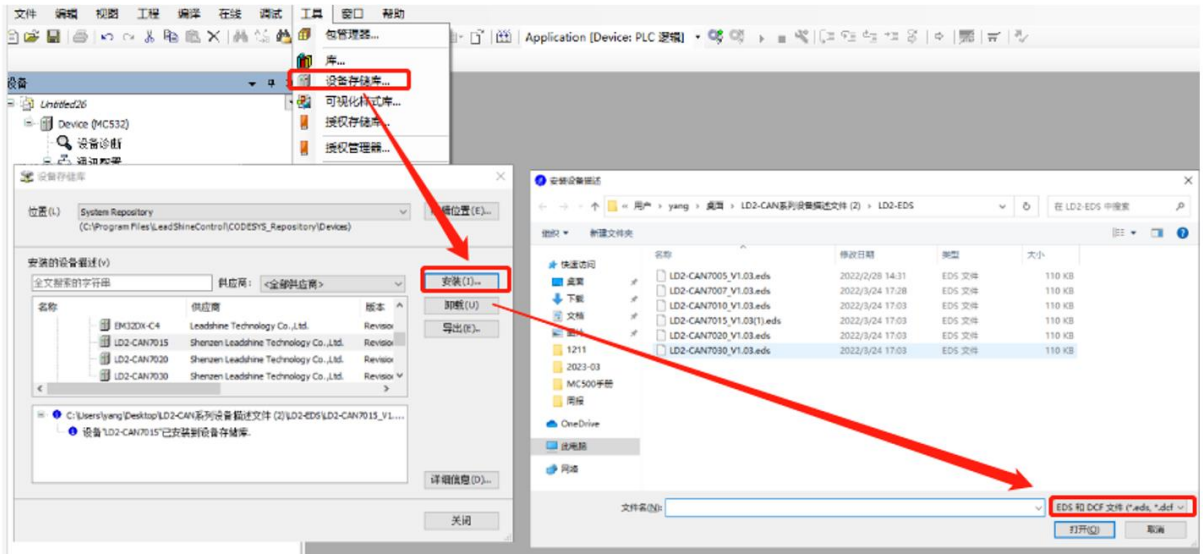


图 5.25　添加 LD2-7030 驱动器描述文件

添加库文件

1）单击“菜单栏-工具”选项，选择“库”，点击安装。找到 CANopen 库文件“CANopenLib.compiled-library”打开，添加到“Application”中，如图 5.26 所示。

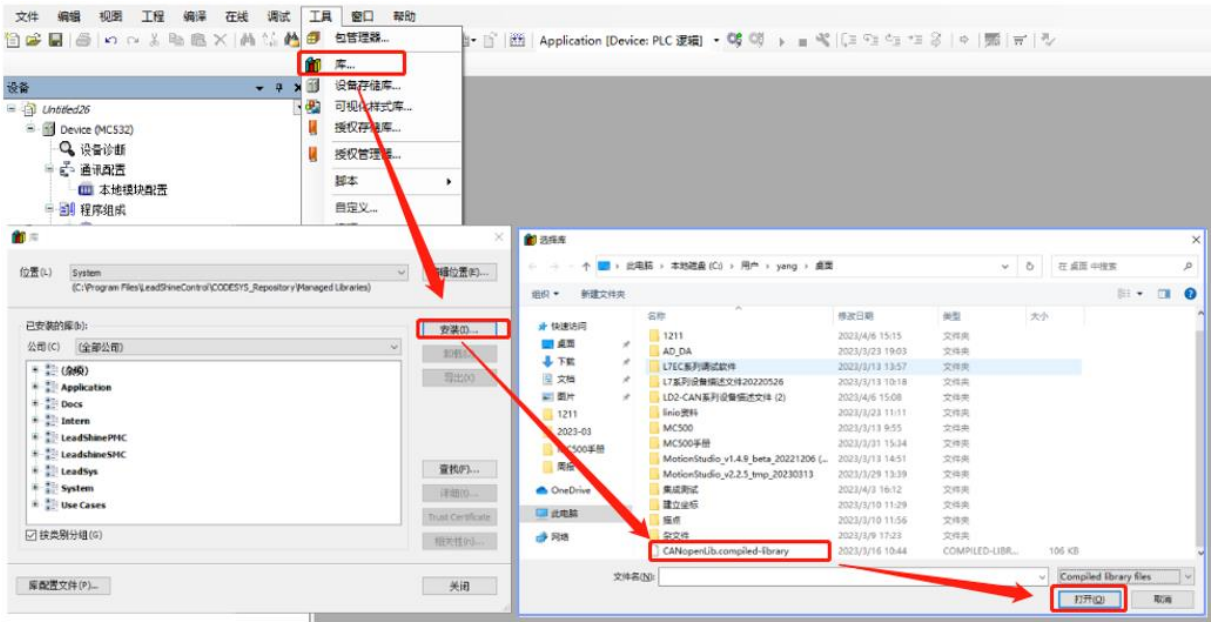


图 5.26　添加“CANopenLib.compiled-library”

2） 双击左侧项目树的“库管理器”，选择添加库，在“Application”中找到“CANopenLib”，点击确定进行添加，如图 5.27 所示。

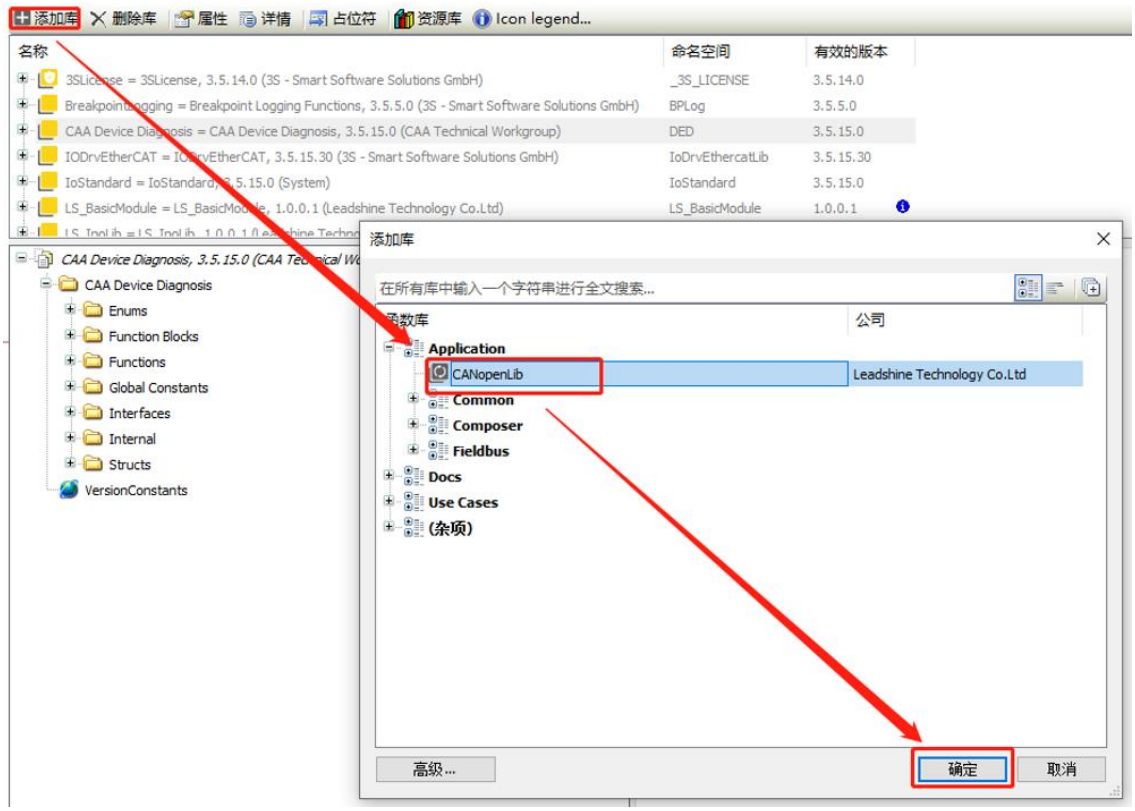


图 5.27　添加“CANopenLIB”库文件

5.3.2 添加 CAN 总线轴

CAN 总线轴添加方法有两种。

1、直接添加

1）选中通讯配置后，双击鼠标左键进入模块配置页。选中“CAN_0”进入 CAN 总线配置，勾选“CANopen 主站”后，左侧项目树出现 CANbus 总线，如图 5.28 所示。

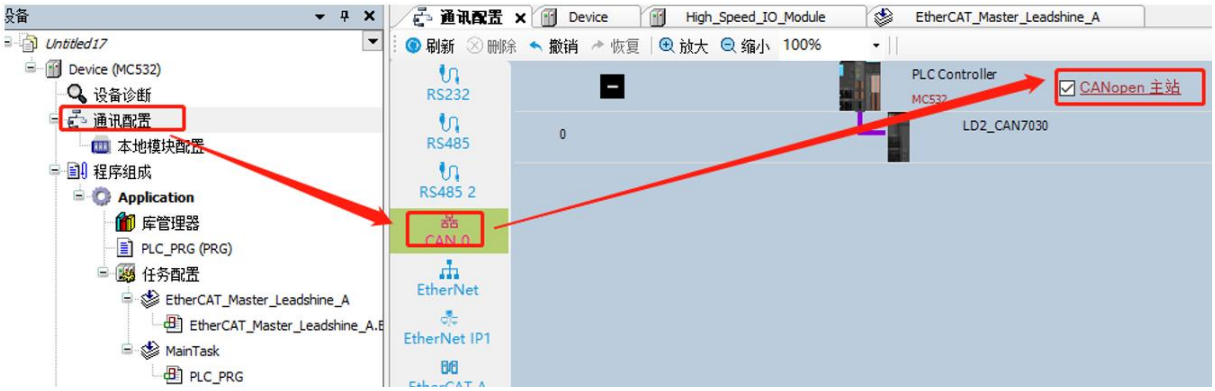


图 5.28　添加 CANopen 主站

2）双击或者拖拽添加左侧工具箱中网络设备，如图 5.29 所示。

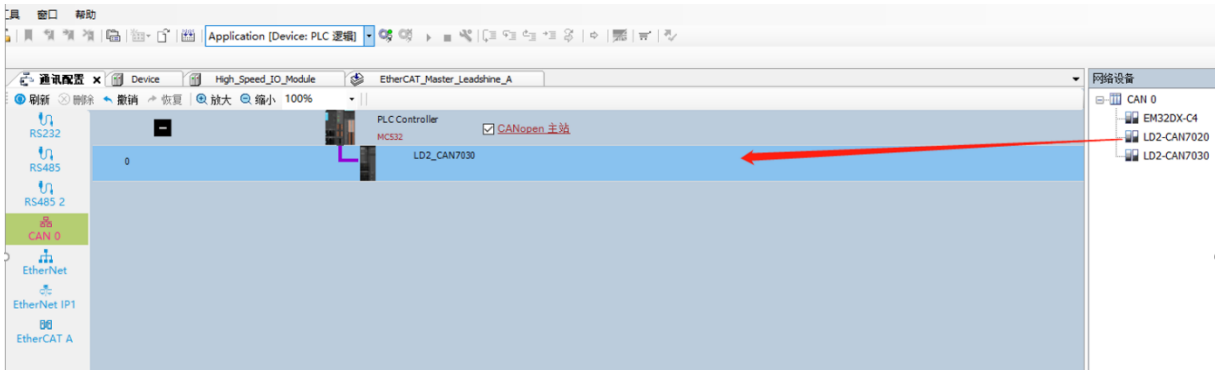


图 5.29　添加 CAN 从站设备

2、扫描添加

1）选中通讯配置后，双击鼠标左键进入模块配置页。选中“CAN_0”进入 CAN 总线配置，勾选“CANopen 主站”后，左侧项目树出现 CANbus 总线，如图 5.30 所示。

2）选中“CANopen_Manager”，点击鼠标右键选择扫描到的设备，如图 5.30 所示，添加“LD2_CAN7030”轴。

注意：推荐使用通过扫描方式进行设备添加，扫描过程中会自动配置设备站号、波特率。

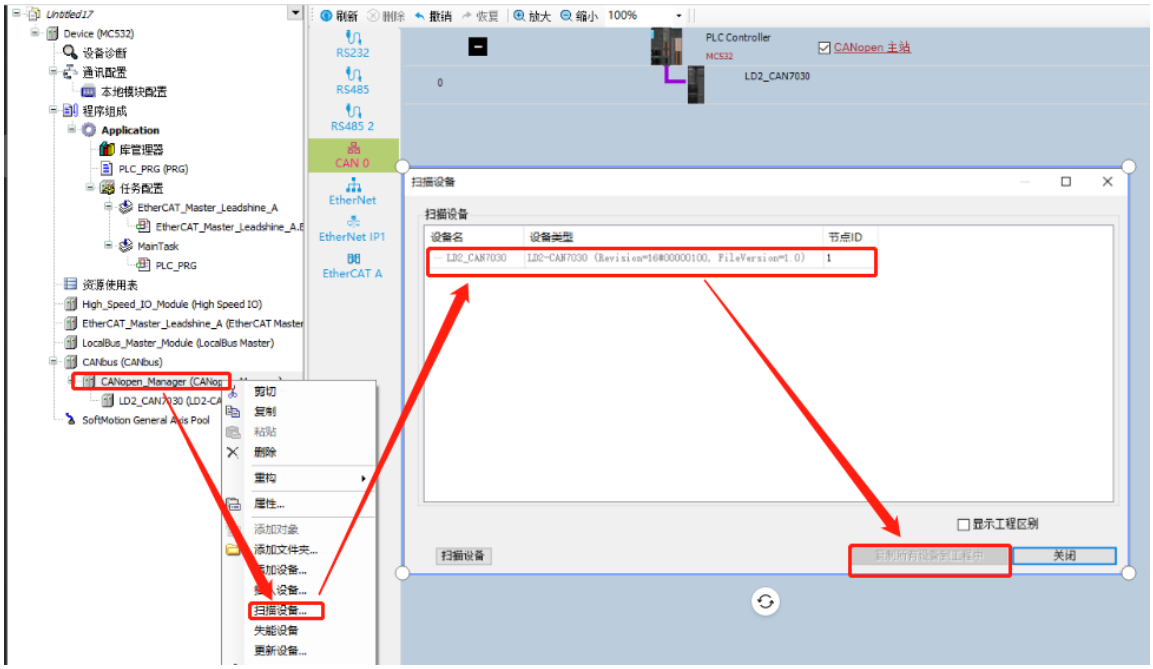


图 5.30　EtherCAT 设备扫描窗口

5.3.3 设置 CAN 总线轴参数

MC300CS PLC 控制 CAN 总线电机前，需要设置 CAN 总线通讯参数、主站和从站的参数。

1） CAN 总线通讯参数的设置

用鼠标在项目树中点击“CANbus”，将 CANbus 参数设置页打开，如图 5.31 所示。

图 5.31　CANbus 配置页

其中：

“网络”：对应 PLC 上的 CAN 总线通讯端口号。MC300CS PLC 只有一个 CAN 口，编号为 0。

“波特率”：CAN 总线通讯的波特率，单位为 Kbit/s，设置范围是 10~1000 Kbit/s。

2） CAN 主站参数设置

用鼠标在项目树中点击“CANopen_Manager”，将 CANopen_Manager 界面打开，如图 5.32 所示。

图 5.32　CAN 主站参数设置

图 5.32 中注释序号的参数说明如表 5.1 所示。

表 5.1　界面参数说明

编号	选项名称	参数说明
1	节点 ID	主站在 CANopen 网络唯一标示号，默认 127，范围 1~127
2	检查与修正配置	当多个从站被添加到 CANopen 系统中，由于不同从站 EDS 文件可能默认配置了 COB-ID 或者修改了从站的节点 ID，可能导致从站或者主站的节点 ID 重复或者 COB-ID 冲突。在 CANopen 主站配置界面点击“检查与修正配置”，再点击“对所有都应用此建议”，可以解决重复的节点 ID 或者冲突的 COB-ID。
3	使能心跳产生	启用使能心跳产生时，主站将发送心跳信息
4	节点 ID	发送心跳信息的唯一标识符，默认为主站节点 ID，可设置范围为：1~127
5	Produter Time	心跳信息发送的时间间隔，单位为毫秒，范围可设置为 2~32767ms，并且与总线任务时间呈现整数倍关系
6	启动同步生成	启动同步生产时，主站将发送同步信息，一个 CAN 总线系统只能由一个站启用同步生产。同步类型 PDO 在同步信息发送后根据设置类型发送信息
7	COB-ID	通讯对象标识
8	循环周期	同步信息循环周期定义的时间间隔发送，同步周期的单位为微秒，范围为 2000~4294967000us，并且是总线任务时间的整数倍。
9	窗口长度	用于同步 PDO，以 ms 为单位的时间窗长度。
10	启动时间生成	启用时间生成时，主站将按照设定的时间发送信息
11	COB-ID	通讯对象标识
12	生产时间	主站以设置的时间将发送信息，并且是总线任务时间的整数倍，可设置的范围 2000~4294967000us。

3） CAN 从站参数设置

用鼠标在项目树中点击 CAN 总线电机“LD2_CAN7030”，即 CAN 从站，将从站通讯参数设置界面打开，如图 5.33 所示。

图 5.33　CAN 从站通讯参数设置界面

图 5.27 中注释序号的参数说明如表 5.2 所示。

表 5.2 从站参数说明

编号	选项名称	参数说明
1	节点 ID	从站在 CANopen 网络中唯一的标识号范围 1~127，需要与驱动器设置的硬件拨码一致。
2	使能专家设置	激活专家模式后，用户可以配置参数，如从站节点保护、心跳生产、紧急情况。
3	使能同步发生器	激活功能使能同步发生器时，从站将发送同步信息，一个 CANopen 总线系统只能有一个启用同步生产。同步发送参数使用主站的同步配置参数。
4	可选设备	暂不支持
5	未初始化	暂不支持
6	复位节点	暂不支持
7	使能节点保护	激活节点保护功能，节点保护和心跳生产是互斥的。主站在保护时间内定时发送节点保护，如果从站没有在节点守护时间（保护时间 x 生命周期因子）内给出包含特定防护 COB-ID（通信对象标识）的响应，则从站认为掉线状态。
8	保护时间	主站定时发送节点保护帧间隔，范围为 10~65535ms，并且为总线任务周期的整数倍。
9	生命周期因子	和保护时间共同使用，如果在节点守护时间（保护时间 x 生命周期因子）内，从站没有响 应，主站认为从站丢失。范围为 1~255。
10	使能心跳产生	激活从站心跳产生，从站以生产时间间隔定时发送心跳生产帧，和节点保护互斥。
11	发送端时间	主站定时发送节点保护帧间隔，范围为 10~65535ms，并且为总线任务周期的整数倍。
12	心跳消费	打开一个对话框，设置从站消费的心跳生产者。通过设置心跳消费，此从站可以检查 对应的心跳生产从站在线状态。一般从站消费主站的心跳生产。
13	使能紧急情况	激活使能紧急情况后，从站将通过紧急报文 COB-ID 发送紧急消 息。这些紧急信息可以通过 CiA405 library (RECV_EMCY_DEF, RECV_EMCY) 函数库提供的函数获取紧急消息。
14	激活时间创建	暂不支持
15	激活时间消耗	暂不支持
16	检查供应商	激活后，从站将检查对象字典中 ID（索引 1018，子索引 01）和从站本身的供应商 ID 是否匹配，如果不匹配，从站将不能正常运行。
17	检查产品号	激活后，从站将检查对象字典中产品好（索引 1018，子索引 02）和从站本身的产品号是否匹配，如果不匹配，从站将不能正常运行。
18	检查修订号	激活后，从站将检查对象字典中版本（索引 1018，子索引 03）和从站本身的版本是否匹配，如果不匹配，从站将不能正常运行。

4） 用 PDO 和 SDO 设置电机参数

CAN 总线通讯有两个重要参数需要设置，过程数据对象和服务数据对象。

过程数据对象（PDO）用于主站和从站之间的实时数据传输，接收 PDO 为主站向从站发送的实时数据。包含通信参数和映射参数。通信参数包括通信唯一标示 COB-ID、传输类型、传输控制等。如图 5.34 所示。

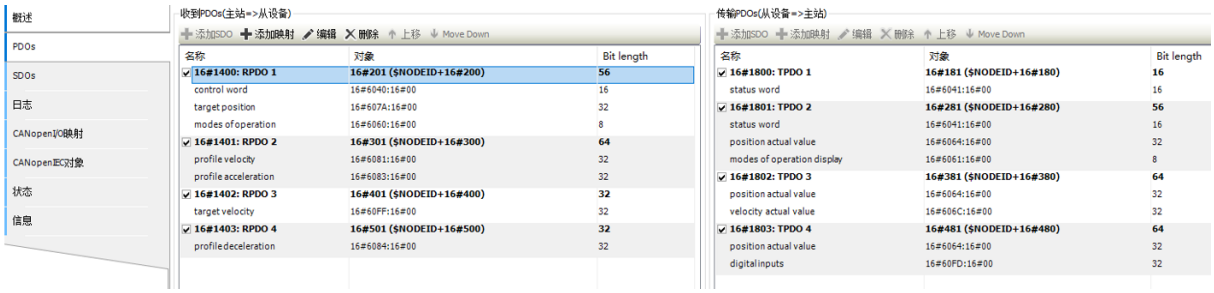


图 5.34　设置 PDO 参数、工作模式

服务数据对象（SDO）用来在设备之间传输低优先级数据且数据量较大的数据，典型的是用来配置 CANopen 网络上的设备，主要用于从站节点运行过程中，读取或者写入 SDO 对象值。读取或者写入一个 SDO 对象需要确定 SDO 对象的索引、子索引、位长度，写入时还需要写入的值。可以通过设置 SDO，设置驱动器脉冲当量，具体参考 10.2.3 小节数据服务对象。也可以设置其他启动参数，如驱动器工作模式。如图 5.35 所示。

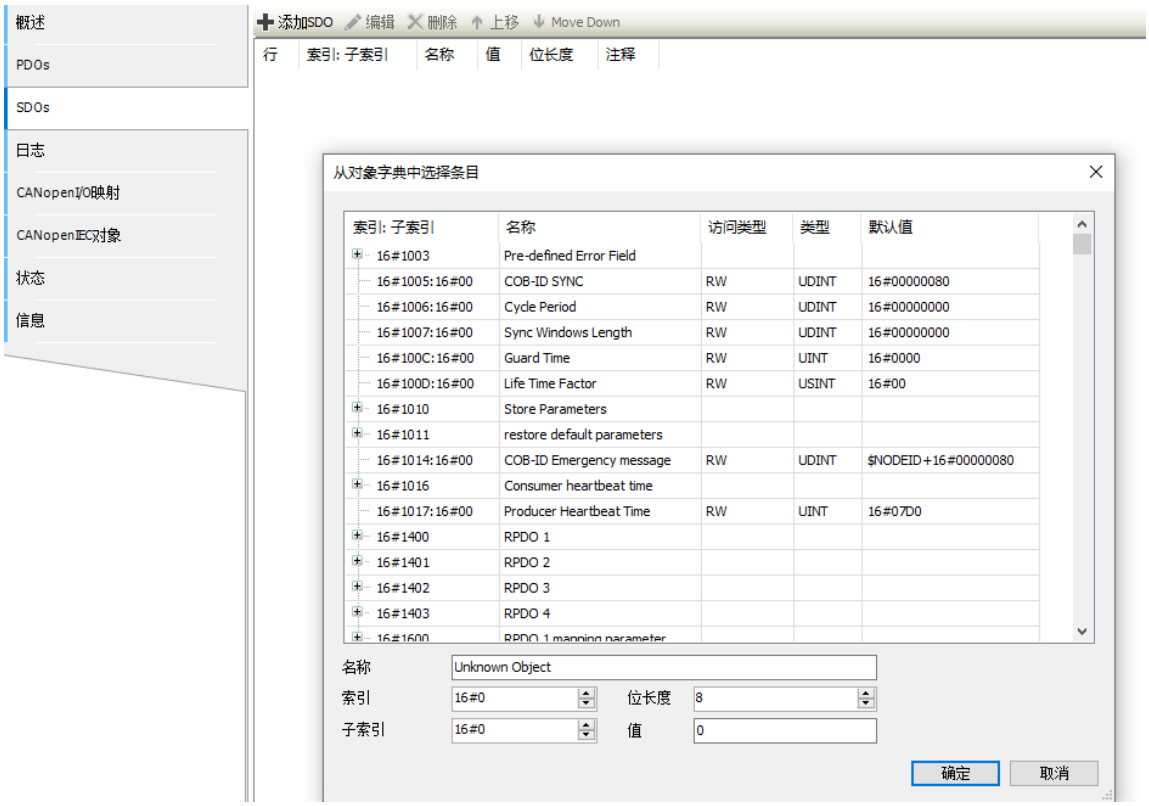


图 5.35　添加 SDO 参数

下面以 CAN 总线电机 LD2-CAN7030 为例，通过 MC300CS PLC 对电机进行写 PDO 和 SDO 操作。通过写 SDO 设置电机的脉冲当量；通过写入 PDO 将驱动器工作模式设置为

协议位置模式，协议位置设置为 10000、协议速度设置为 1000、协议加速度设置为 10000、协议加减速度为 1000000，启动电机运行。具体操作过程如下。

选择项目树中的“LD2-7030”，双击鼠标左键进入配置页。选择“PDO”选项卡进入页面。将 RPDO1-4 中多余的参数删除。如图 5.36 所示。

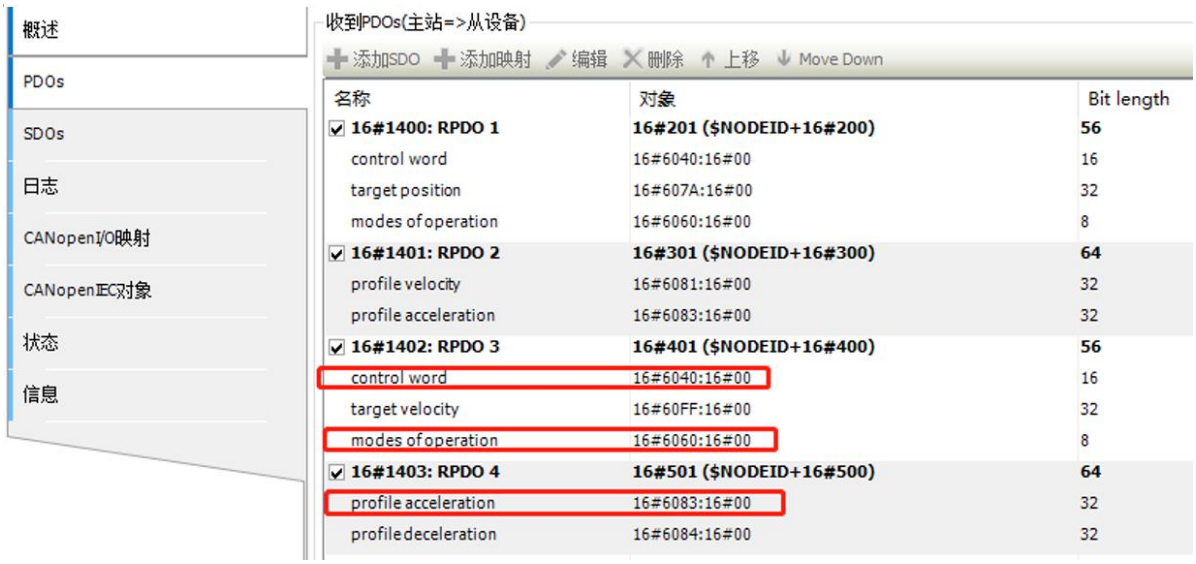


图 5.36　RPDO 通道参数配置

设置驱动器脉冲当量：

选中“SDO”选项卡。点击“添加 SDO”。在对象字典中找到“16#2008”索引，将其设置为 10000。如图 5.37 所示。

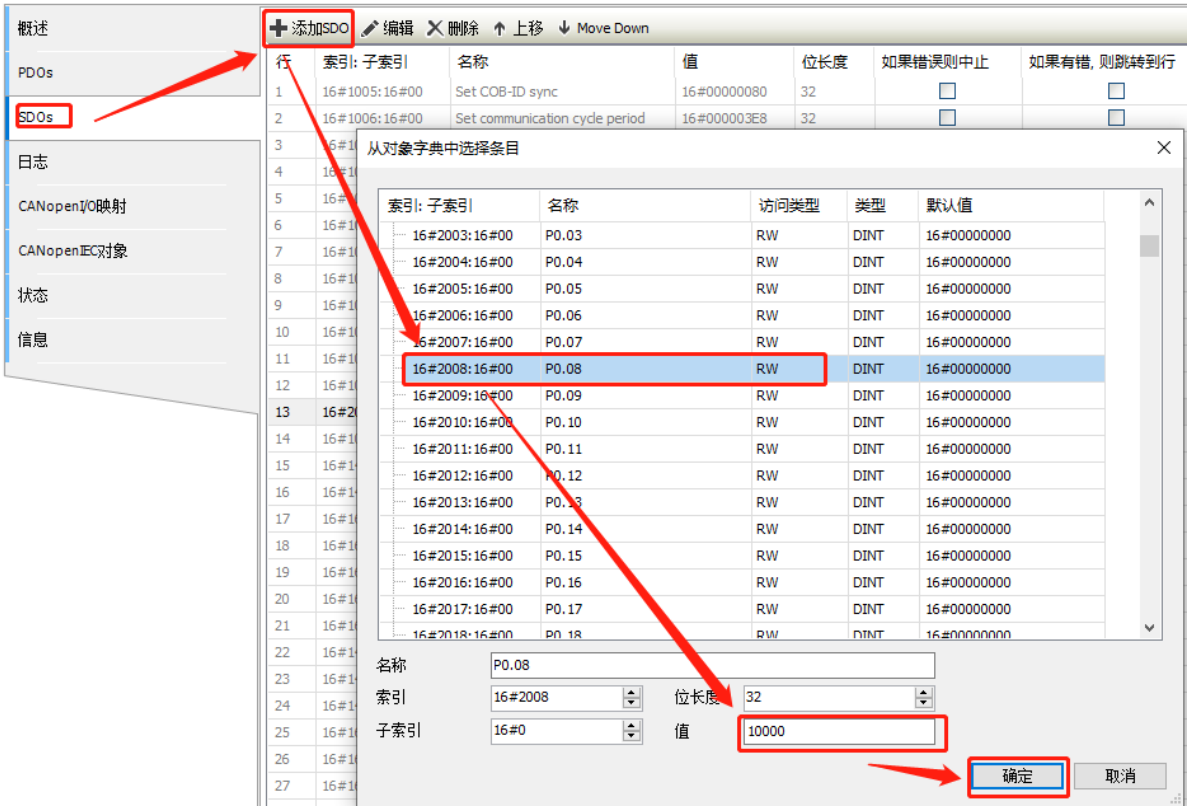


图 5.37　设置伺服驱动器脉冲当量

选中“CANopenIO 映射”选项卡，选择图中框选的“使能 1（如果未在任何任务中使用则使用总线任务）”。如图 5.38 所示。

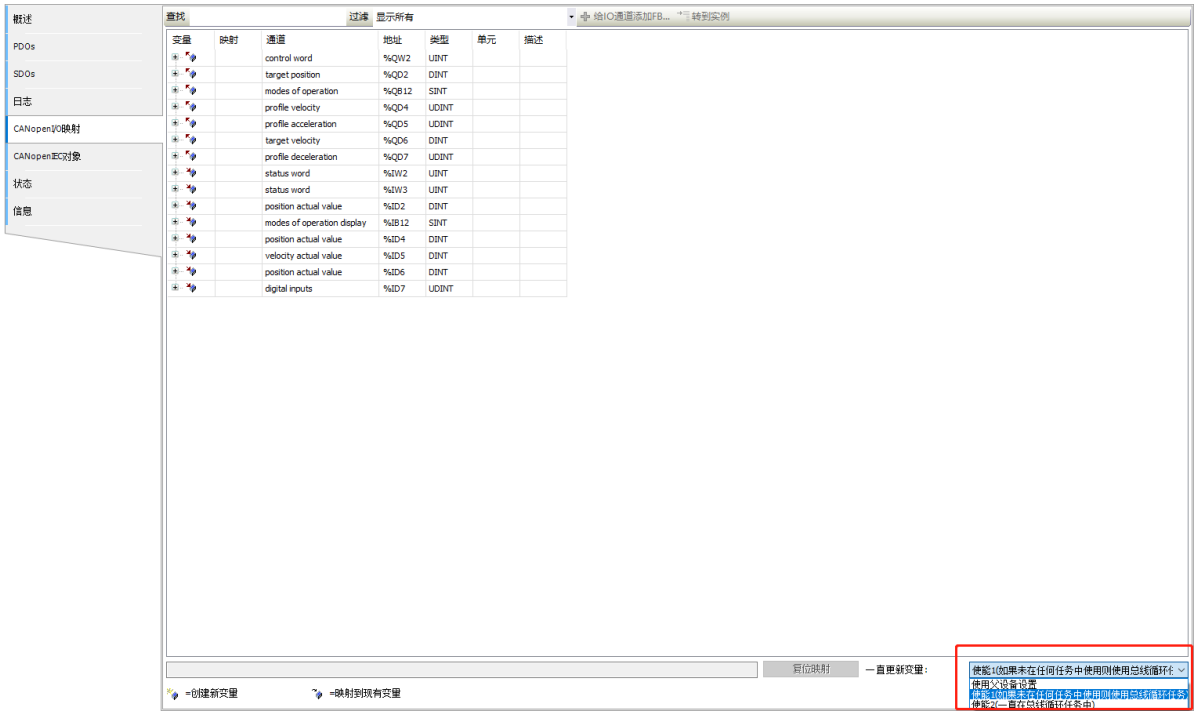


图 5.38　刷新 CANopenIO

点击  进行进入在线模式，将通道“modes of operation”设置为 1 后，设置通道“target position”设置为 10000，通道“profile velocity”设置为 1000，通道“profile accelerattion”设置为 10000，通道“profile deceleration”设置为后 10000。切换通道“control word”为 6→7→79→95。点击能够正常运行。如图 5.39 所示。

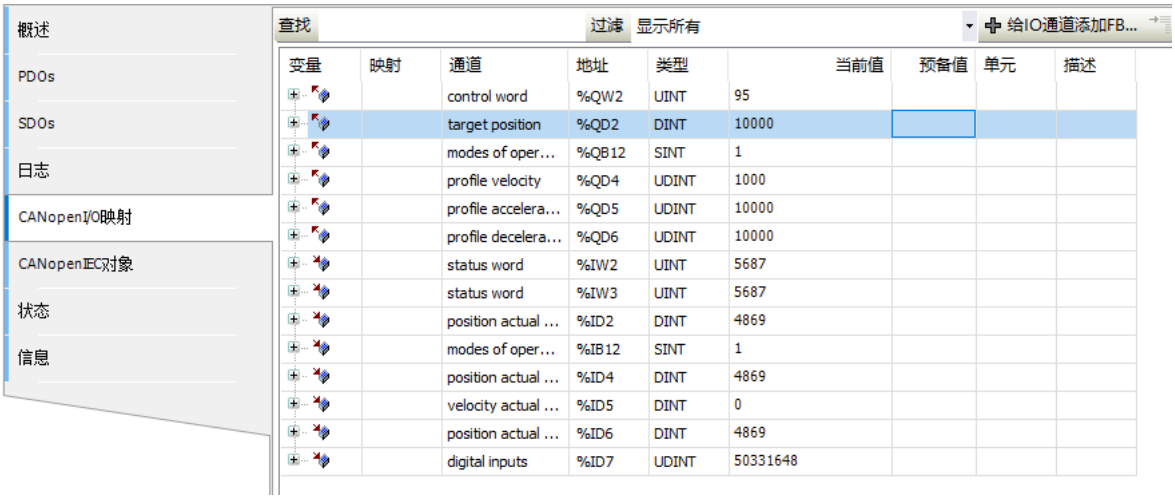


图 5.39　PDO 映射地址操作区

5.3.4 CAN 总线轴例程

本例使用 MC300CS 控制 LD2-7030 CAN 总线伺服驱动器及电机执行点位运动。

注意：CAN 总线轴不能设置脉冲当量，只能使用脉冲为位移单位。

1、搭建硬件系统

将 MC300CS PLC 与 LD2-70300 驱动器用 CAN 总线连接。

2、编写软件

先添加描述文件、库文件；再扫描添加从站设备。

设置 CANbus 网络号为 0，波特率为 125 kbits/s；从站节点号为 1。

软件设置的节点号需要与驱动器上的拨码号设置一致。驱动器上的波特率拨码设置也与软件参数一致。详见 CAN 总线电机及驱动器说明书。

删除多余的 PDO 参数，如图 5.40 所示。

分别双击“RPDO1”，“RPDO2”，“RPDO3”和“RPDO4”，在传输类型中选择“异步的-特定于设备配置文件 (类型 255)”。如图 5.40 所示。

分别双击“TPDO 1”和“TPDO 2”，在传输类型中选择“异步的-特定于设备配置文件 (类型 255)”，设置抑制时间为 2000us；分别双击“TPDO 3”和“TPDO 4”，在传输类型中选择“异步的-特定于设备配置文件 (类型 255)”，设置抑制时间为 10000us，如图 5.41 所示。

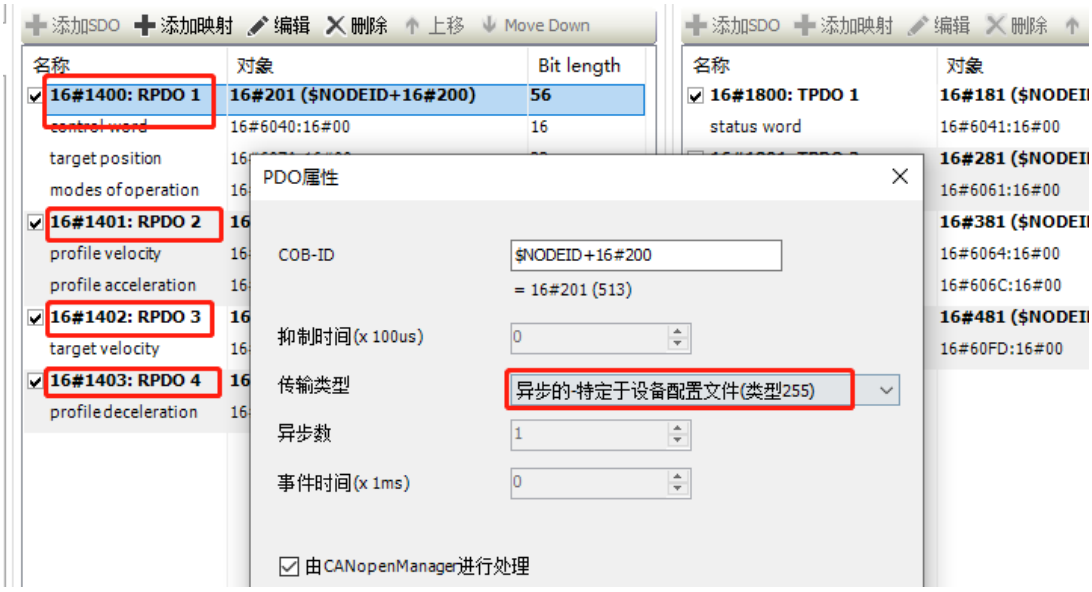


图 5.40　配置 RPDO 通道

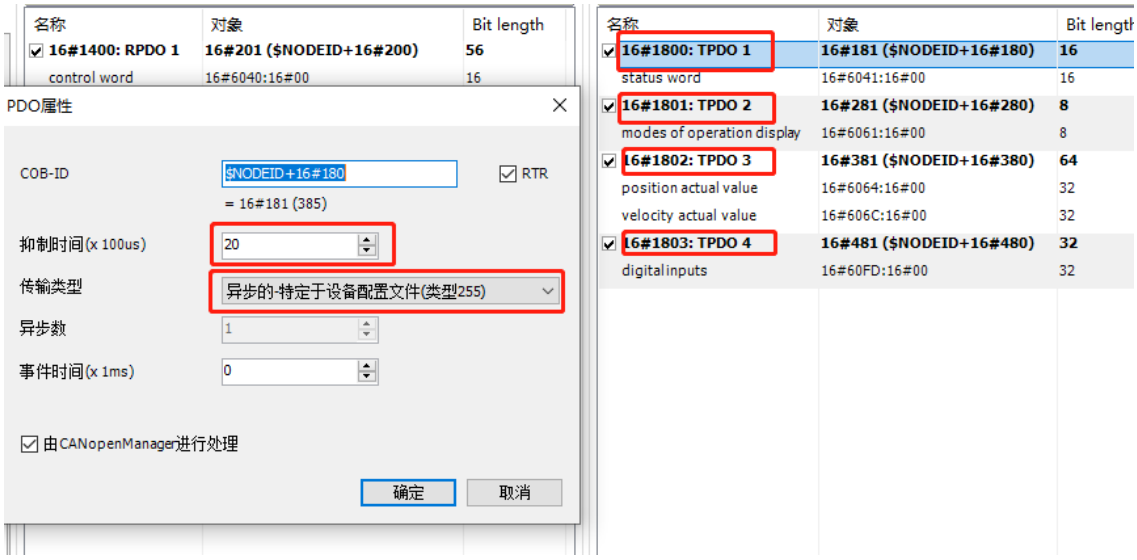


图 5.41　配置 TPDO 通道

选择“使能 1（如果未在任何任务中则使用总线循环任务）”，如图 5.33 所示。

CAN 总线轴与脉冲轴一样也需要设置轴号结构体。在 CAN 总线轴中的“MC_AxisConfi_CAN”需要用到结构体“Str_LTCANopenAxis”，可参照本例程新建结构体。

先使用“MC_AxisConfi_CAN”指令设置 CAN 总线轴参数，使用“MC_Power_CAN”指令进行使能 CAN 总线轴，然后才能调用“MC_MoveRelativeIme_CAN”指令进行点位运动控制。指令详细说明请参考 6.4 小节 CANopen 总线型电机的运动控制指令。

程序代码如下。

```
PROGRAM PLC_PRG
VAR
    MC_AxisConfi_CAN_1:MC_AxisConfi_CAN;    //轴变量绑定
    LD2_CAN7030_axis_1: CANopenLib.Str_LTCANopenAxis;//实例化 CAN 总线轴参数配置
    ControlWord AT%QW2:UINT;//绑定控制字地址
    StatusWord AT%IW2:UINT;//绑定状态字地址
    MC_Power_CAN_1:MC_Power_CAN;//实例化 CAN 总线使能指令
    MC_MoveRelativeIme_CAN_1:MC_MoveRelativeIme_CAN;//实例化 CAN 总线轴相对运动指令
    Button_Pmove AT %IX0.0: BOOL;//绑定启动按钮
END_VAR

MC_AxisConfi_CAN_1(Axis:= LD2_CAN7030_axis_1, Enable:= TRUE, NetWork:= 0, NodeID:= 2,
                   ControlWord_Address:= ADR(ControlWord),
                   StatusWord_Address:= ADR(StatusWord), Done=> , StateMachine=> ,
                   StatusWord=> , Error=> , ErrorID=> ); // 设置 CAN 总线参数
MC_Power_CAN_1(Axis:= LD2_CAN7030_axis_1, Enable:= TRUE, Done=> , Busy=> , Error=> ,
                ErrorID=> );                              // 使能 CAN 总线轴
MC_MoveRelativeIme_CAN_1(Axis:= LD2_CAN7030_axis_1, Execute:= Button_Pmove,
                         Distance:= 10000, Velocity:= 10000, Acceleration:=100000 ,
                         Deceleration:=100000, Done=> , Busy=> , CommandAborted=> ,
                         Error=> , ErrorID=> );          // 点位运动
```

第6章 单轴运动控制

单轴控制是指通过位置控制、速度控制、转矩控制这 3 种模式控制轴运动。在单轴控制中，脉冲型、EtherCAT 总线型电机的常用控制指令如表 6.1 所示，指令的详细说明请参考指令手册。

表 6.1 脉冲型、EtherCAT 总线型电机常用的单轴控制指令

单轴控制指令	指令名称	说明
MC_Power	轴使能	将驱动器切换为可运行状态
MC_SetPosition	设置轴位置	修改当前轴位置寄存器的位置。该操作电机不会移动，仅产生坐标偏移
MC_ReadStatus	读取轴的状态	读取轴的状态数据
MC_Jog	点动	以指定速度运动，常用于点动控制
SMC_Inch	寸动	手动控制轴一段一段的朝指定方向运动
MC_MoveRelative	相对点位运动	以当前位置为起点，移动指定距离
MC_MoveAbsolute	绝对点位运动	以绝对坐标控制目标位置
MC_MoveAdditive	叠加相对点位运动	在当前运行指令的基础上，再做相对点位运动
MC_MoveVelocity	速度控制	使用伺服驱动器的位置模式，控制伺服电机以指定速度运行
LS_Home_P	脉冲型电机回零	控制脉冲型电机回原点
MC_Home	总线型电机回零	控制总线型电机回原点
MC_Halt	暂停	命令电机暂停运行，可再次调用 MC_Movexxx 指令触发运行
MC_Stop	停止	可命令电机以指定减速度减速后停止
MC_Reset	轴报警复位	当伺服电机出现报警停止后，可运行该指令复位。如轴状态 Axis.nAxisState 为 errorstop 需要切换到 StandStill 要用该指令复位
SMC3_ReinitDrive	重启驱动器	在总线同步帧丢失后、轴配置参数修改后需要使用该指令重新初始化轴
SMC_SetControllerMode	切换轴运行模式	使用该指令可以让伺服驱动器选择位置、速度或力矩模式
SMC_SetTorque	力矩控制	命令伺服电机以指定的力矩运行

CANopen 总线型电机的控制指令比较特殊，将在 6.4 节中单独介绍。

6.1. 位置控制：点位运动

点位运动是指：运动控制器控制运动平台从当前位置开始按照加速度曲线、速度曲线设定的规律运动，并在指定位置准确地停止。

加速度 a、速度 v、位移 s 之间的关系如下：

$$v = \int a\ dt$$
$$s = \int v\ dt \qquad (6.1)$$

或：

$$a = dv/dt$$
$$v = ds/dt \qquad (6.2)$$

MC300CS 提供了 4 种速度曲线模式：梯形速度曲线、SIN^2 速度曲线、二次速度曲线、二次（平滑）速度曲线。速度曲线类型的选择在轴参数中设定。如图 6.1 所示。

图 6.1 单轴运动速度模式的选择

梯形速度曲线对应的加速度曲线有 4 处会产生突变，故运动平稳性较差，但计算点位运动时间简单。

其他三种速度曲线非常平滑，运动平稳性较好，但计算运动时间较复杂。

下面以点位运动指令为例，详细分析四种速度曲线的特点。

相对点位运动指令的格式如下。

```
MC_MoveRelative(Axis:= 轴号, Execute:= 启动信号, Distance:= 运动距离,
                Velocity:= 最大速度, Acceleration:= 加速度, Deceleration:= 减速度,
                Jerk:= 加加速度, BufferMode:= 缓存模式,
                Done=> 运动结束, Busy=> 运动中, Active=> 已控制该轴,
                CommandAborted=> 运动被终止, Error=> 指令出错, ErrorID=> 错误代码 );
```

6.1.1 梯形速度曲线及相关参数分析

当速度模式为“梯形”时，执行一条相对点位运动指令如下。

```
PmoveX(Axis:=X, Execute:=startX, Distance:=10, Velocity:=20,
       Acceleration:=50, Deceleration:=100, Jerk:= , BufferMode:= ,
       Done=>doneX, Busy=> , Active=> , CommandAborted=> , Error=> , ErrorID=> );
```

即：PmoveX 是 MC_MoveRelative 函数实例化的名称。运动距离 s 为 10mm，最大速度 V_{max} 为 20mm/s，加速度 a1 为 50mm/s^2，减速度 a2 为 100mm/s^2。位移、速度、加速度曲线如图 6.2 所示。

从图中可看出：运动过程分为三段：加速段、匀速段、减速段。加速是为了让电机平稳地过渡到最大速度，减速是为了让电机能准确地在指定位置停住。

速度上升段：加速度 a1 大于零，为常数；速度线性增加，直至速度等于最大速度 V_{max}；位移是二次曲线。

匀速段：加速度为零；速度等于最大速度 V_{max}；位移线性增加。

速度下降段：加速度 a2 小于零，为常数；速度线性减小，直至速度为零；位移是二次曲线。

加速度曲线在运动开始和结束处、匀速运动开始和结束处，加速度值有突变。

由 F = ma 知，在这 4 个点上，作用在运动平台上的推力有突变，即有冲击力。故以梯形速度曲线运动，平稳性较差。

由式（6.2）知：

速度上升段时间为：dt1 = dv/a1= (V_{max}-0)/a1 = (20-0)/50 = 0.4 s

速度下降段时间为：dt2 = dv/a2= (0-V_{max})/a2 = (0-20)/(-100) = 0.2 s

根据式（6.1）可计算各段位移量。

加速段对应的位移：

s1 = V_{max}×dt1/2 = 20×0.4/2 = 4 mm （速度曲线所围的面积）

减速段对应的位移：

s2 = V_{max}×dt2/2 = 20×0.2/2 = 2 mm

匀速段位移：s3 = s-s1-s2

匀速段时间：dt3 = s3/V_{max} = (10-4-2)/20 = 0.2 s

点位运动的总时间：t = dt1 + dt2 + dt3 = 0.4+0.2+0.2 = 0.8 s

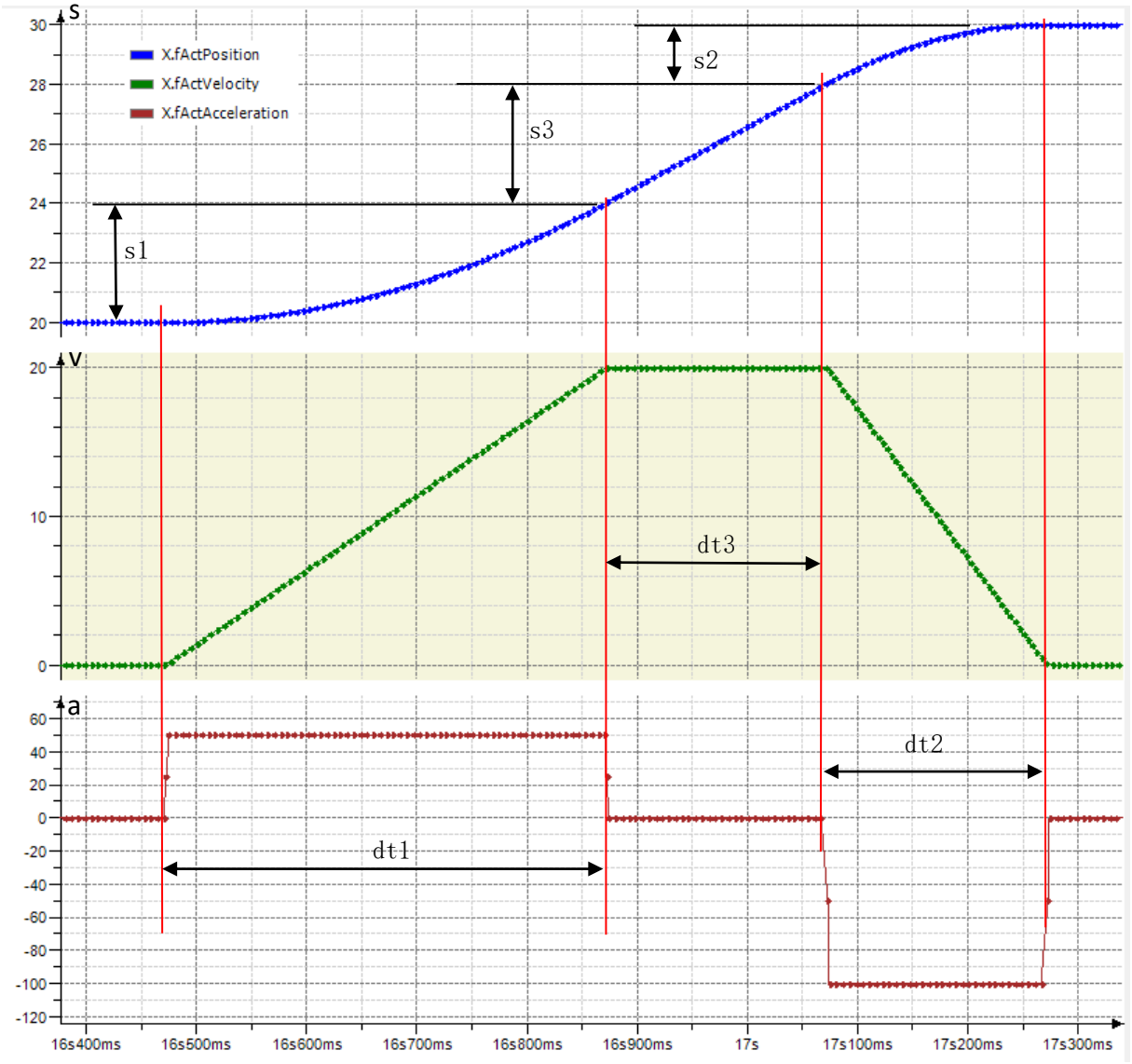


图 6.2　梯形速度曲线及对应的加速度曲线、位移曲线

6.1.2 SIN^2 速度曲线及相关参数分析

当速度模式为“sin^2”时，执行的相对点位运动指令与“梯形”模式时相同，其位移、速度、加速度曲线如图 6.3 所示。

显然，速度上升段的速度曲线的函数为：$v = V_{max} \times \sin^2(\omega t)$

当速度 $v = V_{max}$ 时，$\omega t = \pi/2$　　（6.3）

加速度曲线的函数为：$a = dv/dt = V_{max} \times \omega \times \sin(2\omega t) = a1 \times \sin(2\omega t)$

即：　$\omega = a_{max}/V_{max}$　　（6.4）

将（6.4）式代入（6.3）式，得：

$t1 = \pi/(2a1/V_{max}) = 3.14159/(2\times50/20) = 0.628$ s

同理，速度下降段的时间 $t2 = \pi/(2a2/V_{max}) = 3.14159/(2\times100/20) = 0.314$ s

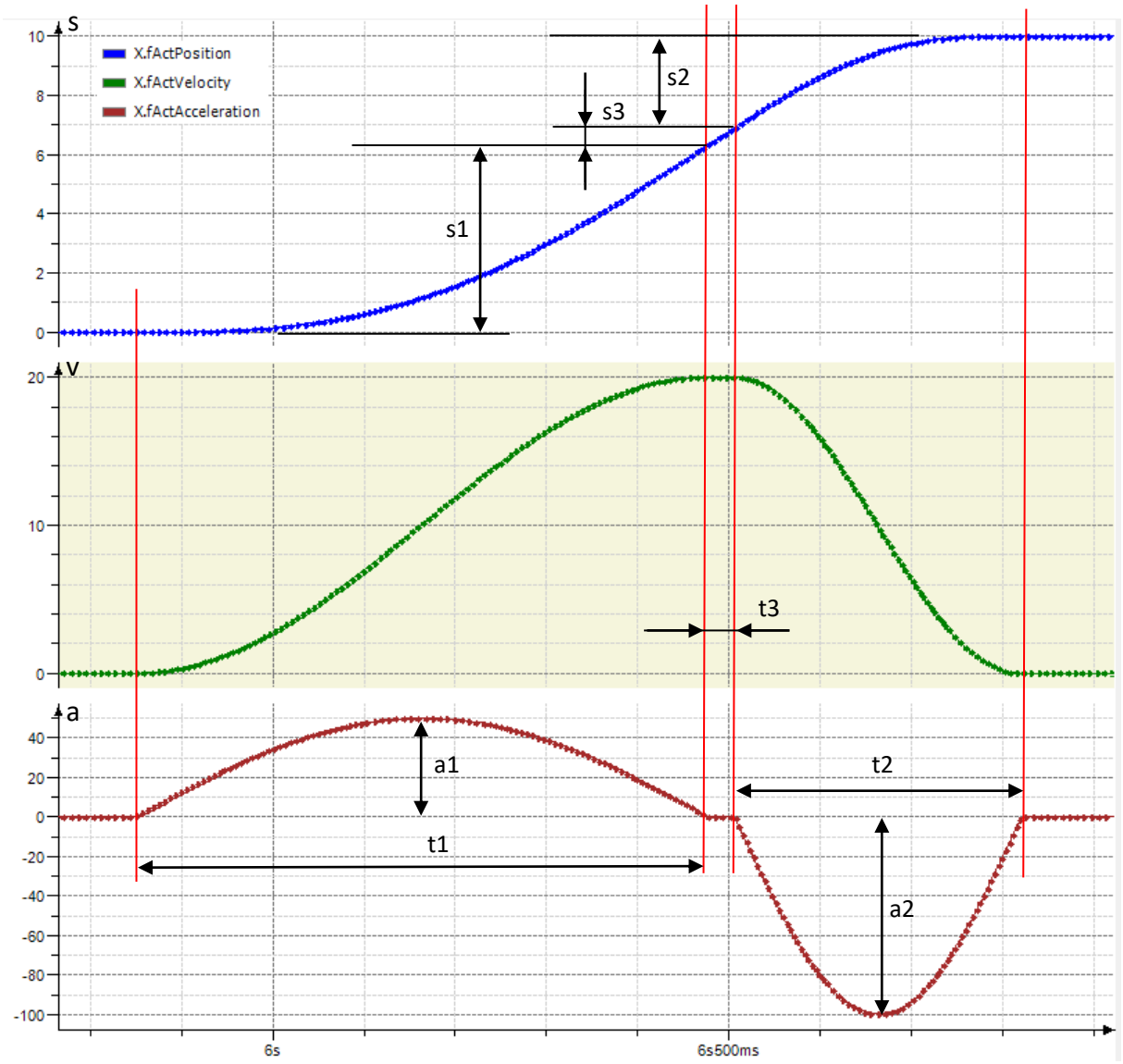


图 6.3　sin^2 速度曲线及对应的加速度曲线、位移曲线

速度上升段的位移 $s1 = \int V_{max} \times \sin^2(\omega t)\,dt$

$= V_{max}(t/2 - \sin(2\omega t)/(4\omega)) = 20(0.628/2 - 0) = 6.28$ mm

同理，速度下降段的位移 s2 = 20(0.314/2 - 0) = 3.14 mm

匀速段的时间 t3 = s3/V_{max} = (s-s1-s2)/V_{max}

= (10-6.28-3.14)/20 = 0.03 s = 30 ms

点位运动的总时间：

t = t1 + t2 + t3 = 0.628 + 0.314 + 0.03 = 0.972 s = 972 ms

6.1.3 二次速度曲线及相关参数分析

当速度模式为“二次”时，执行的相对点位运动指令如下。

```
PmoveX(Axis:=X, Execute:=startX, Distance:=5, Velocity:=20,
       Acceleration:=50, Deceleration:=100, Jerk:=1000, BufferMode:= ,
       Done=>doneX, Busy=> , Active=> , CommandAborted=> , Error=> , ErrorID=> );
```

注意：使用“二次”速度模式时，一定要给定加加速度 J。

指令中位移、速度、加速度的参数与“梯形”速度模式相同，加加速度 J 为 1000mm/s^3。位移、速度、加速度的曲线如图 6.4 所示。

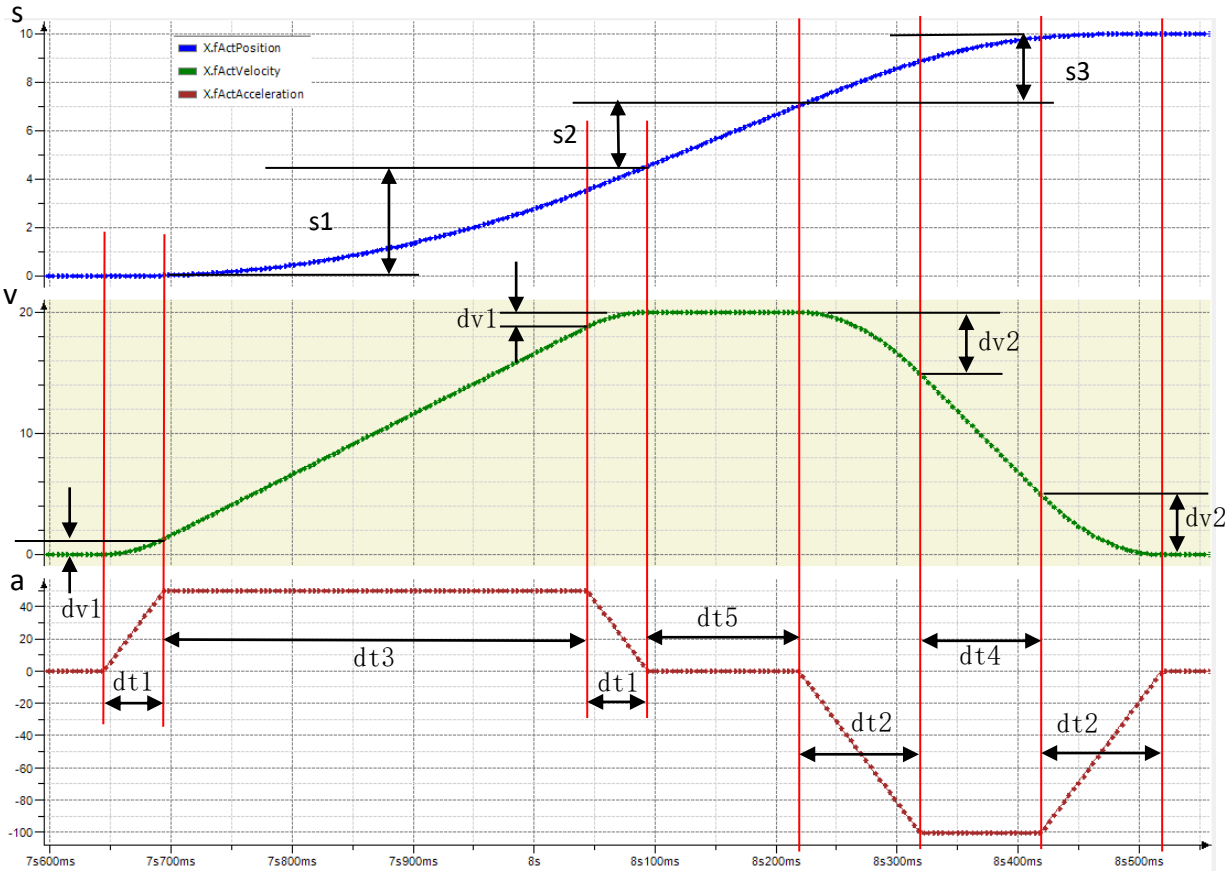


图 6.4　二次速度曲线及对应的加速度曲线、位移曲线

从图中可看出，与“梯形”速度模式相比，“二次”速度模式的加速度曲线中，不管加速度是正还是负，都有加速度上升段、加速度恒定段和加速度下降段。

因为加加速度相同，故加速度的上升段和下降段斜率相同，上升段和下降段是对称

的。

因为，加加速度 J = da/dt

所以，正加速度的上升段和减速段的时间均为：

dt1 = da1/J = (50-0)/1000 = 0.05 s = 50 ms

同理，负加速度的上升段和减速段的时间均为：

dt2 = da2/J = (100-0)/1000 = 0.1 s = 100 ms

因为，$v = \int a\ dt = \int (J \times t) dt$

所以，正加速度上升段对应的速度曲线为二次曲线，该段终点的速度为：

$dv1 = J \times dt1^2/2 = 1000 \times 0.05^2/2 = 1.25$ mm/s

同理，加速度由零变为 a2 时，对应的速度变化量为：

$dv2 = J \times dt2^2/2 = 1000 \times 0.1^2/2 = 5$ mm/s

正加速度恒定段对应的速度曲线为一次曲线，该段时间为：

$dt3 = (v_{max} - dv1 - dv1)/a1 = (20-1.25-1.25)/50 = 0.35$ s

同理，负加速度恒定段的时间时，

$dt4 = (v_{max} - dv2 - dv2)/a2 = (20-5-5)/100 = 0.1$ s

速度上升段对应的位移量：s1 = ds1 + ds2 + ds3

其中：ds1、ds2、ds3 是加速度上升段、恒定段和下降段对应的位移量。

$ds1 = \int_0^{dt1} (J \times t^2/2) dt = J \times dt1^3/6$

$ds2 = \int v\ dt = (dv1 + v_{max} - dv1) \times dt3/2 = v_{max} \times dt3/2$

$ds3 = (v_{max} - dv1) dt1 + \int_0^{dt1} (a1 \times t - J \times t^2/2) dt$

$= (v_{max} - dv1) dt1 + a1 \times dt1^2/3$

故：$s1 = 1000 \times 0.05^3/6 + 20 \times (0.35/2) + (20-1.25) \times 0.05 + 50 \times 0.05^2/3 = 4.5$ mm

同理，速度下降段对应的位移量：

$s3 = J \times dt2^3/6 + v_{max} \times dt4/2 + (v_{max} - dv2) dt2 + a2 \times dt2^2/3$

$= 1000 \times 0.1^3/6 + 20 \times 0.1/2 + (20-5) \times 0.1 + 100 \times 0.1^2/3 = 3$ mm

匀速段的位移：s2 = s - s1 - s3 = 10 - 4.5 - 3 = 2.5 mm

匀速段的时间：$dt5 = s2/v_{max} = 2.5/20 = 0.125$ s

点位运动的总时间：t = dt1 + dt3 + dt1 + dt5 + dt2 + dt4 + dt2

= 0.05 + 0.35 + 0.05 + 0.125 + 0.1 + 0.1 + 0.1

= 0.875 s = 875 ms

注意：当运动距离太短时，最大速度和最大加速度都有可能小于设定值。

例如：当上例的速度、加速度不变，运动距离改为 1mm 时，最大速度和最大减速度都未达到设定值，如图 6.5 所示。

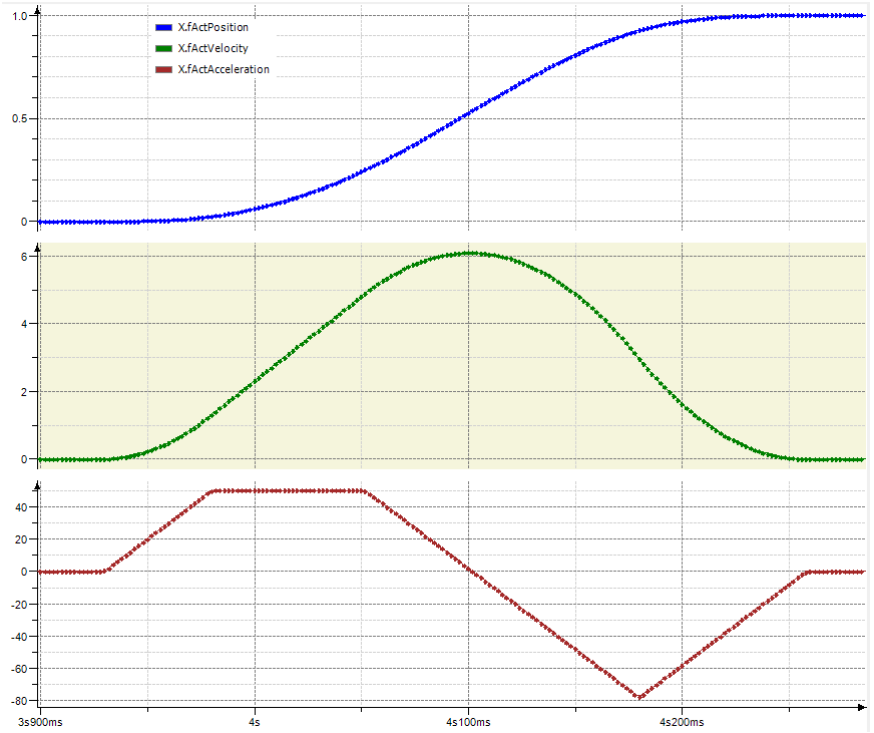


图 6.5　运动距离很小时，最大速度和最大加速度小于设定值

6.1.4 二次(平滑)速度曲线

“二次（平滑）”速度模式下的速度曲线及对应的加速度曲线、位移曲线如图 6.6 所示。与“二次”速度模式相比，其加速度曲线作了平滑处理，点位运动时间略长一点。可以用“二次”速度模式的计算公式估算“二次（平滑）”速度模式下的运动时间。

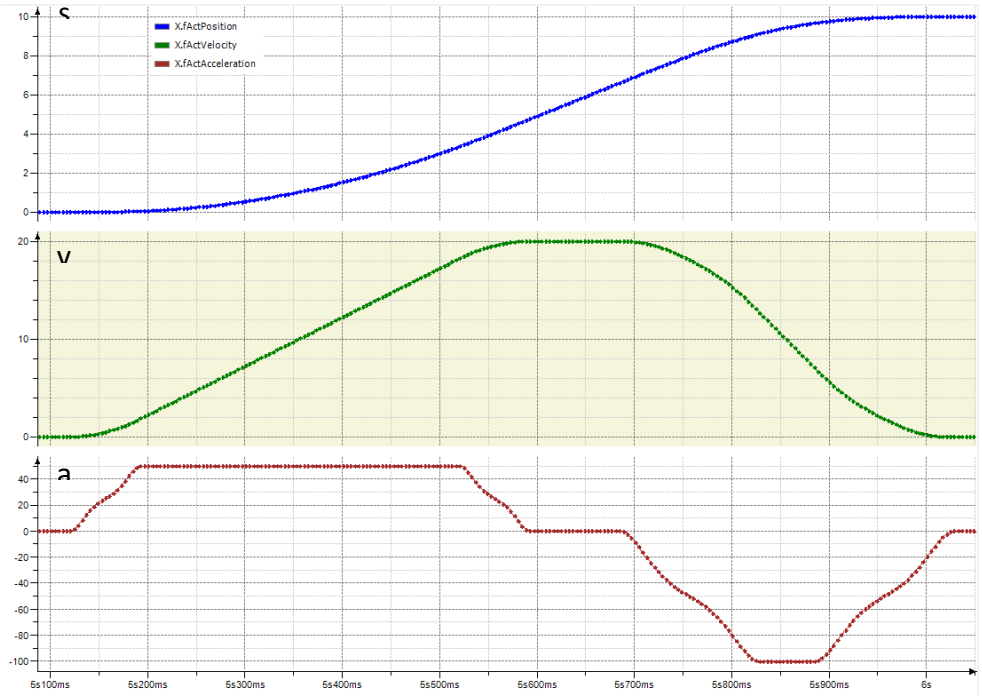


图 6.6　二次（平滑）速度曲线及对应的加速度曲线、位移曲线

6.1.5 多轴点位运动

多轴同时做点位运动为多轴联动。由于软件处理速度远比机械系统响应速度快，虽然多条点位运动指令是一条接一条地发出，各轴的启动时间有微秒级的差距，但对机械系统而言，可认为是同时启动。

点位运动只关注终点坐标，对运动轨迹的精度没有控制。如果每个轴的运动速度相同，则多轴运动的轨迹就可能是折线，如图 6.7 所示。

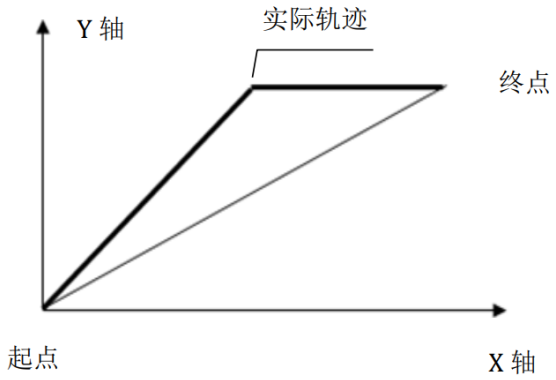


图 6.7 两轴复合点位运动的轨迹

6.2. 速度控制：Jog 运动

Jog（慢跑）运动指令是一条速度控制指令，常用于手动控制平台的运动：当“向前”按键按下，运动平台向前运动，按键抬起，平台运动停止；当“向后”按键按下，平台向后运动，按键抬起，平台停止。

Jog 运动指令的格式如下。

```
MC_Jog( Axis：=轴号， JogForward：=向前启动信号， JogBackward：=向后启动信号，
        Velocity：=最大速度， Acceleration：=加速度， Deceleration：=减速度，
        Jerk：=加加速度， Busy=>运动中， CommandAborted=>运动被终止，
        Error=>指令出错，ErrorID=>错误代码);
```

注意：JogForward 和 JogBackward 信号同时为 TRUE 时，电机不会运动。

例如：X 轴以 $\sin^2$ 速度曲线运行 Jog 指令：

```
JogX( Axis:=X, JogForward:=in1, JogBackward:=in2, Velocity:=5,
      Acceleration:=30, Deceleration:=30, Jerk:=500,
     Busy=> , CommandAborted=> , Error=> , ErrorId=> );
```

Jog 指令的时序图如图 6.8 所示。

从图中可看出：当前进信号 in1 由 FALSE 转为 TRUE 后，X 轴在位置为 0 的地方开始按 $\sin^2$ 速度曲线加速运动，当速度升至最大速度 5mm/s 后开始匀速运动，当 in1 由 TRUE 转为 FALSE 时，X 轴开始减速，直到停止。X 轴向前运动了 9.5mm 左右。

当后退信号 in2 由 FALSE 转为 TRUE 后，X 轴按 $\sin^2$ 速度曲线负方向加速，当速度升至-5mm/s 后开始匀速运动，当 in2 由 TRUE 转为 FALSE 时，X 轴开始减速，直到停止。

这时 X 轴后退至约 2.3mm 的位置。

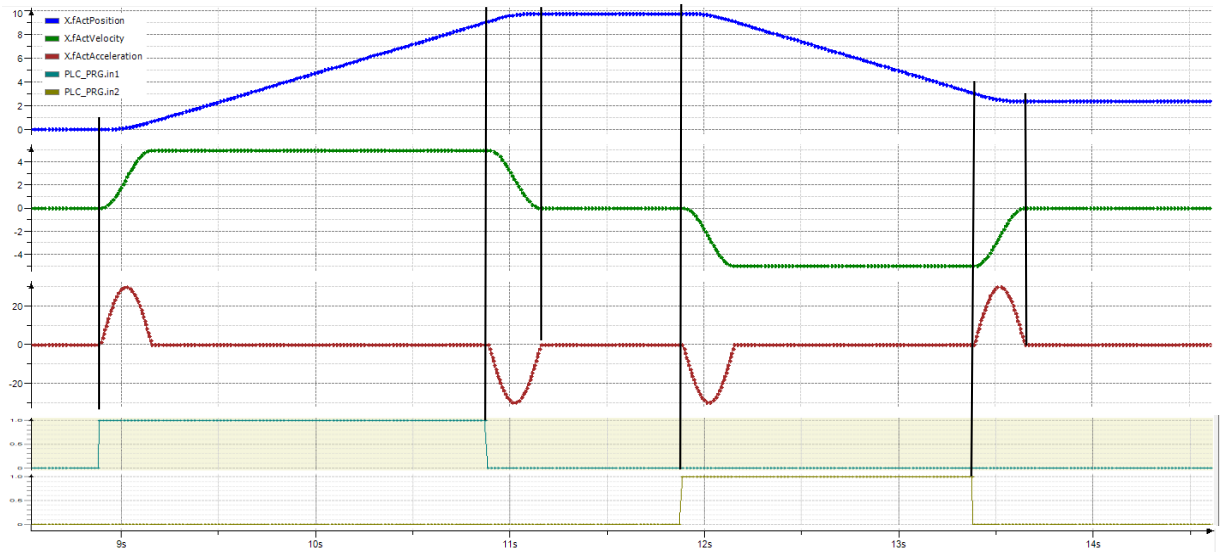


图 6.8　Jog 运动的时序图

6.3.　回原点运动

在运动平台上，每个轴都有一个位置传感器用于设置一个位置参考点，即原点位置，以便进行位置控制。在正常运动之前，都需要用回零指令控制平台向原点方向运动，当 PLC 检测到原点信号 ORG 后，平台自动停止，并将停止位置作为该轴的原点。参见图 6.9。

一般习惯将原点开关安装在电机一侧，且将运动平台向电机方向运动设为负方向。

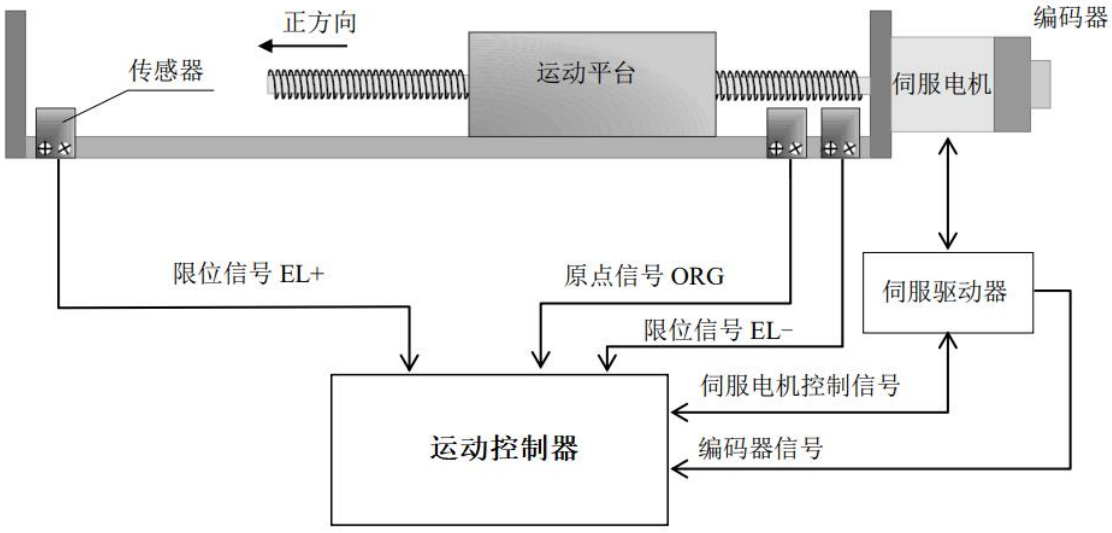


图 6.9 运动平台传感器信号及电机控制信号与运动控制器的关系

6.3.1 脉冲型电机回零指令

MC300CS 系列 PLC 本地的脉冲轴回零指令的格式如下。

```
LS_Home_P( Axis:= 轴号, xExecute:= 启动信号, xDone=> 运动结束,
           xBusy=> 运动中, xCommandAborted=> 运动被终止,
           xError=> 指令出错, nErrorID=> 错误代码 );
```

回零指令中没有设置回零模式、回零方向、回零速度、回零加速度、回零减速度等参数，这些参数在电机参数设置栏中设置，如图 6.10 所示。

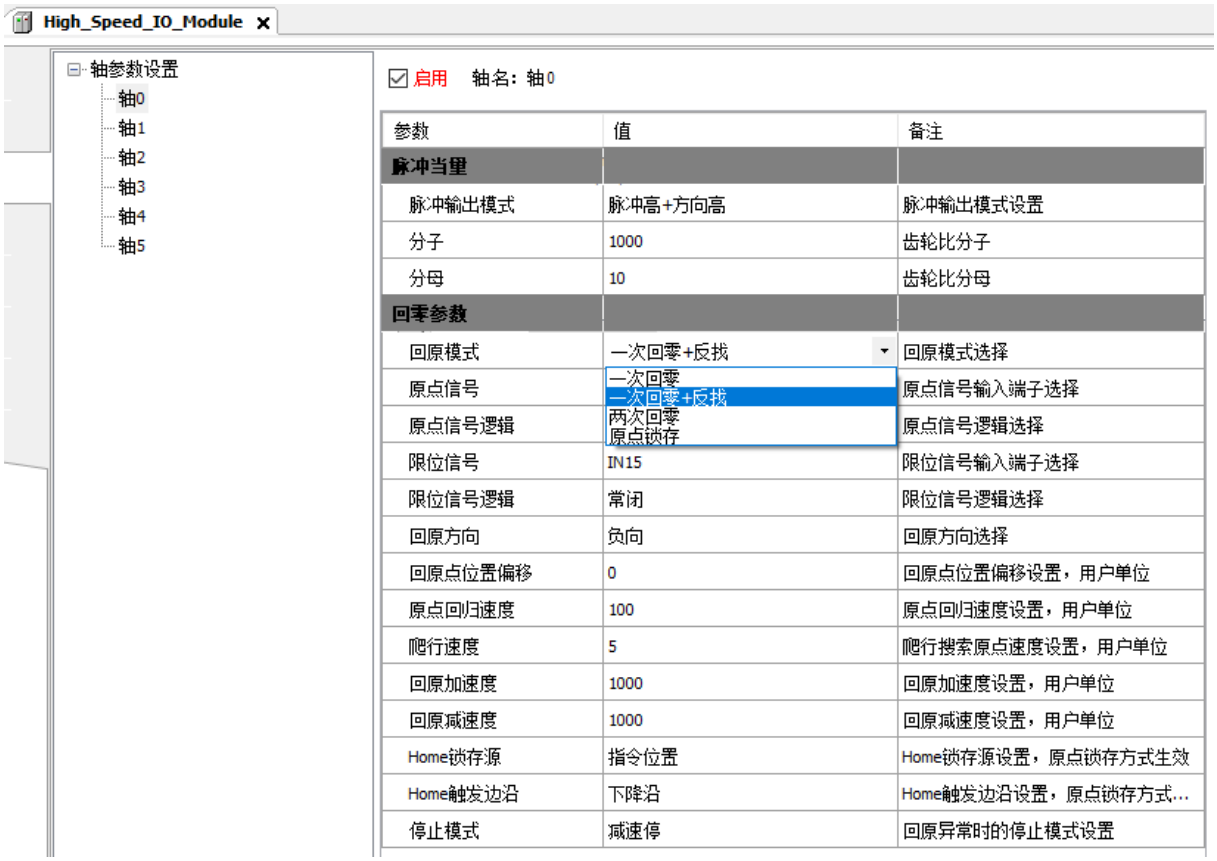


图 6.10 脉冲型电机回零参数的设置界面

MC300CS 主机上的 6 个脉冲轴的回零模式有四种：一次回零、一次回零加反找、二次回零、原点锁存。其回零过程与特点如下。

1. “一次回零”模式

平台在远离原点开关位置开始回零运动，电机驱动平台以回零速度向负方向运动，当平台到达原点开关位置，原点信号被触发，电机减速停止；并将停止位置设为原点位置，如图 6.11 中的长轨迹所示。

若平台停在原点开关有效范围内开始回零运动，电机驱动平台以回零速度向负方向运动；当平台运动到负限位位置触发负限位开关，平台减速停止后开始以回零速度向正方向运动，直到离开原点信号有效范围；然后平台减速停止，接着以回零速度向负方向运动；平台再次触发限位开关信号，电机减速停止；并将停止位置设为原点位置，如图 6.11 中的短轨迹所示。

一次回零模式的定位精度与回零速度、回零减速度有关。速度越低，定位越准。一般用于行程短、定位精度要求不高的场合。

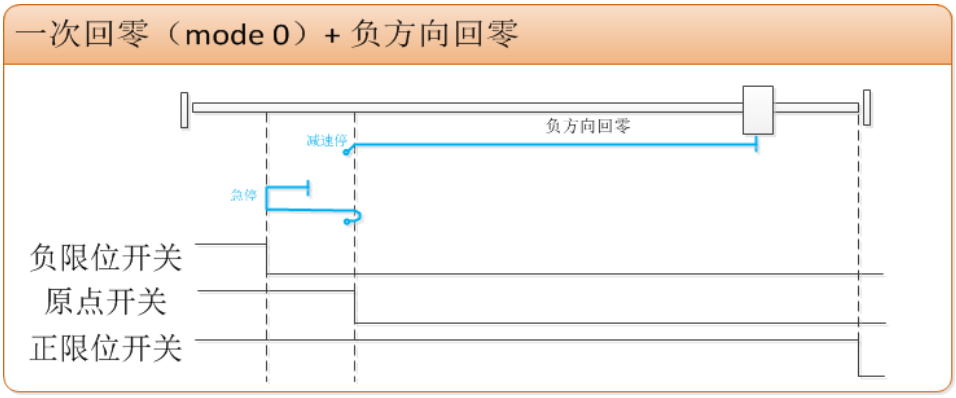


图 6.11 负方向“一次回零”的运动过程

2. “一次回零+反找”模式

平台在远离原点开关位置开始回零运动，电机驱动平台以回零速度向负方向运动，当平台到达原点开关位置，原点信号被触发，电机减速停止；这个过程即为一次回零。

然后平台以爬行速度向正方向运动，当平台离开原点信号有效范围后，平台减速停止并将停止位置设为原点位置，此过程为反找。

一次回零+反找的运动轨迹如图 6.12 中的长轨迹所示。

若平台停在原点开关有效范围内开始回零运动，电机驱动平台以回零速度向负方向运动；当平台运动到负限位位置触发负限位开关，平台减速停止后开始以回零速度向正方向运动，直到离开原点信号有效范围；然后平台减速停止，接着以回零速度向负方向运动；平台再次触发限位开关信号，电机减速停止。最后平台以爬行速度开始反找过程。运动轨迹如图 6.12 中的短轨迹所示。

一次回零+反找模式的定位精度与爬行速度、减速度有关，速度越低，定位越准。该模式一般用于回零行程长、定位精度要求高的场合。

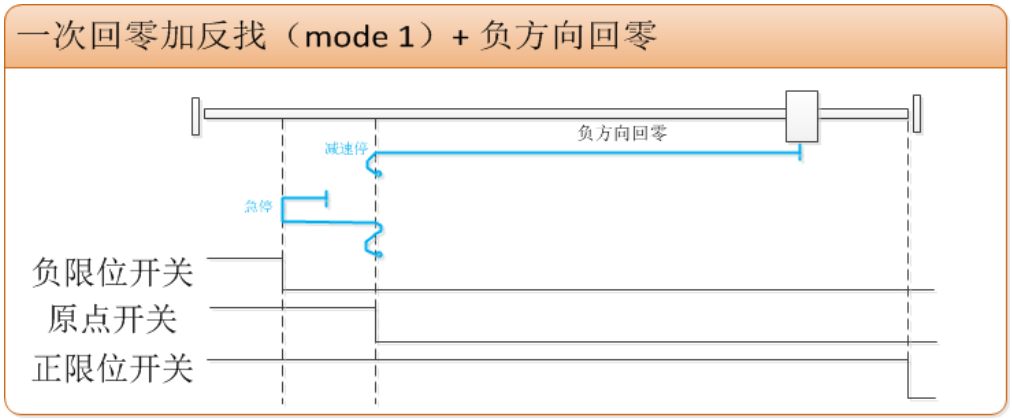


图 6.12 负方向“一次回零+反找”的运动过程

3. “二次回零”模式

二次回零模式的运动过程前半部分与一次回零+反找的过程一样，后面多了一个二次回零的过程，即以爬行速度负方向运动，再次触发原点信号后，减速停止并将停止位置设为原点位置。

一次回零+反找的运动轨迹如图 6.13 中的长轨迹所示。

若平台停在原点开关有效范围内，该回零模式的前半段过程与一次回零+反找的一样，后面多了一个二次回零过程。该运动轨迹如图 6.13 中的短轨迹所示。

二次回零模式与一次回零+反找的不同之处是多了一个回零过程，故时间长；且原点位置在原点开关有效范围内。

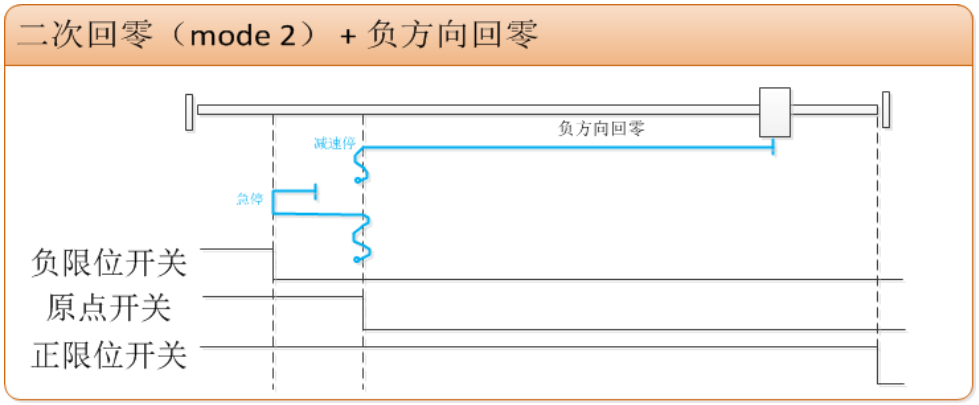


图 6.13 负方向“二次回零”的运动过程

4. “原点锁存”模式

平台在远离原点开关位置开始回零运动，电机驱动平台以回零速度向负方向运动，当平台到达原点开关位置，原点信号被触发，PLC 自动将该点位置锁存，并让电机减速停止；然后电机以回零速度向正方向运动，并在锁存位置停止，且将该位置设为原点，该过程的轨迹如图 6.14 中的长轨迹所示。

若平台停在原点开关有效范围内开始回零运动，电机驱动平台以回零速度向负方向运动；当平台运动到负限位位置触发负限位开关，平台减速停止后开始以回零速度向正方向运动，直到离开原点信号有效范围；然后开始做一次原点锁存回零动作。运动轨迹如图 6.14 中的短轨迹所示。

原点锁存回零模式的定位精度与回零速度、回零减速度有关。速度越低，定位越准。

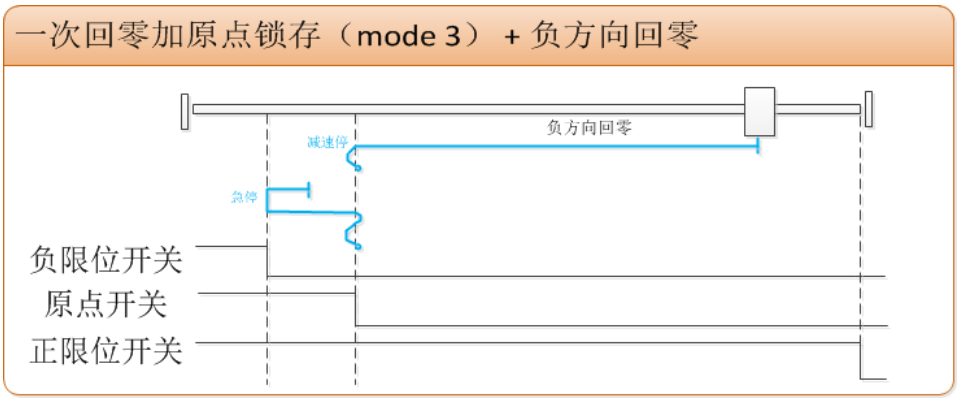


图 6.14 负方向“原点锁存”回零的运动过程

6.3.2 EtherCAT 总线电机回零指令

MC300CS 系列 PLC 控制的 EtherCAT 总线电机回零指令的格式如下。

```
MC_Home( Axis:= 轴号, Execute:= 启动信号(上升沿有效), Position:= 零点偏置量,
         Done=> 运动结束, Busy=> 运动中, CommandAborted=> 运动被终止,
         Error=> 指令出错, ErrorID=> 错误代码 );
```

回零指令中没有设置回零模式、回零速度、回零加速度、回零爬行速度等参数，这

些参数在总线电机参数设置栏中设置，如图 6.15 所示。

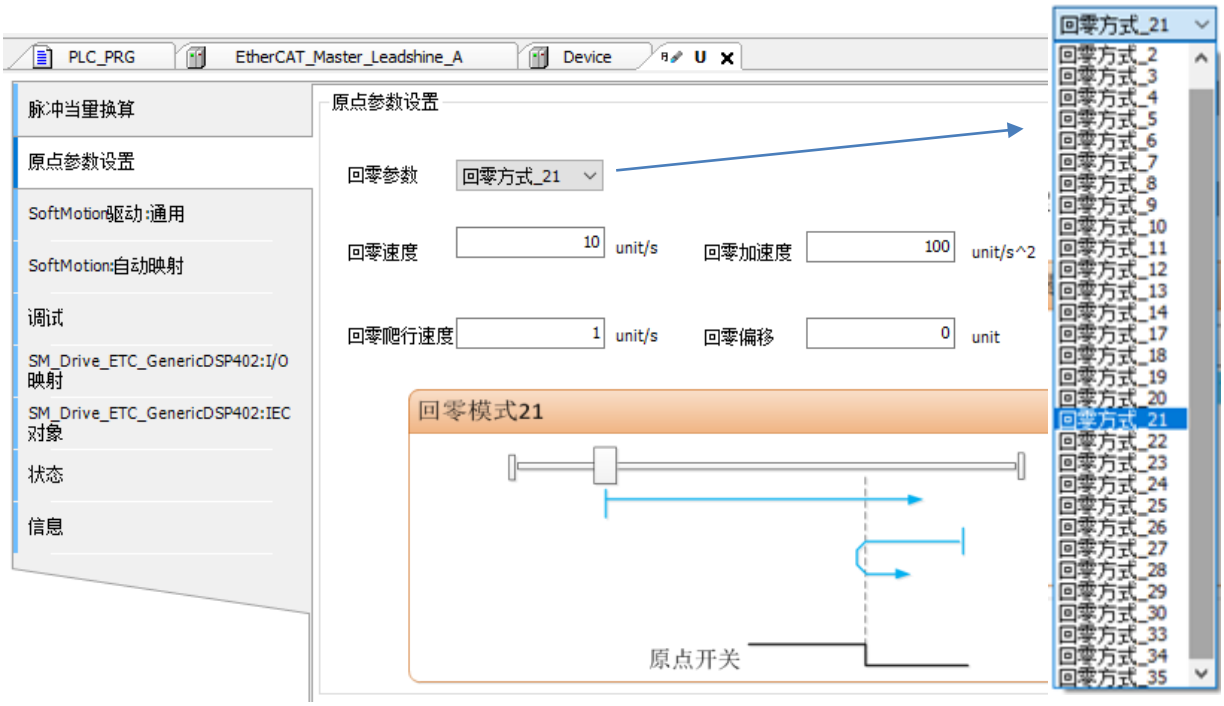


图 6.15　EtherCAT 总线型电机的回零参数设置界面

MC300CS 系列 PLC 可控制 EtherCAT 总线电机以 35 种模式回零，其回零过程符合 CiA402 控制协议的规定。

常用的回零模式有：

19 号回零方式：正方向一次回零+反找，即：电机正方向高速回原点，检测到原点信号后，低速后退，出零点检测范围后停止。

21 号回零方式：负方向一次回零+反找。

20 号回零方式：正方向二次回零，即：电机正方向高速回原点，检测到原点信号后，低速后退，出零点检测范围后再低速负方向进入原点信号范围后停止。

22 号回零方式：负方向二次回零。

雷赛公司的 EtherCAT 总线电机的各种回零模式详见雷赛 EtherCAT 总线型交流伺服系统使用手册、雷赛 EtherCAT 总线型步进驱动器使用手册。

6.4. CANopen 总线型电机的运动控制指令

MC300CS 系列 PLC 控制 CANopen 总线电机只能做单轴运动，无插补运动；而且只能用脉冲作为位移单位。

CANopen 总线型电机的运动控制指令如表 6.2 所示。

表 6.2 CANopen 总线型电机的运动控制指令

单轴控制指令	指令名称	说明
MC_AxisConfi_CAN	设置轴参数	
MC_Power_CAN	轴使能	将驱动器切换为可运行状态
MC_MoveRelativeIme_CAN	相对点位运动	以当前位置为起点，移动指定距离；可在线变速
MC_MoveAbsoluteIme_CAN	绝对点位运动	移动到以绝对坐标定位的终点 可在线变位、变速
MC_JOG_CAN	点动	以指定速度运动，常用于点动控制
MC_MoveVelocity_CAN	速度控制	使用伺服驱动器的位置模式，控制伺服电机以指定速度运行
MC_HomeSetPara_CAN	设置回零参数	控制脉冲型电机回原点
MC_Home_CAN	电机回零	控制总线型电机回原点
MC_Halt_CAN	运动暂停	命令电机暂停运行，可再次调用 MC_Movexxx 指令触发运行
MC_Stop_CAN	运动停止	可命令电机以指定减速度减速后停止
MC_Reset_CAN	轴复位	当伺服电机出现报警停止后，可运行该指令复位。 如轴状态 Axis.nAxisState 为 errorstop 需要切换到 StandStill 要用该指令复位
MC_ReadStatus_CAN	读轴的状态	

1. 设置轴参数指令 MC_AxisConfi_CAN：

```
MC_AxisConfi_CAN( Axis:=轴号, Enable:=使能信号, NetWork:=轴的网络号,
                  NodeID:=轴的节点 ID, ControlWord_Address:= PDO 控制字的地址,
                  StatusWord_Address:= PDO 状态字的地址 , Done=>指令完成,
                  StateMachine=>轴的状态机, StatusWord=>轴的状态字,
                  Error=>出错, ErrorID=>错误代码 );
```

说明：MC300CS 只有一个 CAN 口，故网络号 NetWork = 0。

从站节点 ID 应与从站上的 ID 拨码一致。节点 ID 从 1 开始。

2. 轴使能指令 MC_Power_CAN：

```
MC_Power_CAN( Axis:=轴号, Enable:=使能信号, Done=>轴已使能, Busy=>指令执行中,
              Error=>出错, ErrorID=>错误代码 );
```

3. 相对坐标点位运动指令 MC_MoveRelativeIme_CAN：

```
MC_MoveRelativeIme_CAN( Axis:=轴号, Execute:=启动信号(上升沿有效),
                        Distance:=运动距离, Velocity:=最大速度,
                        Acceleration:=加速度, Deceleration:=减速度,
                        Done=>运动完成, Busy=>运动中, CommandAborted=>指令被中断,
                        Error=>出错, ErrorID=>错误代码 );
```

4. 绝对坐标点位运动指令 MC_MoveAbsoluteIme_CAN：

```
MC_MoveAbsoluteIme_CAN( Axis:=轴号, Execute:=启动信号(上升沿有效),
                        Position:=终点位置, Velocity:=最大速度,
                        Acceleration:=加速度, Deceleration:=减速度,
                        Done=>运动完成, Busy=>运动中, CommandAborted=>指令被中断,
                        Error=>出错, ErrorID=>错误代码 );
```

5. 点动指令 MC_JOG_CAN：

```
MC_JOG_CAN( Axis:=轴号, JogForward:=正向点动信号, JogBackword:=反向点动信号,
                        Velocity:=最大速度,Acceleration:=加速度, Deceleration:=减速度,
                        Done=>运动完成, Busy=>运动中, CommandAborted=>指令被中断,
                        Error=>出错, ErrorID=>错误代码 );
```

6. 恒速运动指令 MC_MoveVelocity_CAN：

```
MC_MoveVelocity_CAN( Axis:=轴号, Execute:=启动信号(上升沿有效),
                     Velocity:=最大速度, Acceleration:=加速度, Deceleration:=减速度,
                     InVelocity=>达到最大速度信号, Busy=>运动中,
                     CommandAborted=>指令被中断, Error=>出错, ErrorID=>错误代码 );
```

7. 设置回零参数指令 MC_HomeSetPara_CAN：

```
MC_HomeSetPara_CAN( Axis:=轴号, Exec:= 启动信号(上升沿有效), HomeMode:=回零模式,
                    HomeHighSpeed:=最大速度, HomeLowSpeed:=低速, Acceleration:=加速度,
                    Done=>指令完成, Busy=>指令进行中, Error=>出错,
                    Error_CANopenKenel=> , Error_SDO=> );
```

说明：回零模式符合 CiA402 控制协议。

8. 回零指令 MC_Home_CAN：

```
MC_Home_CAN( Axis:=轴号, Execute:=启动信号(上升沿有效), Position:=原点偏移位置,
             Done=>运动完成, Busy=>运动中, CommandAborted=>指令被中断,
             Error=>出错, ErrorID=>错误代码);
```

9. 运动停止指令 MC_Stop_CAN：

```
MC_Stop_CAN( Axis:=轴号, Execute:=启动信号(上升沿有效),
              Done=>指令完成, Busy=>指令进行中, Error=>出错, ErrorID=>错误代码 );
```

10. 运动暂停指令 MC_Halt_CAN：

```
MC_Halt_CAN( Axis:=轴号, Execute:=启动信号(上升沿有效), Deceleration:=减速度 ,
              Done=>指令完成, Busy=>指令进行中, Error=>出错, ErrorID=>错误代码);
```

11. 轴复位指令 MC_Reset_CAN：

```
MC_Reset_CAN( Axis:=轴号, Execute:=启动信号(上升沿有效),
               Done=>指令完成, Busy=>指令进行中, Error=>出错, ErrorID=>错误代码 );
```

12. 读轴状态指令 MC_ReadStatus_CAN：

```
MC_ReadStatus_CAN( Axis:=轴号, Enable:=使能信号, valid=>指令生效中, Busy=>指令执行中,
                   Error=>出错, ErrorID=>错误代码, ErrorStop=>轴发生故障,
                   Disabled=>轴未使能, stopping=>轴正在停止, Homing=>轴正在回零,
                  Standstill=>轴已使能, DiscreteMotion=>轴正在执行点位运动,
                  ContinuousMotion=>轴正在执行速度指令 );
```

第7章 插补运动控制

MC300CS 系列 PLC 不支持插补运动。

第8章 多轴联动

8.1. 电子齿轮

齿轮减速器的内部结构如图 8.1 所示。主动轴为输入轴，从动轴 1、2 为输出轴。

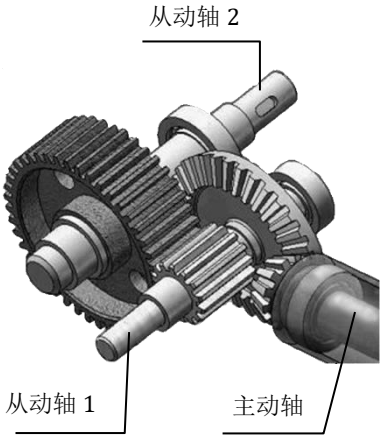


图 8.1　齿轮减速器内部结构

一对平行轴外啮合齿轮的传动比 i 定义为：

$$i = -\frac{\omega_1}{\omega_2} = -\frac{\theta_1}{\theta_2} = -\frac{r_2}{r_1} = -\frac{z_2}{z_1}$$

式中：ω1、θ1、r1、z1，ω2、θ2、r2、z2 分别为齿轮 1 和齿轮 2 的角速度、角位移、半径、齿数。

MC300CS 系列 PLC 提供电子齿轮功能，以此取代齿轮减速器。这样不但可以降低硬件成本、而且具有可任意设置旋转方向、传动比的设置不受齿轮尺寸的限制、调整传动比方便等优点。

8.1.1 电子齿轮的指令

MC300CS 系列 PLC 与电子齿轮相关的主要指令如下。

表 8.1　电子齿轮相关指令

名称	功能
MC_GearIn	设置主从轴、齿轮比，并启动电子齿轮
MC_GearInPos	设置主从轴、同步距离、齿轮比，并启动电子齿轮
MC_GearOut	断开电子齿轮
MC_Phasing	从动轴跟随主动轴

1. 齿轮耦合指令 MC_GearIn

该指令用于设置电子齿轮比及相关参数，并启动电子齿轮。其格式如下。

```
MC_GearIn (Master:=主动轴号, Slave:=从动轴号, Execute:=启动信号,
           RatioNumerator:=齿轮比分子, RatioDenominator:=齿轮比分母,
           Acceleration:=加速度, Deceleration:=减速度, Jerk:=加加速度,
           BufferMode:=缓存模式, InGear=>齿轮啮合完成, Busy=>齿轮工作中,
           Active=>轴在工作, CommandAborted=>指令被中断,
           Error=>出错, ErrorID=>错误码 );
```

功能说明：

MC_GearIn 是速度同步类指令，执行该指令后，从动轴将处于同步状态；要解除耦合，必须调用 GearOut 指令。

当从动轴的速度到达设定速度时，InGear 信号变为 TRUE，此后，

从动轴的位移量 = 主动轴的位移量 × RatioNumerator / RatioDenominator

MC_GearIn 指令处于运行状态时，重新触发 MC_GearIn 指令就可改变电子齿轮比，不需要调用 MC_GearOut 指令暂停电子齿轮运动。

注意：执行该指令过程中，请不要使用 MC_SetPosition 指令，以免引起电机转速突变而发生事故。

2. 齿轮平滑耦合指令 MC_GearInPos

该指令与 MC_GearIn 指令略有不同。它可以指定开始同步时的主动轴位置、从动轴位置、主动轴开始同步的距离，并以此来完成切入电子齿轮的动作。其格式如下。

```
MC_GearInPos( Master:= 主动轴号, Slave:= 从动轴号, Execute:= 启动信号,
              RatioNumerator:= 齿轮比分子, RatioDenominator:= 齿轮比分母,
              MasterSyncPosition:= 主动轴同步位置, SlaveSyncPosition:= 从动轴同步位置,
              MasterStartDistance:= 主动轴开始距离, BufferMode:= 缓存模式,
             AvoidReversal:= 禁止反转,  StartSync=> 同步开始,
             InSync=> 同步完成, Busy=> 齿轮工作中, Active=> 轴在工作,
             CommandAborted=> 指令被中断, Error=> 出错, ErrorID=> 错误码 );
```

3. 齿轮脱离指令 MC_GearOut

该指令将从动轴与主动轴之间的耦合断开，但对主动轴的动作无影响。其格式如下。

```
MC_GearOut( Slave：= 从动轴号,  Execute：= 启动信号,  Done=> 执行完成,
            Busy=> 执行中,  Error=> 出错, ErrorID=> 错误码 );
```

4. 位置差指令 MC_Phasing

该指令将结束 MC_GearIn 或 MC_CamIn 指令，但并不停止从动轴的运动，而是以设置的速度、加速度和加加速度调整从动轴的运动，使得从动轴与主动轴保持一个固定的位置差。其格式如下。

```
MC_Phasing( Master:=主动轴号, Slave:=从动轴号, Execute:= 启动信号, PhaseShift:=位置差,
            Velocity:=速度, Acceleration:=加速度,Deceleration:=减速度, Jerk:=加加速度,
           Done=>指令完成, Busy=>指令进行中, CommandAborted=>指令被中断,
           Error=>出错, ErrorID=>错误码  );
```

8.1.2 电子齿轮的例程

设 X 轴为主动轴，Y、Z 轴为从动轴。Y 轴与 X 轴的减速比为-1000/2000，Z 轴与 X 轴的减速比为 1000/3000。

当按键 input0 按下，X 轴执行相对点位运动指令，做往复运动；往复运动 3 次后自动停止；Y、Z 轴按设置的减速比运动。

若输入信号 input1 有效,Z 轴终止电子齿轮指令,并跟随 X 轴运动,位移差为 50mm。

例程的代码如下。三个轴的运动轨迹如图 8.2 所示。

```
PROGRAM PLC_PRG
VAR
    axes:dut_pulse_axis;  // 声明脉冲轴结构体
```

```
    PowerX: MC_Power;      // 声明轴使能
    PowerY: MC_Power;
    PowerZ: MC_Power;
    MoveRelative: MC_MoveRelative; // 声明相对点位运动
    GearIn1:MC_GearIn;             // 声明齿轮耦合
    GearIn2:MC_GearIn;
    Phasing: MC_Phasing;           // 声明位置差指令
    GearOut1: MC_GearOut;          // 声明齿轮脱离
    GearOut2: MC_GearOut;
    input0: BOOL;                  // 按键 input0
    input1: BOOL;                 // 输入信号 input1
    startGear: BOOL;              // 齿轮启动信号
    startMove: BOOL;              // 点位运动启动信号
    moveDistance: LREAL;          // 点位运动的运动距离
    state: INT;                   // 控制 X 轴的状态值
    moveCounter: INT;             // 往复运动计数器
    gearStop: BOOL;               // 齿轮脱离启动信号
    startPhasing: BOOL;           // 位置差跟随启动信号
END_VAR

axes.pulaxis_0:=ADR(X);// 初始化脉冲轴。ADR 运算符生成轴 X 的参数地址
axes.pulaxis_1:=ADR(Y);
axes.pulaxis_2:=ADR(Z);
LS_MotionControl_P(stAxis:=axes, xClearErr:=, fLimtAxisSpeedJump:=,
                   xDone=>, xError=>, eErrorID=>, xLimitAxisMoveFlag=> );
// 轴使能：
PowerX(Axis:=X, Enable:=TRUE, bRegulatorOn:=TRUE, bDriveStart:=TRUE, Status=>,
       bRegulatorRealState=>, bDriveStartRealState=>, Busy=>, Error=>, ErrorID=> );
PowerY(Axis:=Y, Enable:=TRUE, bRegulatorOn:=TRUE, bDriveStart:=TRUE, Status=>,
       bRegulatorRealState=>, bDriveStartRealState=>, Busy=>, Error=>, ErrorID=> );
PowerZ(Axis:=Z, Enable:=TRUE, bRegulatorOn:=TRUE, bDriveStart:=TRUE, Status=>,
       bRegulatorRealState=>, bDriveStartRealState=>, Busy=>, Error=>, ErrorID=> );
// X 轴做相对点位运动：
MoveRelative(Axis:=X, Execute:=startMove, Distance:=moveDistance, Velocity:=20,
           Acceleration:=100, Deceleration:=100, Jerk:=1000, BufferMode:= ,
           Done=> , Busy=> , Active=> , CommandAborted=> , Error=> , ErrorID=> );
// X、Y 轴耦合为电子齿轮：
GearIn1(Master:=X, Slave:=Y, Execute:=startGear, RatioNumerator:=-1000,
       RatioDenominator:=2000, Acceleration:=100, Deceleration:=100, Jerk:=1000,
       BufferMode:= , InGear=> , Busy=> , Active=> ,
       CommandAborted=> , Error=> , ErrorID=> );
// X、Z 轴耦合为电子齿轮：
GearIn2(Master:=X, Slave:=Z, Execute:=startGear, RatioNumerator:=1000,
       RatioDenominator:=3000, Acceleration:=100, Deceleration:=100, Jerk:=1000,
       BufferMode:= , InGear=> , Busy=> , Active=> ,
       CommandAborted=> , Error=> , ErrorID=> );
// Z 轴与 X 轴保持位置差：
Phasing(Master:=X, Slave:=Z, Execute:=startPhasing, PhaseShift:=50,
       Velocity:=60, Acceleration:=120, Deceleration:=120, Jerk:=1500,
       Done=> , Busy=> , CommandAborted=> , Error=> , ErrorID=> );
// 电子齿轮脱离：
GearOut1(Slave:=Y, Execute:=gearStop, Done=> , Busy=> , Error=> , ErrorID=> );
GearOut2(Slave:=Z, Execute:=gearStop, Done=> , Busy=> , Error=> , ErrorID=> );
```

```
IF input0 = TRUE THEN  // 如果按键 input0 按下
    startGear := TRUE; // 电子齿轮耦合
    state := 1;        // X 轴开始往复运动
END_IF

IF input1 = TRUE THEN      // 如果输入信号 input1 有效
    startPhasing := TRUE;  // 启动 Z 轴对 X 轴的跟随
END_IF

CASE state OF              // 控制 X 轴往复运动
1:
   moveDistance := 100;       // 设置运动距离 100mm
   startMove := TRUE;         // 启动相对点位运动
   state := 2;                // 进入下一步
2:
   IF MoveRelative.Done THEN  // 如果点位运动结束
      startMove := FALSE;     // 准备重启点位运动
      state := 3;             // 进入下一步
   END_IF
3:
   moveDistance := -100;      // 设置运动距离-100mm
   startMove := TRUE;         // 启动相对点位运动
   state := 4;                // 进入下一步
4:
   IF MoveRelative.Done THEN            // 如果点位运动结束
      startMove := FALSE;               // 准备重启点位运动
      state := 1;                       // 返回到第 1 步
      moveCounter := moveCounter + 1;   // 往复运动次数加 1
   END_IF
   IF moveCounter = 3 THEN              // 如果完成 3 次往复运动
      startGear:= FALSE;                // 准备下一次啮合齿轮
      gearStop := TRUE;                 // 齿轮啮合脱离
      startPhasing := FALSE;            // Z 轴结束位置跟随
      state := 0;                       // 恢复初始状态
      moveCounter := 0;                 // 计数器清零
   END_IF
END_CASE
```

从图 8.2 中可看出，Y 轴的位移量是 X 轴的-1/2；Z 轴的位移量是 X 轴的 1/3；当输入信号 input1 有效后，Z 轴与 X 轴的位移差保持在 50mm。

8.2. 电子凸轮

如图 8.3 所示，凸轮机构由凸轮和从动杆（也称为挺杆）组成。凸轮是一个具有曲线轮廓的构件，通常作等速转动；从动件在凸轮的驱动下，按照凸轮轮廓曲线定义的运动规律上下运动。

MC300CS 系列 PLC 提供电子凸轮功能。用 PLC 代替凸轮机构，不但可降低硬件成本，而且改变凸轮运动规律简单、方便。

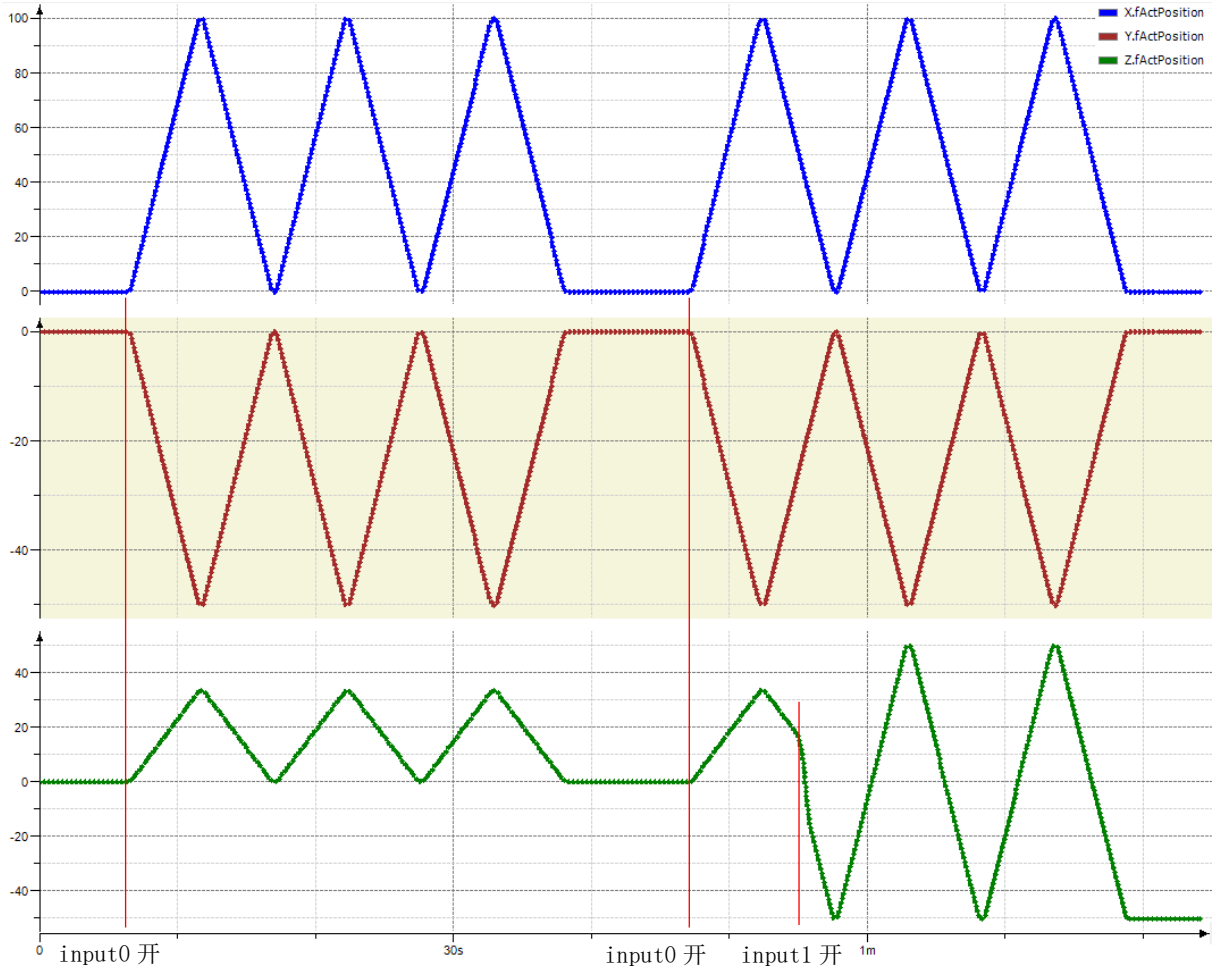


图 8.2　电子齿轮减速器的轨迹图

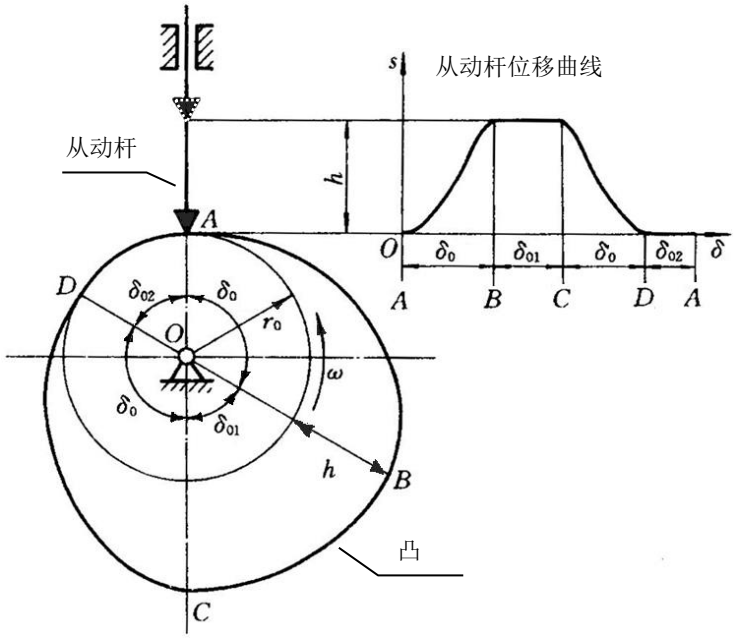


图 8.3　凸轮机构及其运动规律

8.2.1 电子凸轮的指令

MC300CS 系列 PLC 与电子凸轮相关的主要指令如下。

表 8.2 电子凸轮相关指令

名称	功能
MC_CAM_REF	定义凸轮表参数
MC_CamTableSelect	选择 cam 表
MC_CamIn	设置主从轴参数并启动电子凸轮
MC_CamOut	断开电子凸轮

1. 凸轮表参数定义指令 MC_CAM_REF

如果使用程序生成凸轮表，需要调用该指令；如果使用 LeadSys 软件手动生成凸轮表，就不用调用该指令。

```
MC_CAM_REF( wCamStructID:=CamStruct 的 ID, byType:=凸轮轮廓类型,
            byVarType:=凸轮轮廓数据类型, xStart:=主动轴开始位置, xEnd:=主动轴结束位置,
            nElements:=凸轮轮廓数据点个数, nTappets:=挺杆数量, pce:=凸轮地址指针,
            pt:= 挺杆表指针, dwTappetActiveBits:= ,strCAMName:=,
            byInterpolationQuality:=插补质量, byCompatibilityMode:= ,
            bChangedOnline:= , xPartofLM:= );
```

其中：

凸轮轮廓类型：0 – 五阶多项式；

1 – 等距，根据从动轴位置表；

2 – 非等距，根据主从轴相对位置表；

3 – 多项式，根据主动轴位置，从动轴位置、速度和加速度。

凸轮轮廓数据类型：1 – INT，2 – UINT，3 – DINT，4 – UDINT，5 – REAL，6 – LREAL

插补质量：1 –直线插补，3 – 三次插补

2. 选择凸轮表指令 MC_CamTableSelect

该指令用于凸轮表选择，输出凸轮表 ID 号；需要和指令 MC_CamIn 配合使用。

```
MC_CamTableSelect( Master：=主动轴轴号， Slave：=从动轴轴号， CamTable：=凸轮表名，
                   Execute：=启动信号（上升沿有效）， Periodic：=重复模式，
                   MasterAbsolute：=主动轴位置模式， SlaveAbsolute：=从动轴位置模式，
                   Done=>执行完成， Busy=>执行中， Error=>出错， ErrorID=>错误代码，
                   CamTableID=>凸轮表 ID );
```

其中：

重复模式：TRUE – 凸轮重复转动；FALSE – 凸轮转一周。

主动轴位置模式：TRUE – 绝对坐标，FALSE – 相对坐标

从动轴位置模式：TRUE – 绝对坐标，FALSE – 相对坐标

3. 启动凸轮指令 MC_CamIn

设置主动轴和从动轴的参数并启动电子凸轮。

```
MC_CamIn( Master：=主动轴轴号， Slave：=从动轴轴号， Execute：=启动信号（上升沿有效），
          MasterOffset：=主动轴位置偏移， SlaveOffset：=从动轴位置偏移，
          MasterScaling：=主动轴位置比例， SlaveScaling：=从动轴位置比例，
          StartMode：=从动轴啮合方式， CamTableID：=凸轮表 ID，
          VelocityDiff：=速度， Acceleration：=加速度， Deceleration：=减速度，
          Jerk：=加加速度， TappetHysteresis：=挺杆滞后位置，
          InSync=>同步中， Busy=>执行中， CommandAborted=>指令被中断， Error=>出错，
          ErrorID=>错误代码， EndOfProfile=>凸轮曲线结束， Tappets=>挺杆表 );
```

4. 断开凸轮指令 MC_CamOut

该指令解除指定的从动轴与主动轴的凸轮耦合关系。

```
MC_CamOut( Slave：=从动轴轴号， Execute：=启动信号（上升沿有效）， Done=>执行完成，
           Busy=>执行中， Error=>出错， ErrorID=>错误代码 );
```

注意：如果从动轴在执行该指令前速度不为 0，即使凸轮耦合断开，从动轴仍然按照耦合时的速度运行。故 MC_CamOut 指令通常会与 MC_Stop 指令配合使用。

8.2.2 用软件手动生成凸轮表及使用方法

下面通过一个例程介绍手动生成凸轮表及其使用的方法。

设 X 轴为凸轮，Y 轴为挺杆。X 轴匀速运动，即 X = V×t

Y 轴的运动轨迹为：

$$Y = 100\times SIN((X-90) \times 3.1416/180) + 100, \quad (X = 0\sim 360)$$

X、Y 轴的位移关系如表 8.3、图 8.4 所示。

8.3 电子凸轮表

X	Y
0	0.00
30	13.40
60	50.00
90	100.00
120	150.00
150	186.60
180	200.00
210	186.60
240	150.00
270	100.00
300	50.00
330	13.40
360	0.00

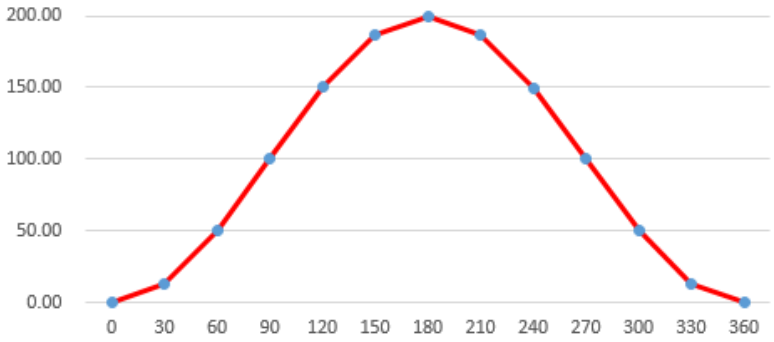


图 8.4 挺杆位移曲线

创建凸轮表的步骤如下：

1. 打开 LeadSys Studio 软件，新建工程、添加脉冲轴 X、Y
2. 鼠标右键单击工程设备栏中的“Application”，选择“添加对象”-“cam 表”，如图 8.5 所示；命名为 Cam1，并打开凸轮表；
3. 然后将表 8.3 中的 X、Y 轴的位置填入凸轮表，如图 8.6 所示；凸轮表自动生成速度 V。注意：“段类型”要选 Line。

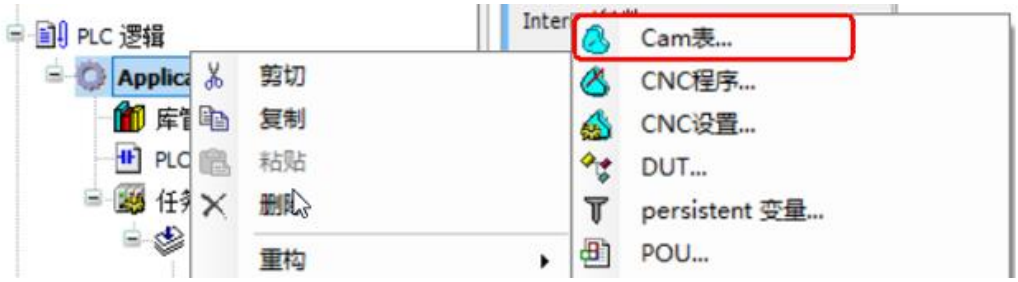
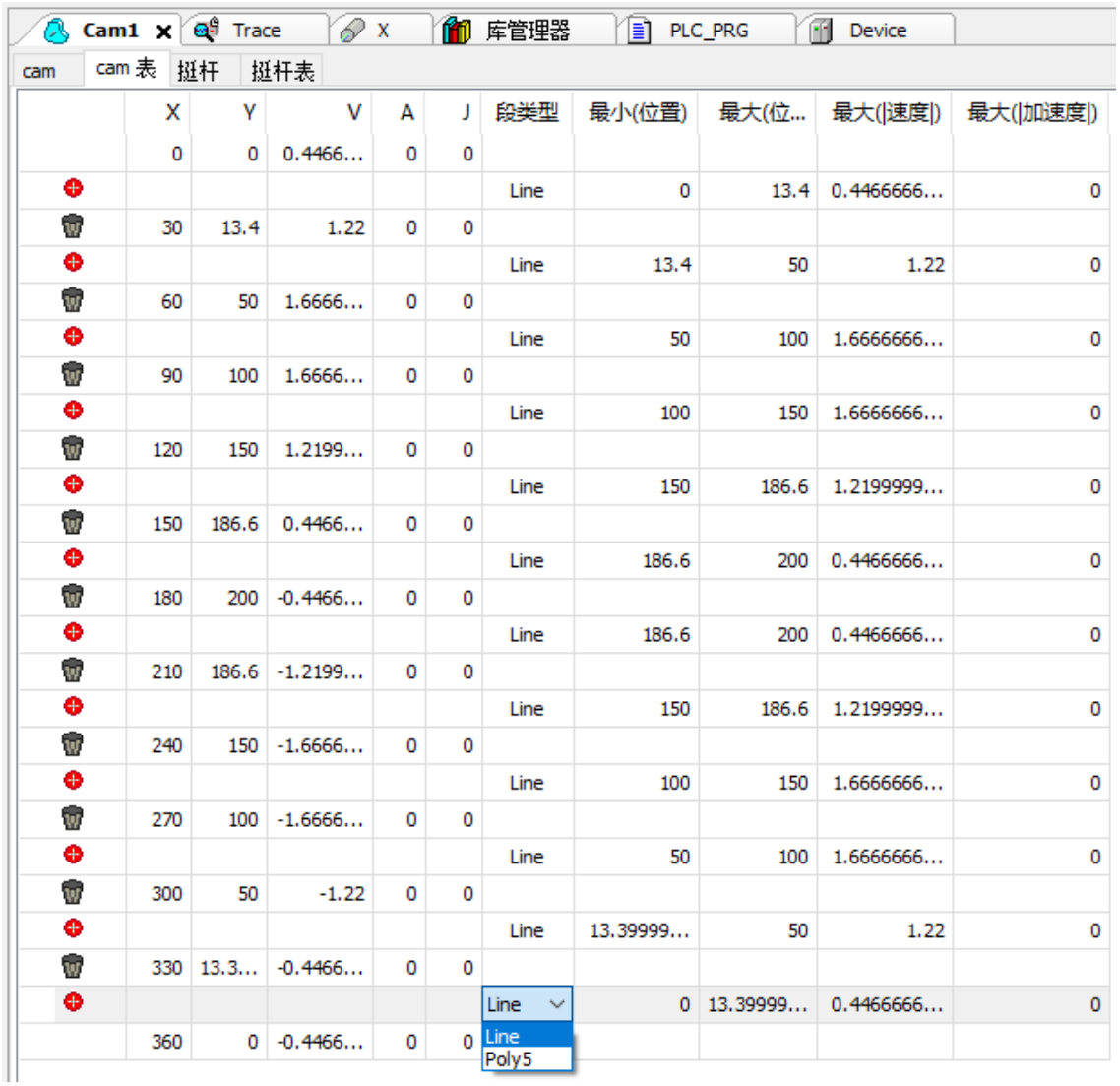


图 8.5　凸轮机构及其运动规律



	X	Y	V	A	J	段类型	最小(位置)	最大(位...	最大(|速度|)	最大(|加速度|)
	0	0	0.4466...	0	0					
						Line	0	13.4	0.4466666...	0
	30	13.4	1.22	0	0					
						Line	13.4	50	1.22	0
	60	50	1.6666...	0	0					
						Line	50	100	1.6666666...	0
	90	100	1.6666...	0	0					
						Line	100	150	1.6666666...	0
	120	150	1.2199...	0	0					
						Line	150	186.6	1.2199999...	0
	150	186.6	0.4466...	0	0					
						Line	186.6	200	0.4466666...	0
	180	200	-0.4466...	0	0					
						Line	186.6	200	0.4466666...	0
	210	186.6	-1.2199...	0	0					
						Line	150	186.6	1.2199999...	0
	240	150	-1.6666...	0	0					
						Line	100	150	1.6666666...	0
	270	100	-1.6666...	0	0					
						Line	50	100	1.6666666...	0
	300	50	-1.22	0	0					
						Line	13.39999...	50	1.22	0
	330	13.3...	-0.4466...	0	0					
						Line	0	13.39999...	0.4466666...	0
	360	0	-0.4466...	0	0					

图 8.6　填写凸轮表

4. 点击“Cam”窗体查看凸轮曲线，如图 8.7 所示。可以用鼠标拖动凸轮位移曲线上的符号⊕，修改凸轮曲线。

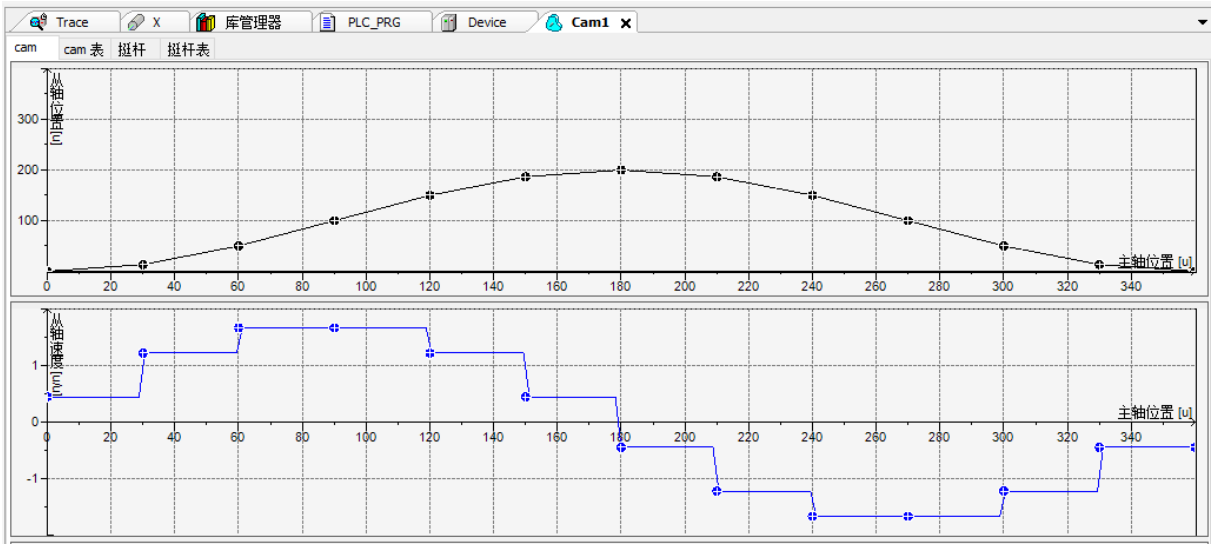


图 8.7　凸轮的位移、速度曲线

5. 然后开始编写程序。

设置凸轮为相对坐标，凸轮循环运动；

在 CamTableSelect 模块中用 CamTableID=>CamID，获取凸轮 ID；在 CamIn 模块中 CamTableID:= CamID。

当按键 input0 按下后，起动 X 轴匀速运动，速度为 72mm/s。

当按键 input1 按下后，启动凸轮运动；

当输入信号 input2 有效时，凸轮运动停止，X 轴运动停止。

程序代码如下。

```
PROGRAM PLC_PRG
VAR
    axes:dut_pulse_axis;                    // 脉冲轴结构体
    PowerX: MC_Power;
    PowerY: MC_Power;
    CamTableSelect: MC_CamTableSelect;
    CamID:  MC_Cam_ID;                      // 凸轮的 ID
    xCamIn: BOOL;
    CamIn: MC_CamIn;
    CamOut: MC_CamOut;
    CamStop: BOOL := FALSE;
    MoveVelocityX: MC_MoveVelocity;
    xGoVel: BOOL;
    StopX: MC_Stop;
    StopY: MC_Stop;
    input0: BOOL ;
    input1: BOOL ;
    input2: BOOL ;
END_VAR
```

```
axes.pulaxis_0:=ADR(X); // 获取 X 轴的地址
axes.pulaxis_1:=ADR(Y); // 获取 Y 轴的地址
// 初始化脉冲轴：
LS_MotionControl_P(stAxis:=axes, xClearErr:=, fLimtAxisSpeedJump:=,
                   xDone=>, xError=>, eErrorID=>, xLimitAxisMoveFlag=> );
//轴使能：
PowerX(Axis:=X, Enable:=1, bRegulatorOn:=1, bDriveStart:=1, Status=>,
       bRegulatorRealState=>, bDriveStartRealState=>, Busy=>, Error=>, ErrorID=> );
PowerY(Axis:=Y, Enable:=1, bRegulatorOn:=1, bDriveStart:=1, Status=>,
       bRegulatorRealState=>, bDriveStartRealState=>, Busy=>, Error=>, ErrorID=> );
//主轴速度指令：
MoveVelocityX(Axis:= X, Execute:= xGoVel, Velocity:= 72, Acceleration:= 100,
              Deceleration:= 100, Jerk:= , Direction:= , BufferMode:= ,
              InVelocity=> , Busy=> , Active=> , CommandAborted=> , Error=> , ErrorID=> );
//凸轮表选择：
CamTableSelect(Master:=X, Slave:=Y, CamTable:=Cam1, Execute:=xCamIn, Periodic:=TRUE,
               MasterAbsolute:=FALSE, SlaveAbsolute:=FALSE,
               Done=> , Busy=> , Error=> , ErrorID=> , CamTableID=>CamID );
//凸轮耦合：
CamIn(Master:=X, Slave:=Y, Execute:=xCamIn, MasterOffset:=0, SlaveOffset:=0,
      MasterScaling:=1, SlaveScaling:=1, StartMode:=1, CamTableID:= CamID,
      VelocityDiff:=, Acceleration:=, Deceleration:=, Jerk:=, TappetHysteresis:=,
      InSync=>, Busy=>, CommandAborted=>, Error=>,
      ErrorID=>, EndOfProfile=>, Tappets=> );
//凸轮脱离：
CamOut(Slave:=Y, Execute:=CamStop, Done=> , Busy=> , Error=> , ErrorID=> );
//轴减速停止：
StopY(Axis:=Y, Execute:=CamStop, Deceleration:=100, Jerk:= ,
       Done=> , Busy=> , Error=> , ErrorID=> );
StopX(Axis:=X, Execute:=CamStop, Deceleration:=100, Jerk:= ,
      Done=> , Busy=> , Error=> , ErrorID=> );

IF input0=TRUE THEN       // 按键 0 按下
   xGoVel :=TRUE;         // 启动 X 轴匀速运动
END_IF

IF input1 = TRUE THEN     // 按键 1 按下
   xCamIn := TRUE;        // 启动凸轮
   camStop:= FALSE;
END_IF

IF input2 = TRUE THEN     // input2 信号有效
   camStop:= TRUE;        // 凸轮停止
   xCamIn := FALSE;
   xGoVel :=FALSE;
END_IF

IF StopY.done AND StopX.done THEN
     CamStop:=FALSE;      // X、Y 轴结束停止状态
END_IF
```

6. 编译、运行程序，如果如图 8.8 所示。

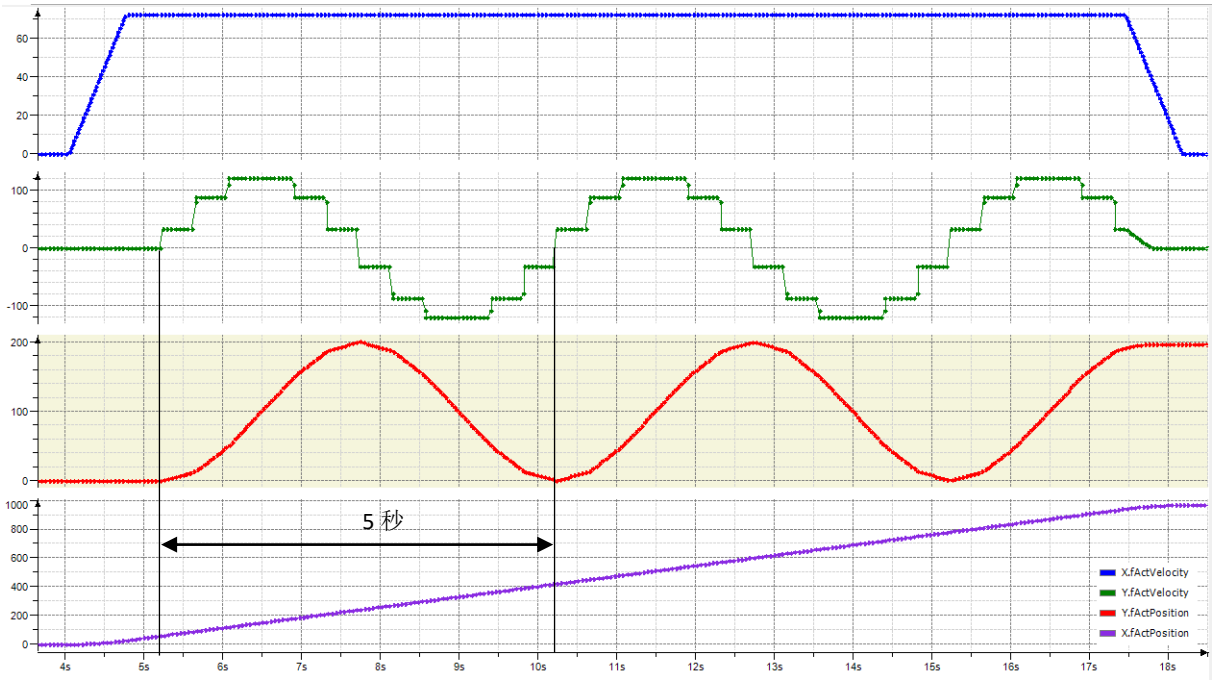


图 8.8　X、Y 轴的位移、速度曲线

当 input1 按下后，凸轮开始运动，且运动周期为 5 秒，每个周期 X 轴相对运动 360mm；Y 轴的位移量与表 8.3 的数据一致。

8.2.3 用程序生成凸轮表及使用方法

当凸轮轮廓数据点较少时，手动生成凸轮表比较简单；但如果要求凸轮轮廓的位置精度高、挺杆运动的速度连续，就应该采用程序生成凸轮表。方法如下。

1. 定义凸轮的参数。

在变量定义区声明凸轮的名称、凸轮的数组。

```
Cam : MC_Cam_REF;                          // 声明一个凸轮
arCam: ARRAY[0..n] OF SMC_CAMXYVA;         // 凸轮数据，n+1 点
```

2. 在程序中计算凸轮的轮廓和速度。

凸轮数组的结构为：

```
arCam[i].dX        // 主动轴第 i 个点的位置
arCam[i].dY        // 从动轴第 i 个点的位置
arCam[i].dV        // 从动轴第 i 个点的速度，dY/dX
arCam[i].dA        // 从动轴第 i 个点的加速度
```

注意：这里 dV = dY/dX，而不是 dY/dt。

与上例相同，X = V×t

Y 轴的运动轨迹为：

Y = 100×SIN((X-90)×3.1416/180) + 100，　（X = 0~360）

故：dY/dX = 100×COS((X-90)×3.1416/180)×3.1416/180.0

3. 编写程序。

程序的其他内容与上例相仿。不同之处有：

1）用 MC_Cam_REF 模块生成凸轮的主要参数：

凸轮轮廓类型 byType:= 3

凸轮轮廓数据类型 byVarType:= 5

主动轴开始位置 xStart:= 0

主动轴结束位置 xEnd:= 360

凸轮轮廓数据点个数 nElements:= 121,

凸轮地址指针 pce:= ADR(arCam)

插补质量 byInterpolationQuality:= 1（直线插补）

2）凸轮循环次数为 1，即 CamTableSelect 模块中的 Periodic:=FALSE

3）按下 input0 键，启动 X 轴匀速运动，并用 FOR 循环语句，计算凸轮轮廓各点参数。

```
FOR n:=0 TO 120 DO                                          // 设置凸轮各点参数：
    arCam[n].dX:=3*n;                                       // 主动轴位置
    arCam[n].dY:=100*SIN((3*n-90)*3.1416/180)+100;          // 从动轴位置
    arCam[n].dV:=100*COS((3*n-90)*3.1416/180)*3.1416/180.0; // 从动轴速度 dY/dX
    arCam[n].dA:=0;                                         // 从动轴加速度
END_FOR
```

4）按下 input1 键，启动凸轮运动；

5）根据模块 CamIn 中的 EndOfProfile=>camDone，判断凸轮轨迹是否完成。若完成，即 camDone=True 后，则让凸轮脱离主动轴，准备下一次启动凸轮。

详细代码如下。

```
PROGRAM PLC_PRG
VAR
    axes:dut_pulse_axis;  // 脉冲轴结构体
    PowerX: MC_Power;
    PowerY: MC_Power;
    Cam : MC_Cam_REF;                       // 声明一个凸轮
    arCam : ARRAY[0..120] OF SMC_CAMXYVA;   // 凸轮数据，121 点
    CamID:  MC_Cam_ID;                      // 凸轮的 ID
    CamTableSelect: MC_CamTableSelect;      // 选择凸轮表
    CamIn: MC_CamIn;                        // 声明电子凸轮啮合模块
    CamOut: MC_CamOut;                      // 声明电子凸轮脱离模块
    xCamIn: BOOL;
    CamStop: BOOL;
    MoveVelocityX: MC_MoveVelocity;         // 声明匀速运动模块
    xGoVel: BOOL;
    StopY: MC_Stop;                         // 声明停止运动模块
    input0: BOOL ;
    input1: BOOL ;
    n: INT;
    camDone: BOOL;
END_VAR


axes.pulaxis_0:=ADR(X); // 获取 X 轴的地址
```

```
axes.pulaxis_1:=ADR(Y); // 获取 Y 轴的地址
// 初始化脉冲轴：
LS_MotionControl_P(stAxis:=axes, xClearErr:=, fLimtAxisSpeedJump:=,
                   xDone=>, xError=>, eErrorID=>, xLimitAxisMoveFlag=> );
// 轴使能：
PowerX(Axis:=X, Enable:=1, bRegulatorOn:=1, bDriveStart:=1, Status=>,
       bRegulatorRealState=>, bDriveStartRealState=>, Busy=>, Error=>, ErrorID=> );
PowerY(Axis:=Y, Enable:=1, bRegulatorOn:=1, bDriveStart:=1, Status=>,
       bRegulatorRealState=>, bDriveStartRealState=>, Busy=>, Error=>, ErrorID=> );
// 主轴速度指令：
MoveVelocityX(Axis:= X, Execute:= xGoVel, Velocity:= 72, Acceleration:= 100,
              Deceleration:= 100, Jerk:= , Direction:= , BufferMode:= , InVelocity=> ,
              Busy=> , Active=> , CommandAborted=> , Error=> , ErrorID=> );
// 定义凸轮参数：
Cam( wCamStructID:= , byType:= 3, byVarType:=5, xStart:= 0, xEnd:=360,
     nElements:= 121, nTappets:= , pce:= ADR(arCam), pt:= , dwTappetActiveBits:= ,
     strCAMName:= , byInterpolationQuality:=1, byCompatibilityMode:= ,
    bChangedOnline:= , xPartofLM:= );
// 凸轮表选择：
CamTableSelect(Master:=X, Slave:=Y, CamTable:=Cam, Execute:=xCamIn, Periodic:=FALSE,
               MasterAbsolute:=FALSE, SlaveAbsolute:=FALSE,
               Done=> , Busy=> , Error=> , ErrorID=> , CamTableID=>CamID );
// 凸轮耦合：
CamIn(Master:=X, Slave:=Y, Execute:=xCamIn, MasterOffset:=0, SlaveOffset:=0,
      MasterScaling:=1, SlaveScaling:=1, StartMode:=1, CamTableID:=CamID,
      VelocityDiff:=, Acceleration:=, Deceleration:=, Jerk:=, TappetHysteresis:=,
      InSync=>, Busy=>, CommandAborted=>, Error=>, ErrorID=>,
      EndOfProfile=>camDone, Tappets=> );
// 凸轮脱离：
CamOut(Slave:=Y, Execute:=CamStop, Done=> , Busy=> , Error=> , ErrorID=> );
// 轴减速停止：
StopY(Axis:=Y, Execute:=CamStop, Deceleration:=100, Jerk:= ,
      Done=> , Busy=> , Error=> , ErrorID=> );

IF input0=TRUE THEN             // 按键 0 按下
    FOR n:=0 TO 120 DO          // 设置凸轮各点参数
        arCam[n].dX:=3*n;                                          // 主动轴位置
        arCam[n].dY:=100*SIN((3*n-90)*3.1416/180)+100;             // 从动轴位置
        arCam[n].dV:=100*COS((3*n-90)*3.1416/180)*3.1416/180.0; // 从动轴速度 dY/dX
        arCam[n].dA:=0;                                            // 从动轴加速度
    END_FOR
END_IF

IF input1 = TRUE THEN      // 按键 1 按下
   xCamIn := TRUE;         // 启动凸轮
END_IF

IF camDone = TRUE THEN     // 凸轮轨迹完成
   camStop:= TRUE;         // 凸轮停止
   xCamIn := FALSE;        // 凸轮复位
   camDone := FALSE;
END_IF
```

```
IF StopY.done  THEN        // Y 轴停止
    camStop:= FALSE;       // MC_Stop 模块复位
END_IF
```

4. 运行程序。结果如图 8.9 所示。

与图 8.8 比较，凸轮的轮廓函数虽然一样，但图 8.9 的速度曲线连续、平滑；位置曲线更精确。

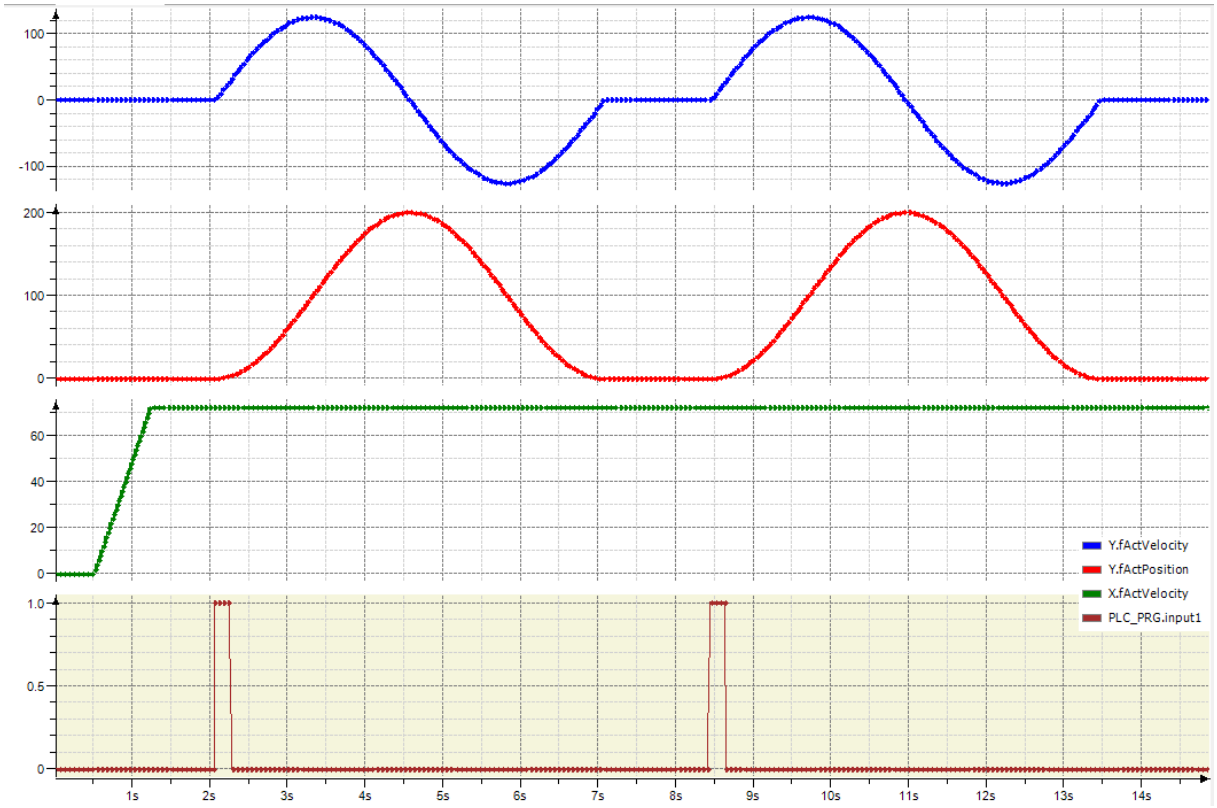


图 8.9　凸轮启动信号和凸轮的位移、速度曲线

注意： 图 8.9 中 Y 轴的速度曲线 Y.fActVelocity 是 dY/dt，而非 dY/dX。

dY/dt = 100×COS((X-90)×3.1416/180)×72×3.1416/180.0

故其最大值为 100×1×72×3.1416/180.0 = 125.664 mm/s

8.3. 多轴样条插值

样条是在造船业下料时放样用的一种极富弹性的细木条或薄钢条，可将一些特定的点（这些点称为样点）连成光滑的曲线。样条插值就是模拟样条画曲线的过程，用高次多项式函数将样点连接起来。

PLC 在拟合各种特殊的运动轨迹时，使用样条插值函数十分方便。MC300CS 系列 PLC 的样条插值函数如表 8.4 所示。指令详细说明请参考《雷赛大中型 PLC 指令手册》。

表 8.4 样条插值运动指令

指令名	功能说明
MC_PositionProfile	通过定义时间-位置表，控制轴的位移
MC_VelocityProfile	通过定义时间-速度表，控制轴的速度
MC_AccelerationProfile	通过定义时间-加速度表，控制轴的加速度

8.3.1 样条插值指令及参数

1. 位置样条插值指令 MC_PositionProfile

使用该指令自定义位置-时间表，可实现单轴位移的自定义运动规律。若多轴同时运行该指令，可实现自定义的轨迹。该指令格式如下。

```
MC_PositionProfile( Axis:=轴号，TimePosition:=位置-时间数据表，
                    Execute:=启动信号(上升沿有效)，ArraySize:=数据点个数，
                    PositionScale:=位置比例，Offset:=偏移量，
                    Done=>执行完成，Busy=>执行中，CommandAborted=>指令被中断，
                    Error=>出错，ErrorID=>错误码 );
```

其中，“位置-时间”数据表的结构体为：

```
TYPE MC_TP_REF
    STRUCT
        Number_of_pairs：INT;                  // 位置-时间数据点的个数
        IsAbsolute：BOOL;                      // 绝对位置、相对位置选择
        MC_TP_Array：ARRAY [1..N] of MC_TP;    // 位置-时间数据
    END_STRUCT
END_TYPE
```

“位置-时间”数据的结构体为：

```
TYPE MC_TP
    STRUCT
        delta_time：TIME;     // 时间
        position：REAL;       // 位置（绝对值或相对值）
    END_STRUCT
END_TYPE
```

2. 速度样条插值指令 MC_VelocityProfile

使用该指令自定义速度-时间表，可实现单轴速度的自定义运动规律。

```
MC_VelocityProfile( Axis:= 轴号，TimeVelocity:=速度-时间数据表，
                    Execute:=启动信号(上升沿有效)，ArraySize:= 数据点个数，
                    VelocityScale:=速度比例，Offset:=偏移量，
                    Done=>执行完成，Busy=>执行中，CommandAborted=>指令被中断，
                    Error=>出错， ErrorID=>错误码 );
```

其中，“速度-时间”数据表的结构体为：

```
TYPE MC_TV_REF
    STRUCT
        Number_of_pairs: INT;                  // 速度-时间数据点的个数
        IsAbsolute: BOOL;                      // 绝对速度、相对速度选择
        MC_TV_Array: ARRAY [1..N] of MC_TV;    // 速度-时间数据
    END_STRUCT
END_TYPE
```

“速度-时间”数据的结构体为：

```
TYPE MC_TV
    STRUCT
        delta_time: TIME;     // 时间
        velocity: REAL:       // 速度（绝对值或相对值）
    END_STRUCT
END_TYPE
```

8.3.2 样条插值指令例程

三叶玫瑰线如图 8.10 所示，可以用 2 条样条插值指令实现该轨迹控制。三叶玫瑰线的极坐标方程为：

$$\rho = a\times\sin(3\theta),\quad \theta = 0\sim180^{\circ}$$

将极坐标方程转换为直角坐标方程如下：

$$\theta = \omega\times t$$

$$x = \rho\times\cos(\theta) = a\times\sin(3\theta)\times\cos(\theta)$$

$$y = \rho\times\sin(\theta) = a\times\sin(3\theta)\times\sin(\theta)$$

设 $\omega = 25^{\circ}/s$，$a = 100$ mm，可得 X、Y 轴在时间轴上的位移曲线如图 8.11 所示。先将图中的样点数据储存在数组内，执行样条插值指令就按照该曲线运动。

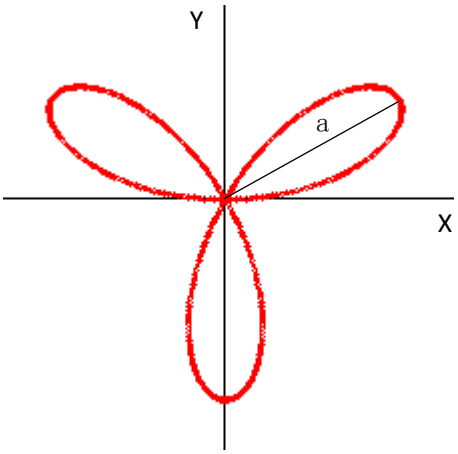
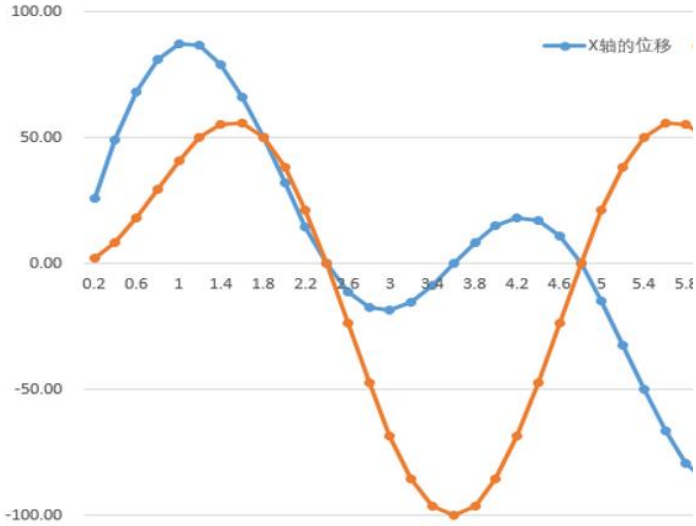


图 8.10　三叶玫瑰线



8.11　三叶玫瑰线对应的 X、Y 轴的位移曲线

编程之前，先在变量定义区声明样条曲线函数和样条曲线的数据表：

```
PositionProfileX:  MC_PositionProfile;
PositionProfileY:  MC_PositionProfile;
PTdataX: MC_TP_REF;
PTdataY: MC_TP_REF;
```

然后在程序中设置样条曲线的参数、计算样点，并保存在数据表中。程序如下：

```
PTdataX.IsAbsolute := TRUE;      // X 轴为绝对坐标
PTdataX.Number_of_pairs := 36;   // X 轴样点的个数
PTdataY.IsAbsolute := TRUE;      // Y 轴为绝对坐标
PTdataY.Number_of_pairs := 36;   // Y 轴样点的个数
FOR i := 1 TO 36  BY 1 DO           // 计算 36 个样点的数据
    t := 5.0 * i * PI / 180.0;      // 计算时间
    p := 100.0 * SIN(3.0*t);        // 计算极径
    px := p * COS(t);               // 计算 X 轴的坐标
    py := p * SIN(t);               // 计算 Y 轴的坐标
    PTdataX.MC_TP_Array[i].delta_time := T#200MS; // 保存 2 点间的时间
    PTdataX.MC_TP_Array[i].position := px;        // 保存 X 轴的位置
    PTdataY.MC_TP_Array[i].delta_time := T#200MS; // 保存 2 点间的时间
    PTdataY.MC_TP_Array[i].position := py;        // 保存 Y 轴的位置
END_FOR;
```

当按下 input0 按键后，开始启动样条插值函数。完整的代码如下。

```
PROGRAM PLC_PRG
VAR
    axes:dut_pulse_axis;  // 脉冲轴结构体
    PowerX: MC_Power;
    PowerY: MC_Power;
    input0: BOOL;
    flag: BOOL := TRUE;
    donePTx: BOOL;
    donePTy: BOOL;
    startPT: BOOL;
    PositionProfileX:  MC_PositionProfile;
    PositionProfileY:  MC_PositionProfile;
    PTdataX: MC_TP_REF;
    PTdataY: MC_TP_REF;
    i: INT;
    p: LREAL;
    px: LREAL;
    py: LREAL;
    t: LREAL;
END_VAR



axes.pulaxis_0:=ADR(X); // 获取 X 轴的地址
```

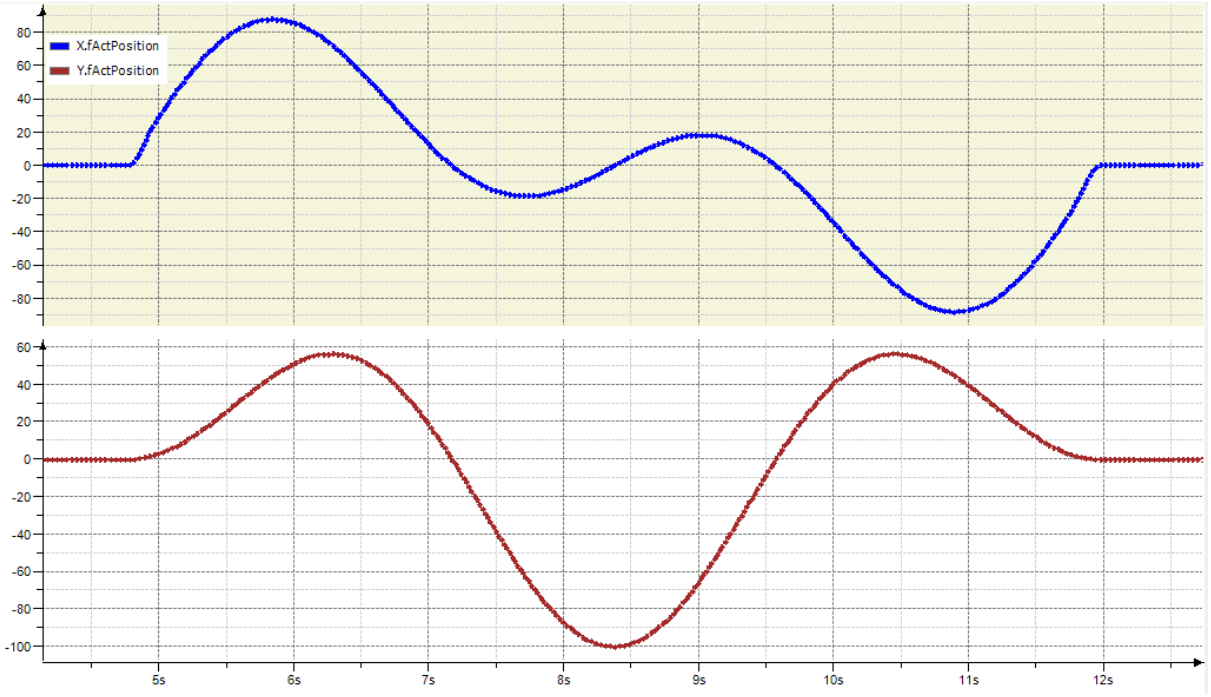
```
axes.pulaxis_1:=ADR(Y); // 获取 Y 轴的地址
// 初始化脉冲轴：
LS_MotionControl_P(stAxis:=axes, xClearErr:=, fLimtAxisSpeedJump:=,
                   xDone=>, xError=>, eErrorID=>, xLimitAxisMoveFlag=> );
//轴使能：
PowerX(Axis:=X, Enable:=1, bRegulatorOn:=1, bDriveStart:=1, Status=>,
      bRegulatorRealState=>, bDriveStartRealState=>, Busy=>, Error=>, ErrorID=> );
PowerY(Axis:=Y, Enable:=1, bRegulatorOn:=1, bDriveStart:=1, Status=>,
      bRegulatorRealState=>, bDriveStartRealState=>, Busy=>, Error=>, ErrorID=> );
//样条插值函数：
PositionProfileX( Axis:=X, TimePosition:=PTdataX , Execute:=startPT , ArraySize:=36,
                  PositionScale:=1, Offset:=0,
                  Done=>donePTx, Busy=> , CommandAborted=> , Error=> , ErrorID=> );
PositionProfileY( Axis:=Y, TimePosition:=PTdataY, Execute:=startPT , ArraySize:=36,
                  PositionScale:=1, Offset:=0,
                  Done=>donePTy, Busy=> , CommandAborted=> , Error=> , ErrorID=> );

IF flag = TRUE THEN                // PT 运动参数的赋值
   PTdataX.IsAbsolute := TRUE;     // 绝对坐标
   PTdataX.Number_of_pairs :=36;   // PT 数据点的个数
   PTdataY.IsAbsolute := TRUE;
   PTdataY.Number_of_pairs :=36;
   FOR i := 1 TO 36  BY 1 DO       // 计算 36 个点的位置坐标
        t := 5.0 * i * PI / 180.0;
        p := 100.0 * SIN(3.0*t);
        px := p * COS(t);
        py := p * SIN(t);
        PTdataX.MC_TP_Array[i].delta_time := T#200MS; // 2 点间的时间
        PTdataX.MC_TP_Array[i].position := px;        // 位置坐标
        PTdataY.MC_TP_Array[i].delta_time := T#200MS;
        PTdataY.MC_TP_Array[i].position := py;
   END_FOR;
   flag := FALSE;   // 只计算一次
END_IF;

IF input0 = TRUE THEN     // input0 按键按下
    startPT := TRUE;      // 同时启动 X 轴和 Y 轴的样条插值函数
    donePTx := FALSE;
    donePTy := FALSE;
END_IF;

IF donePTx = TRUE AND donePTy = TRUE THEN
    startPT := FALSE;   // 样条插值函数执行完成后复位
END_IF;
```

执行该程序，结果如图 8.12 所示，与图 8.11 的理论值一样。



8.12　由位置样条插值函数得到的三叶玫瑰线的 X、Y 轴位移曲线

第9章 扩展模块与高速计数器

MC300CS 系列 PLC 本体可以直接拓展 R2 系列模块，又可以通过 Ethercat 总线与 R2EC 耦合器连接，在 R2EC 上拓展 R2 系列模块。PLC 本体或 R2EC 耦合器最大可拓展 32 个 R2 系列模块。

R2 系列模块包括：IO 模块、A/D 模块、D/A 模块等。

9.1 模块添加方式

9.1.1.MC300CS 本机上添加模块

MC300CS 本机上添加模块的方式有两种。

方式一：选中项目树中“本机模块配置”，进入模块配置页。选中空机架槽后，双击右侧工具箱模模块进行添加或者选中右侧工具箱后，进行拖拽添加。如图 9.1 所示。

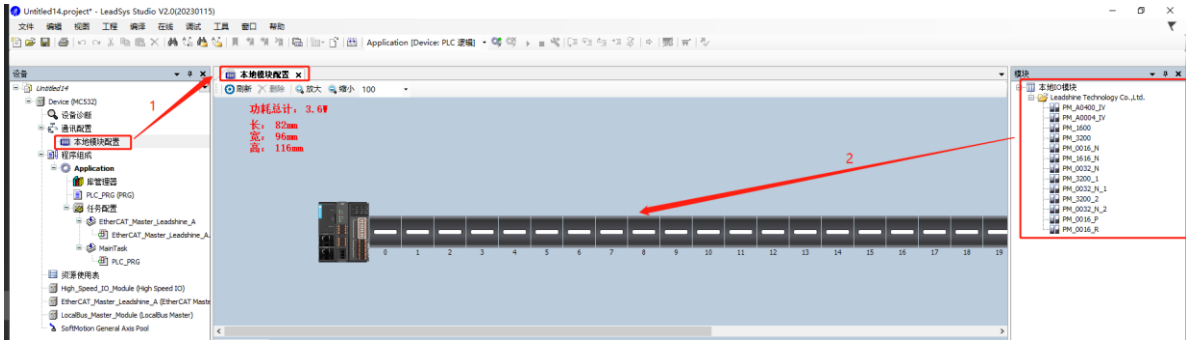


图 9.1 本地模块配置页

方式二：选中项目树中的“LocalBus_Master_Module”，点击鼠标右键。选择“背板扫描”，将设备扫描窗口中对应的设备复制到工程中。如图 9.2 所示。

9.1.2.R2EC 耦合器上添加模块

R2EC 耦合器上添加模块的方式有三种。

方式一：先在项目树中选“EtherCAT_Master_Leadshine_A”，点击右键，弹出列表，选择“添加设备”；点击“Terminal Coupler”，选择“R2EC”，确认添加，如图 9.3 所示。

然后，选中“R2EC”，点击右键，弹出列表，选择“添加设备”后，弹出“添加设备”窗口，选择对应的模块进行添加。如图 9.4 所示。

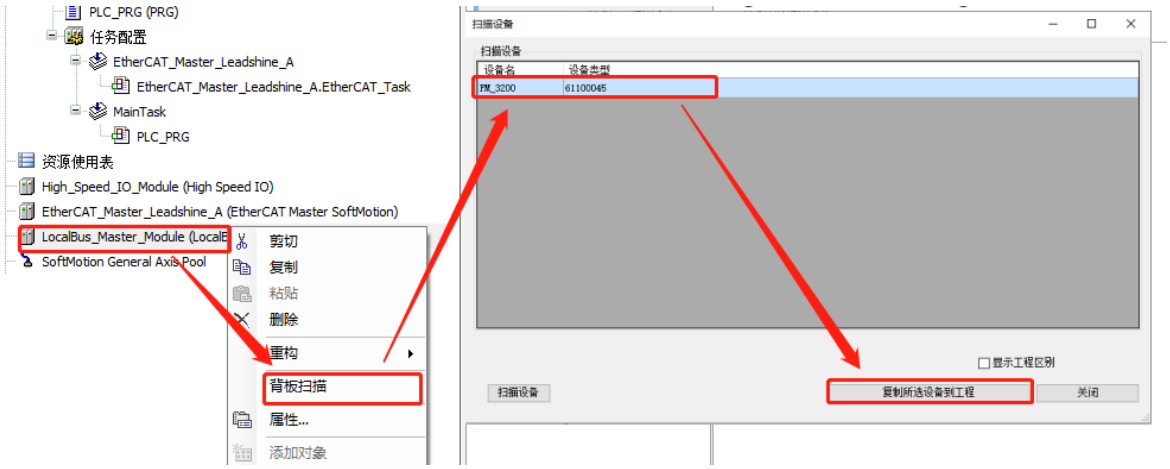


图 9.2 背板扫描方式添加模块

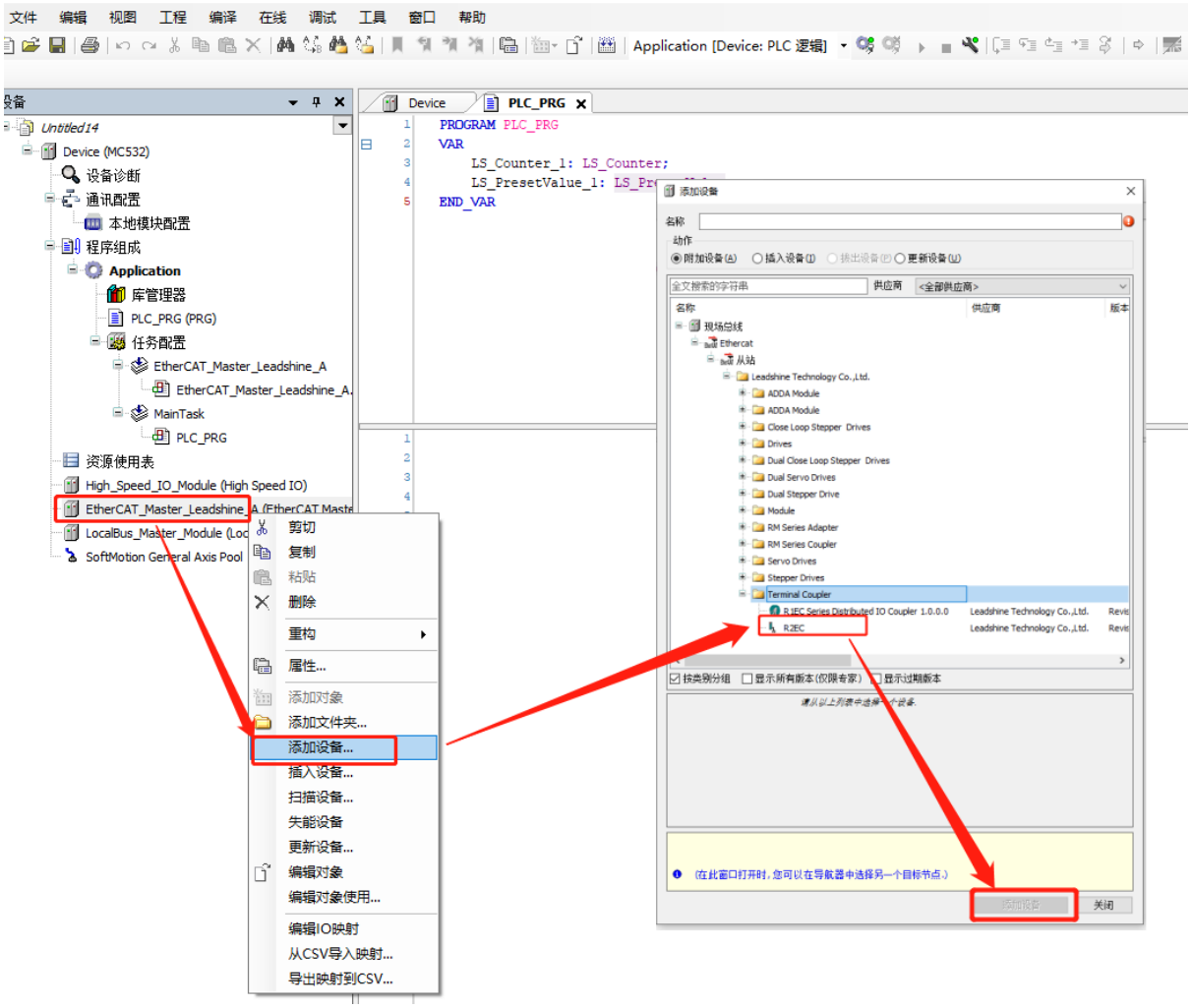


图 9.3　MC300CS 添加 R2EC 模块

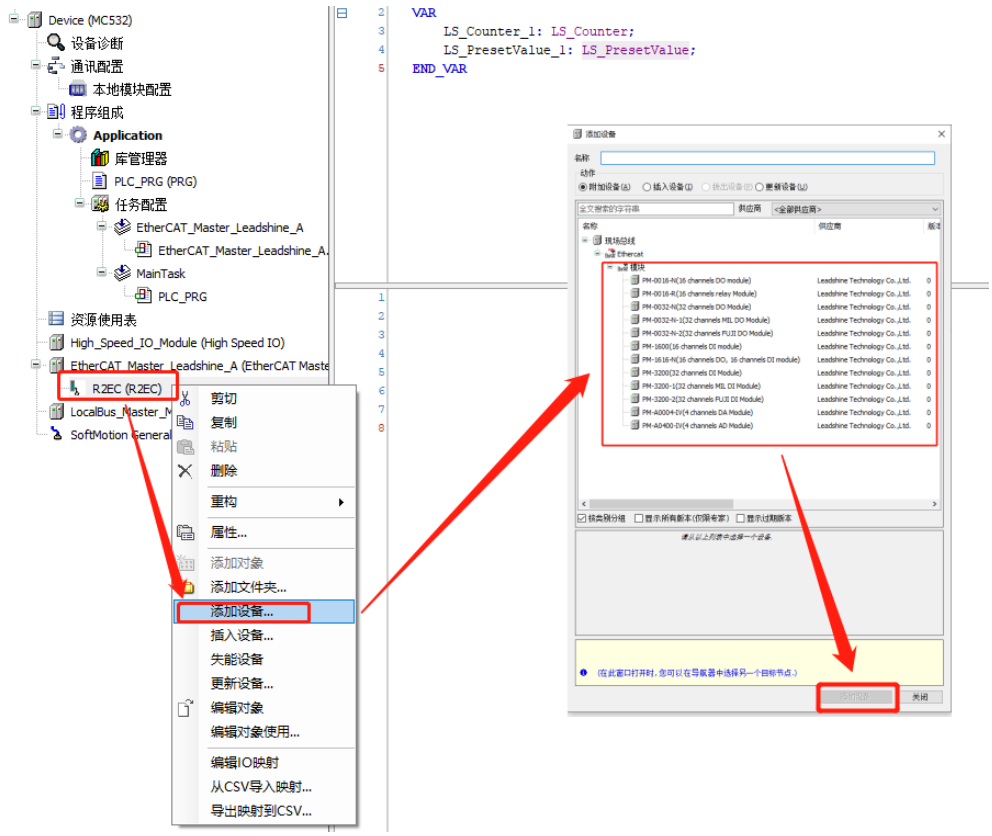


图 9.4　MC300CS+R2EC 添加 R2 模块

方式二：在项目树中选“通讯配置”，鼠标双击进入通讯配置页；选择“EtherCAT A”进入 EtherCAT 通讯配置界面；勾选“EtherCAT 主站”后，选择右侧工具箱“R2EC”进行双击添加或者拖动添加；“R2EC”添加成功后，拖动对应模块或者双击工具箱中的模块进行添加。如图 9.5 所示。

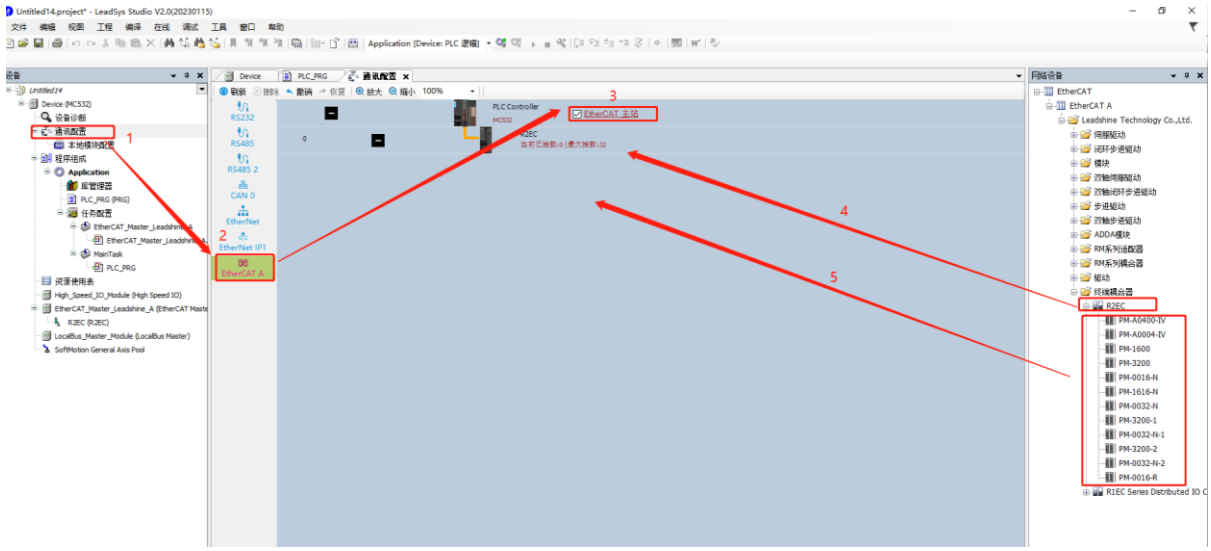


图 9.5　MC300CS+配置 RtherCAT、添加 R2 模块

方式三：当软件处于正常在线状态时，我们可以通过扫描模块的方式进行模块添加，选中“EtherCAT_Master_Leadshine_A”，点击右键，选择“扫描设备”此时弹出设备扫描窗口，选择对应的设备，点击“复制到工程中”。如图 9.6 所示。

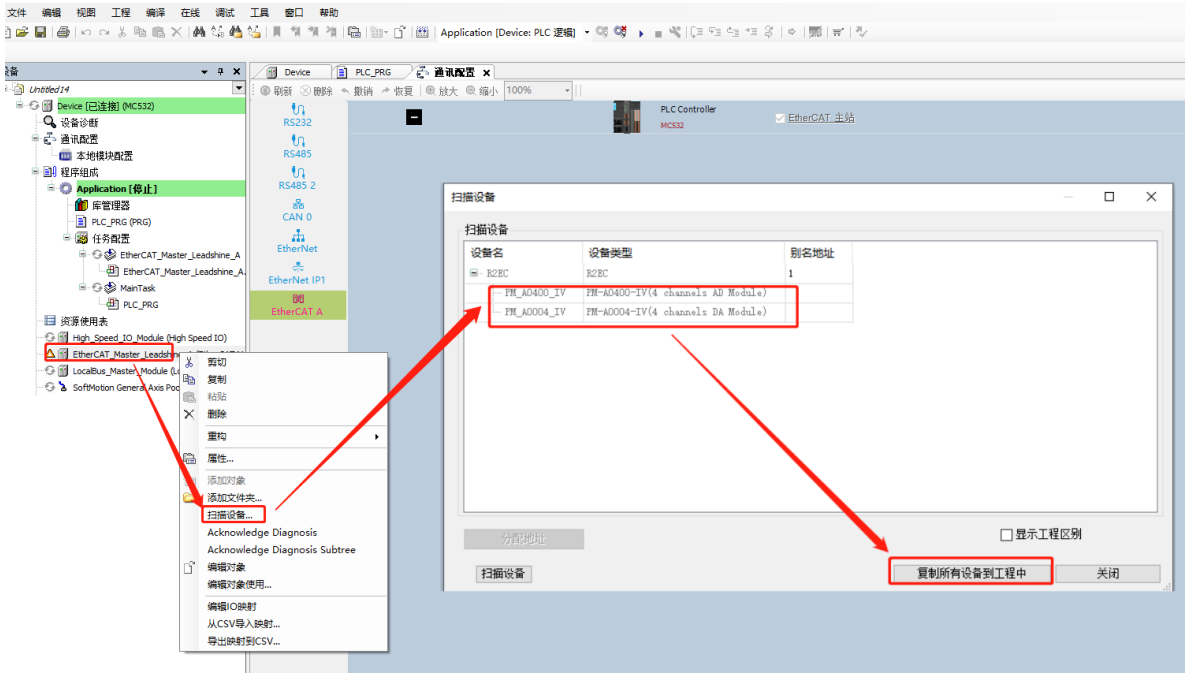


图 9.6　MC300CS+R2EC 扫描模块

9.2　IO 模块

R2 系列 IO 模块有三类：数字量输入模块、数字量输出模块和数字量输入输出模块，详细型号及功能描述见表 9.1。

表 9.1　R2 系列模块型号及描述

模块类型	型号	描述
数字量输入模块	PM-1600	16 路数字量输入模块,源型/漏型输入
	PM-3200	32 路数字量输入模块,源型/漏型输入
	PM-3200-1	32 路数字量输入模块,源型/漏型输入,MIL 接口/40Pin
	PM-3200-2	32 路数字量输入模块,源型/漏型输入,富士通接口/40Pin
数字量输出模块	PM-0016-N	16 路数字量输出模块，漏型输出
	PM-0016-R	16 路数字量输出模块，继电器输出
	PM-0032-N	32 路数字量输出模块，漏型输出
	PM-0032-N-1	32 路数字量输出模块，漏型输出，MIL 接口/40Pin
	PM-0032-N-2	32 路数字量输出模块，漏型输出，富士通接口/40Pin
	PM-0016-P*	16 路数字量输出模块，源型输出
数字量输入输出模块	PM-1616-N	32 路数字量输入输出模块，源型/漏型输入,漏型输出

1.　输入模块

R2 系列输入模块的输入接口等效电路如图 9.7 所示。有漏型输入电路接法和源型输入电路接法。

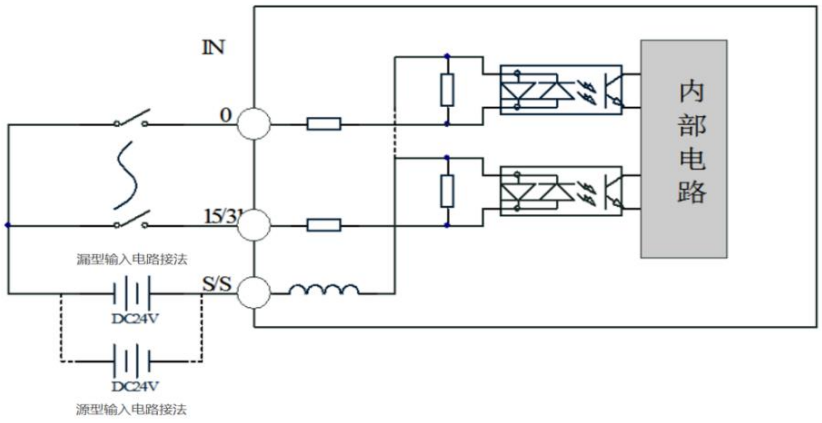


图 9.7　R2 系列输入模块输入口等效电路

2.　输出模块

R2 系列漏型输出模块输出口的等效电路如图 9.8 所示，源型输出模块输出口的等效电路如图 9.9 所示。

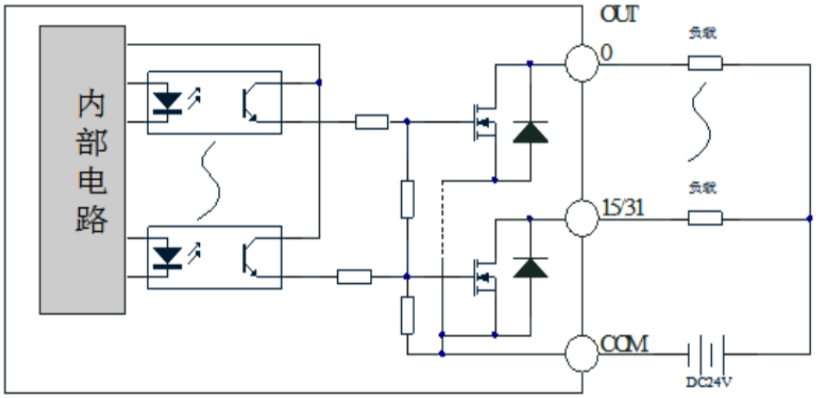


图 9.8　R2 系列漏型输出模块输出口等效电路

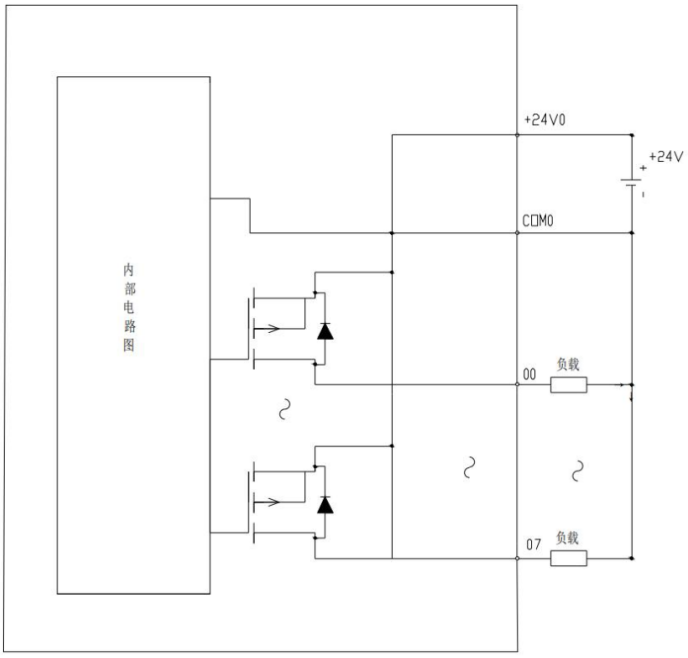


图 9.9　R2 系列源型输出模块输出口等效电路

3. 输入输出模块

R2 系列输入输出模块的输入、输出口的等效电路与图 9.7、图 9.8、图 9.9 相同。

4. 模块上的 IO 口与变量的映射

IO 模块上的 IO 口与变量的映射方法与 PLC 本体上的 IO 口与变量的映射方法相似。

以 PM-1616 模块为例，首先在程序中声明一个 bool 类型的局部变量 out。然后在项目树中选中“本地模块配置”，鼠标左键双击进入本地模块配置页，选中模块机架槽后，选择右侧工具箱中 PM-1616 模块进行添加。如图 9.10 所示。

模块添加完成后，选中添加的模块鼠标双击，进入 PM-1616 模块配置页。选中“Internal/O 映射”选项卡，点击“OUTPUT_0”通道左侧加号，选择“BIT0”一行中的变量区进行双击，点击 ... ，弹出输入助手，在 Application→PLC_PRG 中选中变量 out，点确定，即完成 IO 口与变量的绑定。如图 9.11 所示。

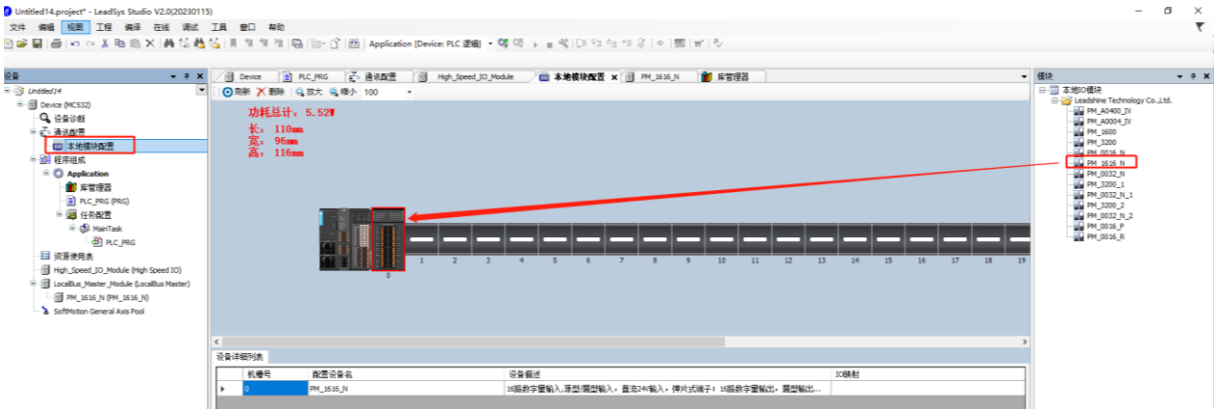


图 9.10　添加 PM-1616 模块

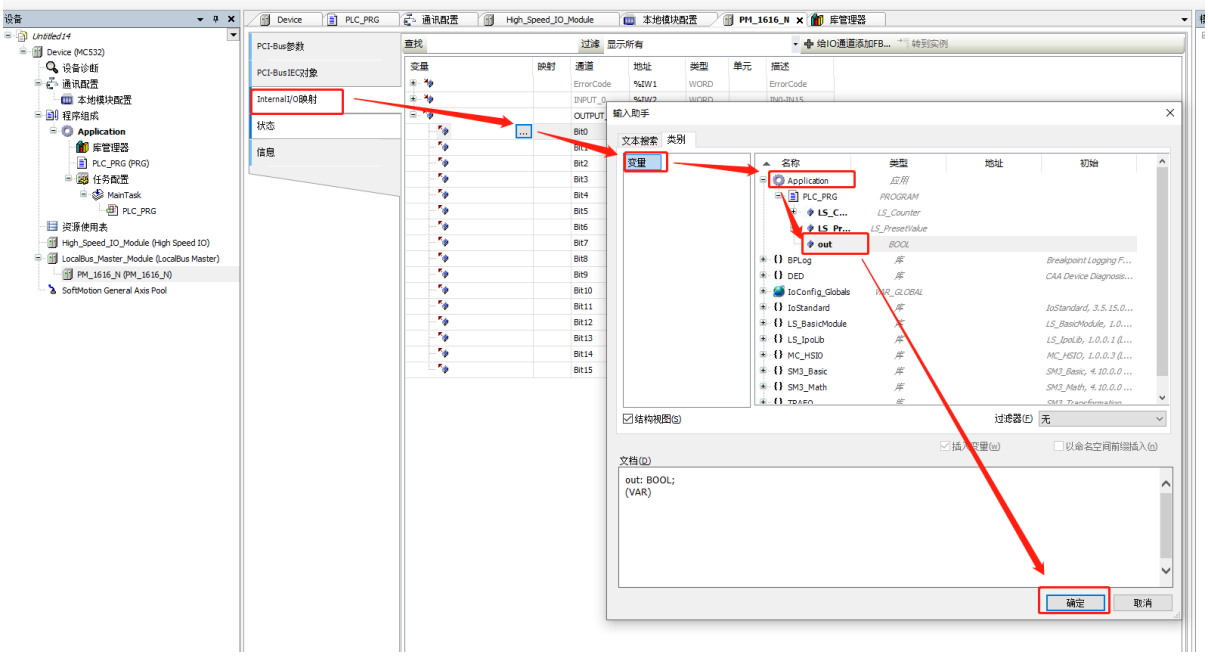


图 9.11　IO 模块添加地址映射

9.3 A/D、D/A 模块

9.3.1 A/D 模块

A/D 模块也称模拟量输入模块，就是将模拟信号转变为数字信号。

MC300CS 搭配 R2 系列中的 PM-A0400-IV 模块，支持 4 路电压/电流输入模式，支持输入电流电压量程为 1V～5V、-5V～5V、0V～10V、−10V～ 10V，4mA～20mA、0mA～20mA；分辨率高达 16 位，4 个通道总响应时间为 1ms；电源支持过流及反接保护。

PM-A0400-IV 模块模拟量输入信号接口原理图如图 9.12 所示。

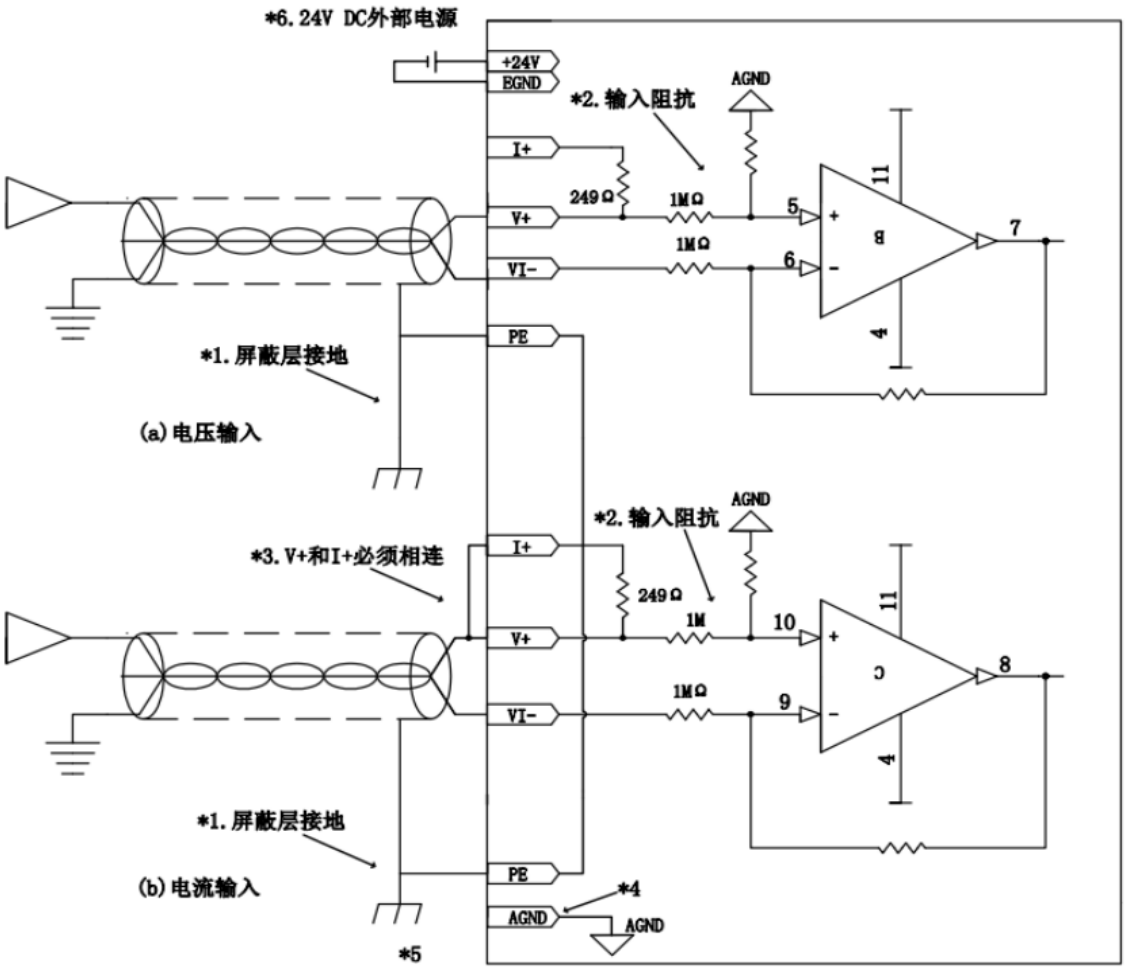


图 9.12　模拟量输入信号接口原理图

注意：

1. 模拟量信号线需采用两芯双绞屏蔽线。
2. A/D 输入阻抗为 1MΩ。
3. 如果以电流信号输入，则必须将 V+和 I+端子连接。
4. 当输入信号为差分信号时，可将“AGND”与兼容设备的模拟地相连接，以消除设备间的共模差，保证模块采样的精度。

5. 模块需要安装在接地良好的金属支架上，并保证模块底部的金属与支架良好接触。
6. 需外接直流 24V 电源。

9.3.2 D/A 模块

D/A 模块也称模拟量输出模块，就是将数字信号转变为模拟信号。

MC300CS 搭配 R2 系列中的 PM-A0004-IV 模块，支持 4 路电压/电流输出模式，输出信号范围可设为 1V～5V、0V～5V、-5V～5V、0V~10V、-10V～10V，0mA～20mA，4mA～20mA；分辨率高达 16 位，4 个通道总响应时间为 1ms；电源支持过流及反接保护。

图 9.13、9.14 分别为模拟量电压输出、模拟量电流输出接口原理图。

电压输出

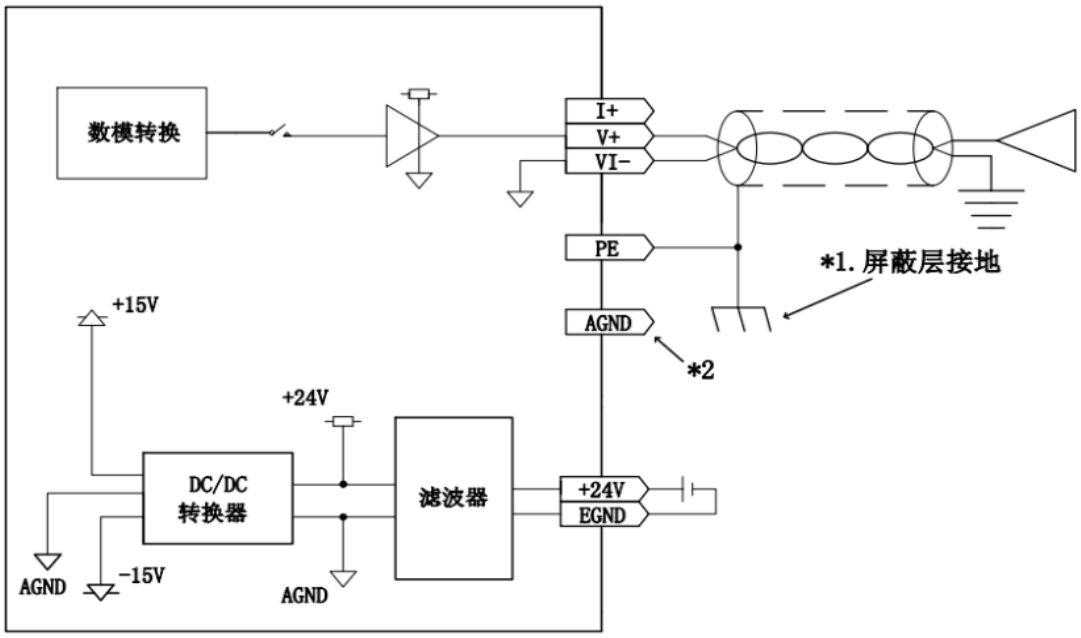


图 9.13　模拟量电压输出接口原理图

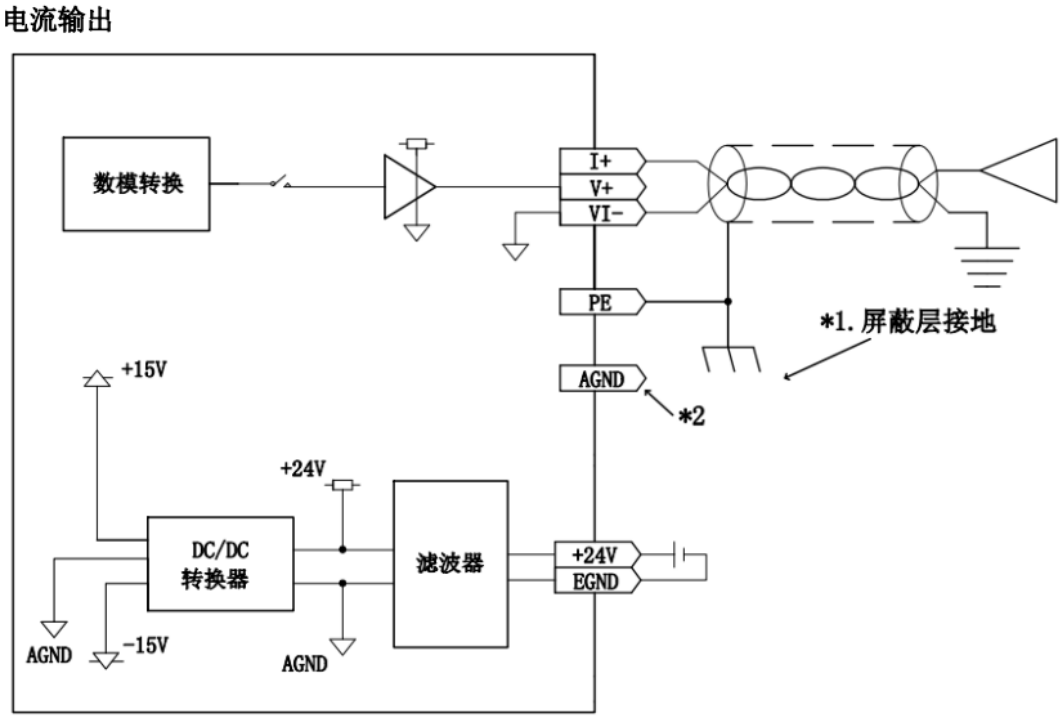


图 9.14　模拟量电流输出接口原理图

注意：

1. 模拟量信号线需采用两芯双绞屏蔽线。
2. 可将“AGND”与兼容设备的模拟地相连接，以消除设备间的共模差，保证模块采样的精度。

9.3.3 A/D、D/A 模块的参数设置

在 MC300CS PLC 上扩展 A/D、D/A 模块后，要设置其参数。先添加设备，然后在项目树中鼠标点击模块名，打开模块参数界面设置相应的参数选择各通道的转模式和信号类型。图 9.15 为 A/D 模块的参数界面。

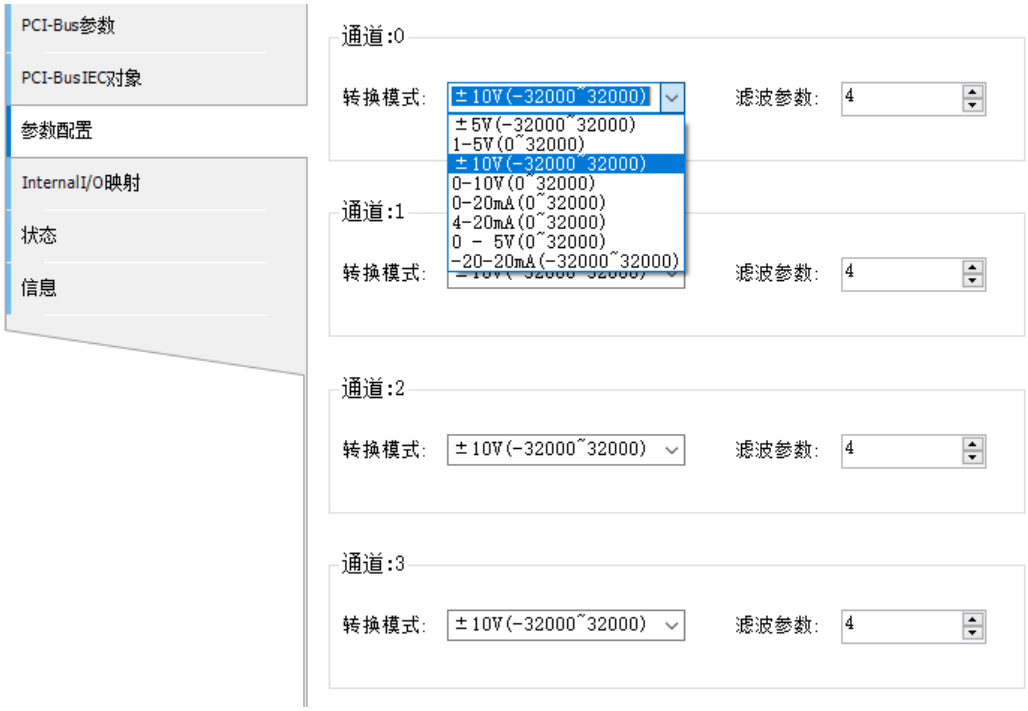


图 9.15　A/D 模块参数设置界面

当 A/D、D/A 模块安装在 R2EC 耦合器上时，需要设置主、从站参数。具体方法见第十章 EtherCAT 总线参数设置及《R2 系列插片式扩展模块用户手册》。

9.3.4 A/D、D/A 模块例程

本例程使用 MC300CS PLC 控制 A/D、D/A 模块，将 D/A 模块输出的电压信号接入到 A/D 模块上；电压信号为 0~10V，并随时间变化；并在 D/A 模块电压信号输出口上接入一个 LED 指示灯，其亮度与 D/A 模块输出电压成正比，呈现呼吸灯效果（由暗到亮，再由亮到暗，反复循环）。

该例程的硬件接线图如图 9.16 所示。

打开 LeadSys Studio 软件，新建工程，添加 A/D、D/A 模块，添加模块方式可参考 9.1.1 小节。

然后在项目树中选中“PM-A0400-IV”，点击“参数配置”选项卡，将通道 0 设为 0-10V(0-32000)，滤波参数为 4。如图 9.17 所示。

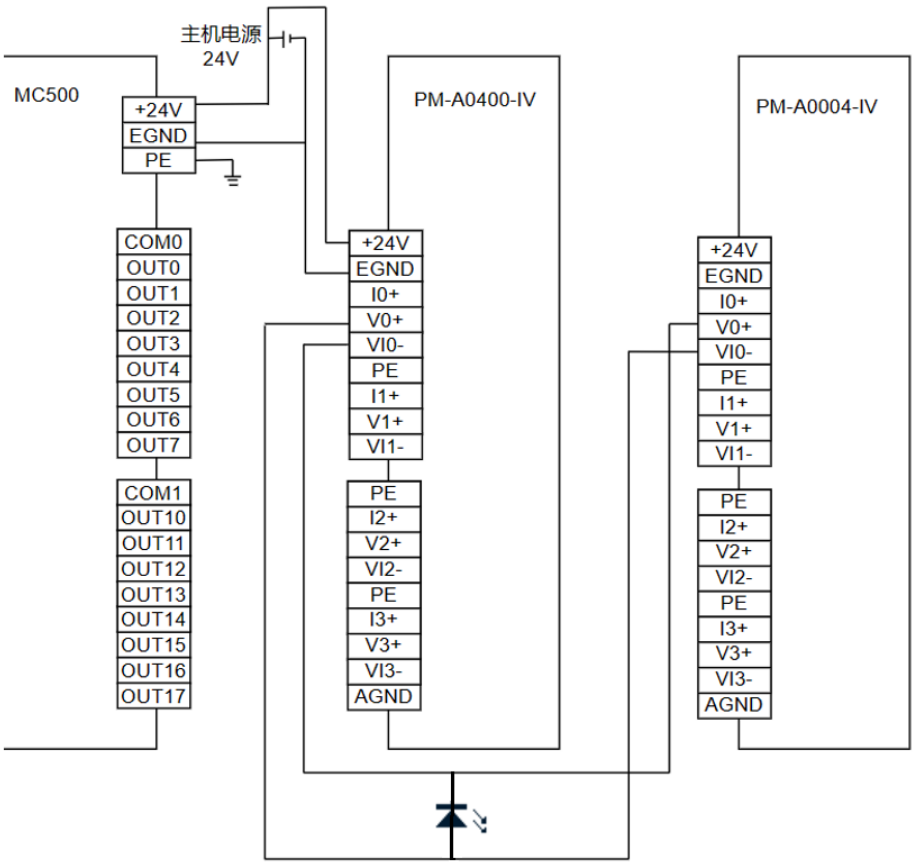


图 9.16　ADDA 模块接线图

图 9.17　PM-A0400-IV 模块通道参数

点击进入“PM-A0004-IV”配置页，选择“参数配置”选项卡后，单击进入。将通道 0 设置为 0-10V（0-32000），“断线后输出状态”为“输出保持”。如图 9.18 所示。

图 9.17、9.18 中的 0-10V“（0-32000）”，含义为：0~10V 对应的数值为 0~32000。

选择“Internal I/O 映射”卡，可以看到 D/A 模块 4 个通道对应的地址，如图 9.19 所示。可以在该界面将通道地址与 D/A 值的变量用手动方式绑定，也可以在变量声明时直接与地址绑定。

A/D 模块 4 个通道对应的地址变量的映射方法与 D/A 模块的相同。

图 9.18　PM-A0004-IV 模块通道参数

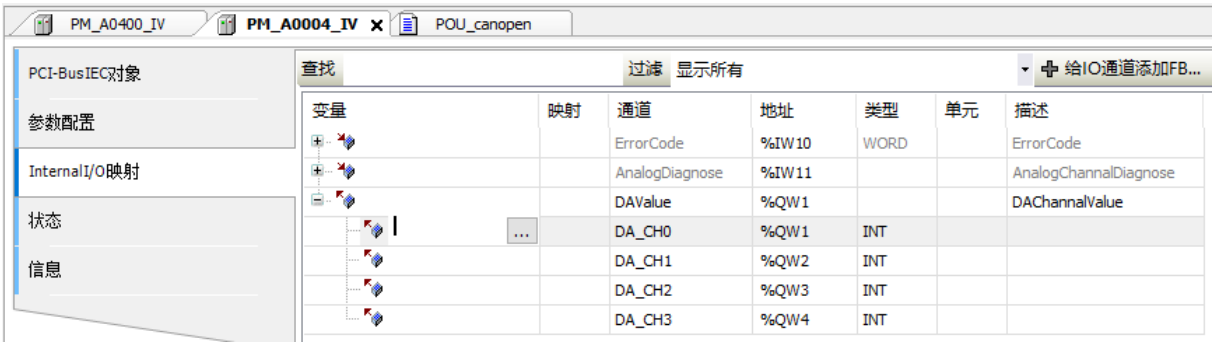


图 9.19　DA 模块输出口通道地址

编写程序前，进行变量声明，代码如下：

```
PROGRAM PLC_PRG
VAR
    DA_CH0_Value AT %QW1:INT:=8000;          // D/A 变量与 D/A 通道 0 地址绑定
    AD_CH0_Value AT %IW2:INT;                // A/D 变量与 A/D 通道 0 地址绑定
    AD_Value:INT;                            // A/D 模块电压值
    TON_0: TON;
    flag: BOOL;
END_VAR
```

程序代码如下：

```
TON_0(IN:=TON_0.Q=FALSE, PT:= T#10MS, Q=> , ET=> );// 定时器 0 每 10ms 启动一次
IF TON_0.Q THEN                                    // 定时器 0 定时完成
    AD_Value:=AD_CH0_Value;                        // 读取 A/D 模块电压值
    IF  flag=FALSE THEN                            // 程序启动时进行标志位判断
        DA_CH0_Value:=DA_CH0_Value+500;            // 上升段：D/A 输出值+500
      ELSE
        DA_CH0_Value:=DA_CH0_Value-500;            // 下降段：D/A 输出值-500
```

```
    END_IF
    IF DA_CH0_Value=32000 OR DA_CH0_Value=0 THEN        // 到达上、下界的判断
        flag:=NOT flag;                                 // 上升下降标志位取反判断
    END_IF
END_IF
```

运行程序，指示灯呈现呼吸灯状态，A/D 模块、D/A 模块电压值变化曲线如图 9.20 所示。

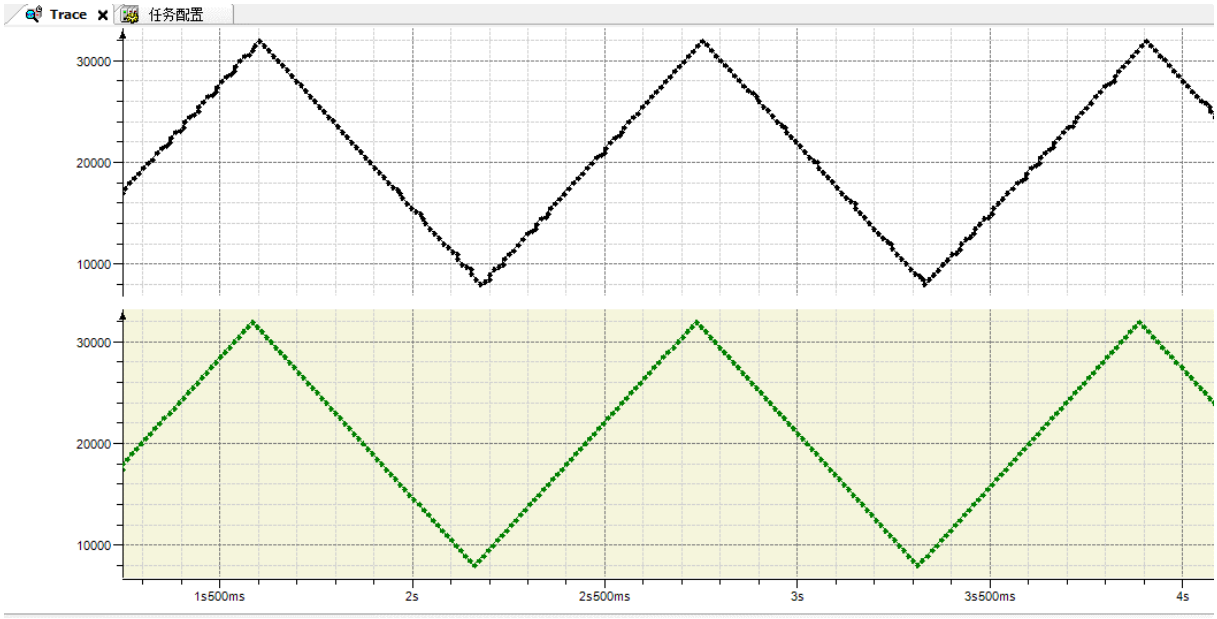


图 9.20　A/D、D/A 模块输入输出电压值

其中：━━ 是 A/D 模块 0 通道的输入电压值。

▬ 是 D/A 模块 0 通道的输出电压值。

9.4 高速计数器

9.4.1 高速输入口的定义

MC300CS PLC 输入口 IN0~IN11 为高速输入口，可复用定义为高速计数器，其输入信号有 4 种类型，如表 9.2 所示。

表 9.2　MC300CS 高速输入口功能说明

端口	信号功能 1	信号功能 2	信号功能 3	信号功能 4	信号功能 5
IN00	通用输入	单相高速计数器 0	AB 相计数器 0 A 相	CW/CCW	脉冲+方向
IN01	通用输入		AB 相计数器 0 B 相	CW/CCW	脉冲+方向
IN02	通用输入	单相高速计数器 1	AB 相计数器 1 A 相	CW/CCW	脉冲+方向
IN03	通用输入		AB 相计数器 1 B 相	CW/CCW	脉冲+方向

IN04	通用输入	单相高速计数器 2	AB 相计数器 2 A 相	CW/CCW	脉冲+方向
IN05	通用输入		AB 相计数器 2 B 相	CW/CCW	脉冲+方向
IN06	通用输入	单相高速计数器 3	AB 相计数器 3 A 相	CW/CCW	脉冲+方向
IN07	通用输入		AB 相计数器 3 B 相	CW/CCW	脉冲+方向
IN08	通用输入	单相高速计数器 4	AB 相计数器 4 A 相	CW/CCW	脉冲+方向
IN09	通用输入		AB 相计数器 4 B 相	CW/CCW	脉冲+方向
IN10	通用输入	单相高速计数器 5	AB 相计数器 5 A 相	CW/CCW	脉冲+方向
IN11	通用输入		AB 相计数器 5 B 相	CW/CCW	脉冲+方向

高速计数器与编码器单端连接。图 9.21 为 MC300CS 与欧姆龙 E6C2-C 增量式编码器的接线图。

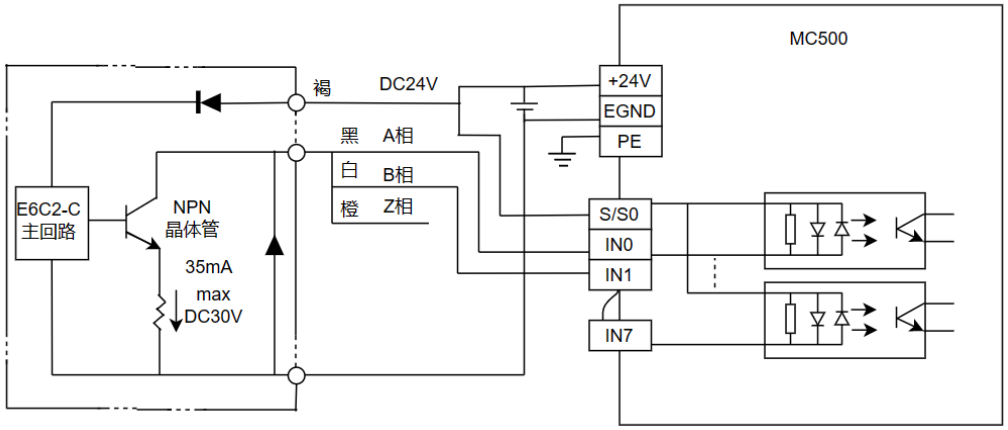


图 9.21　MC300CS 与欧姆龙 E6C2-C 增量式编码器的接线图

9.4.2 高速输入口参数设置

启用高速计数器时，需要对高速计数器进行参数设置，如图 9.22 所示。

图 9.22　高速计数器参数配置

图 9.22 中各参数说明如表 9.3 所示。

表 9.3 高数计数器参数说明

选项名称	参数设置
工作模式	脉冲+方向：接收编码器脉冲信号、方向信号。 单相计数：接收编码器脉冲信号。 CW/CCW：接收编码器双脉冲信号。 A/B 相：接收编码器 AB 相脉冲信号， 可分为 1 倍频、2 倍频、4 倍频等工作模式
计数方向	正向计数、反向计数两种计数方式
计数模式	线性计数：线性计数在最大值和最小值之间计数，当正向计数达到最大值或者反向计数达到最小值时停止计数，溢出标志有效 环形计数：环形计数在最大值和最小值之间计数，当正向计数超过最大值之后转到最小值，当反向计数时小于最小值则跳到最大值
上限值	设置计数值的上限，范围-2147483548~2147483647
下限值	设置计数值的下限，范围-2147483548~2147483647

9.4.3 高速计数器指令及其功能

高速计数器主要用到的指令如表 9.4 所示。

表 9.4 高速计数器指令表

高速计数器指令	指令名称	说明
LS_Counter	计数指令	计数器/编码器计数功能
LS_PresetValue	预设值指令	计数器的预置值功能

计数器/编码器计数指令格式为：

```
LS_Counter( xEnable:= 启动信号, eChannel:= 计数器通道, xValid=>完成信号标志,
            xError=>错误信号标志, eErrorID=>错误代码, diValue=>监测通道计数值);
```

其中：

eChannel：设置计数器通道，0-5 为计数器，6-8 为编码器。

计数器预置值指令格式为：

```
LS_PresetValue( xExecute:=启动信号, eChannel:=计数器通道, eTriggerEdge:=暂不支持,
                diPresetValue:=计数器预设值, xDone=>完成信号标志, xBusy=>进行中标志,
                xError=>错误信号 ,eErrorID=> 错误代码);
```

其中：

eChannel：设置计数器通道，0-5 为计数器，6-8 为编码器

9.4.4 高速计数器例程

本例程用高数计数器记录脉冲轴输出脉冲数。

1、搭建硬件系统

高速计数器口与高速脉冲输出口接线图如图 9.23 所示。

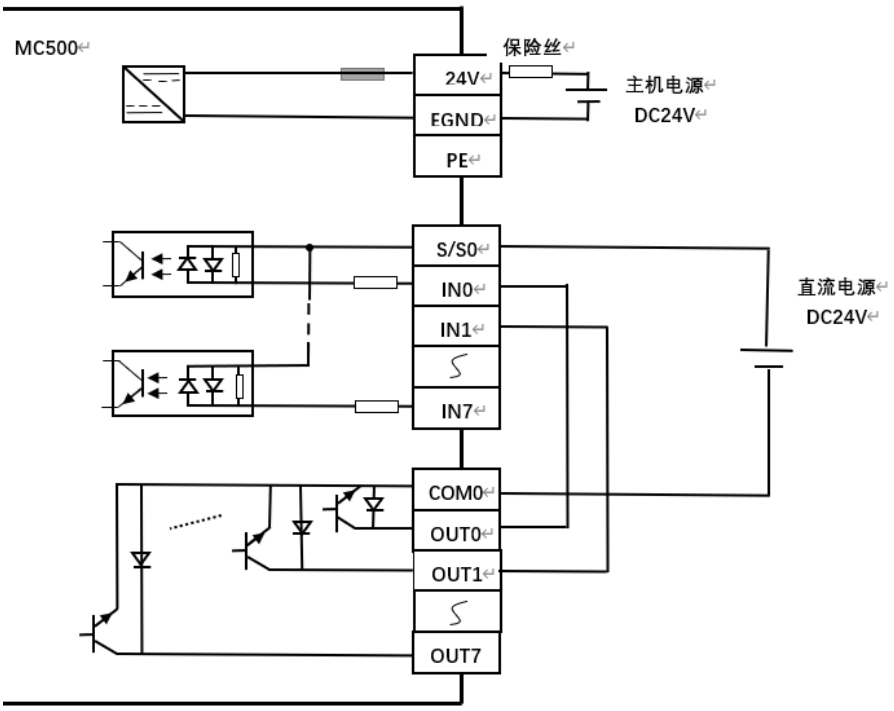


图 9.23　高速计数器硬件接线图

2、编写软件

打开 LeadSys Studio 软件，新建工程后，在左侧项目树中选中“High_Speed_IO_Module”后，双击进入配置页。

由于例程中需要使用到脉冲轴，需要启用脉冲轴，具体参考可 5.2 脉冲轴点位运动例程添加脉冲轴。

在“高速输入设置”下，选择“计数器 0”；再点击“启用”，使方框内出现“√”；若未点击“启用”，不能设置该界面参数。如图 9.24 所示。

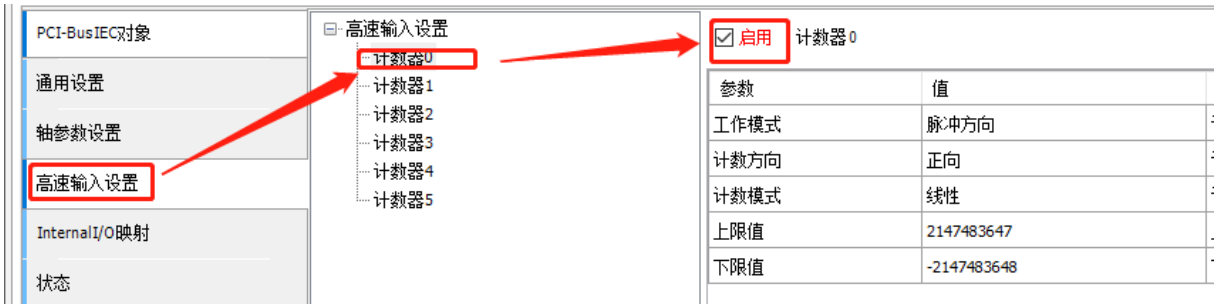


图 9.24　启用高速计数器

设置计数器工作模式：鼠标点击“A/B 相 4 倍频”右侧，出现如图 9.25 所示下拉菜

单，在其中选择所需的计数模式。本例选择“脉冲+方向”计数。计数方向选择“正向”，计数模式为“线性”。

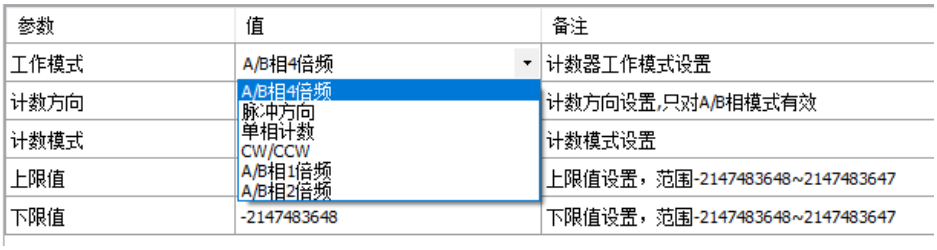


图 9.25　高速计数器工作模式

程序代码如下。

```
PROGRAM PLC_PRG
VAR
    Axis:DUT_Pulse_Axis;                                  //脉冲轴特定结构体
    MC_Power_1: MC_Power;                                 //实例化使能指令
    LS_Counter_1: LS_Counter;                             //实例化高速计数器指令
    VALUE: DINT;                                          //显示计数值
    MC_MoveRelative_1: MC_MoveRelative;                   //实例化相对点位运动指令
    MC_ReadActualPosition_1: MC_ReadActualPosition;       //实例化读取实际位置指令
    Position: LREAL;                                      //显示轴实际位置
    MC_SetPosition_1: MC_SetPosition;                     //实例化设置轴位置指令
END_VAR

Axis.PulAxis_0:=ADR(LS_Axis_0);                                          //取脉冲轴地址
LS_MotionControl_P( stAxis:=Axis , xClearErr:= , fLimtAxisSpeedJump:= , xDone=> ,
                    xError=> , eErrorID=> , xLimitAxisMoveFlag=> ); //初始化脉冲轴
MC_Power_1( Axis:=LS_Axis_0 , Enable:=TRUE , bRegulatorOn:=TRUE , bDriveStart:=TRUE ,
            Status=> , bRegulatorRealState=> , bDriveStartRealState=> , Busy=> ,
            Error=> , ErrorID=> );                                        //使能脉冲轴
MC_SetPosition_1(Axis:=LS_Axis_0 , Execute:=MC_SetPosition_1.Execute , Position:=0 ,
                 Mode:= , Done=> , Busy=> , Error=> , ErrorID=> ); //设置脉冲轴位置
LS_Counter_1( xEnable:=TRUE , eChannel:=0 , xValid=> , xError=> , eErrorID=> ,
              diValue=>VALUE );                                          //计数脉冲个数
MC_ReadActualPosition_1(Axis:=LS_Axis_0 , Enable:=TRUE , Valid=> , Busy=> ,
                        Error=> , ErrorID=> , Position=>Position ); //监控脉冲轴
//点位运动：
MC_MoveRelative_1(  Axis:=LS_Axis_0 , Execute:=MC_MoveRelative_1.Execute ,
                    Distance:=50 , Velocity:=100 , Acceleration:=1000 ,
                    Deceleration:=1000 , Jerk:= , BufferMode:= , Done=> , Busy=> ,
                    Active=> , CommandAborted=> , Error=> , ErrorID=> );
```

当 MC_MoveRelative_1.Execute=TRUE 时，高数计数器能够正常读取到脉冲轴发出脉冲数。如图 9.26 所示，第 36 行 LS_Counter_1 的输出值 VALUE=50，与点位运动指令的运动距离相同。

```
28        Error=> ,
29        ErrorID=> );
30    LS_Counter_1(                    //计数脉冲个数
31        xEnable TRUE :=TRUE ,
32        eChannel Counter0 :=0 ,
33        xValid=> ,
34        xError=> ,
35        eErrorID=> ,
36        diValue 50 =>VALUE 50 );
37    MC_ReadActualPosition_1(     //监控脉冲轴
38        Axis:=LS_Axis_0 ,
39        Enable TRUE :=TRUE ,
40        Valid=> ,
41        Busy=> ,
42        Error=> ,
43        ErrorID=> ,
44        Position 50 =>Position 50 );
45    MC_MoveRelative_1(           //点位运动
46        Axis:=LS_Axis_0 ,
47        Execute TRUE :=MC_MoveRelative_1.Execute TRUE ,
48        Distance 50 :=50 ,
49        Velocity 100 :=100 ,
50        Acceleration 1E+03 :=1000 ,
51        Deceleration 1E+03 :=1000 ,
52        Jerk:= ,
```

图 9.26　监控高速计数器计数

9.5 探针功能及应用

MC300 系列 PLC 不支持探针功能。

9.6 高速比较输出

MC300 系列 PLC 不支持高速比较输出功能。

第10章 EtherCAT 总线通讯

EtherCAT 通讯是一种基于实时工业以太网技术的现场总线通讯。EtherCAT 通讯系统由一个主站设备和多个从站设备组成。与其他总线通讯相比，EtherCAT 具有延时低、同步精度高、拓扑结构灵活、应用容易、成本低等特点。

EtherCAT 总线通过分布式时钟同步从站设备，保证主站与从站的单轴运动或多轴协同运动的驱动器实现精准同步。其中分布式时钟的周期越小，同步的周期时间就越短，通讯实时性就更高；但是在通讯数据量大时会导致通讯数据丢失，出现通讯错误。

10.1. EtherCAT 主站参数设置

MC300CS 系列 PLC 能配置为 EtherCAT 主站，可以与伺服电机驱动器、步进电机驱动器、耦合器模块等 EtherCAT 从站建立通讯。建议勾选“自动配置主站/从站”选项，使用默认配置进行通讯。其中关键参数说明如下。

1. 分布式时钟

分布式时钟可以使 EtherCAT 通讯的从站设备都同步于参考时钟，实现各从站设备之间的准确同步控制。参考时钟来源于 EtherCAT 主站连接的第一个且具有分布时钟功能的从站，将其分布式时钟作为参考时钟。如图 10.1 所示。

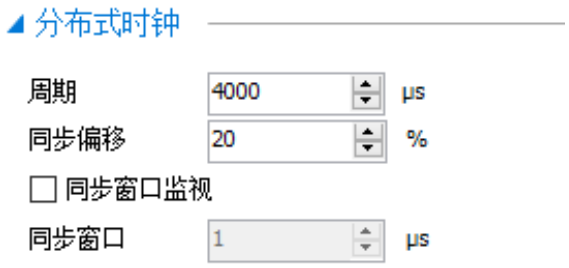


图 10.1 分布时钟参数设置

周期：即 EtherCAT 数据帧发送周期，该周期时间长度与 EtherCAT 任务配置的间隔时间相同。周期越小，通讯实时性更高。周期最小值是 500 微秒。

同步偏移：PLC 任务周期相对于从站的定时器中断的相对偏移的百分比，其范围是 0~50%，默认值为 20%。

2. 选项

如图 10.2 所示，如选中“自动重启从站”项，主站会在通讯异常后自动尝试重启从站。

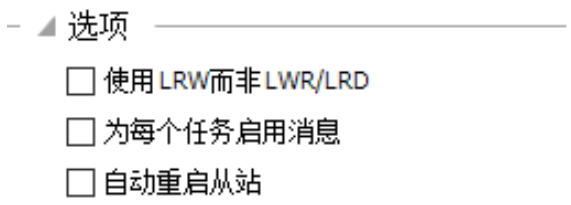


图 10.2　选项参数设置

10.2. 添加 EtherCAT 从站

1. 添加 LeadSys 软件自带设备

添加 EtherCAT 从站设备有两种方式，具体操作如下。

方式一：

选择项目树中的“EtherCAT_Master_Leadshine_A”后，鼠标点击右键。选择“扫描设备”，将扫描到的设备复制到工程中。如图 10.3 所示。通过扫描添加 EtherCAT 从站时，首先确保 PLC 连接的实际从站设备正常通讯，只有处于正常通讯的情况下，才能扫描到从站设备。

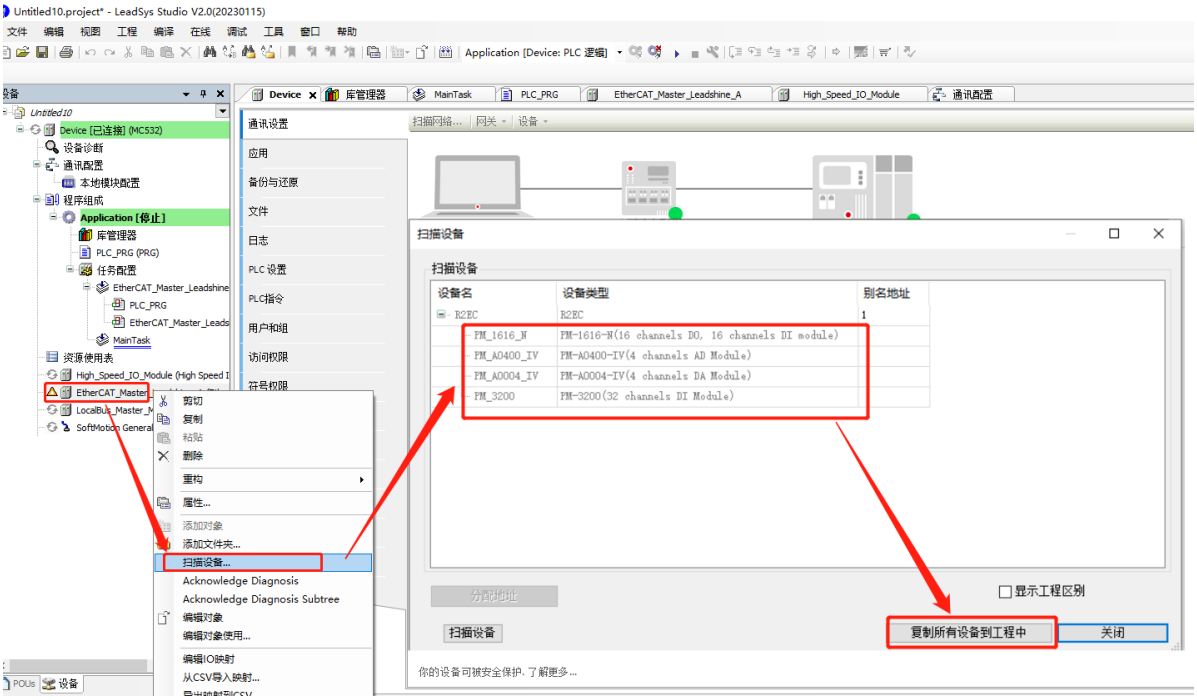


图 10.3　扫描添加

方式二：

鼠标双击“通讯配置”配置页，选择“EtherCAT”选项卡，勾选“EtherCAT 主站”，双击右侧工具箱设备或者将工具箱设备拖拽到 PLC 本体下方，进行设备添加。如图 10.4 所示。

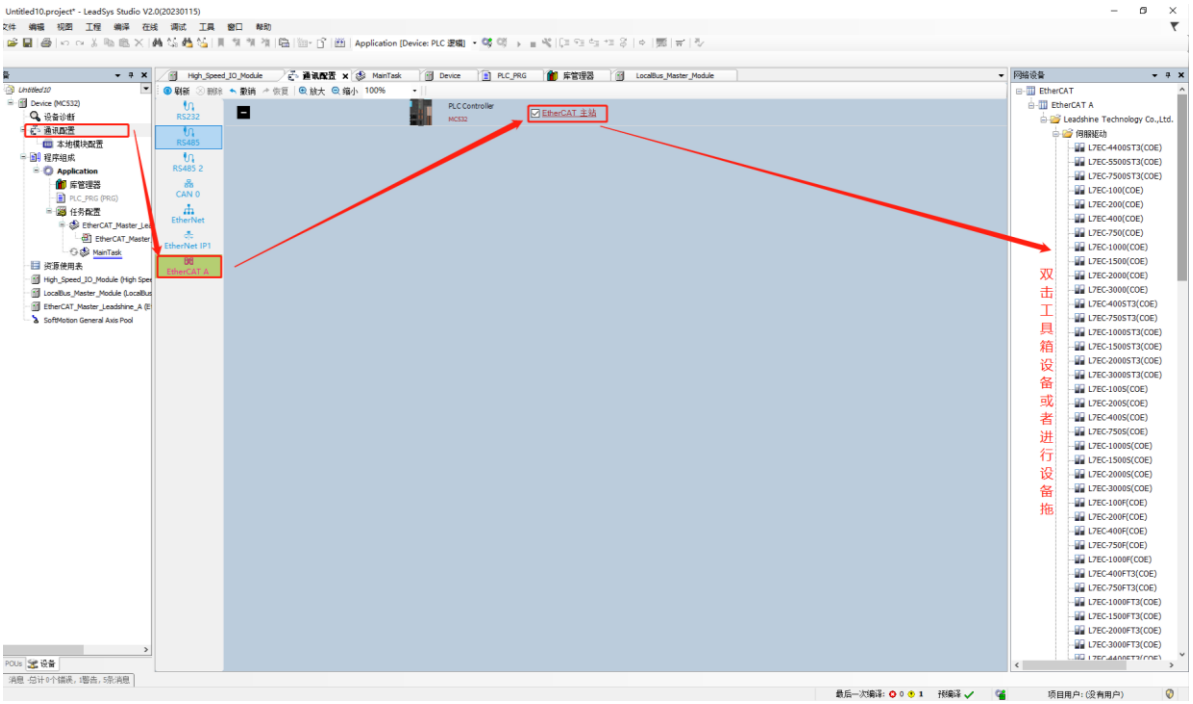


图 10.4　手动添加

2. 添加 LeadSys 软件未包含设备

用户在使用尚未添加设备描述文件的 EtherCAT 从站时，需要自行将对应的描述文件添加到软件中，以添加松下伺服驱动器描述文件为例，具体的操作步骤如下。

打开 LeadSys Studio，选择菜单栏中的“工具”选项卡，在二级菜单中选择“设备储存库”，如图 10.5 所示。

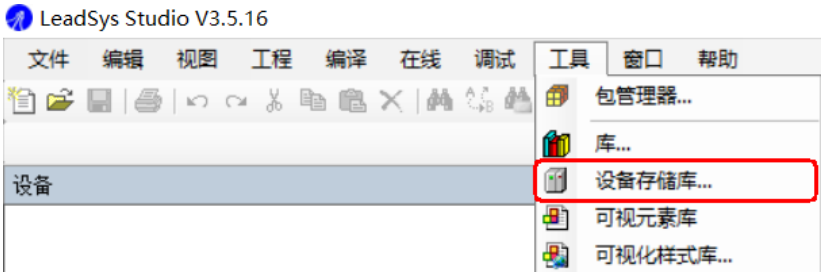


图 10.5　选择“设备存储库”

在弹出的窗口中点击“安装”按钮，如图 10.6 所示。

在弹出窗口中找到存放设备描述文件的路径。

注意：此次要将文件类型选择为 EtherCAT XML 设备描述配置文件，如果选择为其他文件类型，会导致无法正常选取 XML 文件，找到对应的 XML 后，点击“打开”按钮。如图 10.7 所示。

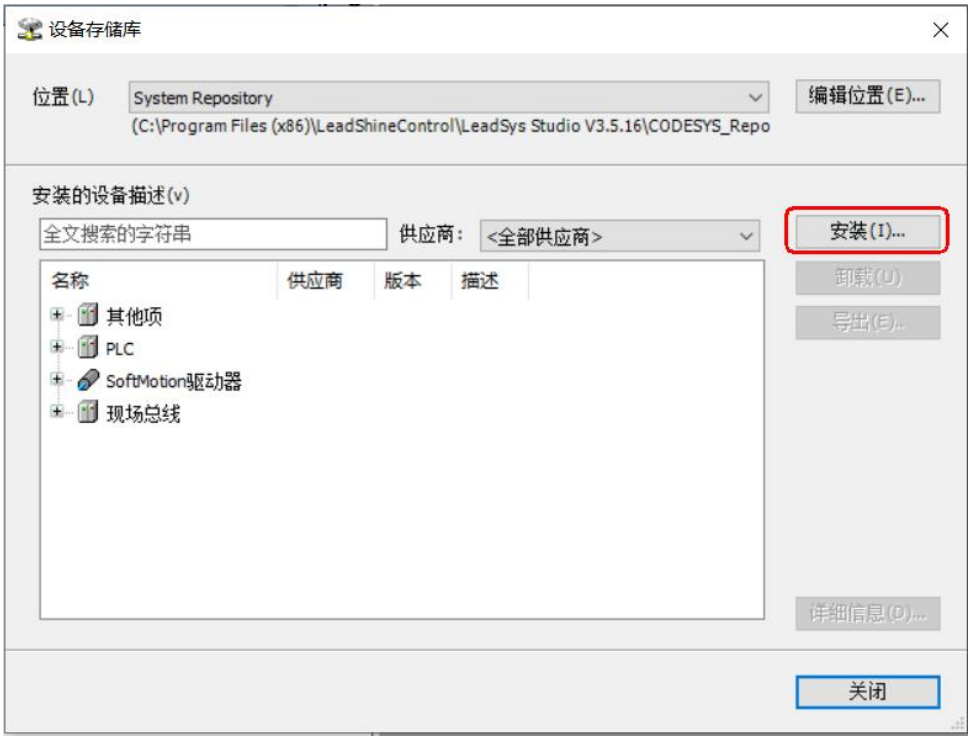


图 10.6　安装“设备存储库”

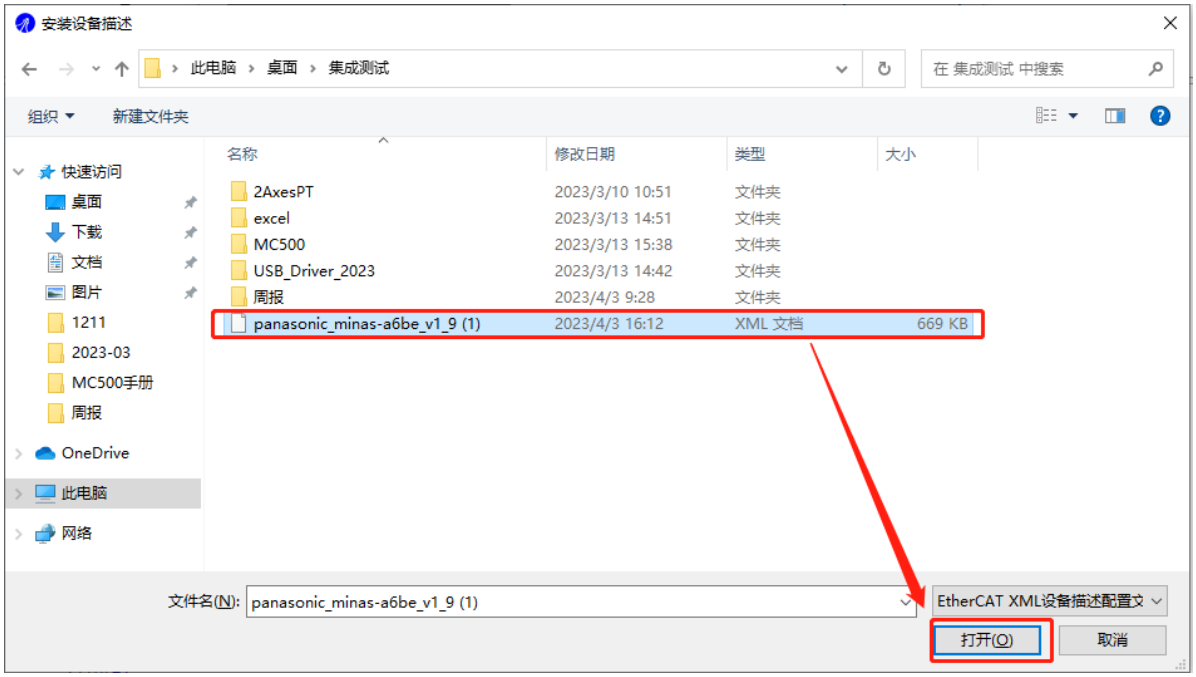


图 10.7　选择“设备描述文件”的路径

打开文件之后，软件会自动导入设备，此时可观察信息输出框，确认安装完成后，点击“关闭”按钮即可。如图 10.8 所示。

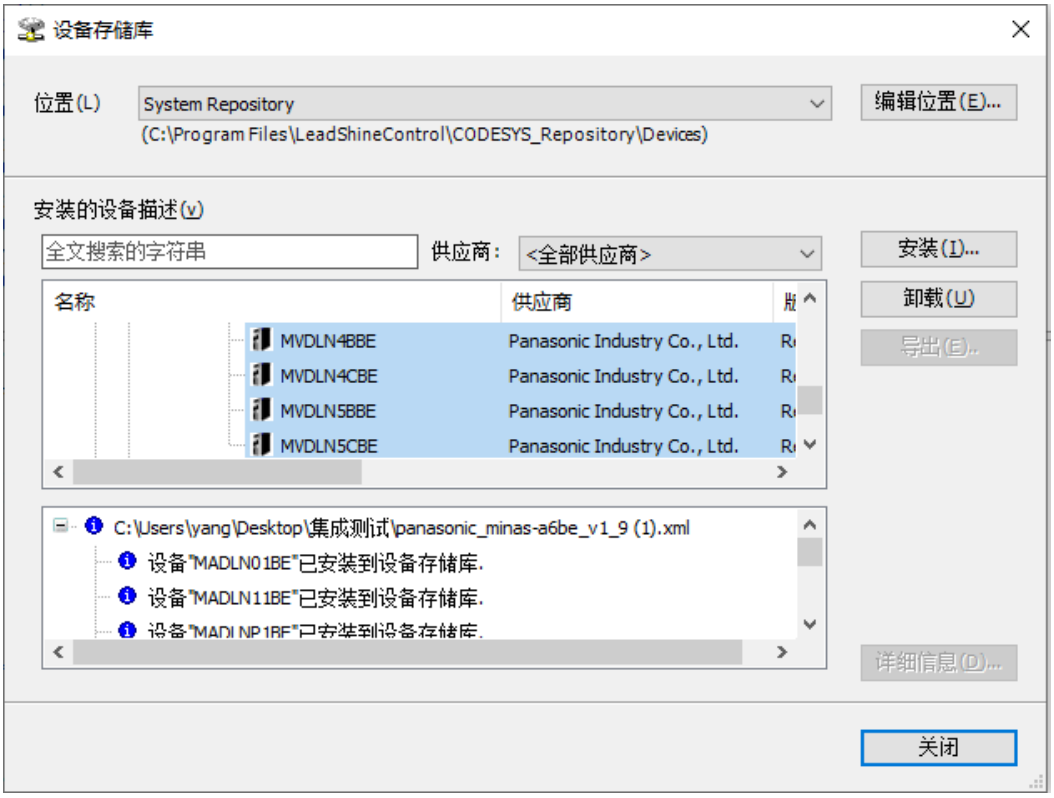


图 10.8 将“新设备描述文件”导入设备

10.3. PDO、SDO 传输方式及例程

EtherCAT 总线通讯中主站与从站进行数据交互的方式主要是过程数据对象（PDO）和服务数据对象（SDO）方式。

PDO 是用于传输周期性的实时数据，可分为传送过程数据（TXPDO）和接收过程数据（RXPDO）;其中 TXPDO 用于发送数据给其它从站,RXPDO 用于接收其它从站发送的数据。

SDO 主要用于传输实时性要求不高的数据，一般用来写入或读取从站的配置参数。

MC300CS 系列 PLC 的 EtherCAT 通讯数据操作指令如表 10.1 所示。详细说明请参考《雷赛大中型 PLC 指令手册》。

表 10.1 EtherCAT 通讯数据操作指令列表

指令类别	名称	功能
EtherCAT 通讯数据操作指令	ETC_CO_SdoReadDWord	以 SDO 方式读取从站地址(Dword)
	ETC_CO_SdoRead4	以 SDO 方式读取从站地址 (4 个 BYTE)
	ETC_CO_SdoRead	以 SDO 方式读取从站地址(大于 4 个 BYTE)
	ETC_CO_SdoWriteDword	以 SDO 方式写从站地址(Dword)
	ETC_CO_SdoWrite4	以 SDO 方式写从站地址(4 个 BYTE)
	ETC_CO_SdoWrite	以 SDO 方式写从站地址(大于 4 个 BYTE)

1. ETC_CO_SdoReadDWord 指令格式

```
ETC_CO_SdoReadDWord(xExecute:=启动信号,xAbort:=终止信号,usiCom:=EtherCAT 主站个数,
                    uiDevice:= 从站 EtherCAT 地址, usiChannel:=保留,
                    wIndex:=主索引地址, bySubindex:=子索引地址, udiTimeOut:=超时时间,
                    xDone=>完成信号, xBusy=>执行中标志, xError=>错误标志,
                    eError=>错误码,udiSdoAbort=>终止标志,dwData=>读取的数据值 ,
                    usiDataLength=>读取的数据长度);
```

2. ETC_CO_SdoRead4 指令格式

```
ETC_CO_SdoRead4(xExecute:=启动信号,Abort:=终止信号, usiCom:= EtherCAT 主站个数,
                uiDevice:= 从站 EtherCAT 地址, usiChannel:= 保留, wIndex:= 主索引地址,
                bySubindex:= 子索引地址, udiTimeOut:= 超时时间, xDone=>完成信号, xBusy=>
                执行中标志, xError=>错误标志, eError=>错误码,
                udiSdoAbort=>终止标志, abyData=>读取的数据值,
                usiDataLength=>读取的数据长度);
```

3. ETC_CO_SdoRead 指令格式

```
ETC_CO_SdoRead(xExecute:= 启动信号, xAbort:= 终止信号, usiCom:= EtherCAT 主站个数,
               uiDevice:= 从站 EtherCAT 地址, usiChannel:= 保留, wIndex:= 主索引地址,
               bySubindex:= 子索引地址,udiTimeOut:= 超时时间,
               pBuffer:= 数据缓冲区的指针, szSize:= 数据缓冲区大小,
                xDone=>完成信号, xBusy=>执行中标志, xError=>错误标志, eError=>错误码,
                udiSdoAbort=>终止信号, szDataRead=>完成读取的数据个数);
```

4. ETC_CO_SdoWriteDWord 指令格式

```
ETC_CO_SdoWriteDWord(xExecute:= 启动信号,xAbort:= 终止信号, usiCom:= EtherCAT 主站个数,
                     uiDevice:= 从站 EtherCAT 地址, usiChannel:= 保留 ,
                     wIndex:= 主索引地址, bySubindex:= 子索引地址,
                      udiTimeOut:= 超时时间, dwData:= 写入数据值,
                      usiDataLength:= 写入的数据长度, xDone=>完成信号,
                      xBusy=>执行中标志, xError=>错误信号,
                      eError=>错误码, udiSdoAbort=>终止标志 );
```

5. ETC_CO_SdoWrite4 指令格式

```
ETC_CO_SdoWrite4( xExecute:=启动信号,xAbort:=终止信号,usiCom:=EtherCAT 主站个数,
                  uiDevice:= 从站 EtherCAT 地址,usiChannel:= 保留,wIndex:= 主索引地址,
                  bySubindex:= 子索引地址 , udiTimeOut:=超时时间 ,
                  abyData:= 写入数据值, usiDataLength:= 写入的数据长度,
                  xDone=>完成信号 , xBusy=>执行中标志 , xError=>错误信号 ,
                  eError=> 错误码, udiSdoAbort=> 终止标志);
```

6. ETC_CO_SdoWrite 指令格式

```
ETC_CO_SdoWrite(xExecute:= 启动信号, xAbort:= 终止信号, usiCom:= EtherCAT 主站个数,
                uiDevice:= 从站 EtherCAT 地址, usiChannel:= 保留, wIndex:= 主索引地址,
                bySubindex:= 子索引地址, udiTimeOut:= 超时时间,
                pBuffer:= 数据缓冲区的指针, szSize:=数据缓冲区大小,
                eMode:= 写入模式,xDone=>完成信号,xBusy=>执行中标志,
                xError=> 错误信号, eError=>错误码, udiSdoAbort=>终止标志,
                szDataWritten=>完成写入的数据长度);
```

本例程以 MC332 与 EtherCAT 总线型伺服驱动器通讯为例，通过指令方式读、写驱动器每转指令脉冲数的参数，该参数主索引地址为 16#2008。其中每转指令脉冲数的参数值采用 DWORD 转换为 BYTE 类型的方式写入，DWORD 类型的方式读取。

详细代码如下。

```
PROGRAM PLC_PRG
VAR
    SDO_Write:BOOL;                                  //写入信号
    SDO_Read:BOOL;                                   //读取信号
    ETC_CO_SdoWrite4_0: ETC_CO_SdoWrite4;            //SDO 写入指令实例化
    ETC_CO_SdoReadDWord_0: ETC_CO_SdoReadDWord; //SDO 读取指令实例化
    data_write: ARRAY [1..4] OF BYTE;                //数据写入区
    Read_Puls_r:DWORD;                               //读取每转指令脉冲数
    Write_Puls_r:DWORD :=20000;                      //写入每转指令脉冲数
    i:INT;                                           //计数
    count:DWORD;
    Write_flag: BOOL;                                //写入标志
    Read_flag: BOOL;                                 //读标志

END_VAR

……
IF  Write_flag THEN                                  //写入标志
    count:=Write_Puls_r;
    SDO_Write:=FALSE;                                //复位写入信号
    FOR i:=1 TO 4 DO                                 //将写入值（DWORD）转换为 4 个 BYTE
        data_write[i]:=(TO_BYTE (count MOD 256 ));
        count:=(count / 256);
    END_FOR
    SDO_Write:=TRUE;                                 //置位写入信号
    Write_flag:=FALSE;                               //复位写入标志
END_IF

IF  Read_flag THEN                                   //读标志
    SDO_Read := TRUE;
    ……
END_IF
……
//写入每转指令脉冲数
ETC_CO_SdoWrite4_0( xExecute:=SDO_Write,xAbort:=,usiCom:=,
                    uiDevice:=1001,usiChannel:=,wIndex:=16#2008,bySubindex:=,
                    udiTimeOut:=5000,abyData:=data_write,usiDataLength:=4,
                    xDone=>,xBusy=>,xError=>,eError=>,udiSdoAbort=>);

//读取每转指令脉冲数
ETC_CO_SdoReadDWord_0(xExecute:=SDO_Read, xAbort:= , usiCom:= , uiDevice:=1001 ,
                usiChannel:= ,wIndex:= 16#2008, bySubindex:= , udiTimeOut:=5000 ,
                xDone=>,  xBusy=>,  xError=>  ,  eError=>  ,  udiSdoAbort=>  ,
                dwData=>Read_Puls_r , usiDataLength=> );
```

第11章 串口通讯

MC300CS 系列 PLC 上有 RS485 和 RS232 串行通讯接口。可以用于控制 485 接口的步进电机、伺服电机，也可以与其他 PLC、触摸屏等外设通讯。本章介绍用 Modbus RTU 协议和自由协议进行串口通讯的方法。

11.1. Modbus 通讯协议

Modbus 协议是一个主/从(master/slave)架构的协议。有一个节点是 master 节点，其他参与通信的节点是 slave 节点；每一个 slave 设备都有一个唯一的地址。在通讯网络中，只有被指定为主节点的节点才可以启动一个 Modbus 通讯指令。

一条 Modbus 指令包含一个将执行该指令的设备的 Modbus 地址。所有设备都会收到该指令，但只有指定地址的设备会执行该指令并回应指令。

所有的 Modbus 指令都包含检查码，以确定接收到的指令没有被破坏。基本的 Modbus 指令能指定一个远程终端单元 RTU（Remote Terminal Unit）改变其寄存器的值，控制或者读取一个 I/O 端口，以及指挥设备回送一个或者多个其寄存器中的数据。

Modbus 协议中按照通信访问的数据宽度划分数据类型，主要有 Bit 型和 Word 型两种数据类型。

依照行业惯例，本文中有时将 Bit 型变量称为“线圈”或“触点”，将 Word 型变量则称为“寄存器”。

Modbus 协议通信数据帧格式如图 11.1 所示。

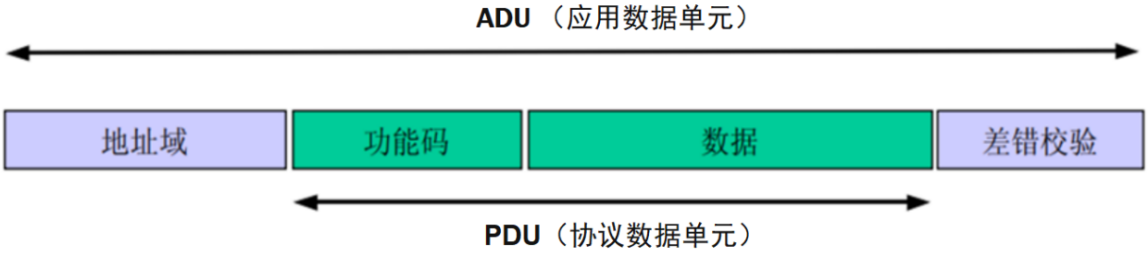


图 11.1 Modbus 通信帧格式

例如：主站要读地址为 1 的从站中的 2 个寄存器的值，两个寄存器的地址为 0004、0005。该指令的功能码为 03。报文交互过程如下：

主站的请求指令的通讯帧内容为：01 03 00 04 00 02 85 ca，详见表 11.1。

从站响应该请求，返回相应数据。该动作的功能码为 03。从站发出的通讯帧内容为：01 03 04 00 00 00 00 21 33，详见表 11.2。

表 11.1　主站发出的通讯帧

从机地址	功能码	从站寄存器 起始地址	读取寄存器个数	CRC 校验
01	03	00 04	00 02	85 ca

表 11.2　从站返回的通讯帧

从机地址	功能码	返回字节个数	寄存器 0005 数据	寄存器 0006 数据	CRC 校验
01	03	04	00 00	00 00	21 33

11.1.1.　Modbus 协议可访问的内部地址

MC300CS 系列 PLC 的变量区有 Q 区、I 区和 M 区，都可以分别按位、按字节、按字和按双字进行访问。寄存器地址索引规则如图 11.2 所示。

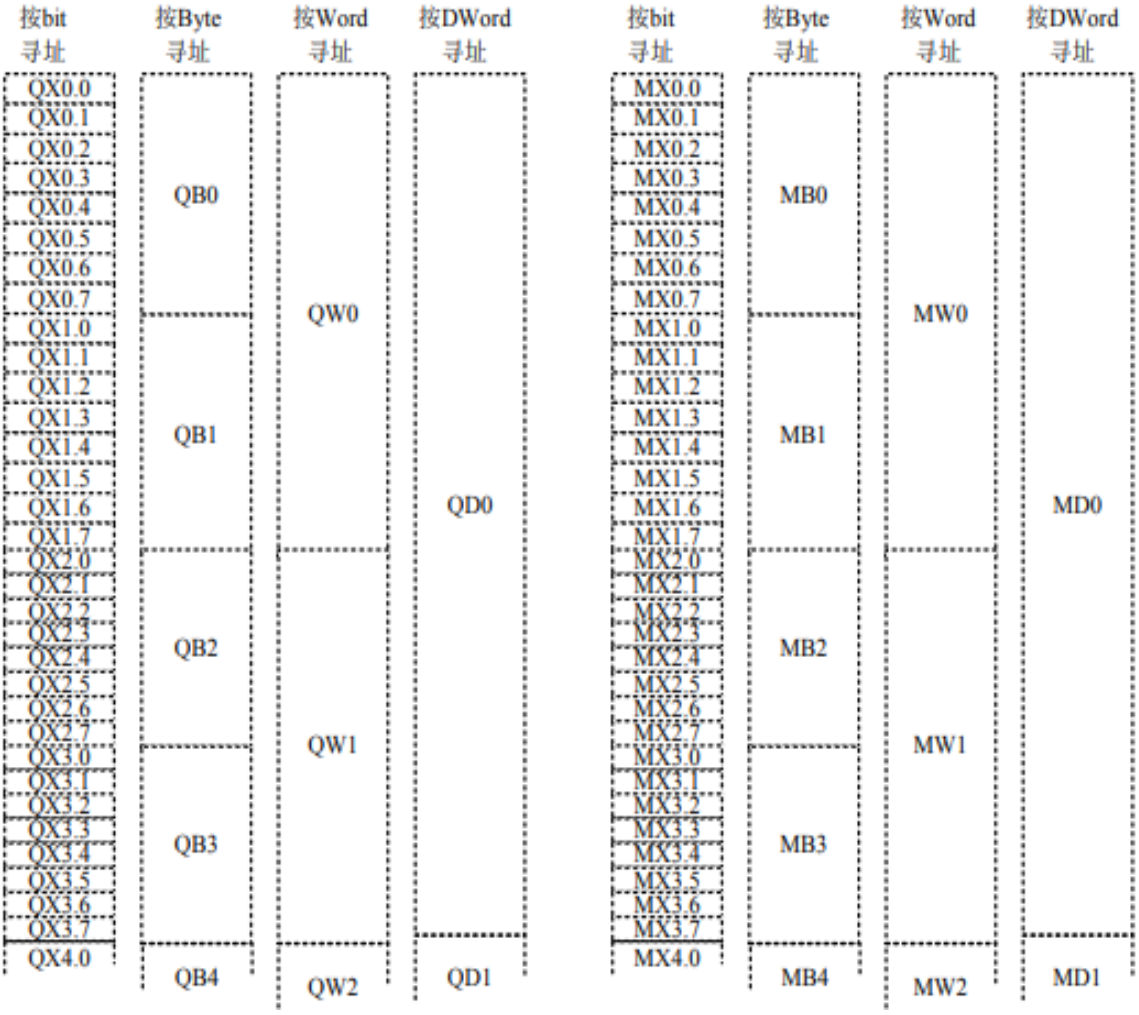


图 11.2　Q 区、M 区寄存器地址索引规则

Modbus 通信都可以访问 MC300CS PLC 内部 I、Q 区，范围如表 11.3 所示。

表 11.3 PLC 内部 I、Q 区范围

地址范围	功能码	起始地址	线圈数量	说明
QW0～QW4095 (QX0.0～QX8191.7)	0X01,0x05	0	65536	通用标准 Modbus 协议都可以访问
IW0～IW4095 (IX0.0～IX8191.7)	0X02	0	65536	通用标准 Modbus 协议都可以访问

Modbus 通信可访问 MC300CS PLC 内部 M 区，范围如表 11.4 所示。

表 11.4 PLC 内部 M 区范围

地址范围	功能码	起始地址	寄存器数量	说明
MW0～MW65535	0x03,0x06, 0x10, 0x0f	0	65536	通用标准 Modbus 协议都可以访问

11.1.2. Modbus 通信功能码

常用的Modbus通信功能码如表11.4所示。若想详细了解Modbus协议可参见《Modbus协议规范》。

表 11.4 功能码详细说明

功能码	访问类型	寄存器个数
01	读线圈状态	1～125
03	读取保持寄存器	1～125
05	写单个线圈	1
06	写单个寄存器	1
15	写多个线圈	1～125
16	写多个寄存器	1～125

11.2. RS485 Modbus RTU 通讯

Modbus RTU 是 Modbus 协议的一种，该通讯协议常用于工业领域。

RS485 Modbus RTU 通讯具有抗干扰强、传输速率快、传输距离远的优点。本节介绍 MC300CS RS485 Modbus RTU 配置方法。

11.2.1. RS485 Modbus RTU 主从站通讯

MC300CS PLC 基于 RS485 接口的 Modbus RTU 通信，通常以一个 PLC 为主站，变频器、伺服电机驱动器和其他 PLC 等设备为从站。

Modbus RTU 主从站通讯过程如图 11.3 所示。Modbus 主站的特点有：

① 可以主动发出指令；
② 具有唯一性，通讯总线上只能有一个主站；
③ 可以对接多个 Modbus 从站。

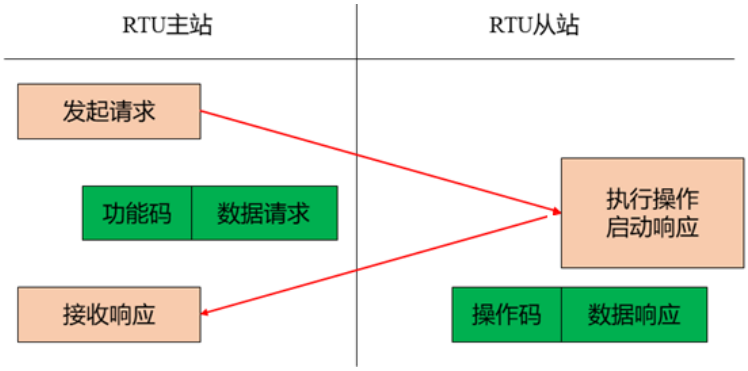


图 11.3　Modbus RTU 主从站通讯过程

MC300CS PLC 作为 Modbus RTU 主站时需要在 LeadSys Studio 软件中对串口参数进行设置，主要参数如表 11.5 所示。

表 11.5　Modbus 主从站串口配置相关参数

参数名	参数说明
波特率	通信时的速率，4800～115200 Hz
奇偶校验	通信帧的校验方式，与停止位匹配使用
数据位	RTU 模式下为 8 位，ASCII 模式下为 7 位
停止位	奇/偶校验时为 1，无校验时为 2
传输模式	RTU
帧间隔	主站接收上一个响应数据帧到下一个请求数据帧之间等待的时间间隔。RTU 模式下，两帧由时长至少 3.5 个字符时间区分,估算公式：（3.5 * 11 * 1000 / 波特率)ms

MC300CS PLC 作为 Modbus RTU 从站时需要设置从站站号和超过时间参数，如表 11.6 所示。超时时间是主站等待从站响应的时间，超过该时间主站会报错。

表 11.6　Modbus 从站配置相关参数

参数名	参数说明
从站站号	从站号范围 1～247
超时时间	超时时间参数设置范围 1～10000ms,默认设置为 5ms

MC300CS PLC 作为主站时还需要设置从站设备通讯数据的存储类型、控制模式、读/写寄存器等参数，如表 11.7 所示。

控制模式有周期和触发两种模式。

周期控制模式是主站向从站以一定间隔时间发送或者读取数据。

触发控制模式是触发变量（触发变量在 IO 映射中设置）电平改变时主站才向从站发送或者读取数据。

表 11.7 Modbus Master 通讯设置相关参数

序号	参数名	参数说明
1	功能码	读线圈状态 （功能码 01） 读取保持寄存器（功能码 03） 写单个线圈 （功能码 05） 写单个寄存器 （功能码 06） 写多个线圈 （功能码 15） 写多个寄存器 （功能码 16）
2	控制模式	周期模式
		触发模式
3	重发次数	本次发生通信故障未获得从站返回帧，则按重发次数进行重新发送。
4	注释	可以对数据进行描述的简短文本区域。
5	起始地址	读\写的寄存器位置开始地址。
6	长度	读取的寄存器个数。
7	错误处理	保持最后的值：使数据保持最后一次的有效值。

11.2.2. RS485 Modbus RTU 主站通讯例程

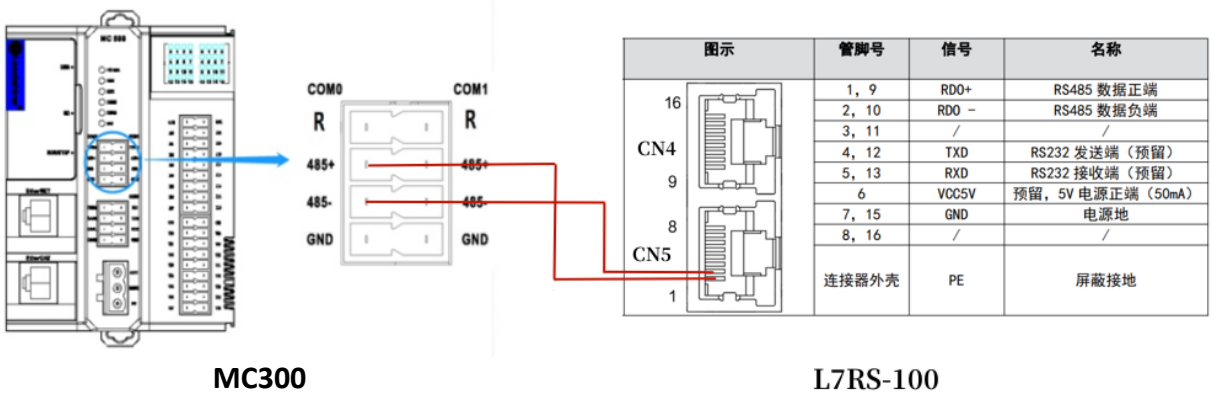
本例程用 MC300CS 作为 Modbus RTU 主站，485 总线电机的伺服驱动器 L7RS-100 作为 Modbus RTU 从站，进行数据交互。MC300CS 与 L7RS-100 通讯连接如图 11.4 所示。

主站 PLC 对从站 L7RS-100 的数据传输是通过周期循环方式进行读取和写入，循环周期时间 100ms。PLC 主站帧间隔为 5ms，从站超时时间为 1000ms。

Modbus RTU 主站将伺服驱动器的控制模式、PR1 模式、PR1 位置、PR1 速度、PR1 加速度、PR1 减速度、使能驱动、IO 组合触发模式等参数循环发送给从站，从站将命令位置循环回传给主站。

程序中判断输入口 IN0 上的启动按键状态，当按键闭合，将使能驱动设为 3，即电机做绝对坐标的点位运动；否则，使能驱动设为 0，即电机停止。

主要步骤如下。



图示	管脚号	信号	名称
	1，9	RDO+	RS485 数据正端
	2，10	RDO －	RS485 数据负端
	3，11	/	/
	4，12	TXD	RS232 发送端（预留）
	5，13	RXD	RS232 接收端（预留）
	6	VCC5V	预留，5V 电源正端（50mA）
	7，15	GND	电源地
	8，16	/	/
	连接器外壳	PE	屏蔽接地

图 11.4 MC300CS 与 L7RS-100 的 RS485 接线图

1. PLC 主从站通讯设置

1）使能 PLC 作为 Modbus RTU 主站

鼠标双击网络组态中的“通讯配置”，打开通讯配置界面，如图 11.5 所示；单击左侧栏中的 RS485，弹出所支持的通信协议的使能窗口。勾选“Modbus 主站”，使能后，变为红色。Modbus 主站使能后，左侧树中，出现了对应的通信配置，如图 11.5 中第 4 处所示“ModbusCOM_Master_1（ModbusCOM-RS485 Master）”。

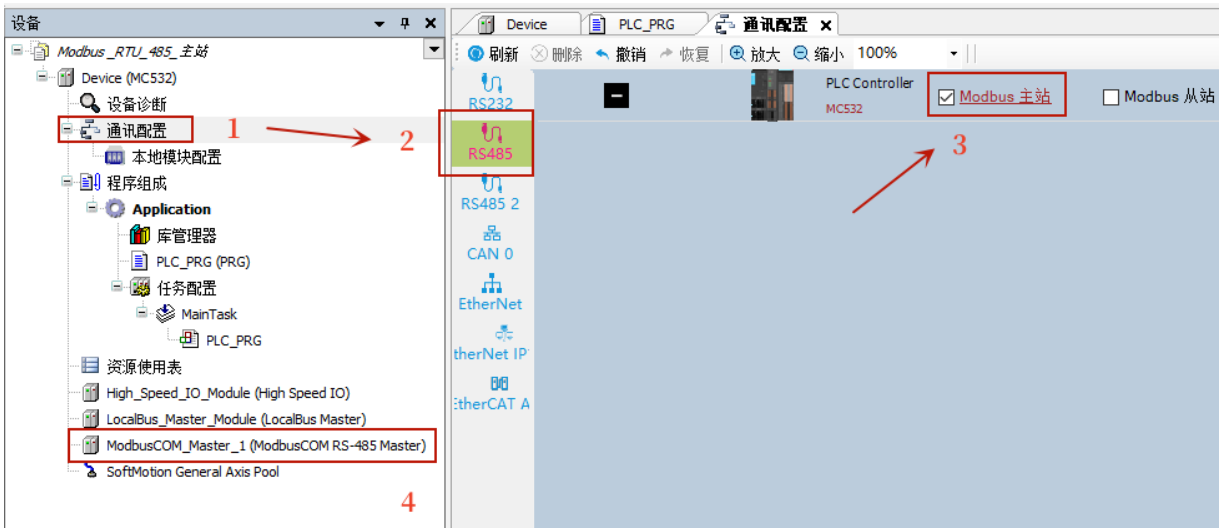


图 11.5　使能 PLC 作为 Modbus RTU 主站

2）添加 Modbus RTU 从站设备

如图 11.6 所示，在右侧“网络设备”中双击“ModbusCOM Slave”从站设备，或者直接拖拽到蓝色区域都可以成功添加从站设备。从站设备添加后，左侧项目树中出现了对应的通信配置“ModbusCOM_Slave（ModbusCOM Slave）”，如图 11.6 中的注释 2 所示。如果有多个从站可以重复添加上述步骤。

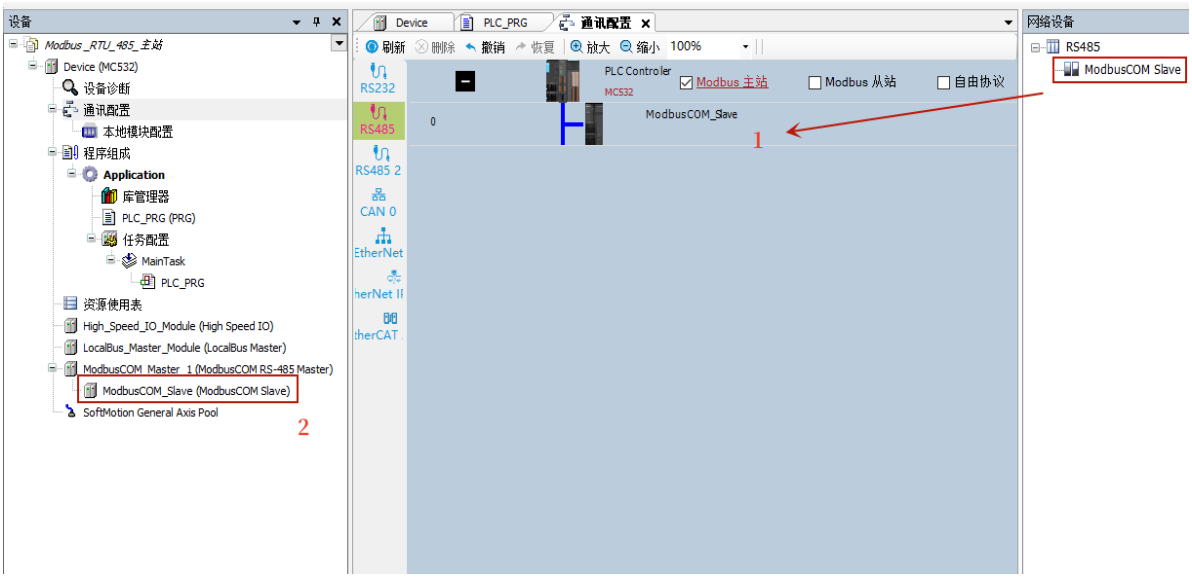


图 11.6　添加从站设备

3）Modbus RTU 主站设置

双击项目树中的主站设备“ModbusCOM_Master_1（ModbusCOM RS-485 Master）”，打开 Modbus 主站配置窗口，设置波特率 115200、偶检验、数据位为 8，停止位 1，默认帧间隔 5ms；如图 11.7 所示。帧间隔设置的值越小，主站响应的速度越快。帧间隔最小值为 1ms。

注意：Modbus 主从站通信参数配置必须一致，才能正常通信。

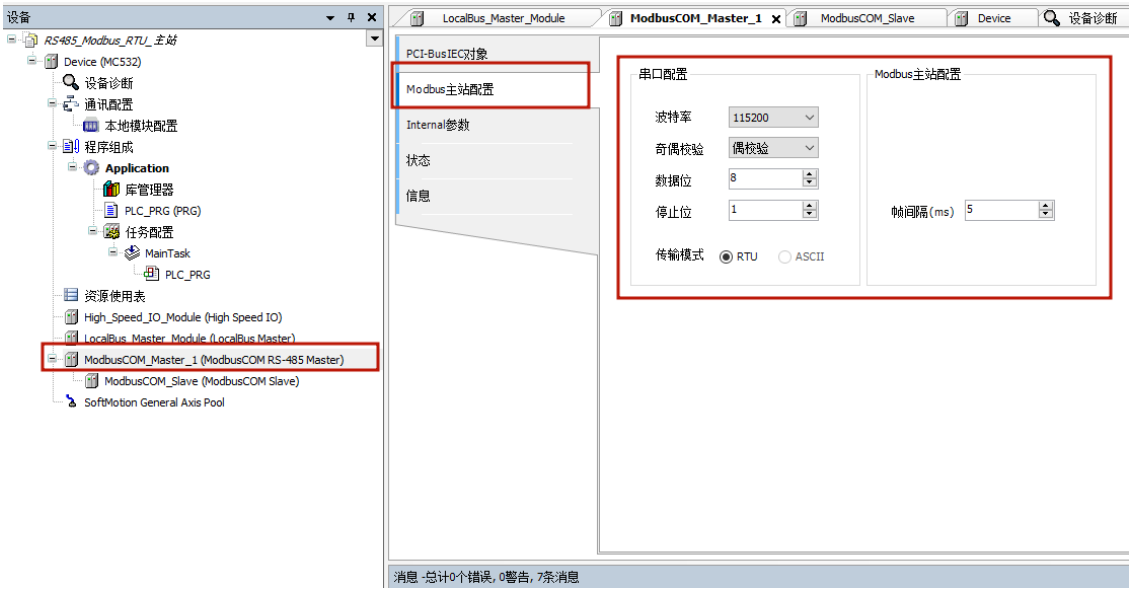


图 11.7　Modbus 主站配置

4）Modbus 从站设置

双击项目树中的主站设备“ModbusCOM_slave（ModbusCOM Slave）”，打开 Modbus 从站设置窗口，设置从站站号为 1，超时时间设置为 1000ms，PLC 主站发送数据后，从站在 1000ms 后没有响应，PLC 就报错接收超时。配置过程如图 11.8 所示。

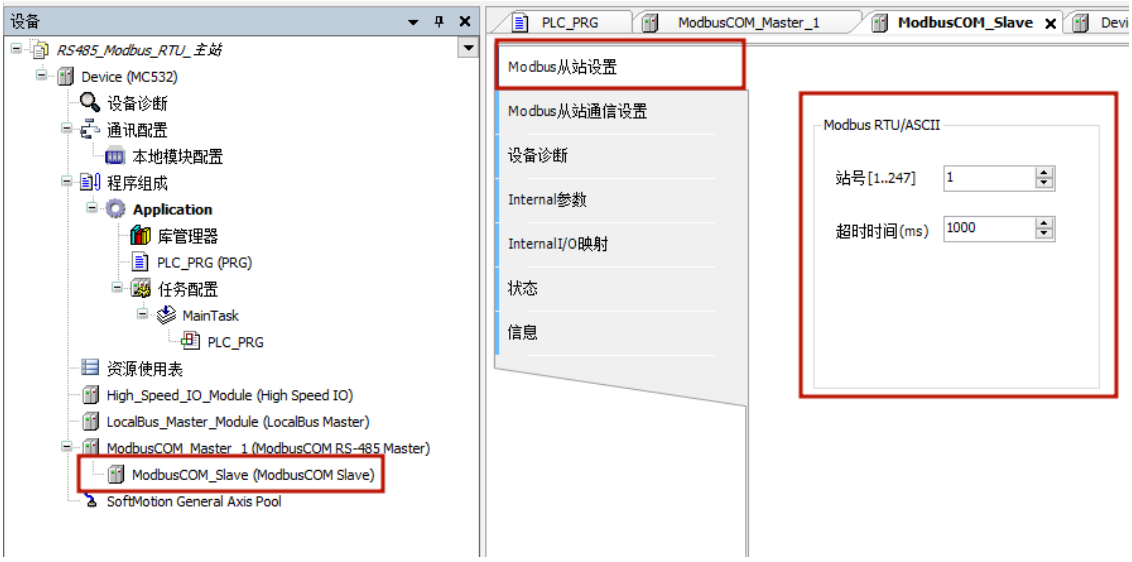


图 11.8　Modbus 从站设备配置-从站设置

5）Modbus 从站通讯参数设置

设置完 Modbus 主从站相关参数后，需要设置从站设备通讯数据的存储类型、控制模式、读写寄存器的起始地址（具体地址详见电机驱动器手册）、长度（WORD）。添加方式如图 11.9 所示。

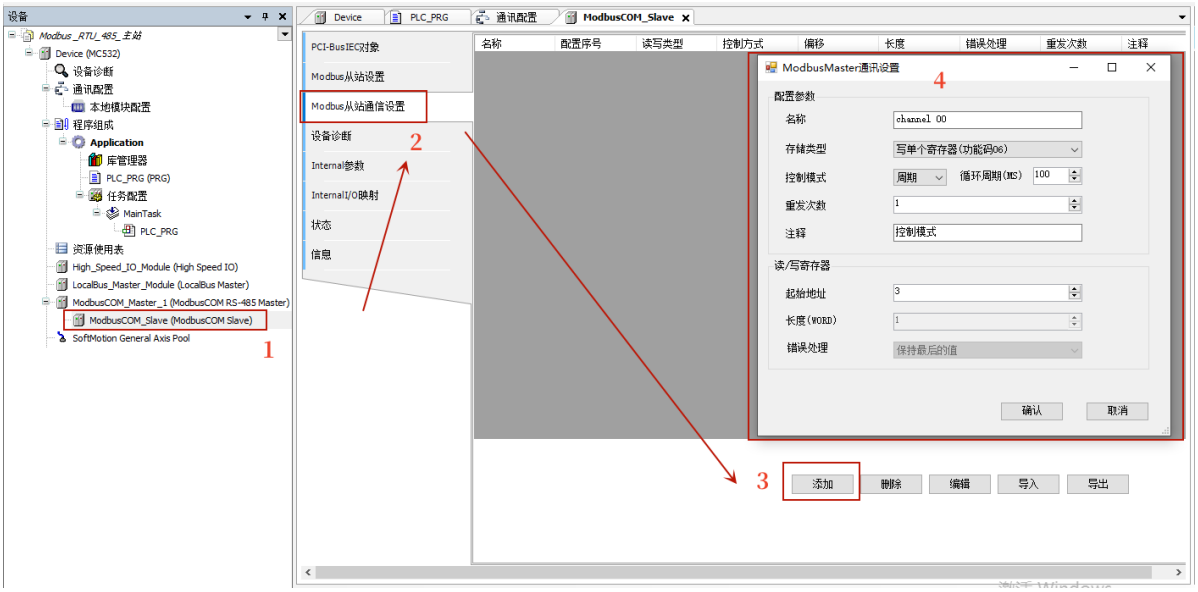


图 11.9　Modbus 从站通讯设置

用上述方法将伺服驱动器 Pr 模式的参数：控制模式设定参数、PR1 路径模式参数、PR1 位置、PR1 速度、PR1 加速度、PR1 减速度、虚拟 IO 参数、IO 组合触发模式、命令位置等要发送、接收的数据逐一添加，完成结果如图 11.10 所示。

以上参数的数据类型都为 WORD,控制模式都设置为周期，循环周期为 100ms。

图 11.10　Modbus 从站设备配置-从站通讯设置

MC300CS 支持 Modbus 从站通讯参数导入、导出功能，即从站通讯参数设置好后，可以保存为文件，下次要用时可以直接导入从站通讯参数文件。导入和导出的文件为 xml 格式。导入、导出参数的过程如图 11.11 和 11.12 所示。

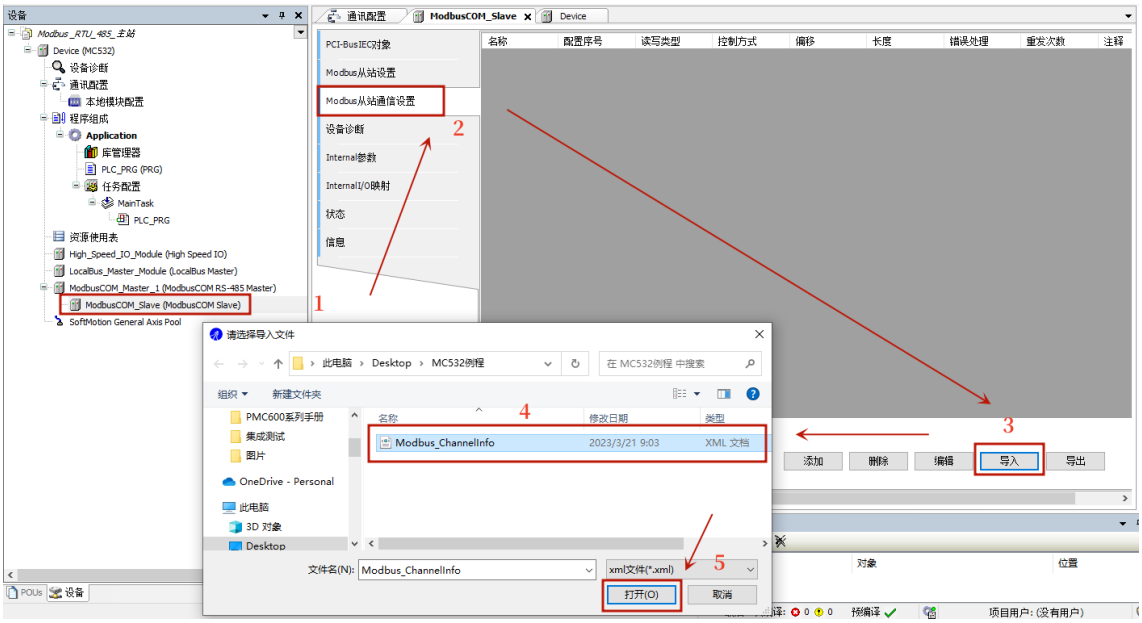


图 11.11　导入从站通讯参数的过程

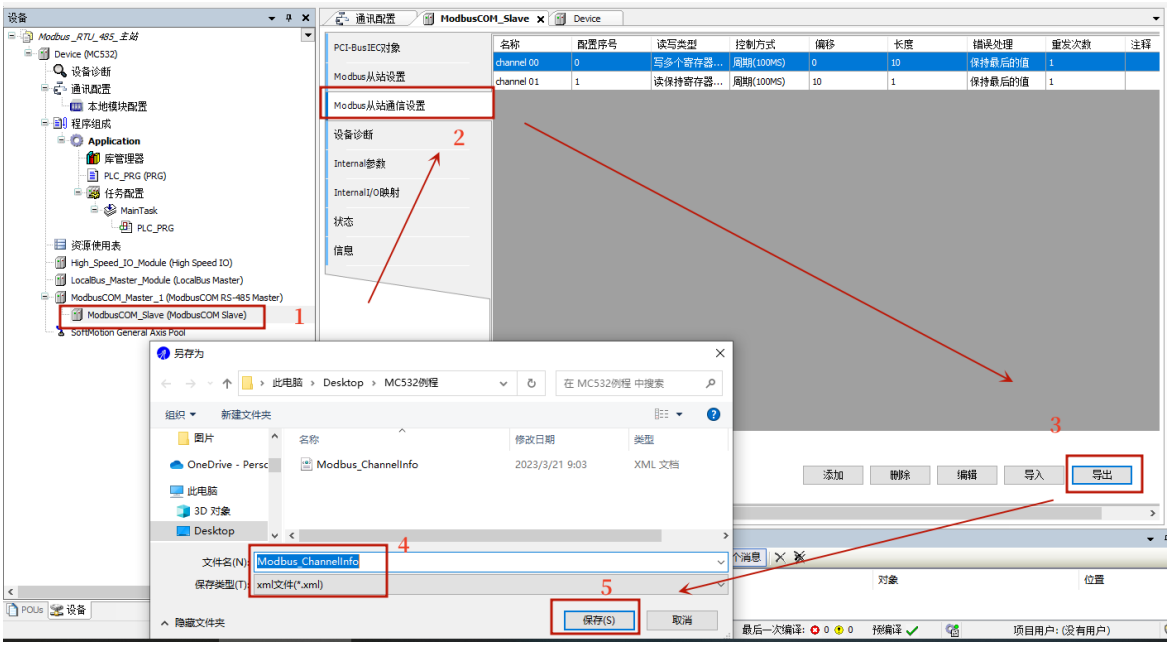


图 11.12　导出从站通讯参数的过程

6）Modbus 从站 Internal I/O 映射

Modbus 从站参数设置完成后，“Internal I/O 映射”卡中会显示各通讯数据自动分配到主站 PLC 的内部映射地址，如图 11.13 所示。如：第二行的%QW1 表示第一个寄存器值将写入到 PLC 内部%QW1 这个地址。

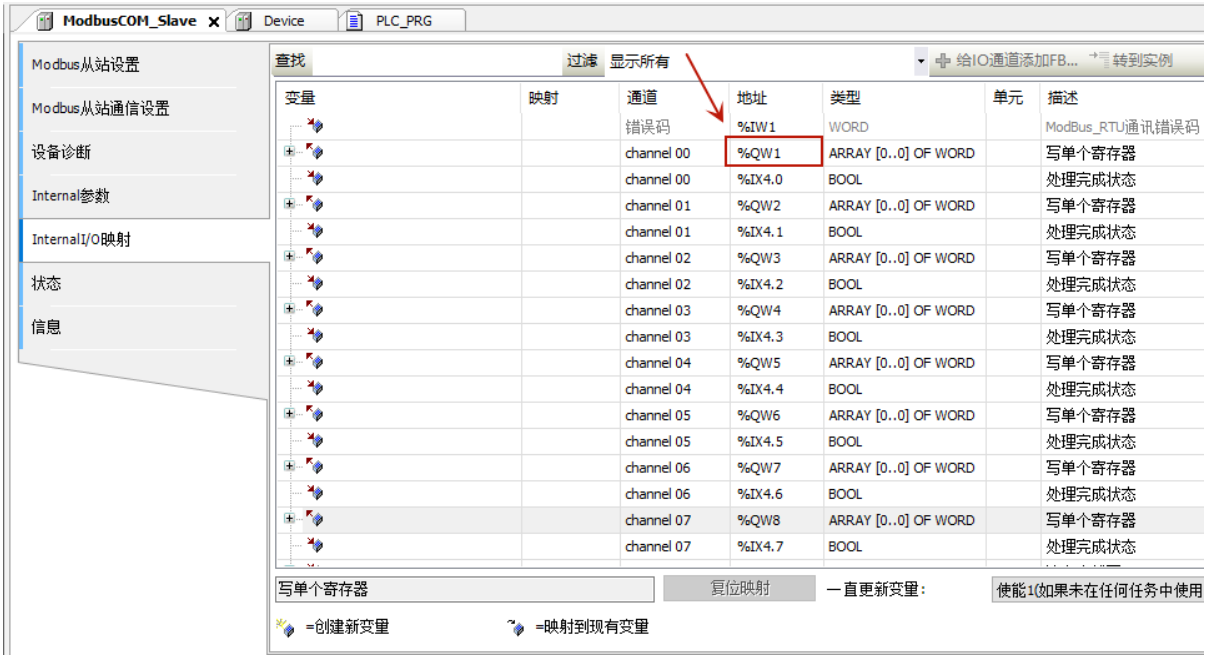


图 11.13　Internal I/O 映射

然后，需要在 Internal/IO 映射卡中手动操作，将通信通道与程序中自定义的变量建立映射关系，如图 11.14 所示的第 3、4、5 步骤。主站发送给从站的数据的变量有：Controlmodel、PR1mode、PR1position、Setspeed、Setacc、Setdec、IOmode，enable，主站读取从站的数据的变量有：Getposition。完成后的结果如图 11.15 所示。

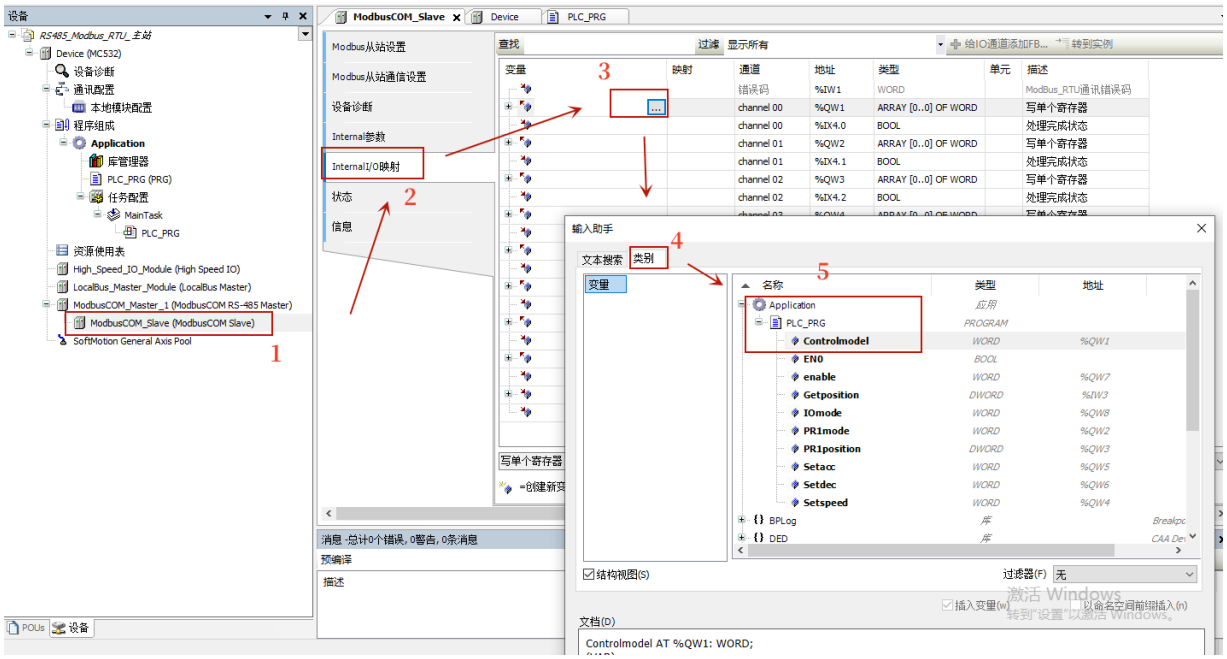


图 11.14　Internal I/O 映射输入变量路径

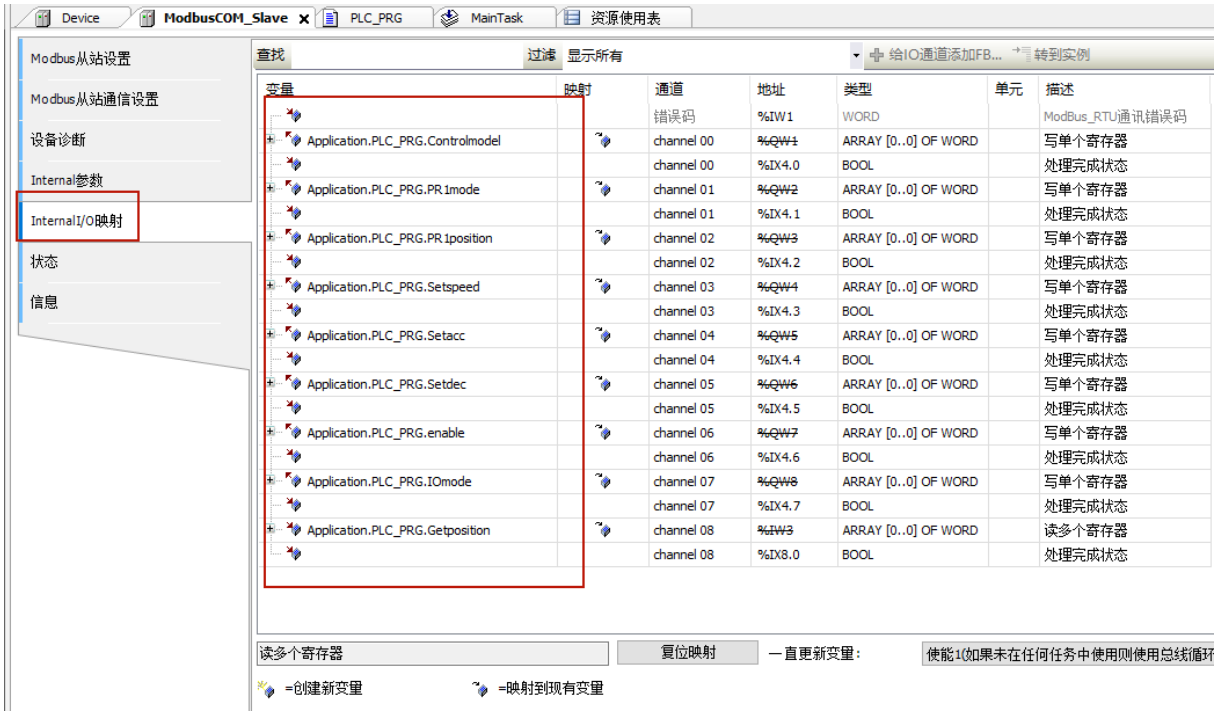


图 11.15　Modbus 从站设备配置-I/O 映射

还有另一种方式也可以声明变量,将程序中的自定义变量映射到 Internal I/O 地址。即在变量后加入 AT（寄存器地址），如图 11.16 所示。

```
PROGRAM PLC_PRG
VAR
    Controlmodel AT%QW1:WORD;          //控制模式
    PR1mode AT%QW2 : WORD;             //运动模式
    PR1position AT%QW3:WORD;           //绝对运动距离
    Setspeed AT%QW4:WORD;              //速度
    Setacc AT%QW5:WORD;                //加速度
    Setdec AT%QW6:WORD;                //减速度
    enable AT%QW7:WORD;                //使能驱动
    IOmode AT%QW8:WORD;                //IO 组合触发模式
    Getposition AT%IW3:DINT;           //获取电机指令位置
END_VAR
```

图 11.16　Internal I/O 映射

2. L7RS-100 伺服驱动器设置

设置 L7RS-100 伺服驱动器 485 串口通讯参数要与 MC300CS 串口通讯参数一致,即：波特率 115200、偶检验、数据位为 8，停止位 1。

用雷赛公司的 L7RS 系列调试软件“MotionStudio”设置 L7RS-100 伺服驱动器通讯参数。图中编号 Pr5.29 和 Pr5.30 设定的值可以查手册《L7RS 系列交流伺服系统使用手册》可知波特率为 115200，有效位 8 位，偶检验位，停止位为 1；如图 11.17 所示。

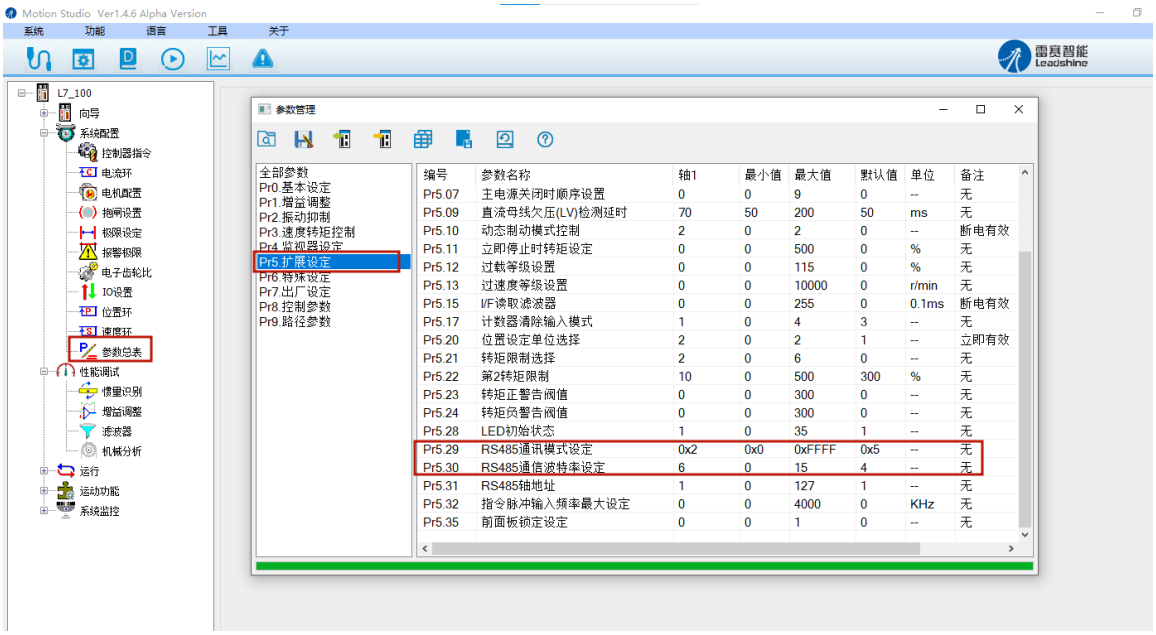


图 11.17　L7RS 系列调试软件设置 L7RS-100 伺服驱动器参数的界面

在 IO 设置界面，将“伺服接通输入”和“路径 0-PR 专用（ADD0）”设置在 Pr5.29 和 Pr5.30，如图 11.18 所示。分别用于伺服驱动器使能和路径 0 触发启动。通过 Internal I/O 映射到变量 enable。当 enable=3 时，Pr5.29 和 Pr5.30 都置为 TRUE。

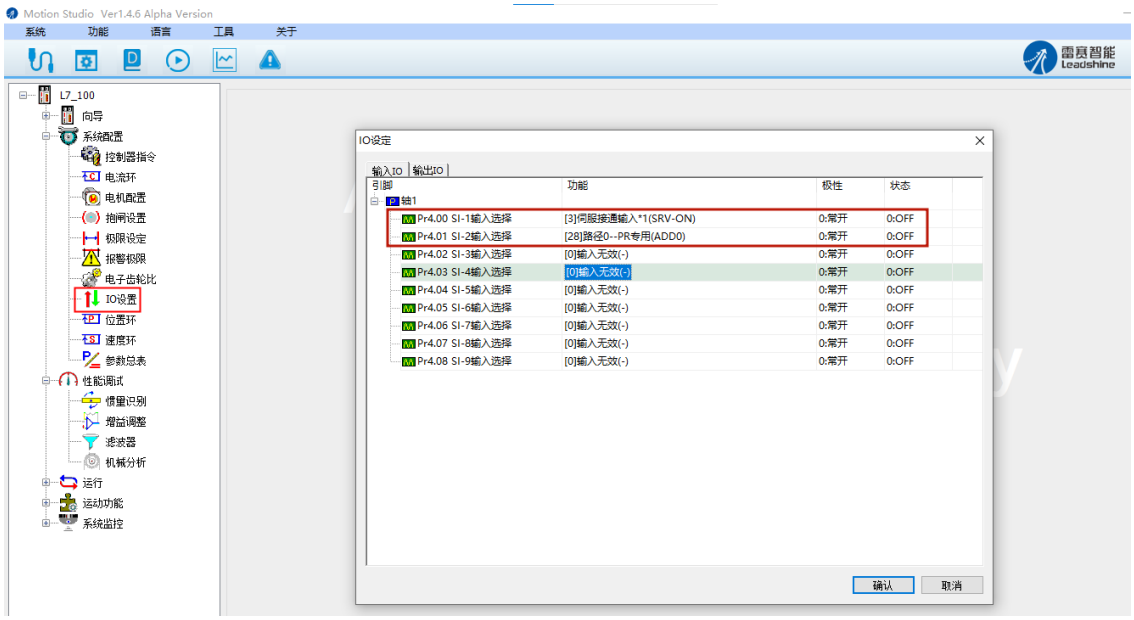


图 11.18　L7RS 系列调试软件设置虚拟 IO 映射

3.　MC300CS Modbus 主站的程序代码

```
PROGRAM PLC_PRG                    //声明变量
VAR
   ENO AT %IX0.0 :BOOL ;           //输入口 IN0 上的启动按键信号
   Controlmodel:WORD;              //控制模式
   PR1mode : WORD;                 //运动模式
   PR1position:WORD;               //绝对运动距离
   Setspeed:WORD;                  //速度
```

```
    Setacc:WORD;                    //加速度
    Setdec:WORD;                    //减速度
    enable:WORD;                    //使能驱动
    Getposition:DINT;               //获取电机指令位置
    IOmode: WORD;                   //IO 组合触发模式
END_VAR

//程序内容
    Controlmodel:=6;                //设置伺服驱动器控制模式
    PR1mode :=1;                    //运动模式：绝对位置
    IOmode:=2;                      //设置伺服 IO 触发模式
    PR1position:=30000;             //设置电机运行的绝对位置
    Setspeed:=80;                   //设置速度
    Setacc:=100;                    //设置加速度
    Setdec:=100;                    //设置减速度
    IF ENO THEN
        enable:=3;                  //使能驱动变量 enable 触发点位运动
    ELSE
        enable:=0;                  //使能驱动变量 enable 令电机停止
    END_IF
```

4. 程序运行结果

程序运行后，PLC 作为 Modbus RTU 主站将 Controlmodel、PR1mode、PR1position、Setspeed、Setacc、Setdec、IOmode，enable 等变量发送给伺服驱动器，当启动按键按下后，伺服驱动器使能，电机做点位运动，即运动到绝对位置 30000 处。主站读取变量 Getposition，其值变化至 30000，如图 11.19 所示。

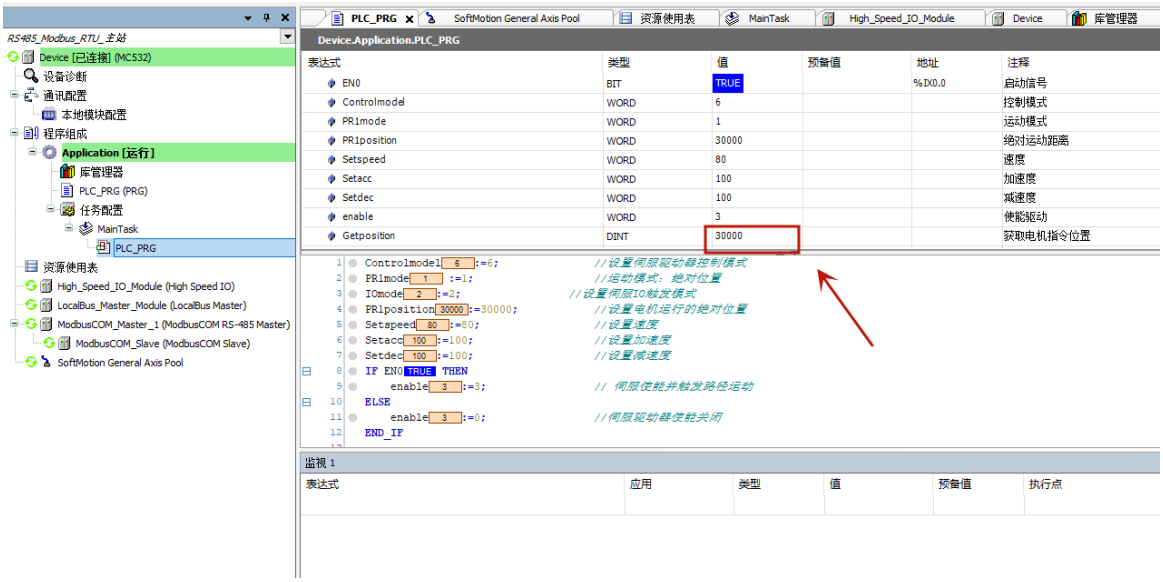


图 11.19　程序运行结果

11.2.3.　RS485 Modbus RTU 从站通讯

本节介绍 MC300CS 作为 Modbus RTU 从站时配置过程，PLC 作为 Modbus 从站特点：

① 只能接收主站发出的指令；

② 不具有唯一性，通讯总线上可以有多个从站；

③ 只能对接一个 Modbus 主站。

设置 Modbus RTU 从站参数的方法如下。

1. 使能 PLC 作为 Modbus RTU 从站

双击项目树中的“通讯配置”，如图 11.20 所示，单击“RS485”，弹出所支持的通信协议的使能窗口。勾选“Modbus 从站”，使能后，变为红色。Modbus 从站使能后，项目中出现对应的通信配置“Modbus_Slave_RS485_COM（Modbus Slave RS485 COM）”，如图 11.20 所示。

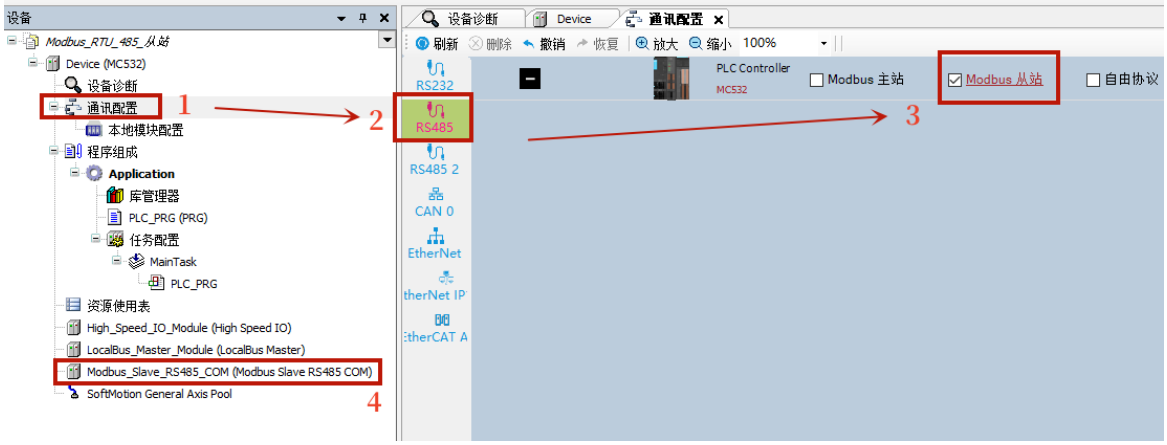


图 11.20　使能 PLC 作为 Modbus RTU 从站

2. Modbus RTU 从站参数设置

双击项目树中的从站设备“Modbus_Slave_RS485_COM”，打开 Modbus 从站配置窗口，配置波特率、奇偶校验等，如图 11.21 所示。有关的配置串口配置参数的可以参考表 11.5。

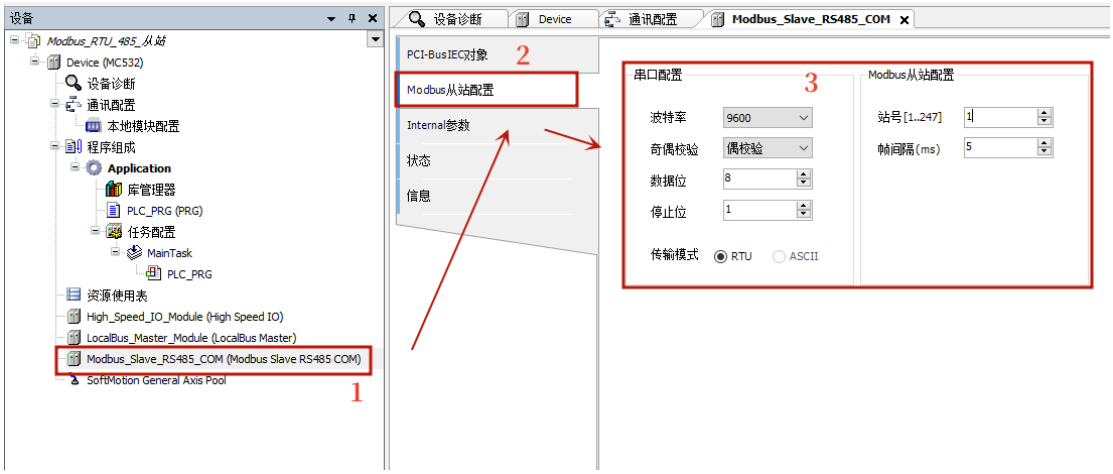


图 11.21　设置从站通讯参数

11.2.4. Modbus RTU 常见故障与错误码

Modbus 主站连接从站发生错误时主要的原因如下：

（1） Modbus 主站与 Modbus 从站配置不一致，导致主站与从站通信无法建立。

（2） Modbus 主站访问 Modbus 从站非法地址，返回错误应答。

（3） Modbus 主站操作 Modbus 从站写寄存器，但是 Modbus 从站该寄存器只支持读不支持写操作，Modbus 主站会收到 Modbus 从站返回的出错应答。

（4） 报文中的错误码与 LeadSys Sudio 编程软件在线调试显示的错误码关系为：在线调试显示的错误码 = error +40000。错误码定义，如表 11.8 所示。

表 11.8　错误码定义

故障码	说明	故障码	说明
40001	非法的功能码	40011	目标设备未能回应
40002	非法的数据地址	40012	无效的 CRC
40003	非法的数据值	40013	无效的数据
40004	从站设备故障	40014	无效的异常码
40005	AK 异常	40015	保留
40006	从站设备忙	40016	数据过多
40007	否定应答	40017	响应与查询地址不匹配
40008	内存奇偶校验错误	40018	Modbus TCP 连接超时
40009	未定义	40019	Modbus TCP 通讯掉线
40010	网关路径不可用		

（5） 错误码查询与错误响应帧格式

① 通过 PLC 在线调试，可以查看当前通信错误码，正常通信时，错误码为 0，如图 11.22 所示。

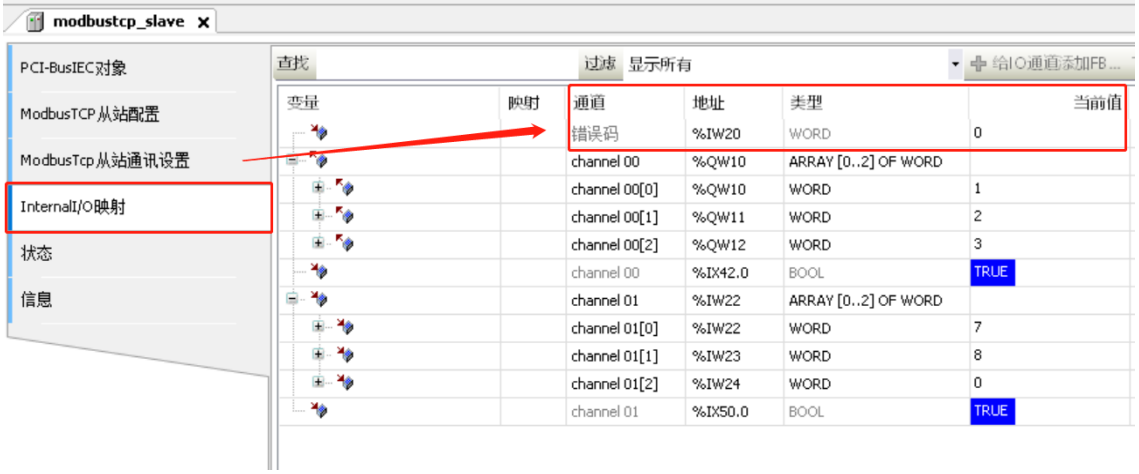


图 11.22　错误码查询

例：如图 11.23 所示，当通信错误时，错误码为 40017，查表 11.8 可知当前错误为“响应与查询地址不匹配”。

图 11.23　错误码

② 也可以在设备诊断中直接查看诊断代码以及对应的诊断状态，如图 11.24 所示。

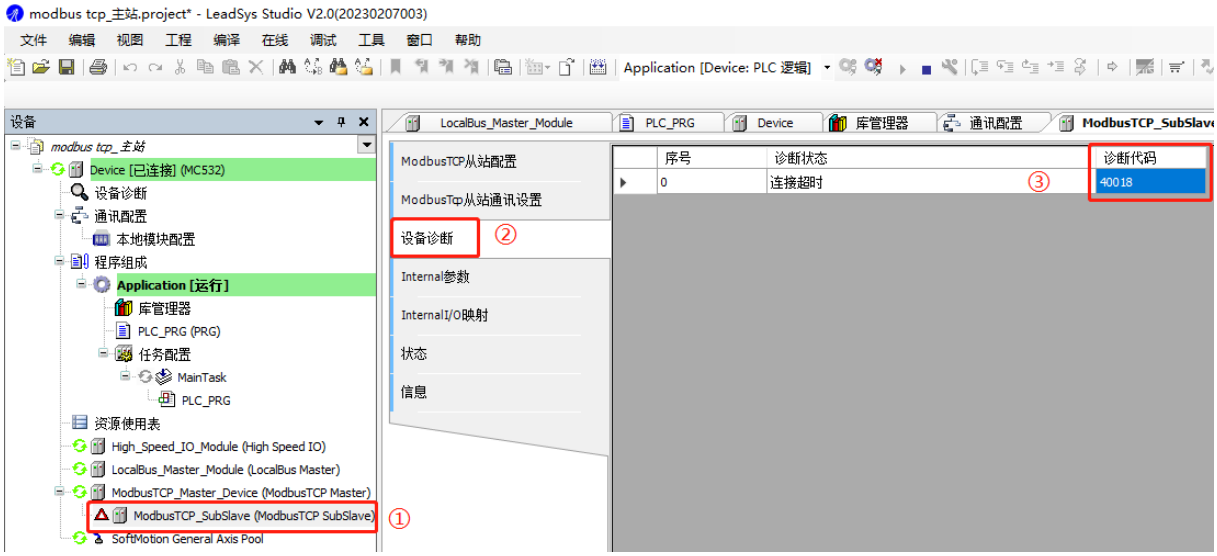


图 11.24　设备诊断

③ 错误响应帧格式：（本错误帧适合所有的操作命令帧）

错误响应帧：从机地址 +（命令码 +0x80）+ 错误码 +CRC 校验。

以从机地址为 1，功能码 03 为例：

主站请求：01 05 00 0e 00 00 ac 09

从站响应：01 85 02 c3 51，如表 11.9 所示。

表 11.9　从站响应

从机地址	错误响应码	返回错误码	CRC 校验
01	85	02	c3 51

0x85 是功能码 0x05 的错误响应码，返回错误码 = 02。

根据 LeadSys Sudio 编程软件对应的显示规则，可知错误码为 40002，查表 11.8 可知，该错误为“非法的数据地址”。

11.3. RS485 自由协议通讯及例程

自由协议通讯也称无协议通讯，需要根据对方设备的通讯数据格式编写一个临时协议，虽然随着标准协议（如：Modbus，USS 等）的普及，自由协议通讯应用越来越少，但一些小的设备，如：扫码枪、LCD 显示屏等，没有集成标准通讯协议，所以只能选用自由协议通讯。

本节介绍 RS485 自由协议通讯相关指令和参数设置方法，以 MC300CS PLC 与 SMC304 控制器通讯为例讲解自由协议通信过程。

11.3.1.　自由协议的指令

自由协议通讯的相关指令在“CAA SerialCom”库文件中，如果函数库中没有“CAA SerialCom”文件，需要手动添加。手动添加过程如图 11.25 所示，完成后如图 11.26 所示。

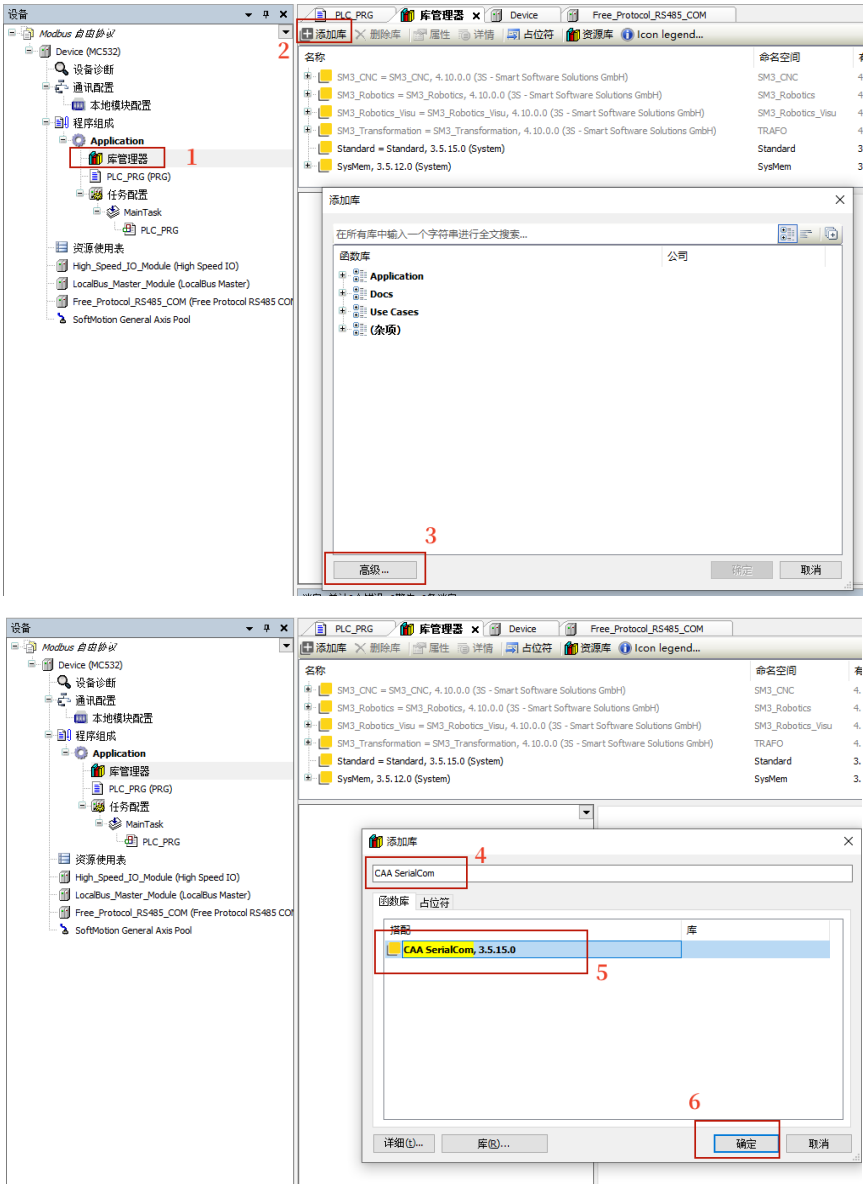


图 11.25　添加 CAA SerialCom 库函数的操作过程

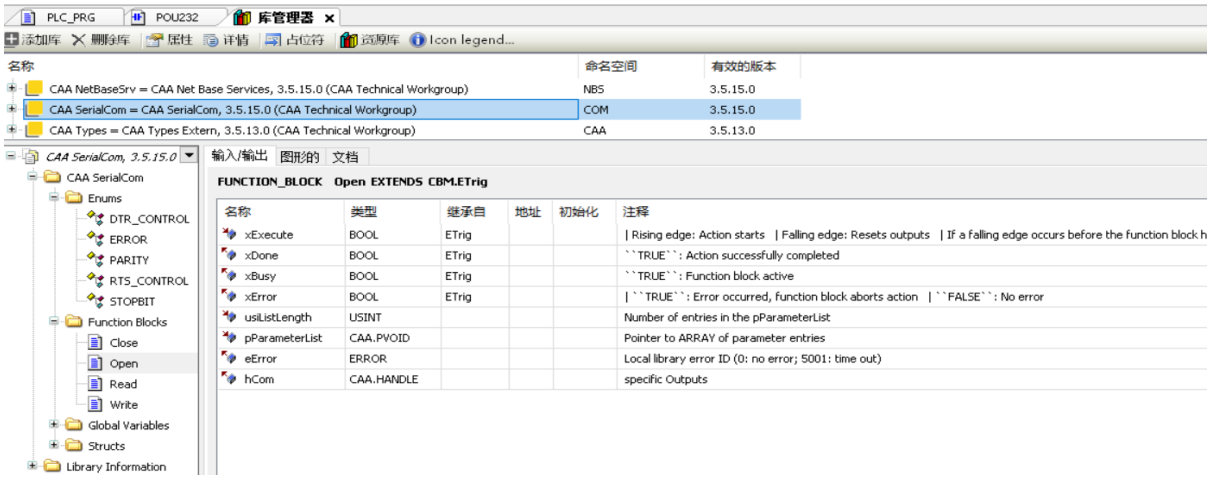


图 11.26　CAA SerialCom 库函数

CAA SerialCom 结构体定义了串行通讯配置参数，配置参数如下：

```
aParamsB1 : ARRAY [1..7] OF COM.PARAMETER := [
(udiParameterId := COM.CAA_Parameter_Constants.udiPort, udiValue := 端口号),
(udiParameterId := COM.CAA_Parameter_Constants.udiBaudrate, udiValue := 波特率),
(udiParameterId := COM.CAA_Parameter_Constants.udiParity, udiValue := 奇偶检验),
(udiParameterId := COM.CAA_Parameter_Constants.udiStopBits,udiValue := 停止位),
(udiParameterId := COM.CAA_Parameter_Constants.udiTimeout, udiValue := 超时时间值),
(udiParameterId := COM.CAA_Parameter_Constants.udiByteSize, udiValue := 数据位值),
(udiParameterId := COM.CAA_Parameter_Constants.udiBinary, udiValue := 二进制参数)];
```

CAA SerialCom 串口自由协议的指令如表 11.10 所示，用以实现串口的打开、关闭，串口的读、写操作。其格式如下。

表 11.10　串口通讯功能块

指令类别	名称	功能
串口通讯	COM.Close	关闭串行端口
	COM.Open	打开串行端口
	COM.Write	从串行端口进行自由协议数据发送
	COM.Read	从串行端口读取自由协议的接收数据

1. 打开串口自由协议通讯

```
COM.Open(xExecute :=启动信号, xDone=>完成信号, xBusy=>执行中信号,
         xError =>错误标志, usiListLength :=参数参数列表条目数,
        pParameterList:=参数列表指针, eError=>错误信号, hCom =>连接句柄 );
```

其中：连接句柄 hCom 等于 0，表示通讯连接失败。

2. 关闭串口自由协议通讯

```
COM.Close (xExecute:= 启动信号, xDone=>完成信号, xBusy=>执行中信号,
           xError =>错误标志, hCom:=连接句柄, eError=>错误信号 );
```

其中：连接句柄 hCom 与指令 COM.Open 的相同。

3. 自由协议通讯数据读取

```
COM.Read(xExecute:=启动信号, xAbort:=功能执行中止信号, udiTimeOut:=超时时间 ,
        xDone=>完成信号, xBusy=>执行中信号, xError=>错误标志,
        xAborted=>操作中止信号, hCom:=连接句柄, pBuffer:=指向接收数据缓存区的指针,
        szBuffer:= 接收数据最大字节数, eError=>错误信号,
        szSize=>pBuffe 中接收到的数据字节数 );
```

其中：连接句柄 hCom 与指令 COM.Open 的相同。

4. 自由协议通讯数据写入

```
COM.Write(xExecute:=启动信号, xAbort:=功能块执行中止信号, udiTimeOut:=超时时间,
          xDone=>完成信号, xBusy=>执行中信号, xError=>错误标志,
          xAborted=>中止信号, hCom:=连接句柄, pBuffer:=指向发送数据缓存区的指针,
          szSize:= 发送数据的字节数, eError=>错误信号);
```

其中：连接句柄 hCom 与指令 COM.Open 的相同。

以上是 CAA SerialCom 库中自由协议通讯的指令，下面两个指令是自由协议通讯读写数据时常常会用到的两个指令：SysMemSet 和 SysMemCpy，这两个指令在 SysMem 函数库中。

5. SysMemCpy 指令

SysMemCpy 指令的主要作用是将内容从源(pSrc)复制到目标缓冲区(pDest)，返回值为完成复制的目标缓冲区的指针，其指令格式如下。

```
SysMemCpy（pDest:=指向要复制到的内存地址指针, pSrc:=指向要从中复制的内存地址指针,
          udiCount:=复制字节数）;
```

6. SysMemSet 指令

SysMemSet 指令是使用指定值初始化内存空间，返回值为已初始化的内存块的指针。如果操作失败，则为 0。其指令格式如下：

```
SysMemSet（pDest:=指向初始化内存块指针, udivalue:=初始化内存值,
          udiCount:= 内存块中初始化的字节数）;
```

11.3.2. 485 总线自由协议通讯例程

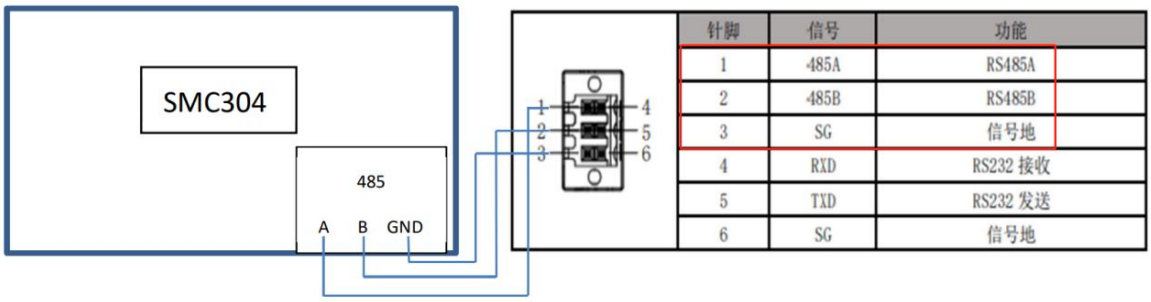
本节以 MC300CS 系列 PLC 与雷赛 SMC304 运动控制器进行 485 总线自由协议通讯为例，介绍实现自由协议通讯的方法。

MC300CS 通过 485 接口给 SMC304 发送数据控制 SMC304 控制器输出 OUT0 上的指示灯亮、灭，并且读取该指示灯的状态。通讯接线方法如图 11.27 所示。

程序中 MC300CS 每间隔 50ms 读一次 SMC304 回传的数据；并循环查询输入口 IN0 上的按键状态，当按键按下后，如果接收到 SMC304 回传的数据为“OFF”，则向 SMC304 控制器发送数据“0”，让 SMC304 的指示灯亮；如果接收到 SMC304 回传的数据为“ON ”，则向 SMC304 控制器发送数据“1”，让 SMC304 的指示灯灭。

SMC304 每隔 200ms 循环读取 MC300CS 发送的数据；接收的数据为“0”，则控制 OUT0 上的指示灯亮，并回送数据“ON ”给 MC300CS；若接收的数据为“1”，则控制 OUT0 上的指示灯灭，并回送数据“OFF”给 MC300CS。

当 IN1 接口上的按键按下后，结束通讯。



针脚	信号	功能
1	485A	RS485A
2	485B	RS485B
3	SG	信号地
4	RXD	RS232 接收
5	TXD	RS232 发送
6	SG	信号地

图 11.27　通讯接线图

1.　485 总线自由协议通讯配置

鼠标双击项目树中的“通讯配置”，如图 11.28 所示，单击“RS485”，弹出所支持的通信协议的使能窗口。勾选“自由协议”，使能后变为红色。自由协议使能后，项目树中出现对应的通信配置 “Free_Protocol_RS485_COM（Free Protocol RS485 COM）”，如图 11.28 所示。

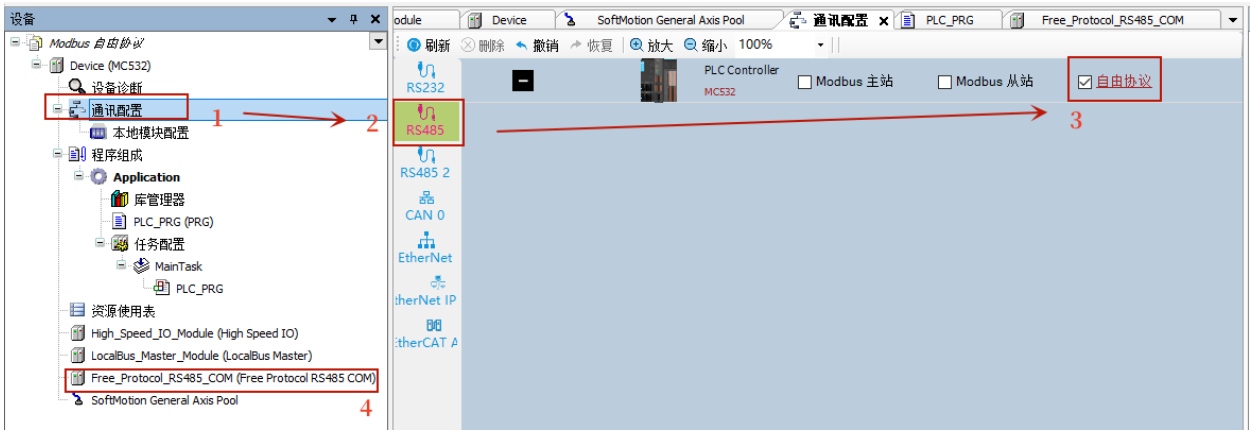


图 11.28　485 总线使能自由协议

设置 MC300CS 通讯参数如图 11.29 所示。帧间隔设置为 100ms，设置接收和发送的数据最大长度为 10 个字节。

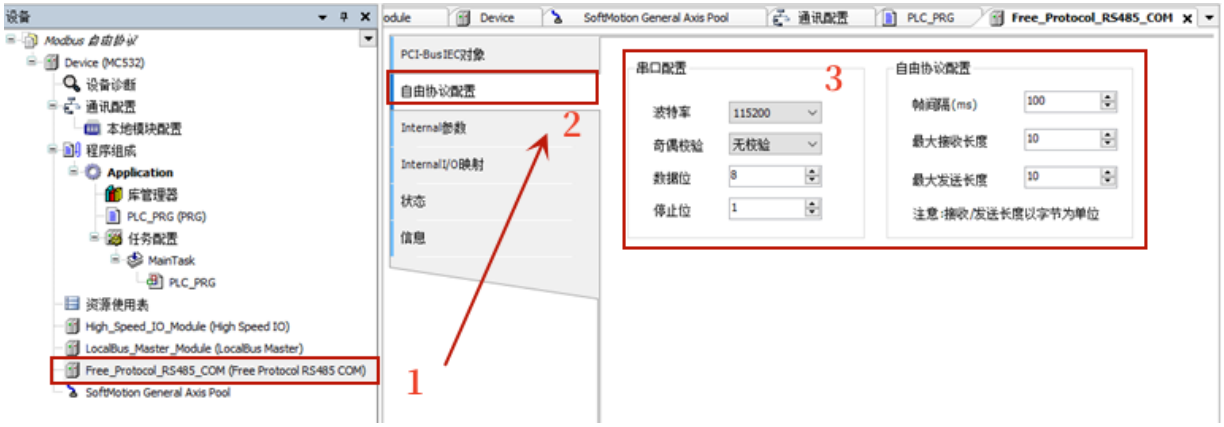


图 11.29　自由协议通讯参数

MC300CS 自由协议通讯是使用 RS232 端口还是 RS485 端口由 COM 端口号决定，相应 udiPort 值如表 11.11 所示，需要在程序中由串口参数结构体设置。本次例程用 COM0

口，udiPort 值为 3。

表 11.11 通讯端口选择表

端口号	COM0(RS485)	COM1(RS485)	COM2(RS232)
udiPort 值	3	4	2

2. 通讯程序

RS485 总线自由协议通讯例程代码如下。

```
PROGRAM PLC_PRG
VAR
  // 设置串口通讯结构体参数：udiPort=3，选择 COM0 口 485 自由协议通讯；波特率=115200，无
  校验位，停止位为 1，
  // 连接超时 500ms，每个字节 8 位，二进制参数为 1
  aParamsB1 : ARRAY [1..7] OF COM.PARAMETER := [
  (udiParameterId := COM.CAA_Parameter_Constants.udiPort, udiValue := 3),
  (udiParameterId := COM.CAA_Parameter_Constants.udiBaudrate, udiValue := 115200),
  (udiParameterId := COM.CAA_Parameter_Constants.udiParity,
    udiValue := INT_TO_UDINT(COM.PARITY.NONE)),
  (udiParameterId := COM.CAA_Parameter_Constants.udiStopBits,
    udiValue := INT_TO_UDINT(COM.STOPBIT.ONESTOPBIT)),
  (udiParameterId := COM.CAA_Parameter_Constants.udiTimeout, udiValue := 500),
  (udiParameterId := COM.CAA_Parameter_Constants.udiByteSize, udiValue := 8),
  (udiParameterId := COM.CAA_Parameter_Constants.udiBinary, udiValue := 1)];
  fbComOpen    : COM.Open;              //COM 口开启函数实例化
  fbComWrite   : COM.Write;             //COM 口写数据函数实例化
  fbComRead    : COM.Read;              //COM 口读数据函数实例化
  fbComClose   : COM.Close;             //COM 口关闭函数实例化
  xRead: BOOL;                          //数据读取
  xSend: BOOL;                          //数据发送
  xComConnect: BOOL;                    //串口连接
  xOpenCom: BOOL:=TRUE;                 //打开串口
  xError: BOOL;                         //错误状态
  hCom: COM.CAA.HANDLE;                 //串口连接句柄
  dataBuf2:ARRAY [0..3] OF BYTE;        //读数据区
  dataBuf: BYTE;                        //写数据
  input1 AT %IX0.0:BOOL;                //发送信号的按键
  data:STRING;                          //读取的字节转换为字符串
  timer1:TON;                           //定时器 1
  timer1On:BOOL :=TRUE;                 //直接启动定时器 1
  timer1Off:BOOL;
  timer2:TON;                           //定时器 2
  timer2On:BOOL;
  timer2Off:BOOL;
  input2 AT %IX0.1:BOOL;                //结束通讯的按键
END_VAR

//程序代码：
  //打开串口：
```

```
fbComOpen(xExecute := xOpenCom,
          usiListLength := SIZEOF(aParamsB1)/SIZEOF(COM.PARAMETER),
          pParameterList := ADR(aParamsB1),xError => xError,  hCom => hCom);
IF hCom <> 0  THEN                                          //判断连接状态
    xComConnect := TRUE;                                    //连接成功
    fbComWrite( xExecute:= xSend, xAbort:= , udiTimeOut:= , xDone=> , xBusy=> ,
                xError=> , xAborted=> , hCom:=hCom , pBuffer:= ADR(dataBuf) ,
                szSize:=1 , eError=> );                     //写数据
    fbComRead(xExecute:=xRead , xAbort:= , udiTimeOut:= , xDone=> ,  xBusy=> ,
              xError=> , xAborted=> , hCom:=hCom , pBuffer:=ADR(dataBuf2),
              szBuffer:=3, eError=>, szSize=> );          //读数据
  ELSE
    xComConnect :=FALSE;                                  //连接失败
END_IF
IF fbComRead.xDone = TRUE THEN                            //接收到新数据
   SysMem.SysMemSet(ADR(data),0,SIZEOF(data));            //清除 data 中上一次接收的数据
   SysMem.SysMemCpy(ADR(data),ADR(dataBuf2),3);           //将新读取数据转换为字符串
END_IF

timer1(In:=timer1On, pt:=T#50MS, Q=>timer1Off, ET=> );    // 定时器 timer1 延时 50ms
timer2(In:=timer2On, pt:=T#50MS, Q=>timer2Off, ET=>);     // 定时器 timer2 延时 50ms

IF timer1Off=TRUE THEN                                    // 如果定时器 timer1 时间到
   timer1On:=FALSE;                                       // 定时器 timer1 启动信号置零
   timer2On:=TRUE;                                        // 启动定时器 timer2
   xRead:=FALSE;
END_IF
IF timer2Off=TRUE THEN                                    // 如果定时器 timer2 时间到
   timer2On:=FALSE;                                       // 定时器 timer2 启动信号置零
   timer1On:=TRUE;                                        // 启动定时器 timer1
   xRead:= TRUE;
END_IF
IF input1 THEN                                            // IN0 上的按键闭合
    xSend:=TRUE;                                          // 向 SMC304 发送信号
  ELSE
    xSend:=FALSE;
END_IF
IF data='OFF' THEN                                        //判断读取的数据
     dataBuf:=0;
  ELSE
     dataBuf:=1;
END_IF
IF  input2 THEN                                           //结束通讯的按键按下
    fbComClose(xExecute:=TRUE, hCom:= hCom, xError => );  //关闭串口
    xOpenCom:=FALSE;                                      //禁止打开串口
END_IF
```

3. SMC304 控制器程序

```
Dim index=2, mode=1, bytes=5, recvsize, readData(8),sendData(8)  '定义变量
setcom(115200,8,1,0,2)                                    ' 设置串口通讯参数
```

```
SMCComSetMode(index,mode)                              ' 设置串口通讯模式为 485
readData(0)= 1                                         ' 指示灯的初始状态为灭
while 1                                                ' 死循环
    delay(200)                                         '延时 200ms
    recvsize=SMCComRead(index,bytes,readData(0))       '读取数据
    IF recvsize>0 then
        For i=0 to recvsize-1
            Print readData(i)                          '显示数据
        Next i
    endif
    IF readData(0)= 0  then                            '接收的指令是 0 吗?
        SMCWriteOutbit(0,0)                            '亮灯
    ELSEIF readData(0)= 1 THEN
        SMCWriteOutbit(0,1)                            '灭灯
    endif
    IF SMCReadOutbit(0)= 0  then                       '灯亮了吗?
        strcpy(sendData(0),"ON ")                      '将“ON ”写入数组 sendData 中
        SMCComWrite(index,3,sendData(0)                '发送数组 sendData 中的数据
    ELSEIF SMCReadOutbit(0)= 1  then                   '灯灭了吗?
        strcpy(sendData(0),"OFF")                      '将“OFF”写入数组 sendData 中
        SMCComWrite(index,3,sendData(0))               '发送数组 sendData 中的数据
    endif
wend
```

4. 程序运行结果

程序运行后，反复按 IN0 上的按钮，SMC304 上的指示灯也跟随着亮、灭交替变化。运行情况如图 11.30 所示。按下 IN1 上的按钮，通讯结束。

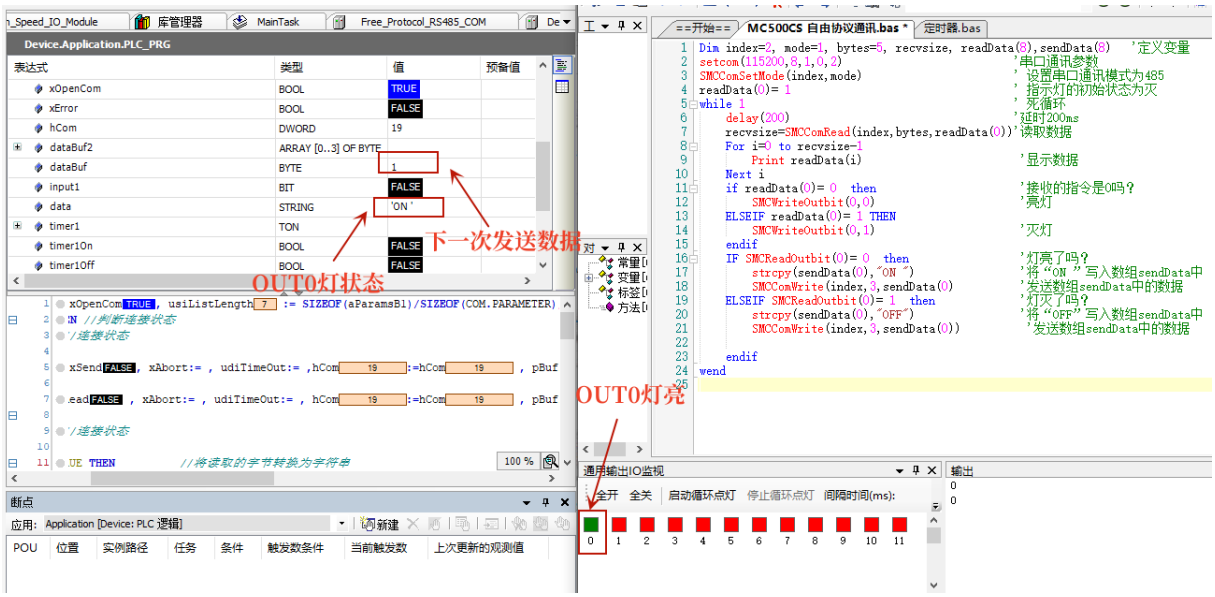


图 11.30 程序运行结果

11.4. RS232 Modbus RTU 通讯

MC300CS 系列 PLC 也可以使用 RS232 串行接口与触摸屏等其他外设进行 Modbus RTU 通讯，但 RS232 接口只能连接一个 Modbus RTU 从站设备。

RS232 Modbus RTU 通讯除硬件接线方法不同外，其他与 RS485 接口进行 Modbus 协议通讯的方法基本相同。本节讲解 MC300CS PLC 分别作 RS232 Moudbus RTU 主站和从站的配置方法。

11.4.1. RS232 Moudbus RTU 主站设置

1. 使能 MC300CS 作为 Modbus RTU 主站

鼠标双击项目树中的“通讯配置”，如图 11.31 所示，单击 RS232，弹出所支持的通信协议的使能窗口。勾选“Modbus 主站”，使能后，变为红色。Modbus 主站使能后，项目树中，出现了对应的通信配置，双击“ModbusCOM_Master_0(ModbusCOM RS-232 Master)”。可以设置 RS232 RTU 主站参数，如图 11.32 所示。

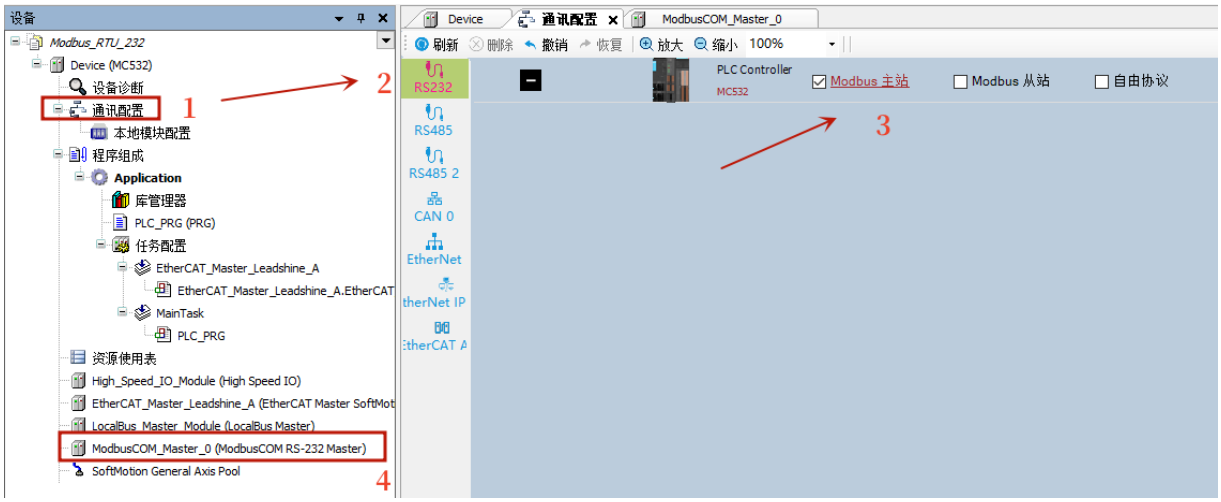


图 11.31　使能 MC300CS 作为 RS232 的 Modbus RTU 主站

图 11.32　RS232 Modbus RTU 主站配置

2. 添加 Modbus RTU 从站设备

使能 PLC 作为 RS232 Modbus RTU 主站后，从右侧“网络设备”中双击“ModbusCOM Slave”从站设备，或者直接拖拽到蓝色区域都可以成功添加从站设备，如图 11.33 所示。从站设备添加后，左侧项目树中出现了对应的通信配置“ModbusCOM_Slave(ModbusCOM Slave)”，如图 11.33 中标注 3 所示。

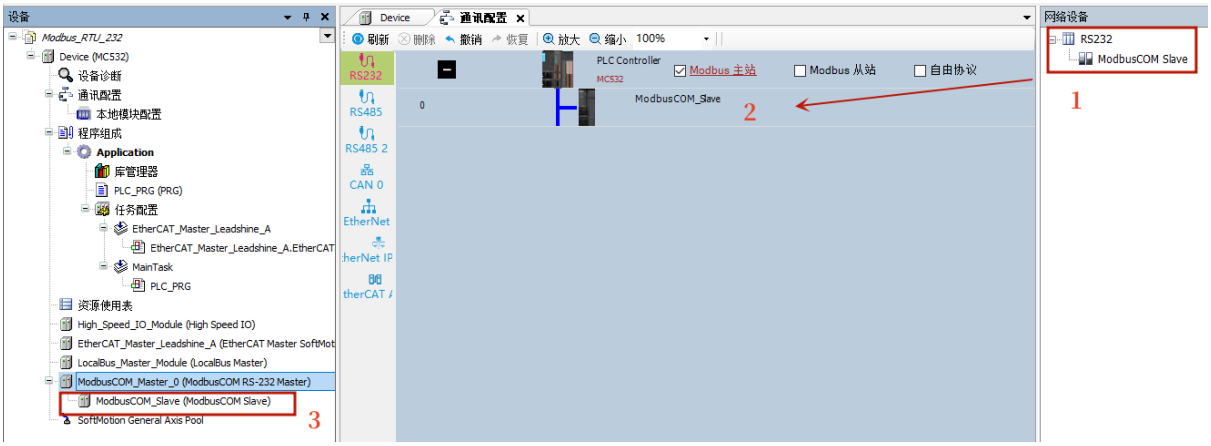


图 11.33　添加 RS232 从站设备

3. 其它相关参数设置

其它相关参数设置方法与 RS485 Modbus RTU 主站通讯的相同，请参见 11.2 节。

11.4.2. RS232 Modbus RTU 从站设置

MC300CS 作 RS232 Moubus RTU 从站时，触摸屏、PLC 或者其它支持 Modbus RTU 通讯的设备可以作为主站，并对 MC300CS PLC 内部存储区的数据进行读写操作。MC300CS Modbus RTU 从站设置方法如下。

1. 使能 MC300CS 作为 Modbus RTU 从站

鼠标双击项目树中的“通讯配置”，如图 11.34 所示，单击“RS232”，弹出所支持的通信协议的使能窗口。勾选“Modbus 从站”，使能后，变为红色。Modbus 从站使能后，左侧项目树中，出现了对应的通信配置“Modbus_Slave_RS232_COM（Modbus Slave RS232 COM）”，如图 11.34 中标注 4 所示。

2. Modbus RTU 从站配置

双击项目树中的从站设备“Modbus_Slave_RS232_COM(Modbus Slave RS232 COM)”，打开 Modbus 从站配置窗口，如图 11.35 所示。

Modbus 从站配置相关参数与配置示例，请参见表 11.5、表 11.6。

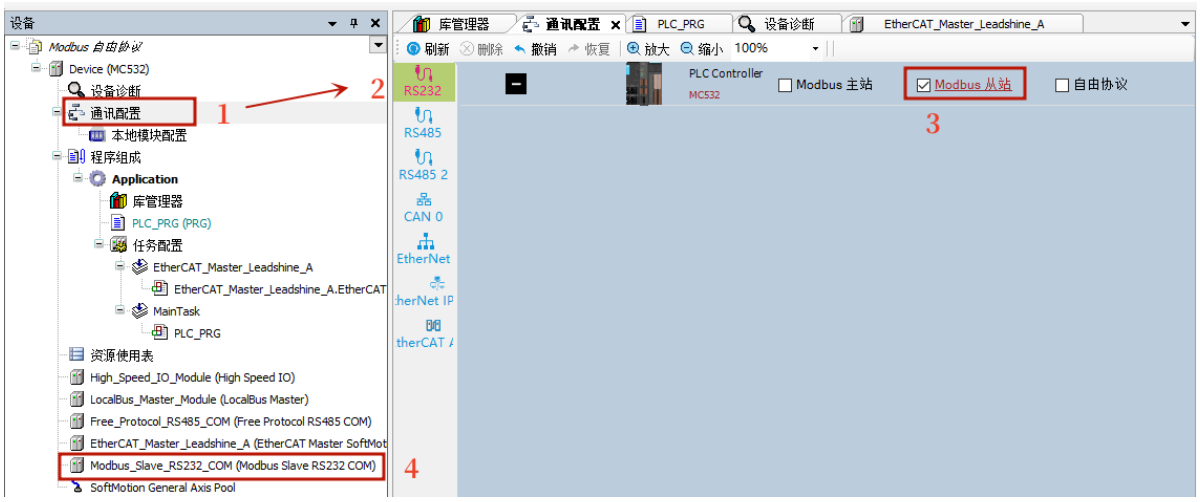


图 11.34　使能 PLC 作为 RS232 Modbus RTU 从站

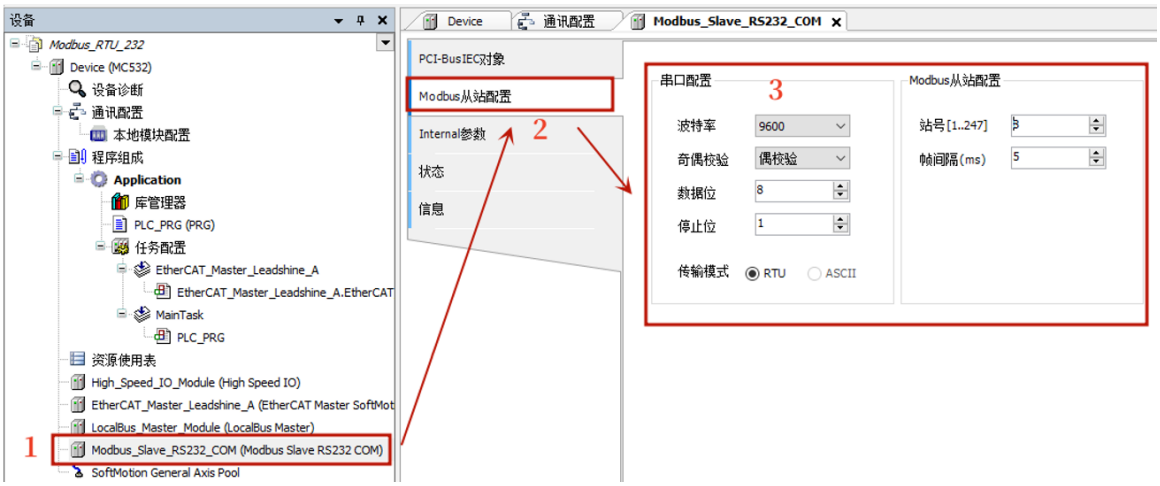


图 11.35　Modbus RTU 从站配置

3. 其它相关参数设置

其它相关参数设置方法与 RS485 Modbus RTU 从站通讯的相同，请参见 11.2 节。

11.5. RS232 自由协议通讯

MC300CS 的 RS232 自由协议通讯与 RS485 自由协议通讯基本相同。只是在选择设备是为 RS232，如图 11.36 所示。

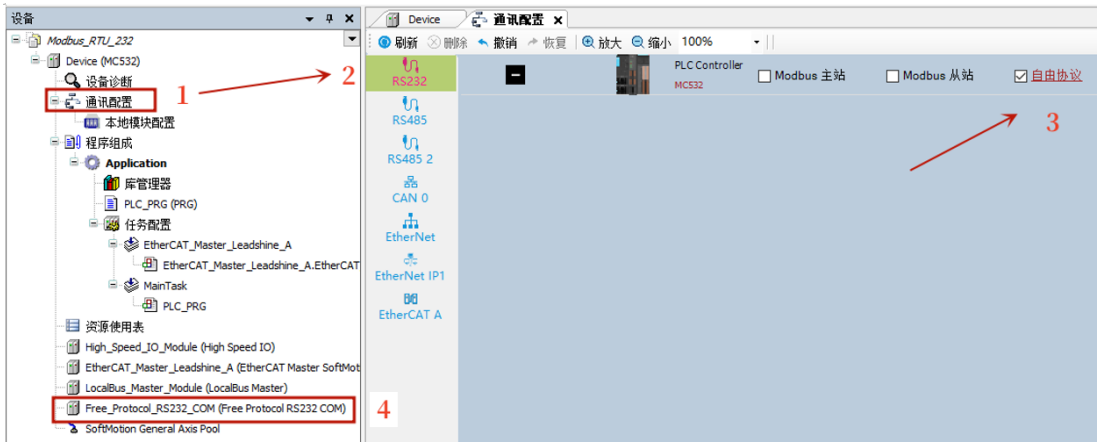


图 11.36　RS232 总线使能自由协议

第12章 以太网通讯

12.1. Modbus TCP 通讯

Modbus TCP 协议是 Modbus 协议中的一种报文类型，采用以太网链路，报文格式在 Modbus RTU 协议的基础上增加一个 MBAP 报文头，并取消了 CRC 校验码。

MC300CS 系列 PLC 通过 100Mbps 的 EtherNET 网口进行 Modbus TCP 通讯，数据交互速度最快能接近 12.5MB/s，实际通讯速度由主从站的帧间隔时间决定，可访问的 PLC 内部寄存器地址与寄存器地址索引规则，请参见 11.1 节。

Modbus TCP 通信可以访问 MC300CS PLC 内部 I、Q、M 区范围请查阅章节 11.1.1。

12.1.1. Modbus TCP 主站通讯及例程

MC300CS 系列 PLC 可作为 Modbus TCP 主站，与其他 PLC、变频器、上位机等作为 Modbus TCP 从站的设备进行通讯。

PLC 作为 Modbus TCP 主站，是发起通讯的一方，可主动对从站通讯数据存储区进行读、写操作。

本例程用两台 MC300CS 系列 PLC 分别作为 Modbus TCP 主站和 Modbus TCP 从站进行数据交互，接线方式如图 12.1 所示。

主站 PLC 对从站 PLC 的数据传输是通过周期循环方式进行数据接收、通过信号触发方式进行数据发送。本例设定主站 PLC 通讯超时时间为 1000ms，从站 PLC 帧响应（即通讯间隔）为 100ms。

主站 PLC 通过读取从站 PLC 的轴状态，判断从站 PLC 的轴是否处于运动状态。当轴处于停止状态时，主站 PLC 向从站发送运动距离、运动速度和启动信号，从站 PLC 接收到这些运动参数后，执行一条相对运动指令，同时以周期循环方式将当前位置和速度以及轴状态返回给主站。

设置 MC300CS 系列 PLC 作为 Modbus TCP 主站的主要步骤有：使能 PLC 作为 Modbus TCP 主站、配置主站参数、添加 Modbus TCP 从站设备、配置从站参数、添加通讯地址以及通讯方式。具体操作方法如下。

MC300CS 系列 PLC 作为 Modbus TCP 从站的配置操作请见下节内容。

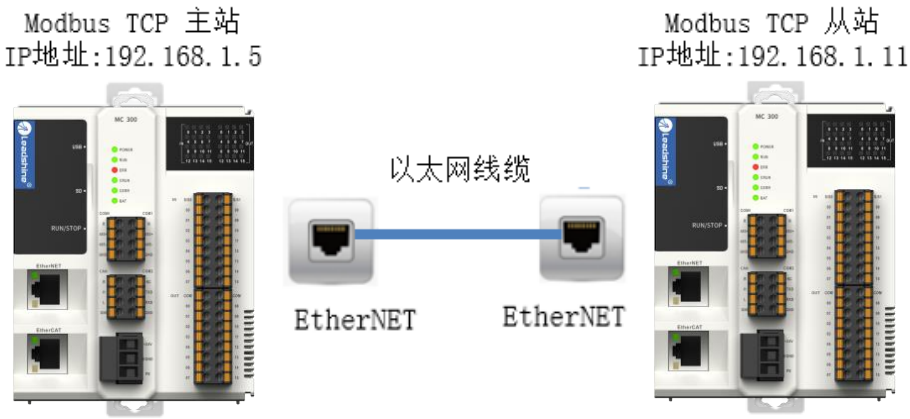


图 12.1　主从站接线方式

1. 使能 PLC 作为 Modbus TCP 主站

双击左侧项目树中的“通讯配置”，选择 EtherNet 通讯，勾选“ModbusTCP 主站”，使能后项目树中，出现了对应的通讯设备“ModbusTCP_Master_Device”，如图 12.2 所示。

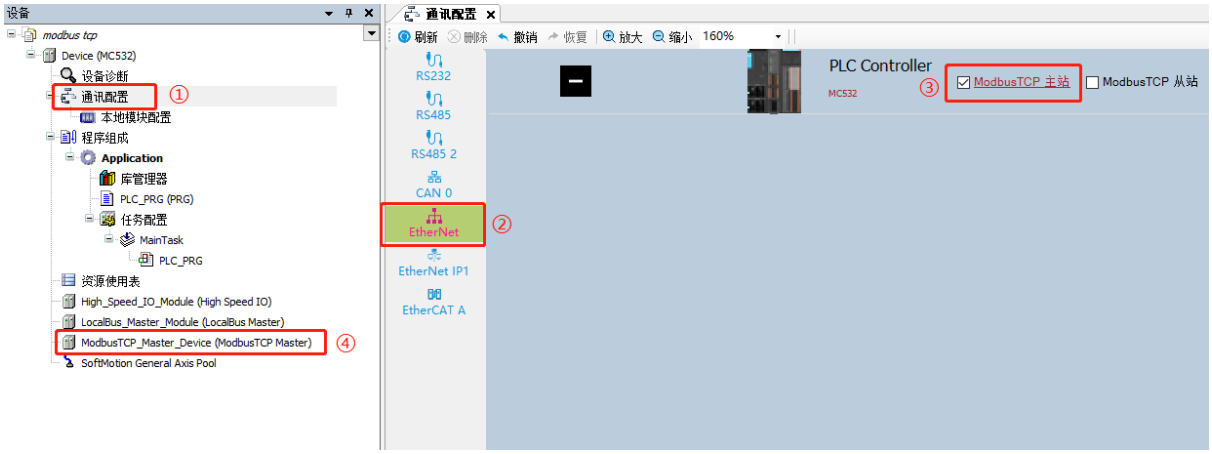


图 12.2　使能 PLC 作为 Modbus TCP 主站

2. Modbus TCP 主站参数配置

双击项目树中的主站设备“ModbusTCP_Master_Device”，选择“ModbusTCP 主站配置”打开配置窗口，如图 12.3 所示。

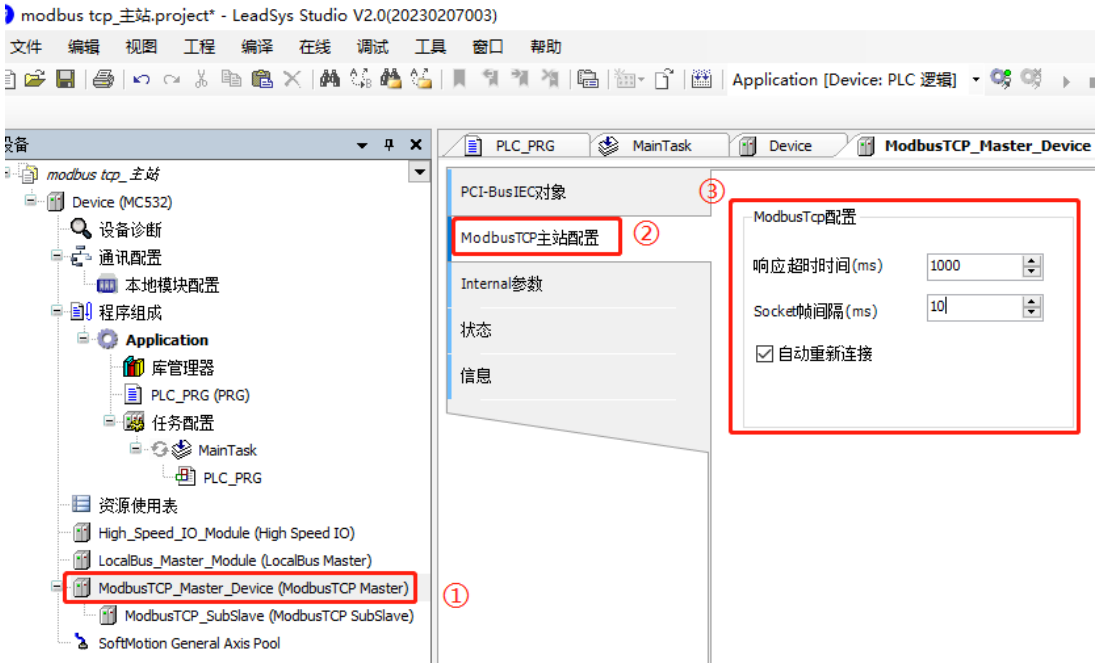


图 12.3　Modbus 主站配置

其中，超时时间为主站等待从站响应的时间，当从站在 1000ms 以内没有发出任何通讯信号时，主站将报错并开始自动重新建立连接；Socket 帧间隔为主站发送数据帧的时

间间隔，该参数越小通讯速度越快，最小值为 1ms。

3. 添加 Modbus TCP 从站设备

完成 Modbus TCP 主站配置后，如图 12.4 所示，从右侧“网络设备”→“Ethernet”中拖入或双击“ModbusTCP SubSlave”添加从站设备。添加从站设备后，项目树中出现了对应的通信设备“ModbusTCP_SubSlave”，如果有多个从站设备，可重复以上步骤进行添加。

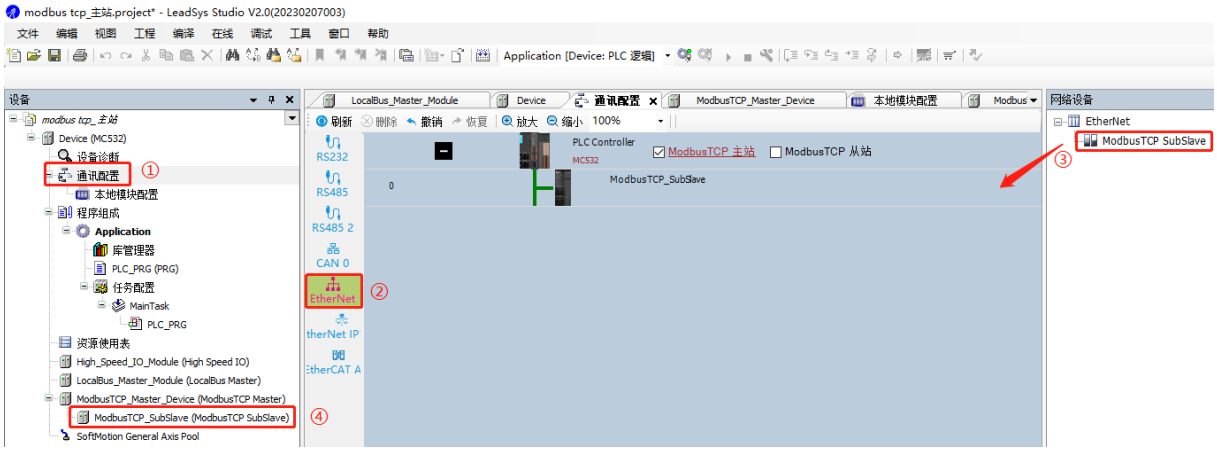


图 12.4　添加从站设备

4. Modbus TCP 从站设备配置

双击项目树中的从站设备“ModbusTCP_slave”，选择“ModbusTCP 从站配置”，打开配置窗口，如图 12.5 所示，设置从站的 IP 地址、端口号、从站地址和超时时间。参数详细说明如表 12.1 所示。

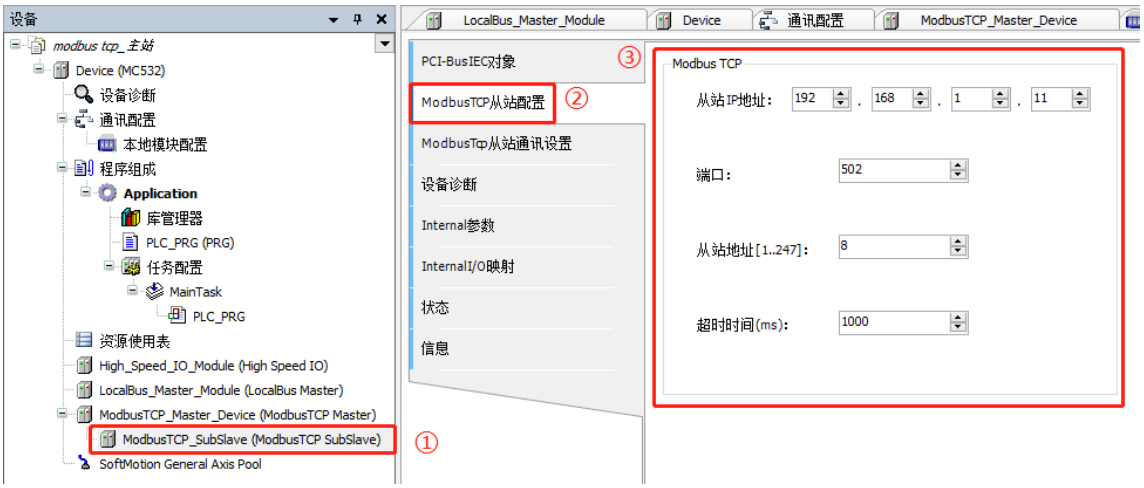


图 12.5　ModbusTCP 从站配置

表 12.1　Modbus TCP 从站配置参数

配置项	参数说明
从站 IP 地址	主站连接 Modbus TCP 从站的 IP 地址
端口	主站连接 Modbus TCP 从站的端口号
从站地址	主站连接 Modbus TCP 从站的站号

超时时间	主站等待从站响应的时间，单位为毫秒(ms)

先添加通道 Channel，再将发送给从站的数据：Start、Distance、Vel，读取从站的数据：State、Xpos、Xspeed、DoneCount 填入 Modbus 从站通讯设置表中，如图 12.6 所示。

Modbus 从站 Internal I/O 映射等其他配置，基本与 Modbus RTU 相同，如图 12.7 所示，其中③为发送给从站的数据：Distance、Vel 为触发写入，Write 变量映射到触发控制位；详细操作过程可参考 11.2 节内容。

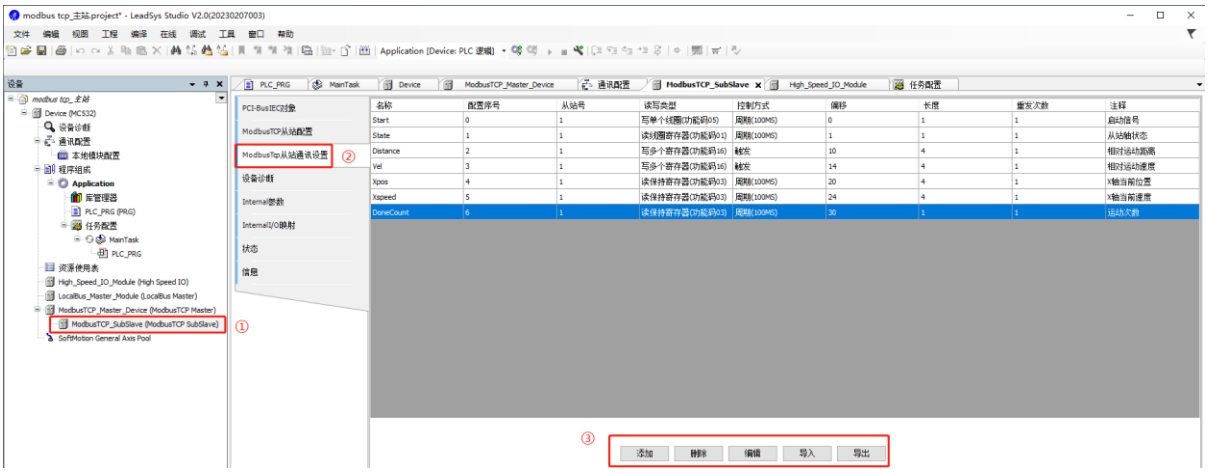


图 12.6　ModbusTCP 从站通讯设置

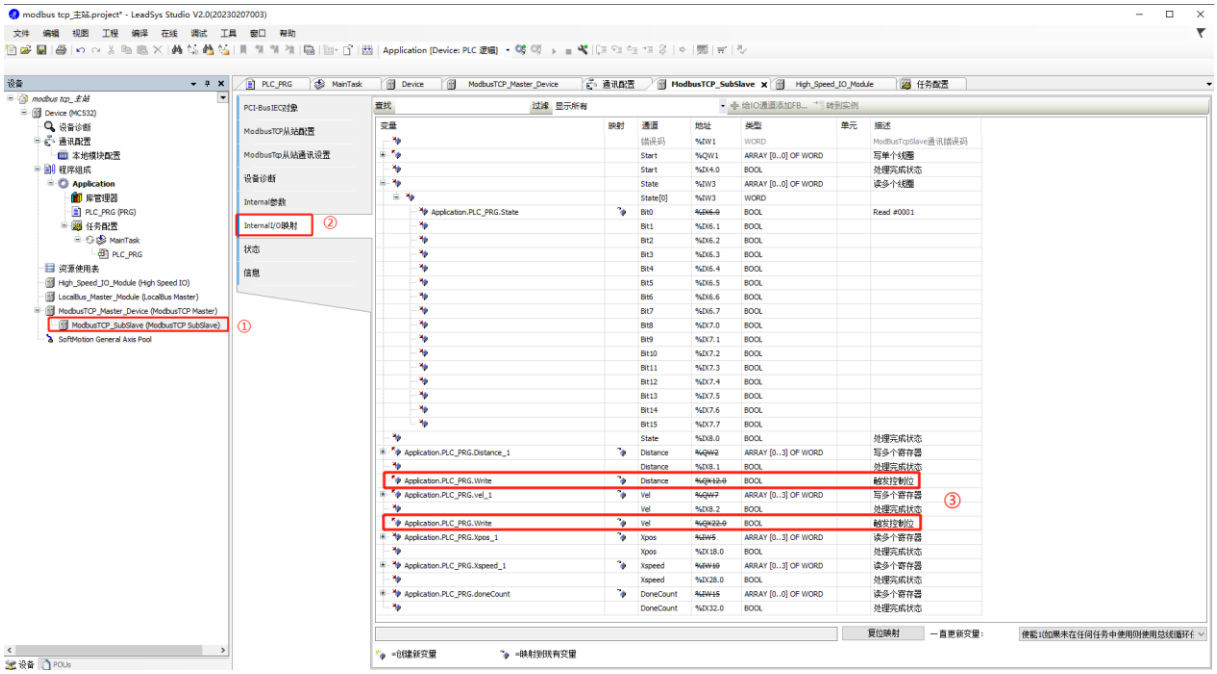


图 12.7　ModbusTCP 从站通讯地址映射

5. 程序代码

1） Modbus TCP 主站 PLC 程序

```
PROGRAM PLC_PRG
VAR
```

```
    Start: BOOL;                    // 启动信号
    State: BOOL;                   // 从站状态，为通讯映射变量
    Write: BOOL;                   // 写入信号
    Distance:LREAL:=10;            // 相对运动距离
    Distance_1:LWORD;              // 相对运动距离，为通讯映射变量
    vel:LREAL:=5;                  // 运动速度
    vel_1:LWORD;                   // 运动速度，为通讯映射变量
    Xpos:LREAL;                    // X 轴的当前位置
    Xpos_1 :ARRAY[0..3]OF WORD ;   // X 轴当前位置的数据;
    Xspeed:LREAL;                  // X 轴的速度
    Xspeed_1 :ARRAY[0..3]OF WORD ;// X 轴当前速度的数据;
    doneCount : INT;               // 运动完成次数
END_VAR

//循环读取从站返回位置和速度：
//从站 X 轴当前位置为 4 个 WORD 型整数，将其转换为有三位小数的实数：
Xpos:=(Xpos_1[0]+Xpos_1[1]*EXPT(2,16)+Xpos_1[2]*EXPT(2,32)+Xpos_1[3]*EXPT(2,64))/1000;
//从站 X 轴当前速度为：
Xspeed:=(Xspeed_1[0]+Xspeed_1[1]*EXPT(2,16)+Xspeed_1[2]*EXPT(2,32)+Xspeed_1[3]*EXPT(2,
64))/1000;
IF  Start=TRUE AND State=FALSE THEN      // 当从站处于运动停止状态且启动信号为 true 时
    Distance:=12.345;                    // 运动距离
    vel:=6.789;                          // 运动速度
    Distance_1:=TO_LWORD( Distance*1000);// 将浮点数转换为整数,保留小数点后三位
    vel_1:=TO_LWORD( vel*1000);          // 将浮点数转换为整数,保留小数点后三位
    Write:=TRUE;                         // 触发写入信号
ELSE
    Write:=FALSE;                        // 状态复位
    start:=FALSE;
END_IF
```

2） Modbus TCP 从站 PLC 程序

```
PROGRAM PLC_PRG
VAR
    axes:dut_pulse_axis;                    // 脉冲轴结构体
    MC_MoveRelative_1: MC_MoveRelative;     // 相对运动实例化
    MC_Power_1: MC_Power;                   // 轴使能实例化
    start AT %QX0.0: BOOL;                  // 启动信号，为通讯变量
    Distance AT %MW10 :ARRAY[0..3]OF WORD ;// 相对运动距离，为通讯变量
    Distance_1: LREAL;                      // 相对运动距离
    vel AT %MW14 :ARRAY[0..3]OF WORD;       // 相对运动速度，为通讯变量
    vel_1: LREAL;                           // 相对运动速度
    State  AT %QX0.1:BOOL;                  // 从站状态，为通讯变量
    Xpos_1 AT %MW20: LWORD;                 // X 轴当前位置，为通讯变量
    Xspeed_1 AT %MW24: LWORD;               // X 轴当前速度，为通讯变量
    doneCount AT %MW30: INT;                // 完成次数，为通讯变量
    R_TRIG_1: R_TRIG;                       // 上升沿计数
END_VAR

axes.pulaxis_0:=ADR(X); // 获取 X 轴的地址
```

```
LS_MotionControl_P(stAxis:=axes, xClearErr:=, fLimtAxisSpeedJump:=,
xDone=>, xError=>, eErrorID=>, xLimitAxisMoveFlag=> ); // 绑定脉冲轴
//轴使能：
MC_Power_1(Axis:=X ,Enable:=1 , bRegulatorOn:=1 ,bDriveStart:=1 , Status=> ,
          bRegulatorRealState=>, bDriveStartRealState=>, Busy=>, Error=>, ErrorID=> );
// 将 X 轴当前位置、速度写入通讯变量，发送到主站：
Xpos_1:=TO_LWORD( X.fActPosition *1000); // 将 X 轴当前位置转换成整数,保留小数点后三位
Xspeed_1:=TO_LWORD( X.fActVelocity *1000);// X 轴当前速度
// 读取主站发送的数据：
Distance_1:=(Distance[0]+Distance[1]*EXPT(2,16)+Distance[2]*EXPT(2,32)+Distance[3]*
          EXPT(2,64))/1000;//相对运动距离，将 4 个 WORD 型数据转换为保留三位小数的实数。
vel_1:=(vel[0]+vel[1]*EXPT(2,16)+vel[2]*EXPT(2,32)+vel[3]*EXPT(2,64))/1000;
                                                          // 相对运动速度
MC_MoveRelative_1(Axis:=X , Execute:=start , Distance:= Distance_1, Velocity:=vel_1 ,
                  Acceleration:=100 , Deceleration:=100 , Jerk:=1000 , BufferMode:= ,
                  Done=> , Busy=>State , Active=> , CommandAborted=> , Error=> ,
                   ErrorID=> ); // 相对运动指令
// 获取运动完成信号的上升沿：
R_TRIG_1(CLK:=MC_MoveRelative_1.Done=TRUE , Q=> );
IF R_TRIG_1.Q THEN
    doneCount:=doneCount+1;       // 更新完成次数
END_IF
```

6. 程序运行结果

当启动信号 Start 为 TRUE 时，读取到从站 PLC 的轴处于停止状态，主站 PLC 向从站发送运动距离、运动速度和启动信号；从站 PLC 接收到数据后，以这些运动参数执行一条相对运动指令；并以周期循环方式向主站发送当前位置和速度以及轴状态。如图 12.8 所示。

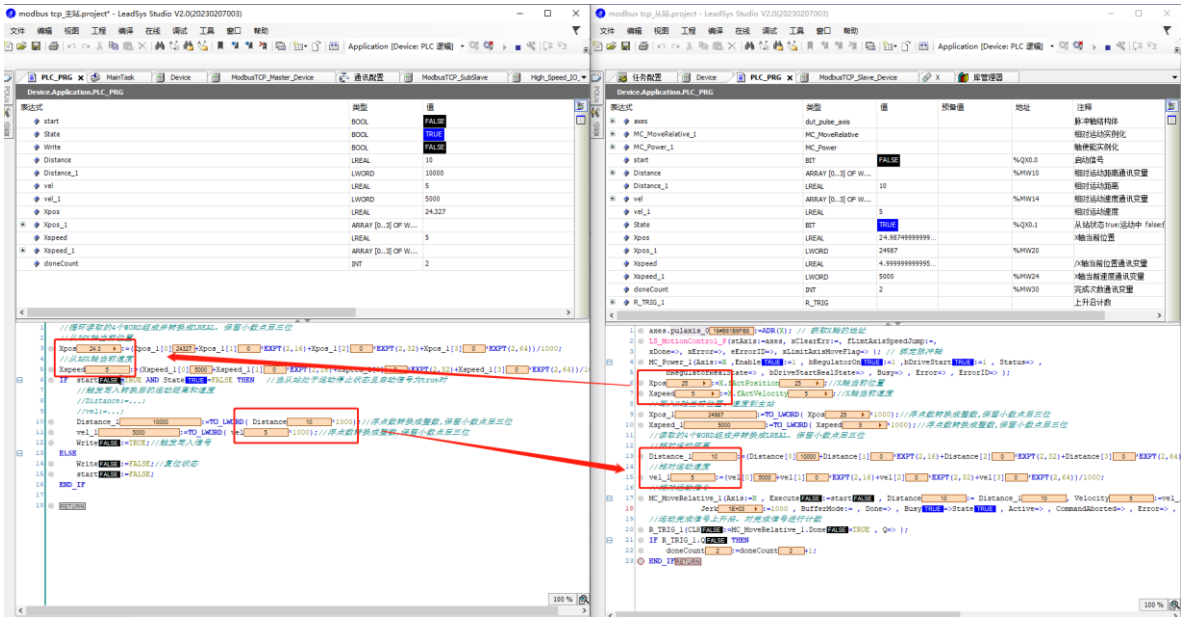


图 12.8　程序运行结果

12.1.2. Modbus TCP 从站通讯及例程

PLC 作为 Modbus TCP 从站时，为 Modbus TCP 响应通讯的一方，等待主站发起数据读写。

该例程中雷赛 LT2070T 触摸屏作为 Modbus TCP 主站，MC300CS 系列 PLC 作为 Modbus TCP 从站，接线方式如图 12.9 所示。新建工程，添加用于通讯的变量地址，包括 X 轴速度、Y 轴速度、Z 轴速度、X 轴位置、Y 轴位置、Z 轴位置。当通讯成功时，将上述变量的当前值显示在触摸屏上。

MC300CS 系列 PLC 作为 Modbus TCP 从站时，需要使能 PLC 作为 Modbus TCP 从站，配置从站参数，具体操作如下。

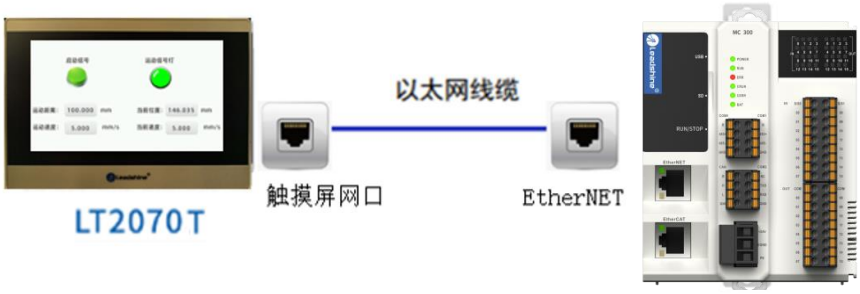


图 12.9　Modbus TCP 通讯接线方式

1. 使能 PLC 作为 Modbus TCP 从站

双击项目树中的“通讯配置”，选择 EtherNet 通讯，勾选“ModbusTCP 从站”，使能后项目树中出现了对应的通讯设备“ModbusTCP_Slave_Device”，如图 12.10 所示。

图 12.10　使能 PLC 作为 ModbusTCP 从站

2. Modbus TCP 从站配置

双击项目树中的从站设备“ModbusTCP_Slave_Device”，打开从站参数设置窗口，如图 12.11 所示。Modbus 从站设置相关参数说明，见表 12.2 所示。

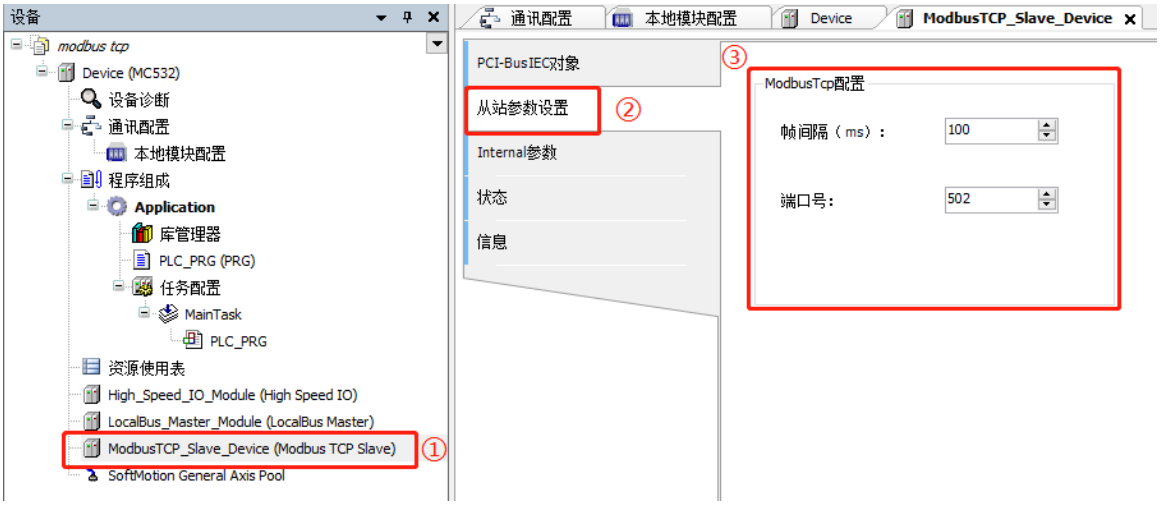


图 12.11　使能 PLC 作为 ModbusTCP 从站

表 12.2　从站设置参数说明

配置项	功能
帧间隔	Modbus TCP 从站接收通信帧后延迟时间发送回复帧。
端口号	Modbus TCP 从站的 TCP 端口号

例程的代码如下。

```
VAR   //触摸屏通讯变量声明
    Start_HMI AT %MX100.0: BOOL;     //启动(触摸屏按钮)，地址为 MX100.0
    Speed  AT %MD0:LREAL;            //合速度，  地址为 MD0
    Xspeed AT %MD2:LREAL;            //X 轴速度，地址为 MD2
    Yspeed AT %MD4 :LREAL;           //Y 轴速度，地址为 MD4
    Zspeed AT %MD6:LREAL;            //Z 轴速度，地址为 MD6
    Xpos AT %MD8:LREAL;              //X 轴位置，地址为 MD8
    Ypos AT %MD10:LREAL;             //Y 轴位置，地址为 MD10
    Zpos AT %MD12:LREAL;             //Z 轴位置，地址为 MD12
END_VAR
```

3. 触摸屏配置

打开雷赛触摸屏软件 LT Studio，新建触摸屏程序文件，选择型号为 LT2070T，添加“网络 PLC”输入 PLC 的 IP 地址和端口号，具体配置如图 12.12 所示。

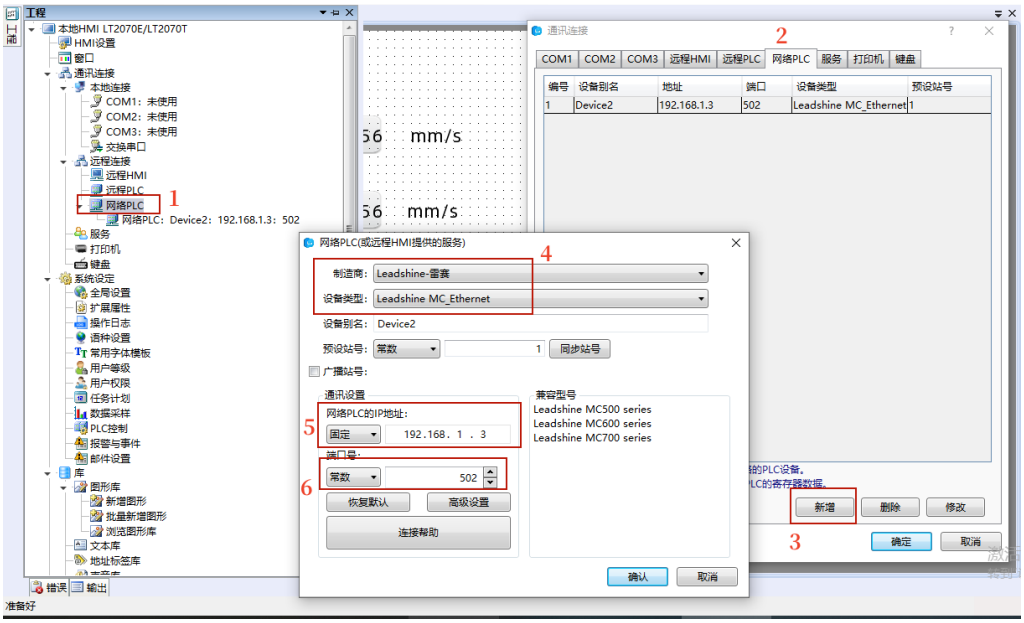


图 12.12　LT Studio 软件界面

根据 PLC 程序的通讯变量声明地址，在触摸屏界面依次添加相关元件，映射通讯变量地址。即：合速度的地址为 MD0、X 速度的地址为 MD2、Y 速度的地址为 MD4、Z 速度的地址为 MD6，X 轴位置的地址为 MD8、Y 轴位置的地址为 MD10、Z 轴位置的地址为 MD12。操作如图 12.13 所示。

图 12.13　新增数值显示的元件

当 PLC 运行时，用于与触摸屏通讯的变量会将当前值显示在触摸屏上，如图 12.14 所示。

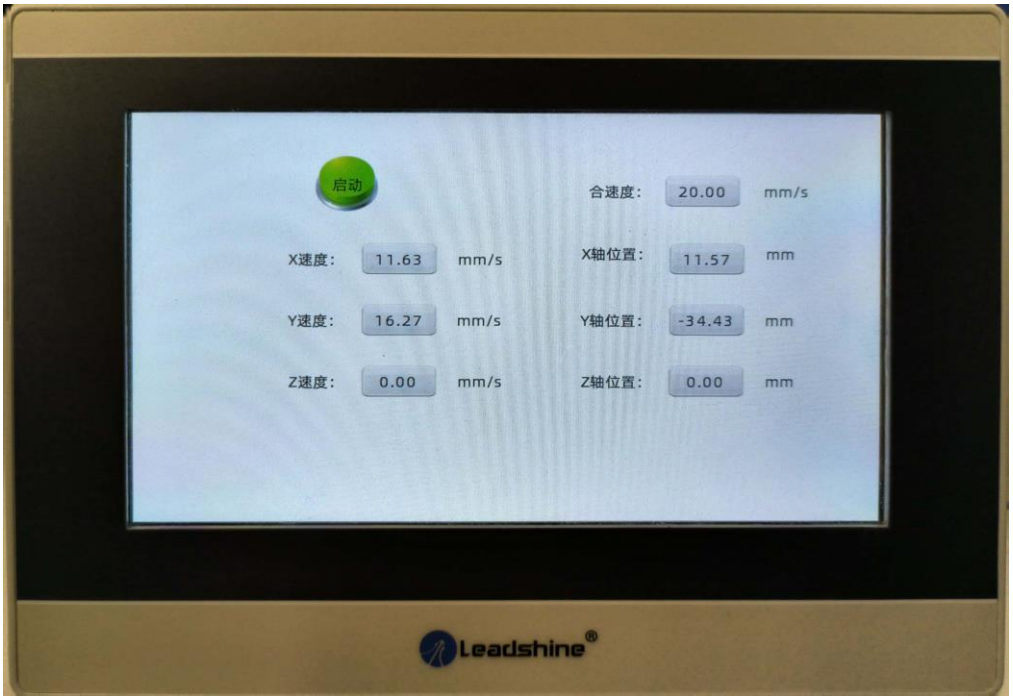


图 12.14　程序运行结果

12.1.3. Modbus TCP 通讯错误码与常见故障

Modbus TCP 错误码(40001~40017)以及查看错误码的方式与本手册介绍“Modbus RTU 常见故障与错误码”相同，错误码定义见表 11.8。

Modbus TCP 主站连接 Modbus TCP 从站时发生的主要故障如下：

1） Modbus TCP 主站连接 Modbus TCP 从站配置 IP 不正确，导致主站与从站通信无法建立。

2） Modbus TCP 主站访问 Modbus TCP 从站非法地址，返回错误应答。

3） Modbus TCP 主站操作 Modbus TCP 从站写寄存器，但是 Modbus TCP 从站该寄存器只支持读不支持写操作，Modbus TCP 主站会收到 Modbus TCP 从站返回的出错应答。

12.2. TCP 自由协议通讯及例程

自由协议通讯也称为无协议通讯，即通讯双方没有共同的通讯协议，只能临时根据某一方的协议进行发送和接收数据，以达到交换数据的目的。

TCP 是面向连接的传输控制协议，相较于 UDP 具有高可靠性，确保传输数据的正确性，不出现丢失或乱序。

使用 TCP 自由协议指令前需要在工程中添加“CAA Types Extern”与“CAA Net Base Services” 库文件，以及用于操作通讯数据的指令“SysMem”库文件。

12.2.1. TCP 自由协议指令

MC300CS 系列 PLC 的 TCP 自由协议指令如表 12.3 所示。自由协议错误码 eError 说明如表 12.4 所示。详细说明请参考《雷赛大中型 PLC 指令手册》。

表 12.3 自由协议指令列表

指令类别	名称	功能
TCP 自由通讯指令	TCP_Client	TCP 客户端通讯
	TCP_Server	TCP 服务器端通讯
	TCP_Connection	连接到服务器
	TCP_Write	TCP 通讯数据写入
	TCP_Read	TCP 通讯数据接收

1. TCP 客户端通讯 TCP_Client

该指令能将 PLC 设置为客户端，连接到指定的 TCP 服务器，输出的客户端句柄用于建立通讯。其格式如下。

```
NBS.TCP_Client(xEnable:=使能信号, xDone=>完成信号, xBusy=>执行中信号,xError=>错误标志,
               udiTimeOut:=超时时间, ipAddr:=服务器 IP 地址, uiPort:=端口号,
               eError=>错误码, xActive=>连接成功标志, hConnection=>客户端句柄 );
```

2. TCP 服务器端通讯 TCP_Server

该指令能将 PLC 设置为服务器，监听 TCP 客户端。通过输出句柄 hServer 用于通过 TCP_Connection 建立连接，若实例化多个 TCP_Connection 建立连接，可实现一个服务器处理多个客户端连接功能。其格式如下。

```
NBS.TCP_Server(xEnable:=使能信号, xDone=>完成信号, xBusy=>执行中信号,xError=>错误标志,
               ipAddr:= 设置服务器 IP 地址,uiPort:= 端口号,eError=>错误信号,hServer=>
               服务器句柄 );
```

3. 连接到服务器 TCP_Connection

该指令能使用 TCP_Server 指令创建的服务器句柄，与客户端建立连接。若有多个客户端需要与 PLC 建立连接，可通过实例化多个 TCP_Connetction 实现，生成的连接句柄用于读写操作。其格式如下。

```
NBS.TCP_Connection(xEnable:= 使能信号, xDone=> 完成信号, xBusy=> 执行中信号,
                   xError=> 错误标志, hServer:= 服务器句柄, eError=> 错误信号,
                   xActive=> 连接成功标志, hConnection=> 连接句柄 );
```

4. TCP 通讯数据写入 TCP_Write

该指令能对 TCP_Client 指令与 TCP_Connetction 指令中已建立连接的句柄 hConnection 进行写入数据操作。其格式如下。

```
NBS.TCP_Write(xExecute:= 启动信号, udiTimeOut:= 超时时间, xDone=> 完成信号 ,
              xBusy=> 执行中信号, xError=> 错误标志, hConnection:=连接句柄 ,
              szSize:= 数据大小, pData:= 发送数据缓存区指针, eError=>错误信号 );
```

5. TCP 通讯数据写入 TCP_Read

该指令能对 TCP_Client 指令与 TCP_Connetction 指令中已建立连接的句柄 hConnection 进行读取数据操作。其格式如下。

```
NBS.TCP_Read(xEnable:=使能信号, xDone=>完成信号, xBusy=>执行中信号, xError=>错误标志,
             hConnection:= 连接句柄, szSize:= 读取数据的大小,
             pData:= 读取数据缓存区指针, eError=> 错误信号,
             xReady=> 读取数据成功标志, szCount=> 实际读取的字数 );
```

表 12.4 自由协议错误码说明

错误码	定义	描述
0	NO_ERROR	无错误
6000	FIRST_ERROR	保留
6001	TIME_OUT	超时
6002	INVALID_ADDR	客户端所连接服务器的 IP 地址无效
6003	INVALID_HANDLE	连接句柄无效
6004	INVALID_DATAPOINTER	数据指针无效
6005	INVALID_DATASIZE	数据大小无效
6006	UDP_RECEIVE_ERROR	UDP 接收错误
6007	UDP_SEND_ERROR	UDP 发送错误
6008	UDP_SEND_NOT_COMPLETE	UDP 发送未完成
6009	UDP_OPEN_ERROR	UDP 打开错误
6010	UDP_CLOSE_ERROR	UDP 关闭错误
6011	TCP_SEND_ERROR	TCP 发送错误
6012	TCP_RECEIVE_ERROR	TCP 发送未完成
6013	TCP_OPEN_ERROR	TCP 打开错误
6014	TCP_CONNECT_ERROR	TCP 连接错误
6015	TCP_CLOSE_ERROR	TCP 关闭错误
6016	TCP_SERVER_ERROR	TCP 服务器错误
6017	WRONG_PARAMETER	参数错误
6018	ERROR_UNKNOWN	未知错误
6019	TCP_NO_CONNECTION	无 TCP 连接
6020	LOCTL_ERROR	内部错误（本机不支持）
6050	FIRST_MF	保留
6099	LAST_ERROR	保留

12.2.2. TCP 自由协议通讯例程

例 1：PLC（服务器）与 PC 机（客户端）的通讯

该例程中 MC300CS 系列 PLC 作为服务器与 PC 机进行通讯，以“通讯连接”、“数据发送”、“数据接收”这三个功能为例，程序执行逻辑流程如图 12.15 所示。其中 PLC 与 PC 机通讯成功后，PC 机向 PLC 发送“Hellow,Nice to meet you!”,PLC 接收成功后向 PC 机发送“Nice to meet you too!”

其中变量 strSendData、strRecvData 为发送和接收的数据，数据类型为 string，根据具体使用情况，也可以使用其它数据类型或结构体。若发送数据量较大，可修改发送和接收的数组长度。

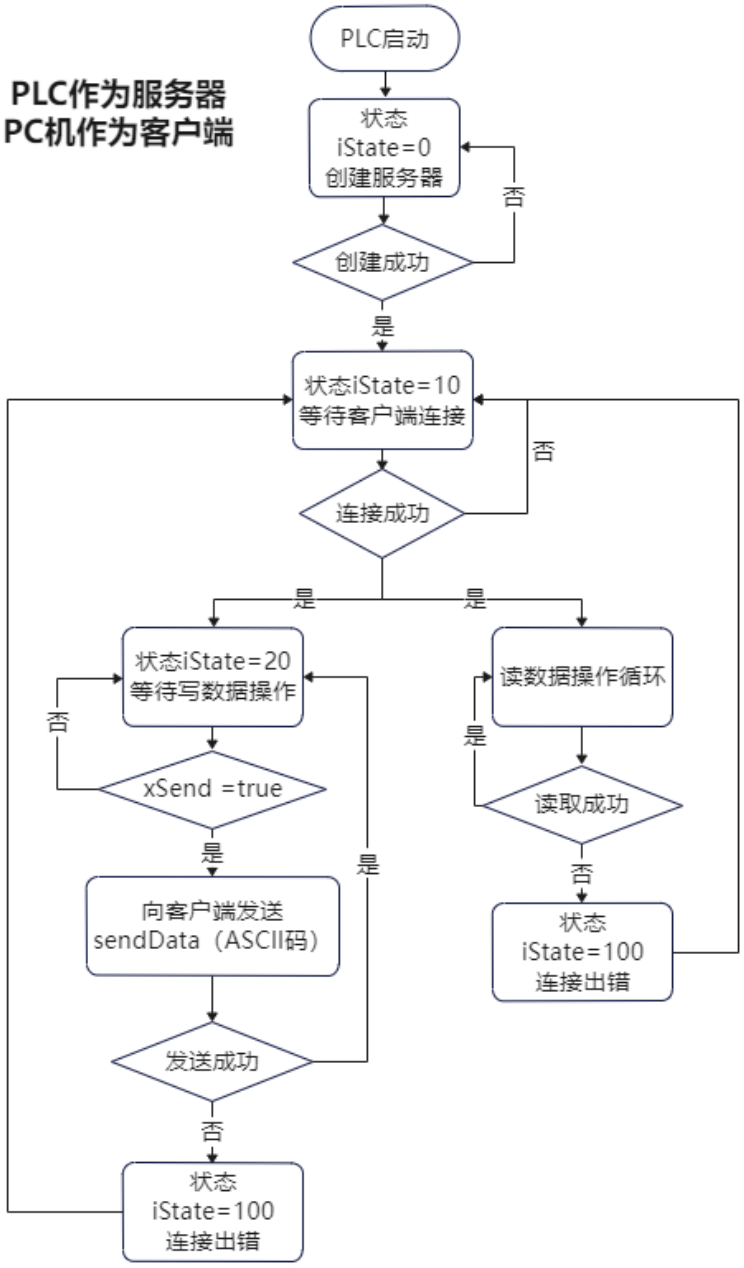


图 12.15 程序执行逻辑流程图

PC 机与 PLC 通过网线连接，如图 12.16 所示；其中 PLC 的 IP 地址为 192.168.1.3，

PC 机 IP 地址为 192.168.1.5，设置方式如图 12.17 所示。

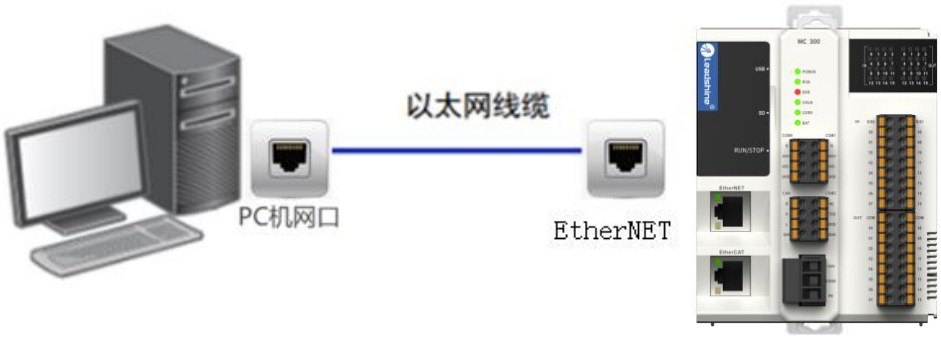


图 12.16　PC 机与 PLC 通过以太网口连接

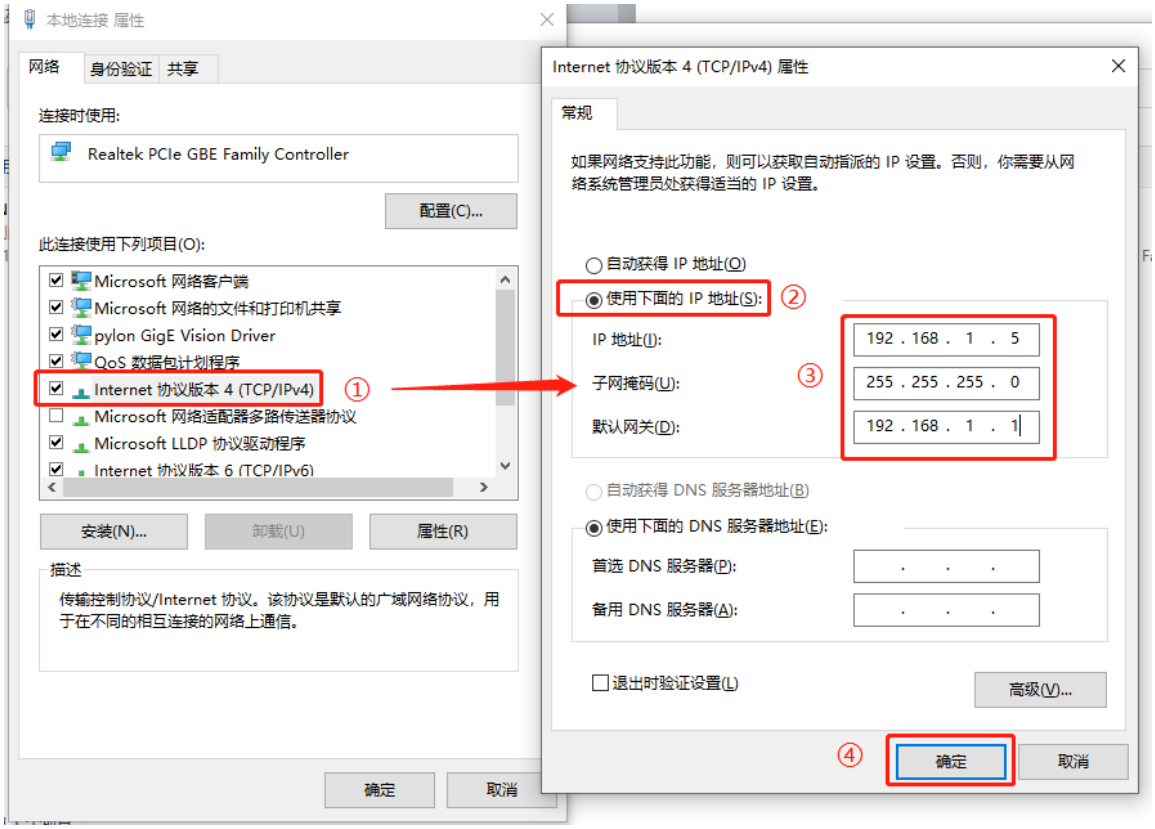


图 12.17　在 Windows 中设置 PC 机的 IP 地址

PLC 作为服务器时，如果需要连接多个客户端，可以实例化多个 TCP_Connect，将 TCP_Server 的句柄 hServer 输入给每个 TCP_Connect。成功建立连接后，将 TCP_Connect 的句柄 hConnection 输入给 TCP_Read 和 TCP_Write 即可进行收发数据。

1）PLC（服务器）程序

```
PROGRAM PLC_PRG  //声明变量名称
VAR
    (******* TCP_Server ********)
    tcpSrver: NBS.TCP_Server;                    //服务器实例化
    IP: NBS.IP_ADDR:=(sAddr:='192.168.1.3');     //本机 PLC 的 IP 地址
    Port: UINT:=8000;                            //端口号
    (******* TCP_Connection ********)
```

```
    tcpConnect: NBS.TCP_Connection;                 //获取连接句柄
    (********* 数据读写 ********)
    read: NBS.TCP_Read;                             //读数据函数
    write: NBS.TCP_Write;                           //写数据函数
    recvData:ARRAY[0..255]OF BYTE;                  //读数据区
    sendData:ARRAY[0..255]OF BYTE;                  //写数据区
    (********* 收发内容 ********)
    strRecvData: STRING;                            //自定义接收数据（类型可改）
    strSendData: STRING;                            //自定义发送数据（类型可改）
    xSend: BOOL;                                    //写数据信号
    iState: INT:=0;                                 //程序运行状态位
    ErrorID: NBS.ERROR;                             //错误标志
    xSendOld: BOOL;                                 //上一次写数据信号
    xError: BOOL;                                   //错误状态
    ton:TON;                                        //重连定时器
    realReadSize: NBS.CAA.SIZE;                     //读取数据大小
END_VAR

tcpSrver(xEnable:= , xDone=> , xBusy=> , xError=> , ipAddr:= IP,
         uiPort:= Port, eError=> , hServer=> ); //开启 TCP 服务器模式
tcpConnect( xEnable:= , xDone=> , xBusy=> , xError=> , hServer:= tcpSrver.hServer,
            eError=> , xActive=> , hConnection=> );                    //获取连接句柄
read(xEnable:= tcpConnect.xActive, xDone=> , xBusy=> , xError=> ,
     hConnection:= tcpConnect.hConnection, szSize:= SIZEOF(recvData),
     pData:= ADR(recvData), eError=> , xReady=> , szCount=> );          //读取数据
write(xExecute:= xSendOld, udiTimeOut:= , xDone=> , xBusy=> , xError=> ,
    hConnection:= tcpConnect.hConnection, szSize:= SIZEOF(sendData),
    pData:= ADR(sendData), eError=> );                                  //写入数据
CASE iState OF                                      //循环状态判断
    0:                                  //建立服务器：
        tcpSrver.xEnable:=TRUE;
        IF tcpSrver.hServer <> 0 THEN               //服务器状态正常
            iState:=10;                             //跳转至创建连接步骤
        END_IF
        IF tcpSrver.xError THEN                     //建立服务器错误
            ErrorID:=tcpSrver.eError;
            iState:=100;                            //跳转至连接出错重连步骤
        END_IF
    10:                                 //创建连接：
        tcpConnect.xEnable:=TRUE;                   //使能获取连接句柄
        IF tcpConnect.xActive THEN                  //连接激活
            iState:=20;                             //跳转至写数据步骤
            xError :=FALSE;
        END_IF
        IF tcpConnect.xError THEN                   //连接错误
            ErrorID:=tcpConnect.eError;
            iState:=100;                            //跳转至连接出错重连步骤
        END_IF
    20:                                 //写数据：
        SysMem.SysMemCpy(ADR(sendData),ADR(strSendData),SIZEOF(strSendData));
```

```
                                        //将写入的数据（BYTE）转换为字符串（类型可改）
        IF xSend AND NOT xSendOld THEN  //上升沿激活写入状态
            SysMem.SysMemSet(ADR(recvData),0,SIZEOF(recvData)); //清除读数据区
        END_IF
        xSendOld:=xSend;                        //保存上次状态（用于产生上升沿）
        IF write.xError THEN                    //写入错误
            ErrorID:=write.eError;
            iState:=100;                        //跳转至连接出错重连步骤
        END_IF
    100:                                //连接出错后定时一秒后重连：
        xError:=TRUE;
        tcpConnect.xEnable:=FALSE;
        ton.IN:=TRUE;
        IF ton.Q THEN
            iState:=10;                          //跳转至创建连接步骤
            ton.IN:=FALSE;
        END_IF
END_CASE
//读数据：
IF read.xReady THEN                             //准备读取数据
    realReadSize:=read.szCount;                 //读取数据的大小
    SysMem.SysMemSet(ADR(sendData),0,SIZEOF(sendData));          //清除写数据区
    SysMem.SysMemSet(ADR(strRecvData),0,SIZEOF(strRecvData));    //清空读取数据
    //将读取的数据（BYTE）转换为字符串：
    SysMem.SysMemCpy(ADR(strRecvData),ADR(recvData),realReadSize);
END_IF
IF read.xError THEN                             //读取出错
    ErrorID := read.eError;
    iState := 100;                          //跳转至连接出错重连步骤
END_IF
ton(IN:= , PT:=T#1S , Q=> , ET=> );     //连接报错后定时一秒后重连
```

2） 上位机（客户端）例程

① C#Socket 通讯流程和主要 API

a） 组装 Socket 通信的地址和端口，建立通信的对象：

```
IPAddress ipAddress = IPAddress.Parse(textBox1.Text);           //获取 IP 地址
int Port = Convert.ToInt32(textBox2.Text);                      //获取端口号
```

b） 创建 Socket 对象，建立与服务器的 TCP 连接：

```
MySocket.BeginConnect(ipAddress,Port,
                      new AsyncCallback(ConnectedCallback), MySocket);
```

c） Socket TCP 通信数据交互：

```
socket.BeginReceive(TCPBuffer, 0, TCPBufferSize, 0,
                    AsyncCallback(ReadCallback), socket);          //接收数据
MySocket.BeginSend(byteArray, 0, byteArray.Length, 0, null, null); //发送数据
```

d） 关闭 Socket 通信：

```
MySocket.BeginDisconnect(false, null, null);                       //断开连接
```

② 程序窗口如图 12.18 所示

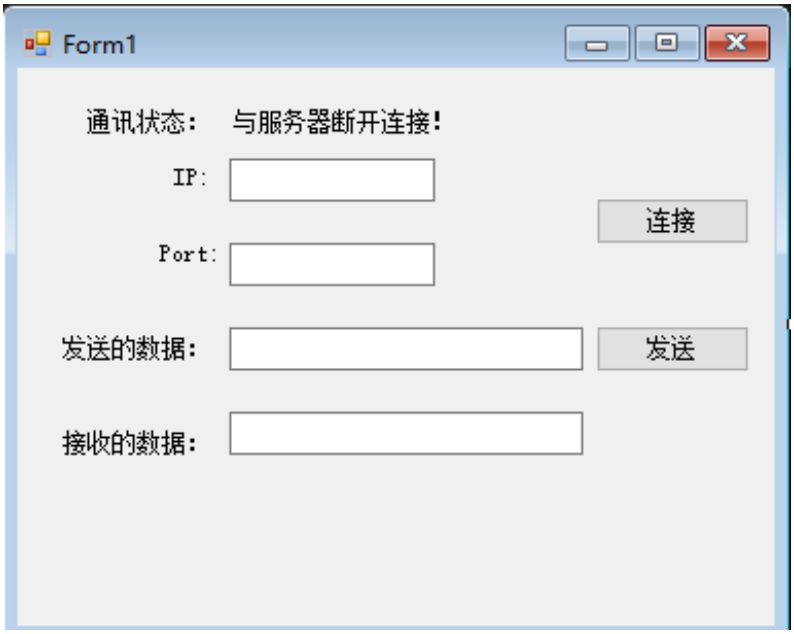


图 12.18 客户端窗口

③ 程序代码：

```
using System;
using System.Net;
using System.Text;
using System.Windows.Forms;
using System.Net.Sockets;                                    //Socket 对应的命名空间
namespace WindowsFormsApp1                                   //文件名称
{
    public partial class Form1 : Form
 {
   private Socket MySocket = null;                           // Socket
   public const int TCPBufferSize = 20000;  //缓存的最大数据个数(PLC 最大缓存数组容量)
   public byte[] TCPBuffer = new byte[TCPBufferSize]; //缓存数据的数组
   string IP = "192.168.1.3";                                //PLC 地址
   int Port = 8000;                                          //PLC 端口号
public Form1()
 {
    InitializeComponent();
  }
private void Form1_Load(object sender, EventArgs e)
{
  textBox1.Text = IP;
  textBox2.Text = Port.ToString(); ;
}
private void button1_Click(object sender, EventArgs e)
{
 if (button1.Text == "连接")
    {   //IP 地址 和 端口号输入不为空
if (string.IsNullOrEmpty(textBox1.Text) == false && string.IsNullOrEmpty(textBox2.Text)
== false)
      {
```

```
        try
       {
        IPAddress ipAddress = IPAddress.Parse(textBox1.Text);//获取 IP 地址
        int Port = Convert.ToInt32(textBox2.Text);          //获取端口号
        MySocket = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
        MySocket.BeginConnect(ipAddress, Port, new AsyncCallback(ConnectedCallback),
        MySocket); //使用 BeginConnect 异步连接
             }
      catch (Exception)
          {
            MessageBox.Show("IP 地址或端口号错误!", "提示");
          }
        }
        else
           {
             MessageBox.Show("IP 地址或端口号为空!", "提示");
           }
          }
        else
           {
         try
           {
              button1.Text = "连接";
              MySocket.BeginDisconnect(false, null, null);//断开连接
              label3.Text = "与服务器连接断开!";          //对话框追加显示数据
           }
           catch (Exception) {}
       }
  }
private void button2_Click(object sender, EventArgs e)
{
  String Str = textBox4.Text.ToString();                    //获取发送文本框里面的数据
  try
   {
    if (Str.Length > 0)
     {
      byte[] byteArray = Encoding.Default.GetBytes(Str);//Str 转为 Byte 值
      MySocket.BeginSend(byteArray, 0, byteArray.Length, 0, null, null);
 //发送数据
      }
     }
     catch (Exception) { }
 }
void ConnectedCallback(IAsyncResult ar)// 连接异步回调函数
 {
  Socket socket = (Socket)ar.AsyncState;
  try
   {
       socket.EndConnect(ar);
```

```
//设置异步读取数据,接收的数据缓存到 TCPBuffer,接收完成跳转 ReadCallback 函数
socket.BeginReceive(TCPBuffer, 0, TCPBufferSize, 0, new AsyncCallback(ReadCallback),
socket);
    Invoke((new Action(() =>
          {
            label3.Text = "成功连接服务器！";     //对话框追加显示数据
             button1.Text = "断开";
             })));
     }
    catch (Exception e)
    {
     Invoke((new Action(() =>
       {
         label3.Text = "连接失败:" + e.ToString();//对话框追加显示数据
        })));
     }
 }
void ReadCallback(IAsyncResult ar)                  // 接收到数据回调函数
 {
  Socket socket = (Socket)ar.AsyncState;           //获取链接的 Socket
  int CanReadLen = socket.EndReceive(ar);          //结束异步读取回调,获取读取的数据个数
  if (CanReadLen > 0)
  {
string str = Encoding.Default.GetString(TCPBuffer, 0, CanReadLen);
//Byte 值根据 ASCII 码表转为 String
    Invoke((new Action(() =>                       //C# 3.0 以后代替委托的新方法
          {
           textBox5.Text = str;                    //对话框追加显示数据
           })));
//设置异步读取数据,接收的数据缓存到 TCPBuffer,接收完成跳转 ReadCallback 函数
socket.BeginReceive(TCPBuffer, 0, TCPBufferSize, 0,
 new AsyncCallback(ReadCallback), socket);
  }
  else//异常
  {
   Invoke((new Action(() =>                        //C# 3.0 以后代替委托的新方法
   {
    button1.Text = "连接";
    label3.Text = "异常断开";                      //对话框追加显示数据
    })));
    try
     {
          MySocket.BeginDisconnect(false, null, null);//断开连接
      }
     catch (Exception) { }
   }
  }
 }
}
```

3） 程序运行结果

作为客户端的 PC 机与作为服务器的 PLC 建立通讯成功后，PLC 通过变量 strRecvData 接收 PC 机发送的字符串数据，如图 12.19 所示 PC 机向 PLC 发送了“Hello,Nice to meet you!”。PLC 通过变量 strSendData 向 PC 机发送字符串数据，如图 12.20 所示 PLC 向 PC 机发送了“Nice to meet you too!”。

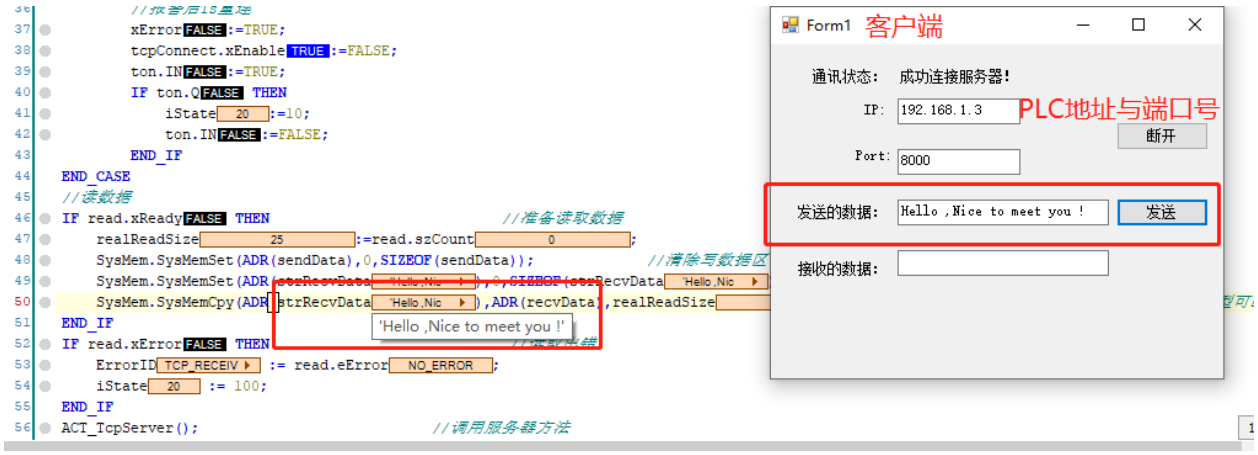


图 12.19　上位机发送字符串到 PLC

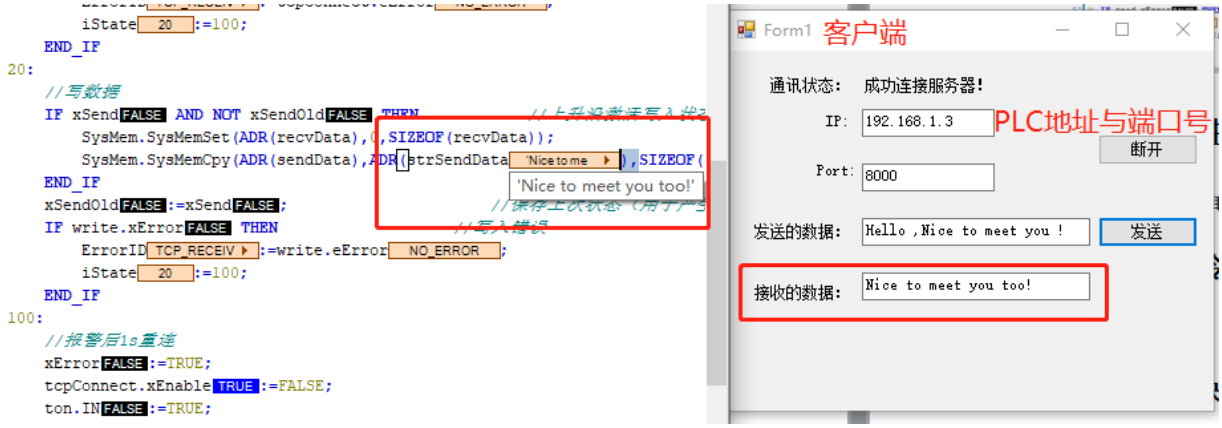


图 12.20　PLC 发送字符串到上位机

例 2：PLC（客户端）与 PC 机（服务器）的通讯

该例程中 MC300CS 系列 PLC 作为客户端与 PC 机进行通讯，以“通讯连接”、“数据发送”、“数据接收”这三个功能为例，程序执行逻辑流程如图 12.21 所示。其中 PLC 与 PC 机通讯成功后，PLC 向 PC 机发送“Hellow,Nice to meet you!”，PC 机接收成功后向 PLC 发送“Nice to meet you too!”

其中变量 strSendData、strRecvData 为发送和接收的数据，数据类型为 string，根据具体使用情况，也可以使用其它数据类型或结构体。若发送数据量较大，可修改发送和接收的数组长度。

PC 机与 PLC 通过网线连接，如图 12.16 所示；其中 PLC 的 IP 地址为 192.168.1.3，PC 机 IP 地址为 192.168.1.5，设置方式如图 12.17 所示。

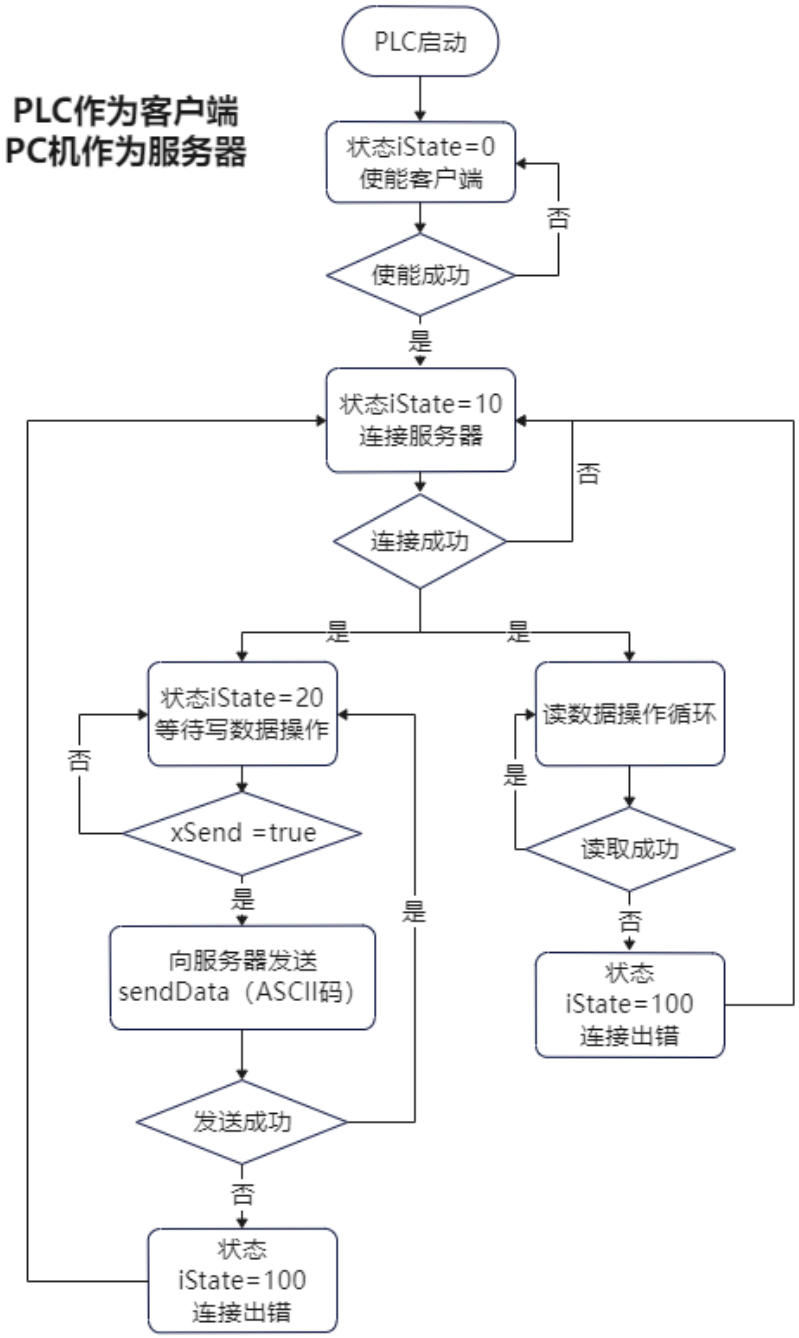


图 12.21　程序执行逻辑流程图

1） PLC（客户端）例程

```
PROGRAM PLC_PRG
VAR
    (******* TCP_Client ********)
    TCPClient :NBS.TCP_Client;
    IP : NBS.IP_ADDR := (sAddr:='192.168.1.123');    //上位机（服务器）IP 地址
    Port :UINT:=8000;                                //上位机（服务器）端口号
    (********* 数据读写 ********)
    read :NBS.TCP_Read;                              //读数据函数
    write:NBS.TCP_Write;                             //写数据函数
    realReadSize :UDINT;                             //读取数据大小
    RecvData :ARRAY[0..255]OF BYTE;                  //读数据区
    SendData :ARRAY[0..255]OF BYTE;                  //写数据区
```

```
    (********* 收发内容 ********)
    strRecvData : STRING;                          //自定义接收数据（类型可改）
    strSendData : STRING;                          //自定义发送数据（类型可改）
    xConnect    :BOOL;                             //连接状态
    xWrite   : BOOL;                               //写入状态
    xWriteOld  : BOOL;                             //上一次写入状态
    xError     :BOOL;                              //错误状态
    ErrorID: NBS.ERROR;                            //错误标志
    ton:ton;                                       //重连定时器
    iState : INT :=0;                              //程序运行状态位
END_VAR

    TCPClient(xEnable:= xConnect, xDone=> , xBusy=> , xError=> ,
              udiTimeOut:= 1000000, ipAddr:= IP, uiPort:= Port,
              eError=> , xActive=> , hConnection=> );      //开启 TCP 客户端模式
    read(xEnable:= TCPClient.xActive, xDone=> , xBusy=> , xError=> ,
        hConnection:= TCPClient.hConnection, szSize:= SIZEOF(RecvData),
        pData:= ADR(RecvData), eError=> , xReady=> , szCount=> );    //读取数据
    write(xExecute:= xWriteOld, udiTimeOut:= , xDone=> , xBusy=> , xError=> ,
        hConnection:= TCPClient.hConnection, szSize:= SIZEOF(SendData),
        pData:= ADR(SendData), eError=> ); //写入数据

    CASE iState OF
        0:                          //连接服务器：
            IF xConnect THEN                       //使能客户端连接
                iState :=10;
            END_IF
        10:                         //判断是否建立连接：
            IF TCPClient.xActive <> 0 THEN         //连接状态
                iState := 20;
            END_IF
            IF TCPClient.xError AND
                (TCPClient.hConnection = CAA.gc_hINVALID) THEN //连接错误
                ErrorID := TCPClient.eError;
                iState :=100;
            END_IF
        20: //写数据：
            //将写入的数据（BYTE）转换为字符串（类型可改）：
            SysMem.SysMemCpy(ADR(SendData),ADR(strSendData),SIZEOF(strSendData));
            IF xWrite AND NOT xWriteOld THEN                    //上升沿激活写入状态
                SysMem.SysMemSet(ADR(SendData), 0, SIZEOF(SendData)); //清除读数据区
            END_IF
            xWriteOld := xWrite;                //保存上次状态（用于产生上升沿）
            IF write.xError THEN                //写入错误
                ErrorID := write.eError;
                iState := 100;
            END_IF
        100:                        //连接报错定时一秒后重连：
            xError :=TRUE;
            xConnect := FALSE;
```

```
            ton.IN := TRUE;
            IF ton.Q THEN
                iState := 0;
                ton.IN := FALSE;
                xConnect := TRUE;
            END_IF
    END_CASE
    //读数据：
    IF read.xReady THEN                          //准备读取数据
        realReadSize := read.szCount;            //读取数据的大小
            SysMem.SysMemSet(ADR(SendData), 0, SIZEOF(SendData));      //清除写数据区
            SysMem.SysMemSet(ADR(strRecvData), 0, SIZEOF(strRecvData)); //清空读取数据
        //将读取的数据（BYTE）转换为字符串：
        SysMem.SysMemCpy(ADR(strRecvData),ADR(RecvData),realReadSize);
    END_IF
    IF read.xError THEN                           //读取出错
        ErrorID := read.eError;
        iState := 100;
    END_IF
    ton(IN:= , PT:=T#1S , Q=> , ET=> );          //连接报错后定时一秒后重连
```

2） 上位机（服务器）例程

① C#Socket 通讯流程和主要 API

a） 组装 Socket 通信的地址和端口，建立通信的对象：

```
IPAddress ipAddress = IPAddress.Parse(textBox1.Text);//获取 IP 地址
int Port = Convert.ToInt32(textBox2.Text);          //获取端口号
```

b） 创建 Socket 对象，建立与服务器的 TCP 连接：

```
MySocket.Bind(new IPEndPoint(ipAddress, Port));     //启用监听
MySocket.Listen(10);                                //监听设备数量 10 个
```

c） Socket TCP 通信数据交互：

```
MySocket.Receive(bytes, bytes.Length, 0);                //接受数据
MySocket.BeginSend(byteArray, 0, byteArray.Length, 0, null, null);//发送数据
```

d） 关闭 Socket 通信：

```
MySocket.Close();                                   //关闭服务器
```

② 程序窗口如图 12.22 所示

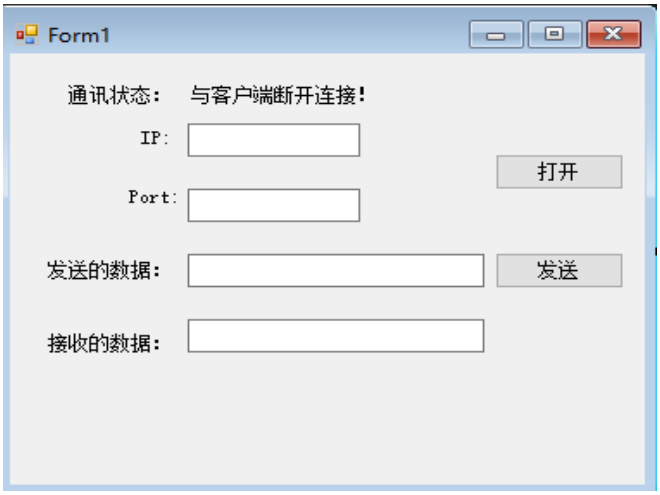


图 12.22 客户端窗口

③ 程序代码：

```
using System;
using System.Net;
using System.Net.Sockets;                                    //Socket 对应的命名空间
using System.Text;
using System.Threading;
using System.Windows.Forms;
namespace WindowsFormsApp1
{
 public partial class Form1 : Form
    {
        private Socket MySocket = null;                      // Socket
        Thread Listen;                                       //监听线程
        Thread Receive;                                      //接收线程
        string IP = "192.168.1.5";                           //PC 地址
        int Port = 8000;                                     //端口号
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            textBox1.Text = IP;
            textBox2.Text = Port.ToString(); ;
        }
        private void button1_Click(object sender, EventArgs e)
        {
            if (button1.Text == "打开")
            {
//IP 地址 和 端口号输入不为空
if (string.IsNullOrEmpty(textBox1.Text) == false && string.IsNullOrEmpty(textBox2.Text)
== false)
  {
    try
    {
  IPAddress ipAddress = IPAddress.Parse(textBox1.Text); //获取 IP 地址
  int Port = Convert.ToInt32(textBox2.Text);           //获取端口号
  MySocket = new Socket(AddressFamily.InterNetwork,SocketType.Stream,ProtocolType.Tcp);
  MySocket.Bind(new IPEndPoint(ipAddress, Port));      //监听
  MySocket.Listen(10);                                 //监听设备数量
  label3.Text = "处于监听状态，等待客户端连接";
  Listen = new Thread(new ThreadStart(ListenC));       //新建监听线程
  Listen.IsBackground = true;           //设置为后台线程，不需要用户主动关闭
  Listen.Start();                                      //启动监听线程
    }
        catch (Exception)
             {
              MessageBox.Show("IP 地址或端口号错误!", "提示");
             }
```

```
    }
        else
             {
               MessageBox.Show("IP 地址或端口号为空!", "提示");
              }
     }
        else
             {
              try
                 {
                  if (Listen != null)
                      Listen.Abort();
                  if (Receive != null)
                     Receive.Abort();
                  MySocket.Shutdown(SocketShutdown.Both);
                  MySocket.Close();
                  MySocket = null;
                  label3.Text = "与客户端连接断开!";  //对话框追加显示数据
                  button1.Text = "打开";
                 }
           catch (Exception) {}
            }
        }
private void button2_Click(object sender, EventArgs e)
{
  String Str = textBox4.Text.ToString();        //获取发送文本框里面的数据
       try
         {
            if (Str.Length > 0)
           {
  byte[] byteArray = Encoding.Default.GetBytes(Str);     //Str 转为 Byte 值
  MySocket.BeginSend(byteArray, 0, byteArray.Length, 0, null, null); //发送数据
            }
           }
            catch (Exception) { }
        }
public void ListenC()
{
  MySocket = MySocket.Accept();            //与 PLC 客户端连接
  Invoke(new MethodInvoker(delegate ()
  {
    button1.Text = "关闭";
    label3.Text = "连接成功！";
    }));
 Receive = new Thread(new ThreadStart(ReceiveC)); //新建接收线程
 Receive.IsBackground = true;   //设置为后台线程，不需要用户主动关闭
 Receive.Start();               //开启接收线程
       }
public void ReceiveC()
 {
```

```
        byte[] bytes = new byte[1024];        //数组容量
        int len = MySocket.Receive(bytes, bytes.Length, 0);
        //将消息转为字符串
        string str = Encoding.Default.GetString(bytes, 0, len);//Byte 转为 String
        Invoke(new MethodInvoker(delegate ()
              {textBox5.Text=Encoding.Default.GetString(bytes, 0, len);
               }));
            }
         }
    }
```

④ 程序运行结果

作为服务器的PC机与作为客户端的PLC建立通讯成功后，PLC通过变量strSendData向PC机发送字符串数据，如图12.23所示，PLC向PC机发送了“Hello,Nice to meet you!”。

PLC通过变量strRecvData接收PC机发送的字符串数据，如图12.24所示，PC机向PLC发送了“Nice to meet you too!”。

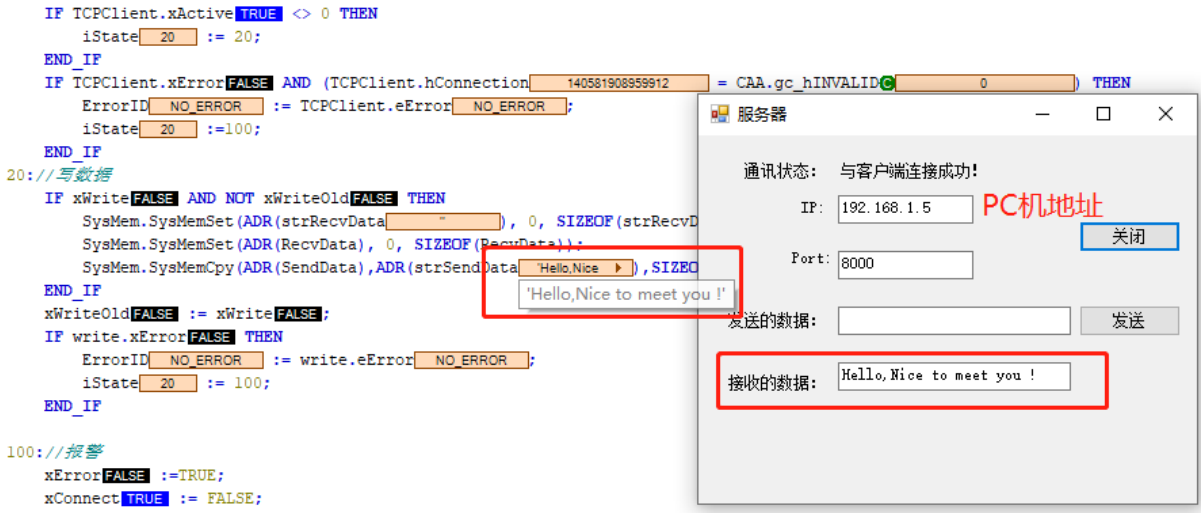


图 12.23　PLC 发送字符串到上位机

图 12.24　上位机发送字符串到 PLC

12.3. UDP 自由协议通讯及例程

UDP 是无连接协议，具有较好的实时性，工作效率较 TCP 协议高，段结构比 TCP 的段结构简单，因此网络开销也小。但是传输数据前不建立连接，不对数据报进行检查与修改，无须等待对方的应答，所以会出现分组丢失、重复、乱序，应用程序需要负责传输可靠性方面的所有工作。

使用 UDP 自由协议指令前需要在工程中添加“CAA Types Extern 库”与“CAA Net Base Services 库”文件，以及用于操作通讯数据的指令“SysMem”库文件。

12.3.1. UDP 自由协议指令

MC300CS 系列 PLC 的 UDP 自由协议指令如表 12.5 所示。自由协议错误码 eError 说明如表 12.4 所示。详细说明请参考《雷赛大中型 PLC 指令手册》。

表 12.5 自由协议指令列表

指令类别	名称	功能
UDP 自由协议通讯指令	UDP_Peer	创建 UDP 通讯连接
	UDP_Receive	数据接收
	UDP_Send	数据发送

1. 创建 UDP 通讯连接 UDP_Peer

该指令能通过设置本机 IP 地址、IP 组播地址、端口号创建 UDP 通讯连接。其格式如下。

```
NBS.UDP_Peer(xEnable:=使能信号, xDone=>完成信号, xBusy=>执行中信号, xError=>错误标志,
             ipAddr:= 本机 IP 地址, uiPort:= 端口号, ipMultiCast:=IP 组播地址 ,
             eError=> 错误内容, xActive=> 连接成功标志, hPeer=>通讯句柄 );
```

其中：ipMultiCast 为 IP 组播地址，即任何用户主机，加入该组播组，就成为了该组成员，可以识别并接收发往该组播组的数据，IP 组播地址即 224.0.0.0 至 239.255.255.255 之间的 IP 地址。也可以将消息发送到在同一广播网络上的每个主机，即将 ipMultiCast 设置为广播地址 255.255.255.255。

2. 数据接收 UDP_Receive

该指令能使用 UDP_Peer 指令创建的通讯句柄 hPeer 进行数据的接收操作。其格式如下。

```
NBS.UDP_Receive(xEnable:=使能信号,xDone=>完成信号,xBusy=>执行中信号,xError=>错误标志,
                hPeer:= 通讯句柄, szSize:=数据大小, pData:= 接收数据缓存区指针,
                eError=>错误内容, xReady=>成功建立连接标志, ipFrom=>当前数据包源 IP,
                uiPortFrom=>数据包源端口号, szCount=>数据大小);
```

3. 数据发送 UDP_Send

该指令能使用 UDP_Peer 指令创建的通讯句柄 hPeer 进行数据的发送操作。其格式如下。

```
NBS.UDP_Send(xExecue:=启动信号, udiTimeOut:= 超时时间 , xDone=>完成信号,
             xBusy=>执行中信号, xError=>错误标志,hPeer:= 通讯句柄, ipAddr:= 目标 IP,
             uiPort:= 通信端口号, szSize:= 数据大小, pData:= 发送数据缓存区指针,
             eError=>错误内容);
```

12.3.2. UDP 自由协议通讯例程

该例程使用两台 MC300CS 系列 PLC 以“通讯连接”、“数据发送”、“数据接收”这三个功能为例，分别说明 UDP 通讯中两台 PLC 之间如何进行通讯，接线如图 12.25 所示。

当两台 PLC 分别创建 UDP 通讯成功后，一台 PLC 向另一台 PLC 发送“Hellow,Nice to meet you!”，另一台接收后发送“Hellow,Nice to meet you too!”。

其中变量 strSendData、strRecvData 为发送和接收的数据，数据类型为 string，根据具体使用情况，可以使用其它数据类型或结构体。若发送数据量较大，可修改发送和接收的数组长度。

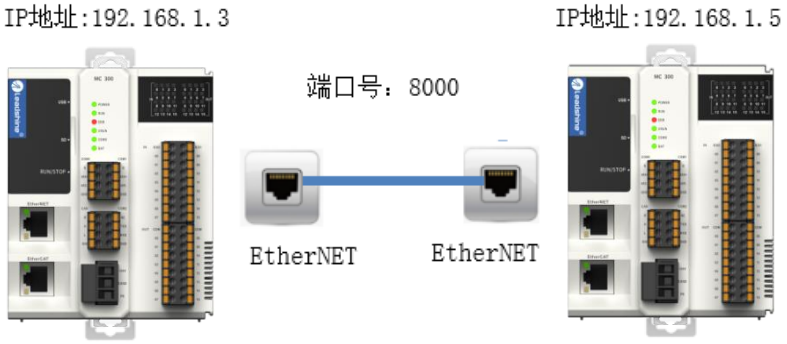


图 12.25 通讯接线方式

以下程序代码为其中一台 PLC 的程序，通过互换 IP.sAddr 和 IP_Target.sAddr 地址，即为另一台 PLC 的程序。

```
PROGRAM PLC_PRG
VAR
    UDP_Peer_0: NBS.UDP_Peer;                  //实例化创建 UDP 通讯功能
    UDP_Receive_0: NBS.UDP_Receive;            //实例化 UDP 数据接收功能
    UDP_Send_0: NBS.UDP_Send;                  //实例化 UDP 数据发送功能
    xPeer_Enable:BOOL:=TRUE;                   //使能 UDP 通讯功能信号
    xSend_Exe AT %IX0.0:BOOL;                  //数据发送功能触发
    xReceieve_Enable AT %IX0.1:BOOL;           //数据接收功能使能
    IP:NBS.IP_ADDR;                            //PLC 本机 IP 地址
    IP_Multicate:NBS.IP_ADDR;                  //IP 组播地址
    IP_Target:NBS.IP_ADDR;                     //UDP 通讯目标设备 IP 地址
    abySendData:ARRAY[0..14] OF BYTE;          //发送的数据内容
    abyReceieveData:ARRAY[0..14] OF BYTE;      //接收的数据内容
    strSendData: STRING;                       //发送的字符串
```

```
    strReceieveData: STRING;                    //接收的字符串
END_VAR

IP.sAddr:='192.168.1.3';                        //PLC 本机 IP 地址
IP_Multicate.sAddr:='255.255.255.255';          //IP 组播地址
IP_Target.sAddr:='192.168.1.5;                  //UDP 通讯目标设备 IP 地址
IF xSend_Exe THEN
  //将写入的数据（字符串）转换为 BYTE：
  SysMem.SysMemCpy(ADR(abySendData),ADR(strSendData),SIZEOF(strSendData));
END_IF
IF xReceieve_Enable THEN
  //将接收的数据（BYTE）转换为字符串：
  SysMem.SysMemCpy(ADR(strReceieveData),ADR(abyReceieveData),SIZEOF(abyReceieveData));
END_IF
//创建 UDP 通讯连接：
UDP_Peer_0(xEnable:=xPeer_Enable,xDone=>,xBusy=> , xError=> , ipAddr:=IP , uiPort:=8000 ,
           ipMultiCast:= IP_Multicate, eError=> , xActive=> , hPeer=> );
//UDP 通讯数据发送：
UDP_Send_0(xExecute:=xSend_Exe AND UDP_Peer_0.xBusy , udiTimeOut:=50000 , xDone=> ,
           xBusy=>,xError=>,hPeer:=UDP_Peer_0.hPeer,ipAddr:=IP_Target,uiPort:=8000,
           szSize:=SIZEOF(abySendData), pData:= ADR(abySendData), eError=> );
//UDP 通讯数据接收：
UDP_Receive_0(xEnable:=xReceieve_Enable AND UDP_Peer_0.xBusy,xDone=>,xBusy=>, xError=> ,
           hPeer:=UDP_Peer_0.hPeer,szSize:=SIZEOF(abyReceieveData),
           pData:=ADR(abyReceieveData), eError=> ,xReady=> , ipFrom=> , uiPortFrom=> ,
           szCount=> );
```

程序运行时，xPeer_Enable 为 TRUE 创建 UDP 通讯连接，连接到地址为 IP_Target 的 PLC，当 IN0 为 TRUE 且 UDP 通讯连接创建成功时，向 IP_Target 的 PLC 发送变量 abySendData 的数据；当 IN1 一直为 TRUE 且 UDP 通讯连接创建成功时，接收 IP_Target 的 PLC 数据并存放于 abySendData 中。

如图 12.26 所示，一台 PLC 向另一台 PLC 发送“Hellow,Nice to meet you!”，另一台接收后发送“Hellow,Nice to meet you too!”。

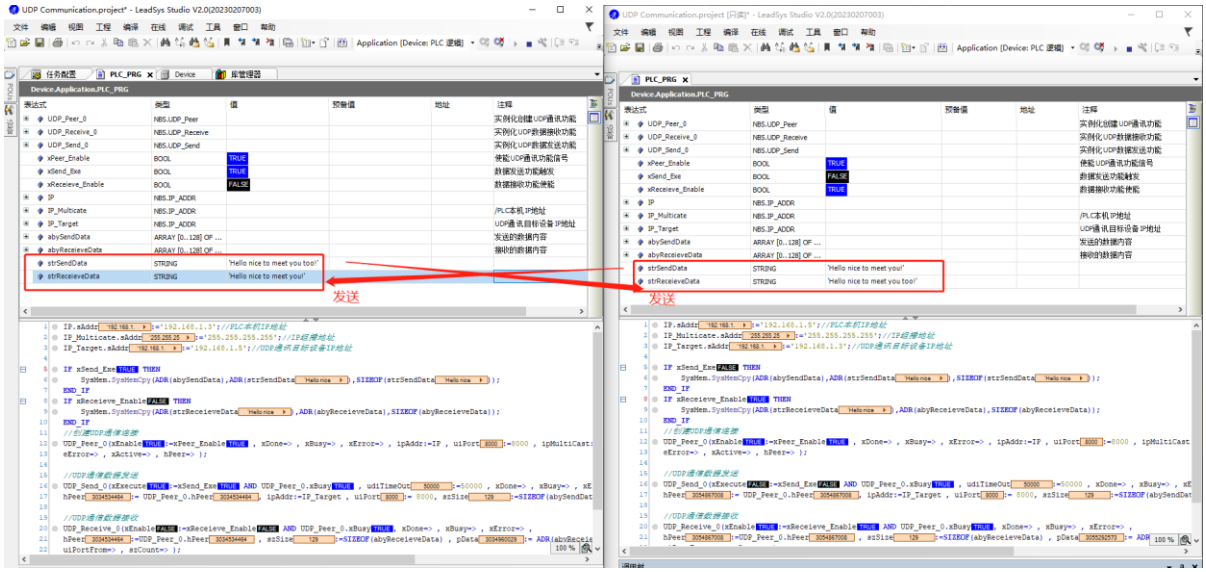


图 12.26　PLC 通过 UDP 通讯发送、接收数据

12.4.　标签通讯及例程

标签通讯基于 CodeSys 协议进行，常用在 PLC 与触摸屏通讯中。程序中的变量声明后，LeadSys Studio 软件可以直接生成用于通讯的 XML 文件；在触摸屏添加该文件并完成配置后可以看到变量名、注释等相关信息。与 Modbus 协议通讯相比，省去了地址定义、数据区规划等繁琐工作。

本例程以MC300CS系列PLC与雷赛LT2070T触摸屏通讯为例，介绍标签通讯的方法，接线方式如图 12.27 所示。当按下触摸屏的启动按钮时，PLC 以触摸屏上设定的运动参数执行一次相对运动，在运动过程中指示灯为绿色，并显示当前轴位置与速度，当运动完成时指示灯为红色。

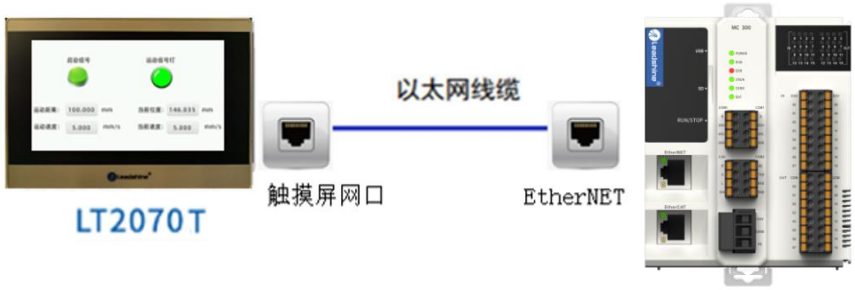


图 12.27 标签通讯接线方式

例程的代码如下。

1) 程序部分

```
PROGRAM PLC_PRG
VAR
    axes:dut_pulse_axis;                    // 脉冲轴结构体
    MC_MoveRelative_1: MC_MoveRelative;     // 相对运动实例化
    MC_Power_1: MC_Power;                   // 轴使能实例化
    Start:BOOL;                             //启动信号
    State:BOOL;                             //运动中信号
    Xpos:LREAL;                             //X 轴当前位置
    Xspeed:LREAL;                           //X 轴当前速度
    Distance:LREAL;                         //运动距离
    Vel:LREAL;                              //运动速度
END_VAR
axes.pulaxis_0:=ADR(X);                     // 获取 X 轴的地址
// 绑定脉冲轴：
LS_MotionControl_P(stAxis:=axes, xClearErr:=, fLimtAxisSpeedJump:=,
                   xDone=>, xError=>, eErrorID=>, xLimitAxisMoveFlag=> );
MC_Power_1(Axis:=X ,Enable:=1 , bRegulatorOn:=1 ,bDriveStart:=1 , Status=> ,
          bRegulatorRealState=>, bDriveStartRealState=>, Busy=>, Error=>,
           ErrorID=> );                     //轴使能
MC_MoveRelative_1(Axis:=X , Execute:=Start , Distance:= Distance, Velocity:=Vel ,
                  Acceleration:=100 , Deceleration:=100 , Jerk:=1000 ,
                  BufferMode:= , Done=> , Busy=>State , Active=> , CommandAborted=> ,
                  Error=> , ErrorID=> );    // 相对运动指令
Xpos:=X.fActPosition;                       // X 轴当前位置
Xspeed:=X.fActVelocity;                     // X 轴当前速度
```

2） 标签通讯配置

在当前项目树中选中“Application”，鼠标右键后，点击“添加对象”，选择“符号配置”，如图 12.28 所示。

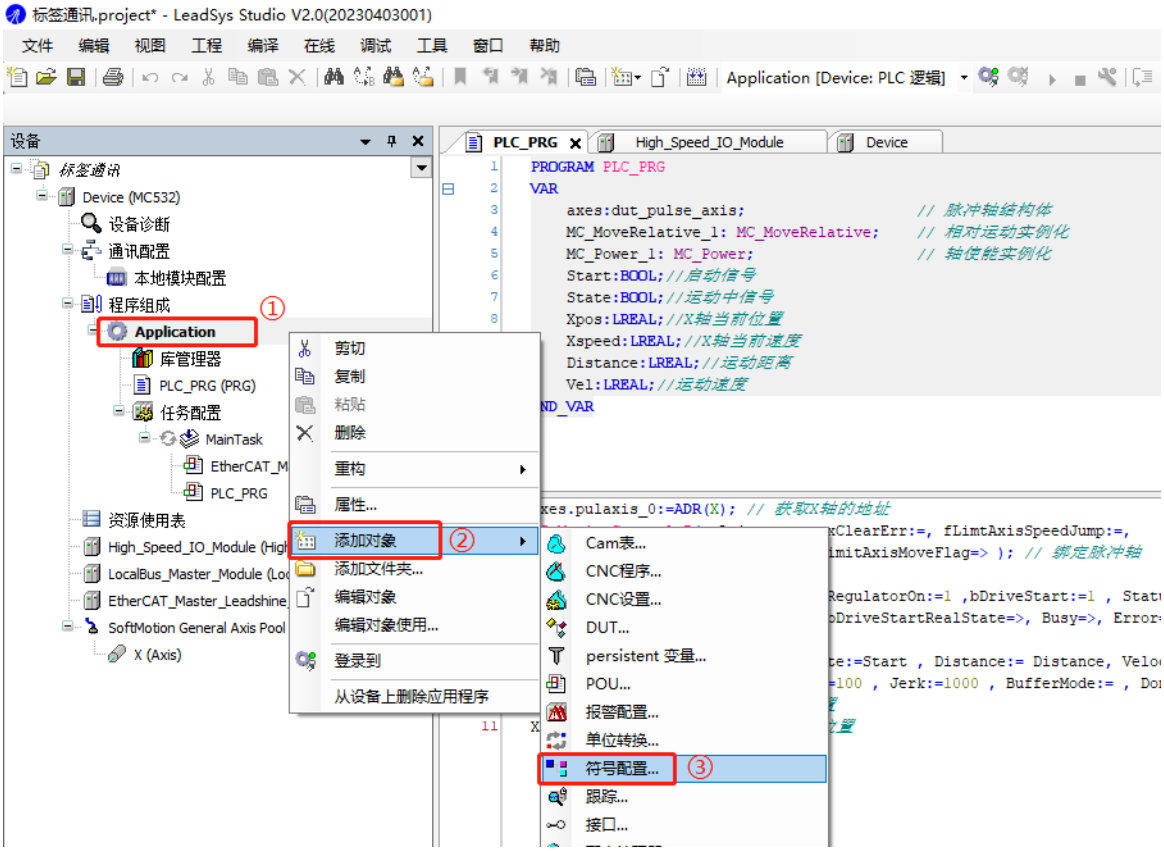


图 12.28　添加“符号配置”

在符号配置窗口中，可勾选“在 XML 中包含注释”选项，选中后标签中将包含程序的注释，如图 12.29 所示。

在“符号配置”界面中勾选需要进行标签通讯的变量，点击“编译”→“生成代码”，编译生成用于标签通讯的 XML 文件，该文件在程序所在的文件夹中，如图 12.30 所示。

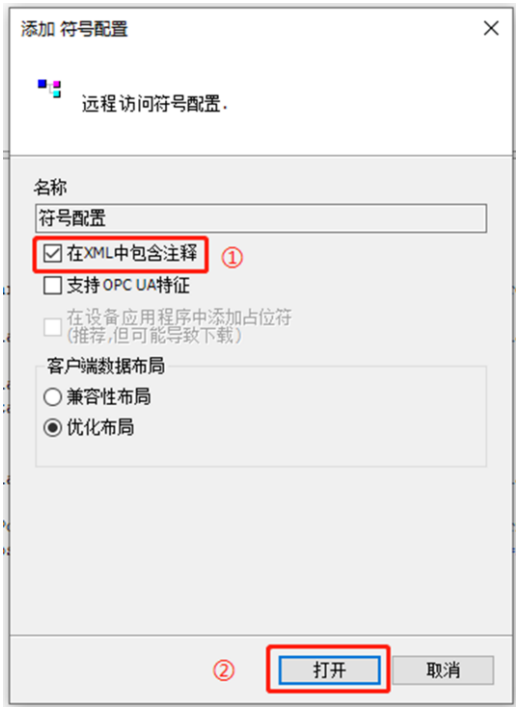


图 12.29　勾选“在 XML 中包含注释”

上述为 MC300CS 系列 PLC 标签通讯配置，下面以雷赛触摸屏 LT2070T 配置为例，介绍触摸屏关于标签通讯的配置。

打开雷赛触摸屏软件 LT Studio，新建触摸屏程序文件，选择型号为 LT2070T，添加“网

络 PLC”输入 PLC 的 IP 地址后，点击“导入变量标签”，在 PLC 程序文件夹中选择 XML 文件导入，如图 12.31 所示。

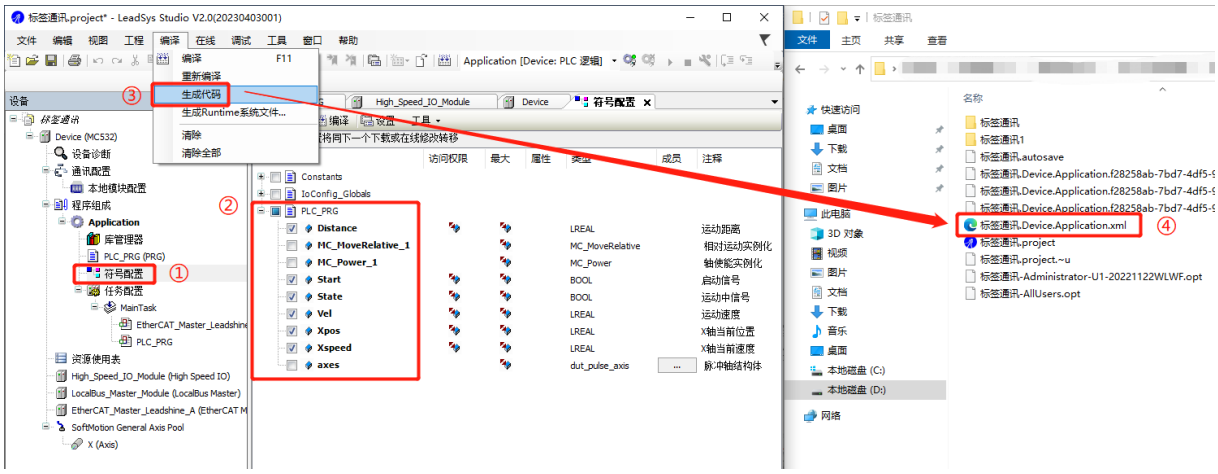


图 12.30　勾选进行标签通讯的变量

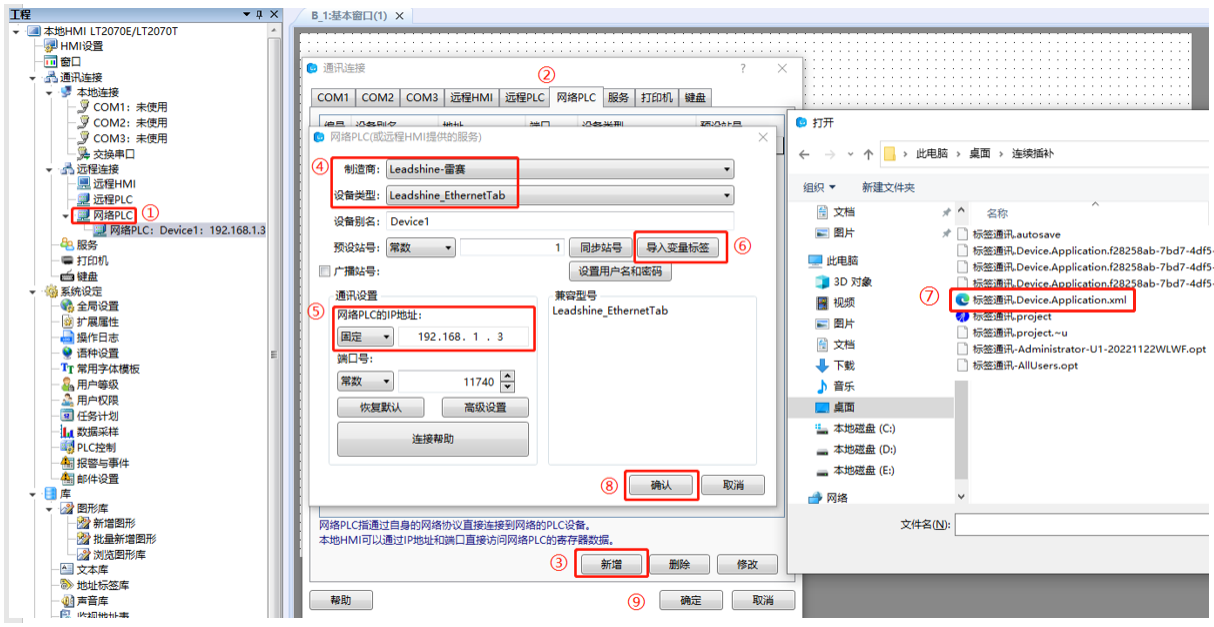


图 12.31　在 LT Studio 软件中导入 XML 文件

在设计界面上依次增加数值显示元件，然后在变量标签库中选择相应的变量名。操作过程如图 12.32 所示。

运行结果如图 12.33 所示。

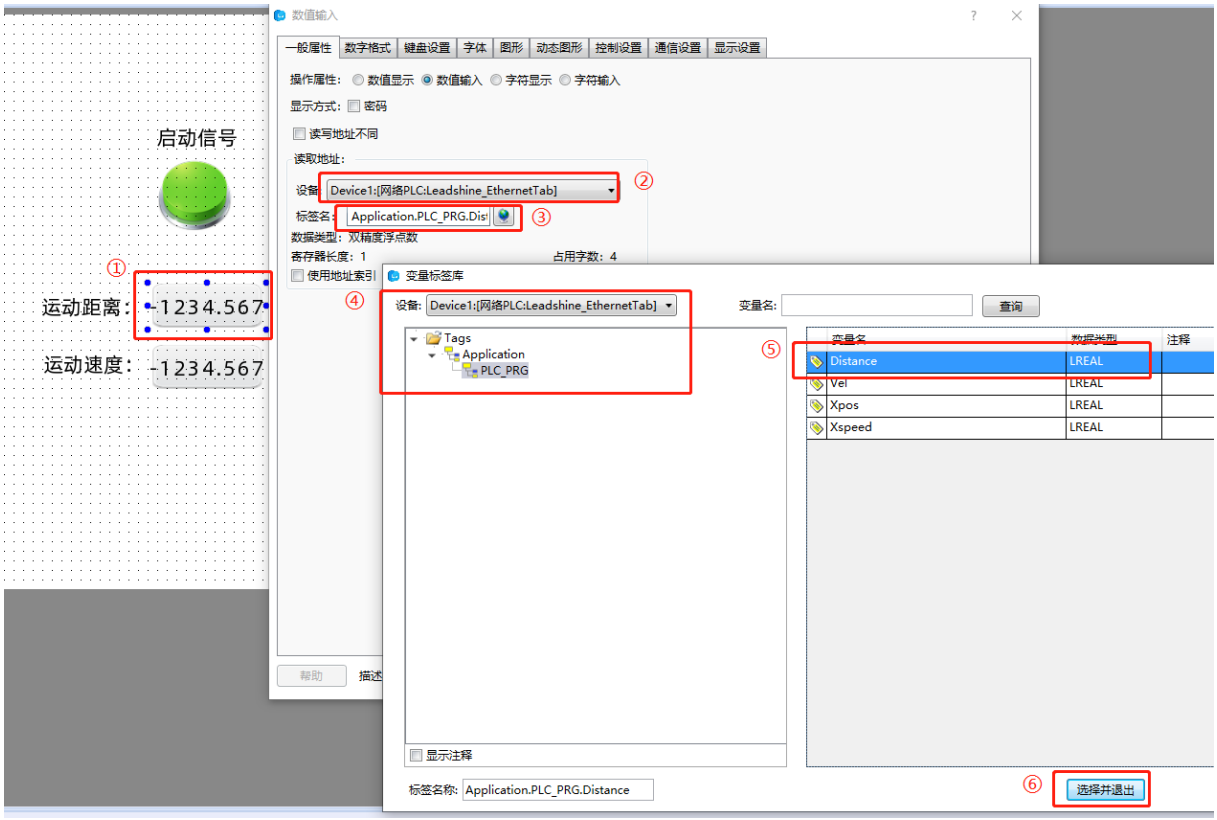


图 12.32 新增数值显示的元件

图 12.33　运行结果

12.5.　OPC UA 服务器通讯及例程

OPC UA(Open Platform Communications - Unified Architecture)是为了应对标准化和跨平台的趋势推出的一个新的 OPC 标准，集成了之前所有 OPC 的特性和信息，支持多种操作系统，以及更复杂的数据类型，如：变量、方法和事件，配置更简单、使用更安全。

MC300CS 系列 PLC 可作为 OPC UA 服务器与 MES（制造执行系统）、SCADA（数据采集与监视控制系统）、ERP（企业资源计划系统）等系统实现设备、生产线的信息交互；也可以与其他支持 OPC UA 通讯的设备进行数据交互。

本例程使用 UaExpert 软件作为客户端与 PLC 进行 OPC UA 通讯,PC 机与 PLC 通过网线连接，如图 12.34 所示。PLC 程序使用“12.4 节 标签通讯”例程作为演示说明。

通讯的变量包括合速度、X 轴速度、Y 轴速度、Z 轴速度、X 轴位置、Y 轴位置、Z 轴位置。当 PLC 执行运动时，将上述变量的当前值发送到 UaExpert 软件上。

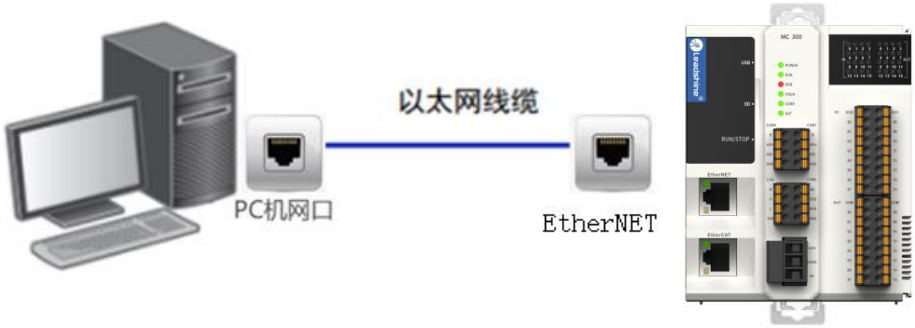


图 12.34　PC 机与 PLC 通过以太网口连接

1） PLC 作为 OPC UA 服务器通讯配置

PLC 作为 OPC UA 服务器的配置与“12.4 节 标签通讯”的配置大致相同，仅需在“符号配置”中勾选支持 OPC UA 即可。 可在添加“符号配置”时勾选“支持 OPC UA 特征”，如图 12.35 所示，也可在添加后在符号配置界面勾选“支持 OPC UA 功能”，如图 12.36 所示。

图 12.35　在添加“符号配置”勾选“支持 OPC UA 特征”

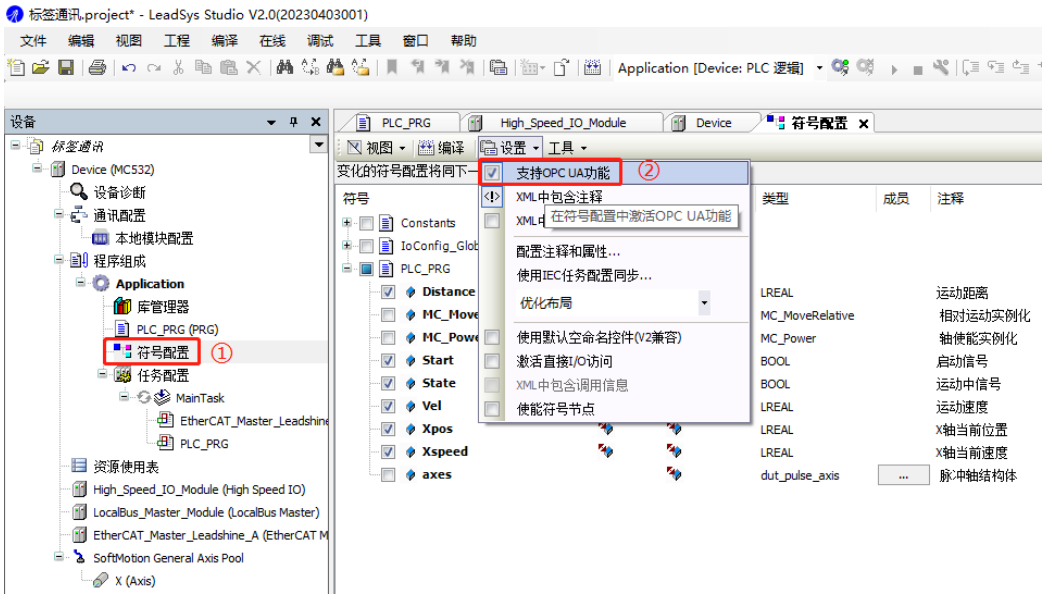


图 12.36　在符号配置界面勾选“支持 OPC UA 功能”

2) UaExpert 软件与 PLC 建立 OPC UA 通讯

以下介绍使用 UaExpert 软件与 PLC 建立 OPC UA 通讯的操作方法。

打开 UaExpert 后，鼠标右键点击“Server”，选择“Add”，添加服务器，如图 12.37 所示。

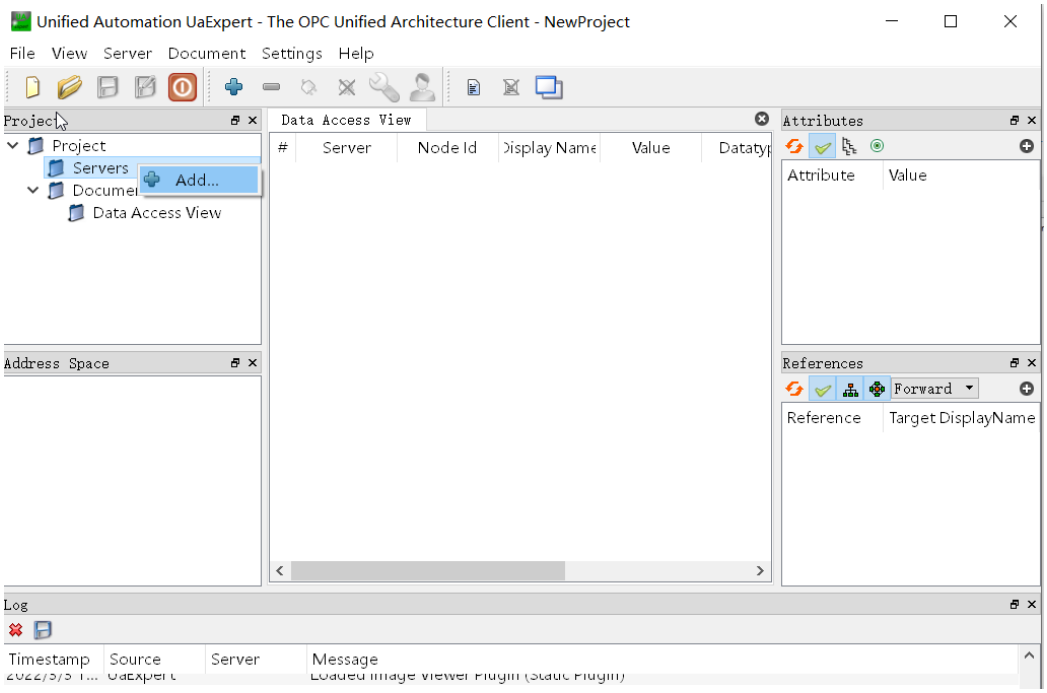


图 12.37　添加 OPC UA 服务器

在弹出的窗口中，在“Configuration Name”中输入自定义 OPC UA 服务器名称，选择“Advance”选项卡，然后输入服务器的 URL，格式为 opc.tcp://<PLC 的 IP 地址>:4840，此例中 PLC 的 IP 地址为 192.168.1.3，故对应的 URL 为 opc.tcp://192.168.1.3:4840，如图 12.38 所示。

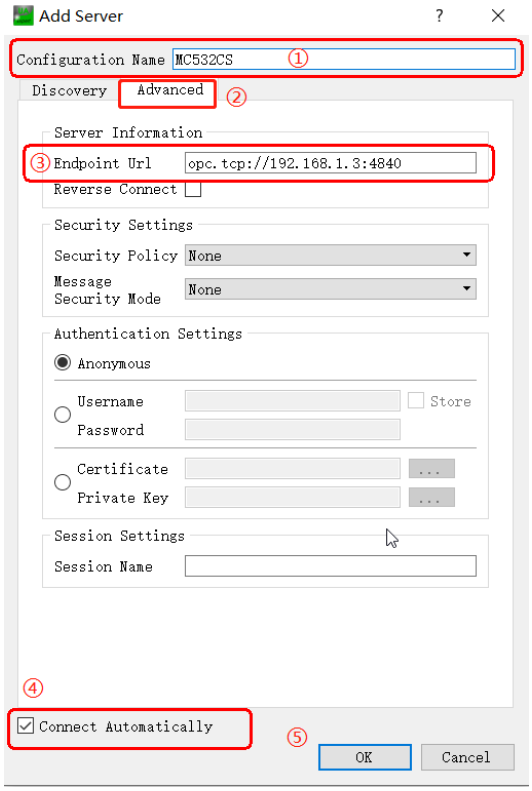


图 12.38　设置服务器参数

设置电脑的网口 IP 地址与 PLC 的 IP 在同一网段，点击“Connect”连接 PLC，右键“Documents”→“Add”，添加“Data Access View”，如图 12.39 至图 12.41 所示。

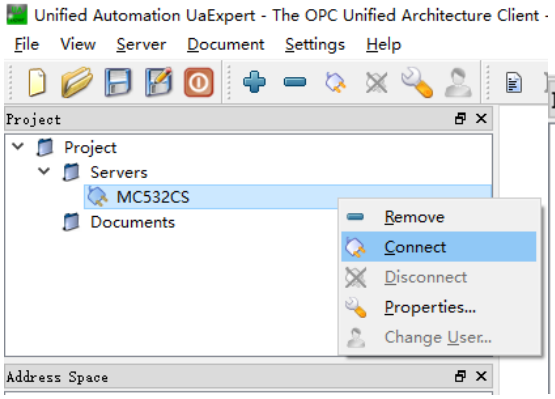


图 12.39　连接 PLC

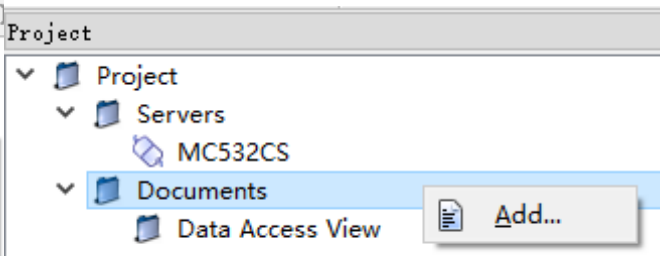


图 12.40　添加数据访问视图

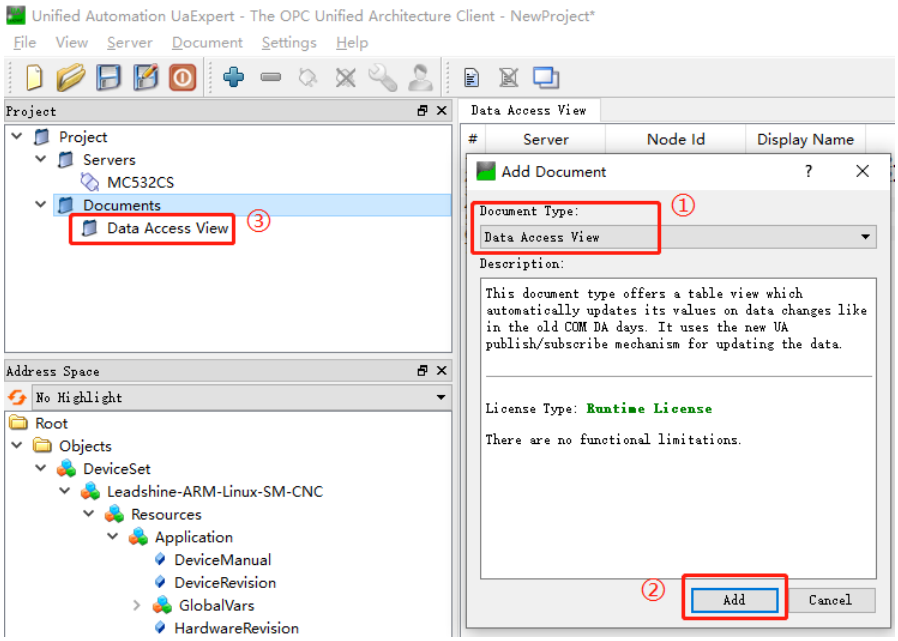


图 12.41　添加数据访问视图

连接成功后，在 Root 根目录下的 Object 文件夹中将通讯的变量拖入“Data Access View”中，如图 12.42 所示，此时运行程序可以看到 Value 中能读取到 PLC 通讯变量的数值，如图 12.43 所示。

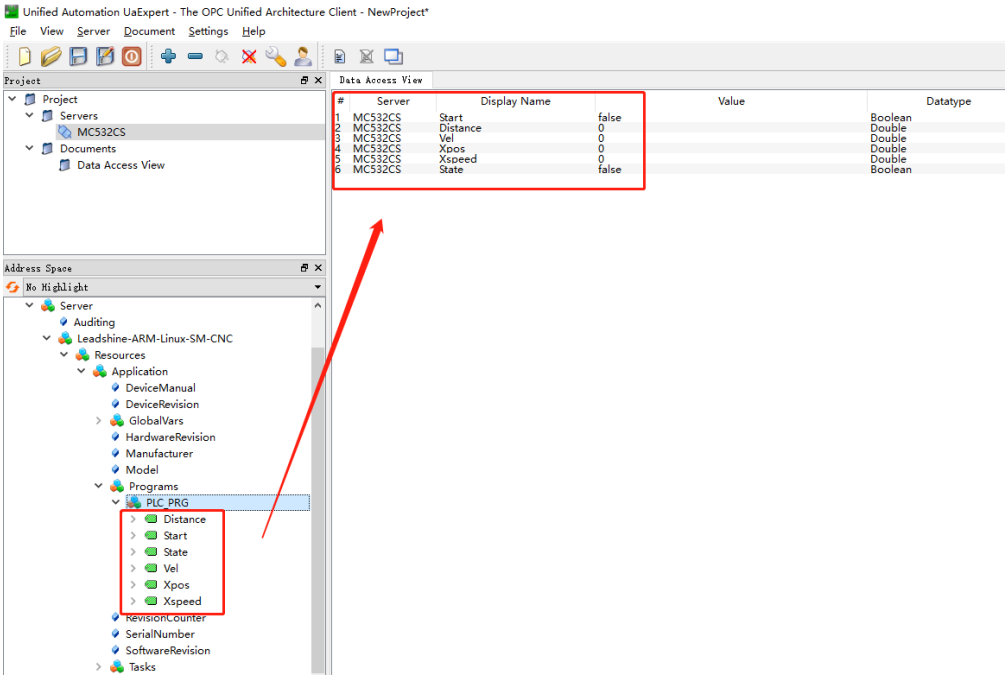


图 12.42　添加通讯变量

Data Access View

#	Server	Display Name	Value	Datatype
1	MC532CS	Start	true	Boolean
2	MC532CS	Distance	100	Double
3	MC532CS	Vel	10	Double
4	MC532CS	Xpos	210.54	Double
5	MC532CS	Xspeed	10	Double
6	MC532CS	State	true	Boolean

图 12.43　读取到 PLC 通讯变量的数值

12.6.　Ethernet/IP 协议通讯及例程

Ethernet/IP 是一个面向工业自动化应用的工业应用层协议，是一种适合工业环境应用的协议体系，基于传统的以太网协议和标准的 TCP/IP 协议，可以实现工业设备之间应用信息的高效交换。

一个 Ethernet/IP 协议通讯网络中可以由一个 Ethernet/IP 扫描器和一个或多个 Ethernet/IP 适配器组成。扫描器是网络中的主机，适配器是从机。

MC300CS 系列 PLC 可以配置为扫描器（Scanner）和适配器（Adapter），可以与支持 Ethernet/IP 协议的设备建立通讯，如 PLC、伺服驱动器等。通讯的速度由所在任务的周期决定，任务周期越小通讯速度越快，最小为 500us。

本例程采用两台 MC300CS 系列 PLC 分别作为扫描器和适配器进行数据交互，接线如图 12.44 所示。其中作为适配器的 PLC 添加了 Big Output Module 和 Big Input Module 这两个模块，Big Output Module 模块是用于将适配器的通讯数据发送给扫描器，Big Input Module 模块是用于接收扫描器发送给适配器的通讯数据。

扫描器通过读取适配器的轴状态，判断适配器的轴是否处于运动状态。当轴处于停止状态时，扫描器向适配器发送运动距离、运动速度和启动信号，适配器接收到这些运动参数后，执行一条相对运动指令，同时将当前位置和速度以及轴状态返回给扫描器。

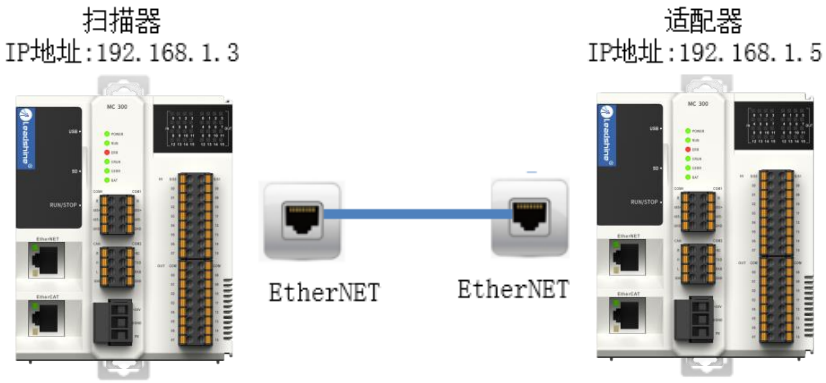


图 12.44　通讯接线方式

12.6.1. 适配器配置

1.　使能 PLC 作为 EtherNet/IP 适配器

双击项目树中“通讯配置”，选择 EtherNet IP1 通讯，勾选“适配器”，使能后项目树中出现了对应的通讯设备“Ethernet_IP_Adapter”，如图 12.45 所示。

2.　添加 EtherNet/IP 通讯模块

使能后需要配置 EtherNet/IP 适配器的通讯参数，再通过添加设备将 EtherNet/IP 通讯模块（EtherNet_IP_Module）添加到适配器中，操作过程如图 12.46 至图 12.47 所示。

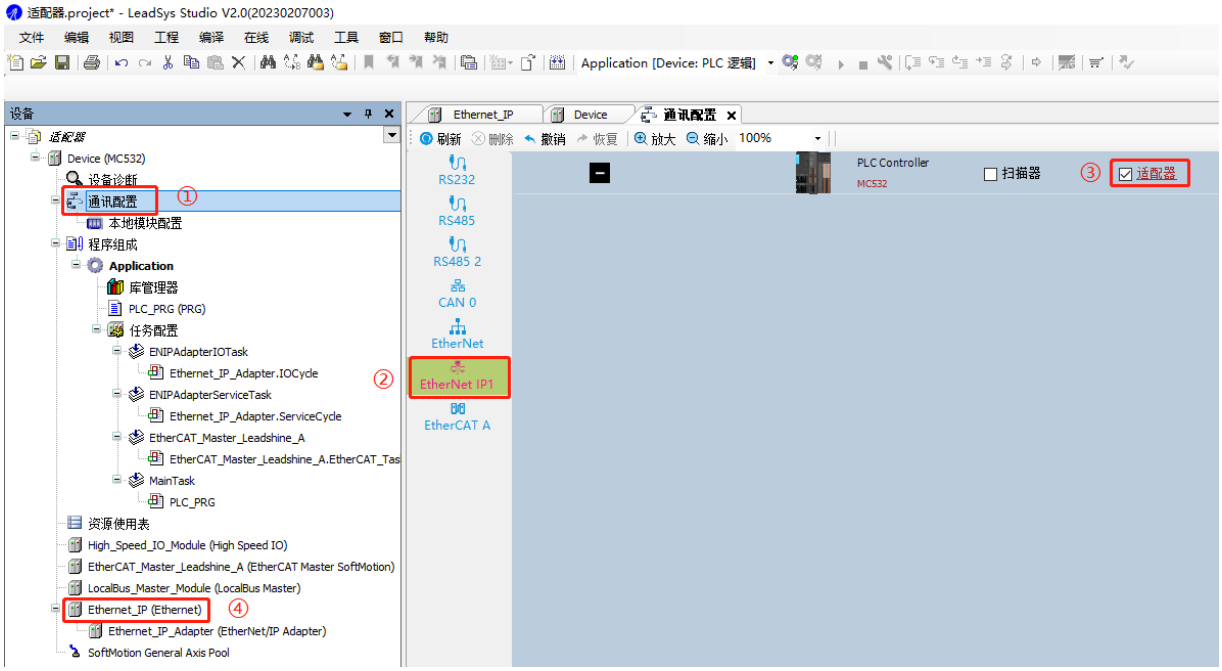


图 12.45　使能 PLC 作为 EtherNet/IP 通讯的适配器

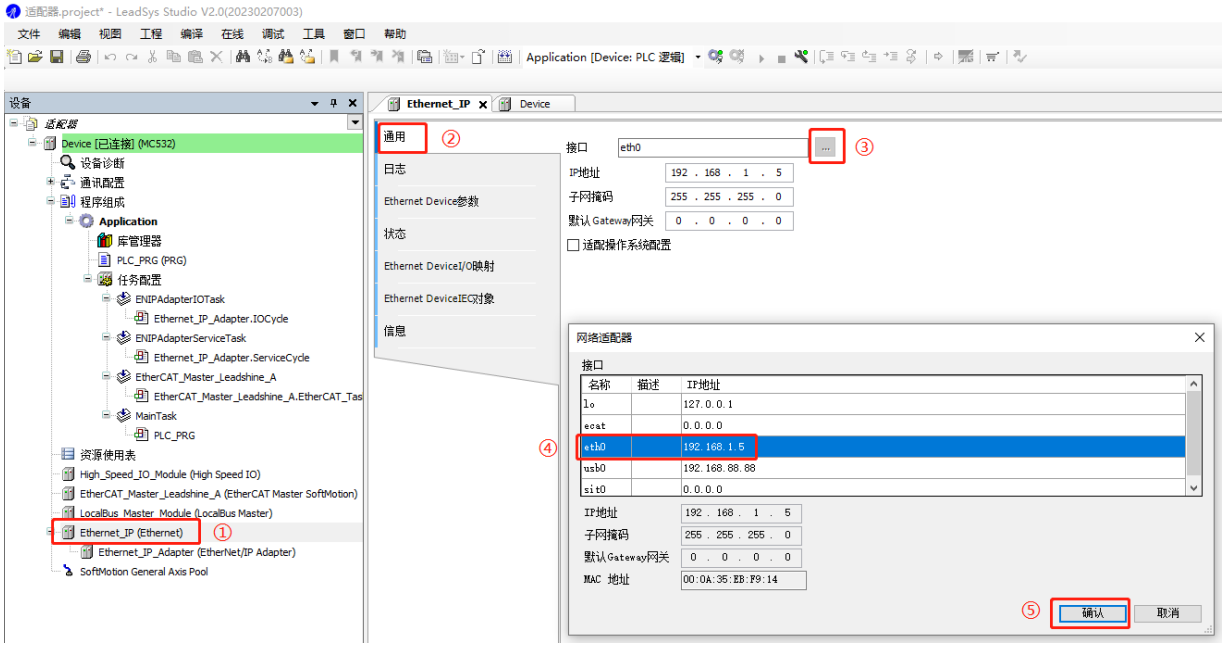


图 12.46　配置通讯的参数

适配器通讯模块类型选择操作如图 12.48 所示，数据类型、数据大小说明如表 12.6 所示。

每个模块都有对应的数据类型，可以直接映射到相同数据类型的变量中，使用 Big Input/ Output Module 模块时，该模块的数据类型为 Byte,在添加变量映射时关联首地址的方法实现映射，模块会自动根据变量大小分配 Byte 的个数。

Ethernet/IP 通讯中适配器最多仅能同时添加一个 Big Input Module 模块与一个 Big Output Module 模块。

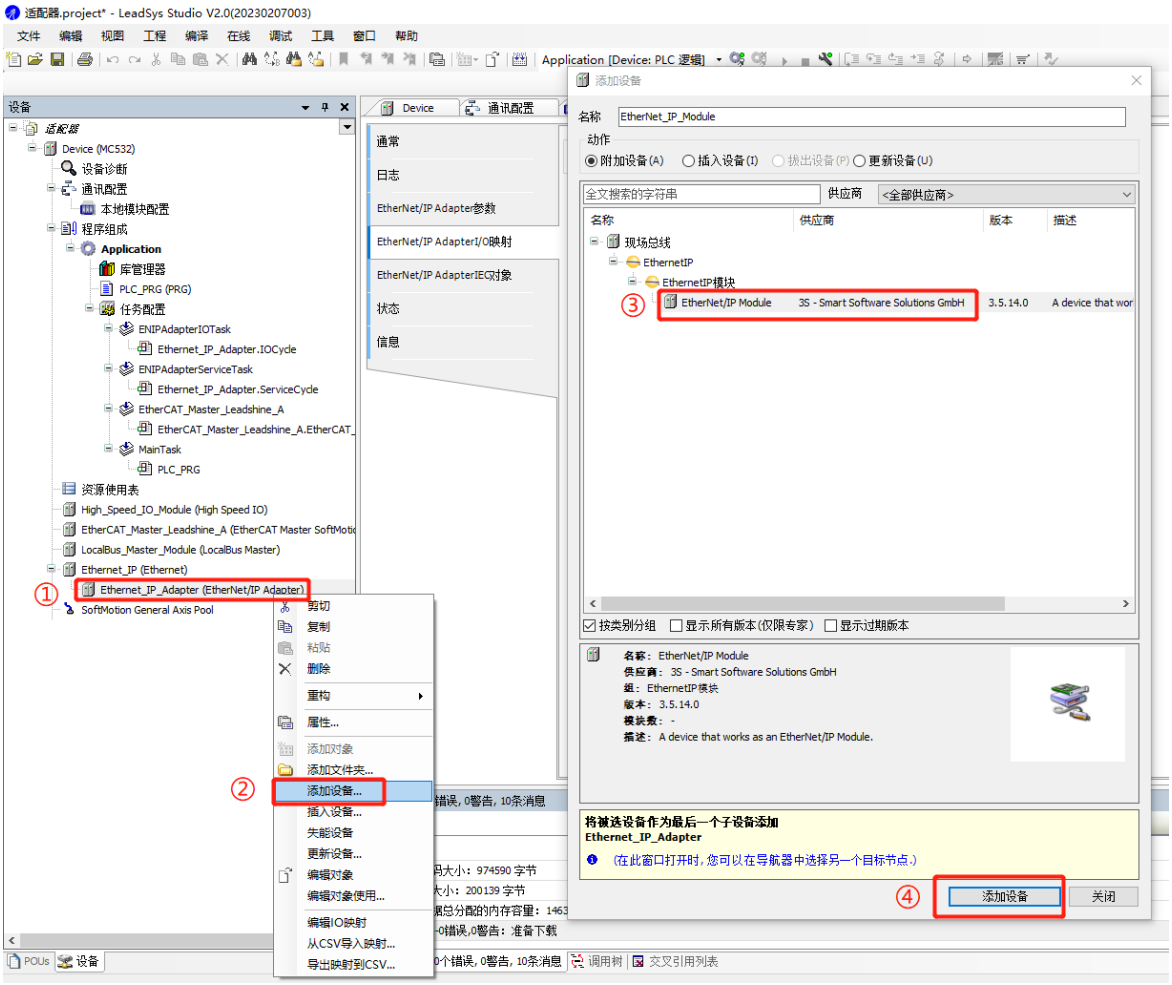


图 12.47　添加 Ethernet/IP 模块

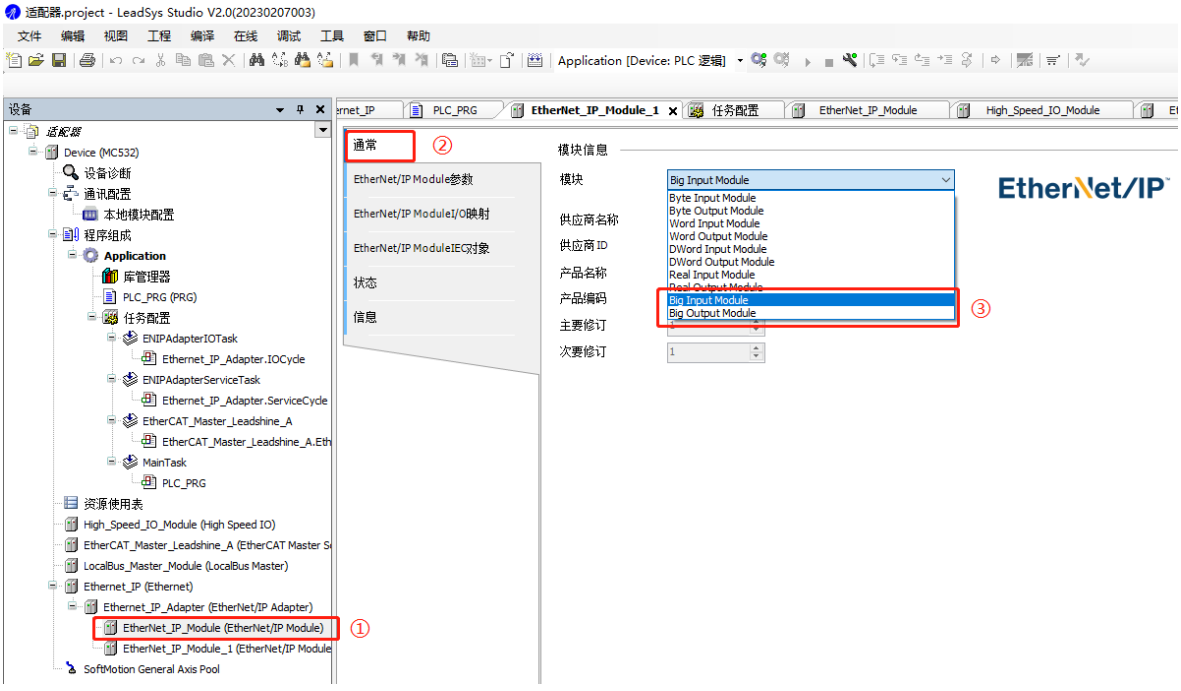


图 12.48　选择模块的类型

表 12.6　Ethernet/IP 适配器模块类型说明

模块名称	数据类型	位数
Byte Input/Output Module	1 个 Byte	8 位，8 个 BOOL
Word Input/Output Module	1 个 Word	16 位，16 个 BOOL
DWord Input/Output Module	1 个 DWord	32 位，32 个 BOOL
Real Input/Output Module	1 个 REAL	32 位
Big Input Module	504 个 Byte	
Big Output Module	508 个 Byte	

如图 12.49 所示，变量可以在③处选择通道进行映射，也可以直接在变量声明中使用通道地址。为了保证通讯正常需要将“一直更新变量：”更改为“使能 2（一直在总线循环任务中）”。

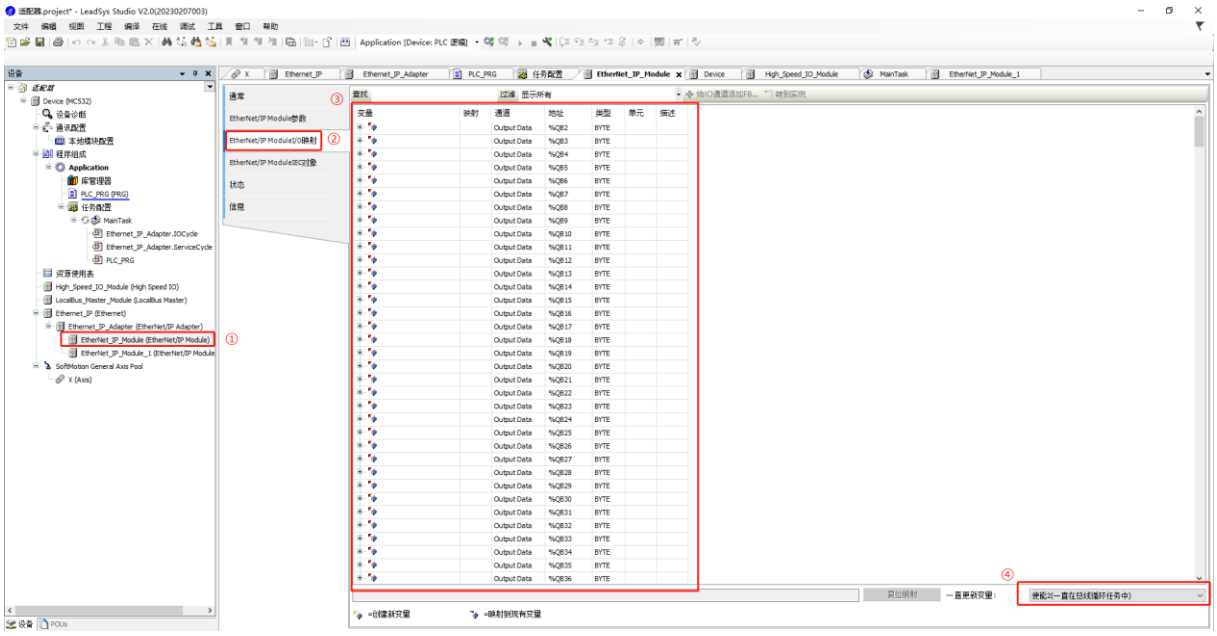


图 12.49　通讯数据的变量映射与设置

完成配置后需要将设备安装到设备库中，用于扫描器在扫描设备时能够正常添加该设备，如图 12.50 所示。

如果适配器与扫描器编程配置时不在同一台电脑操作，则需要将适配器的设备文件导出 EDS 文件，将该文件复制到扫描器编程配置的电脑上并安装到设备存储库中，如图 12.51 所示。

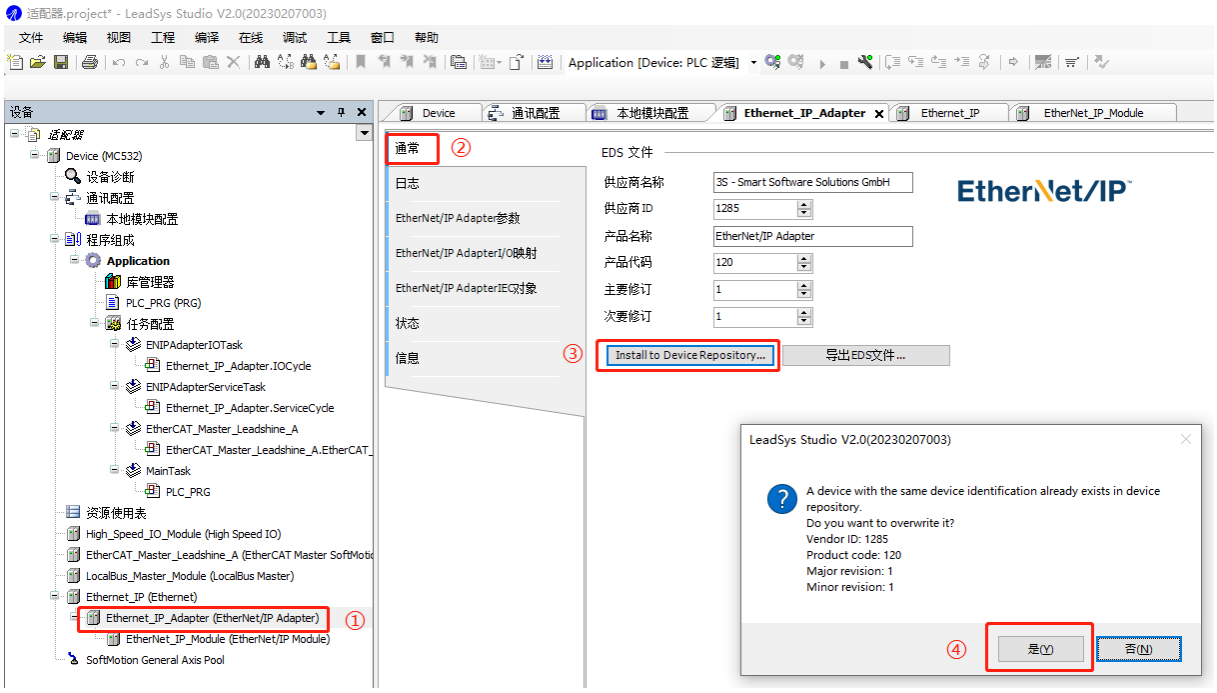


图 12.50　适配器设备文件安装到设备库中

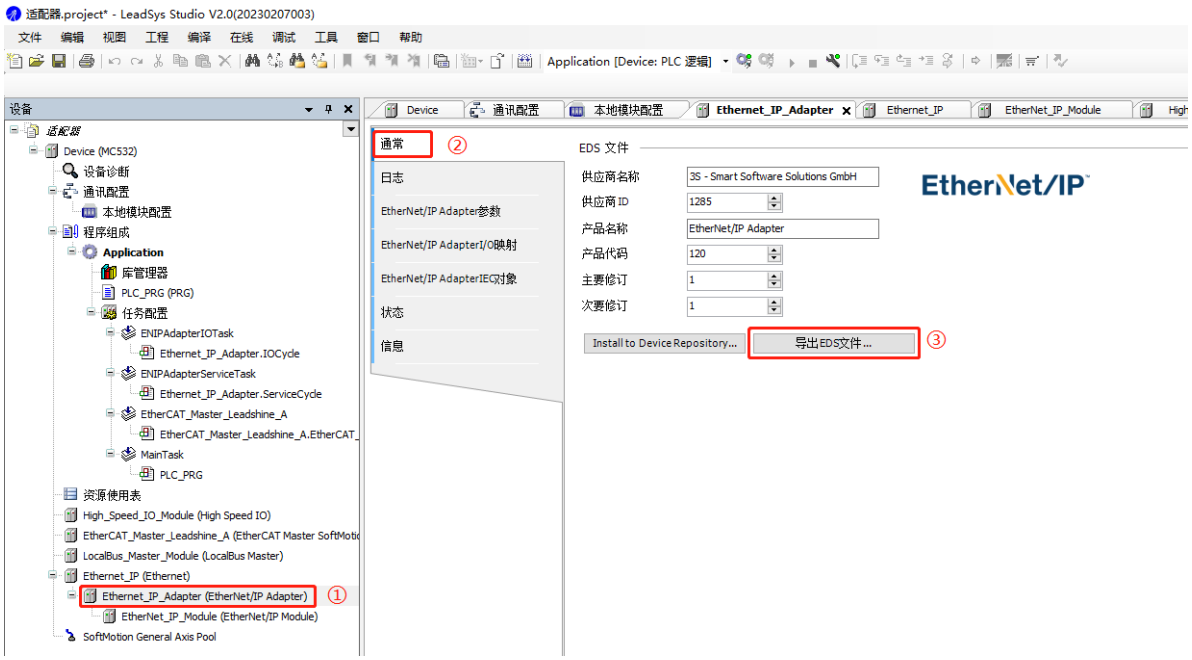


图 12.51　适配器设备文件导出 EDS 文件

12.6.2. 扫描器配置

1)　使能 PLC 作为 EtherNet/IP 扫描器

双击项目树中的“通讯配置”，选择 EtherNet IP1 通讯，勾选“扫描器”，使能后项目树中出现了对应的通讯设备“Ethernet_IP_Scanner”，如图 12.52 所示。

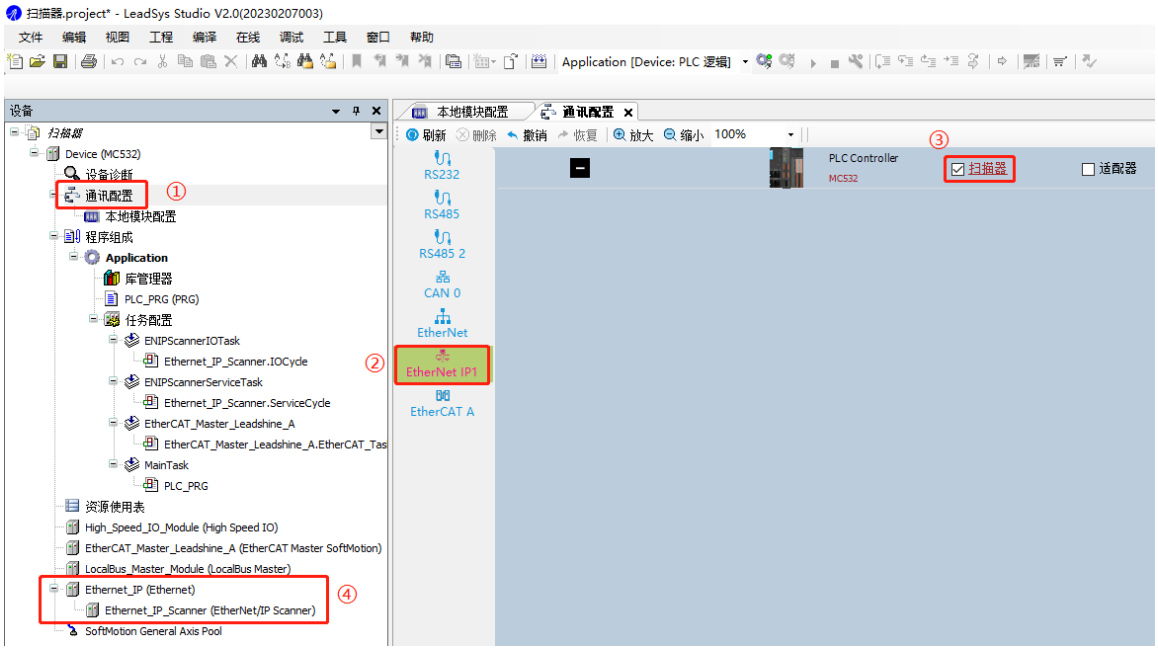


图 12.52　使能 PLC 作为 EtherNet/IP 通讯的扫描器

2）添加 EtherNet/IP 适配器设备

使能后需要配置 EtherNet/IP 扫描器的通讯参数，再通过扫描设备添加其他 EtherNet/IP 适配器的设备，扫描添加设备后需要将“一直更新变量：”选择为“使能 2（一直在总线循环任务中）”，设置过程如图 12.53 至图 12.55 所示。

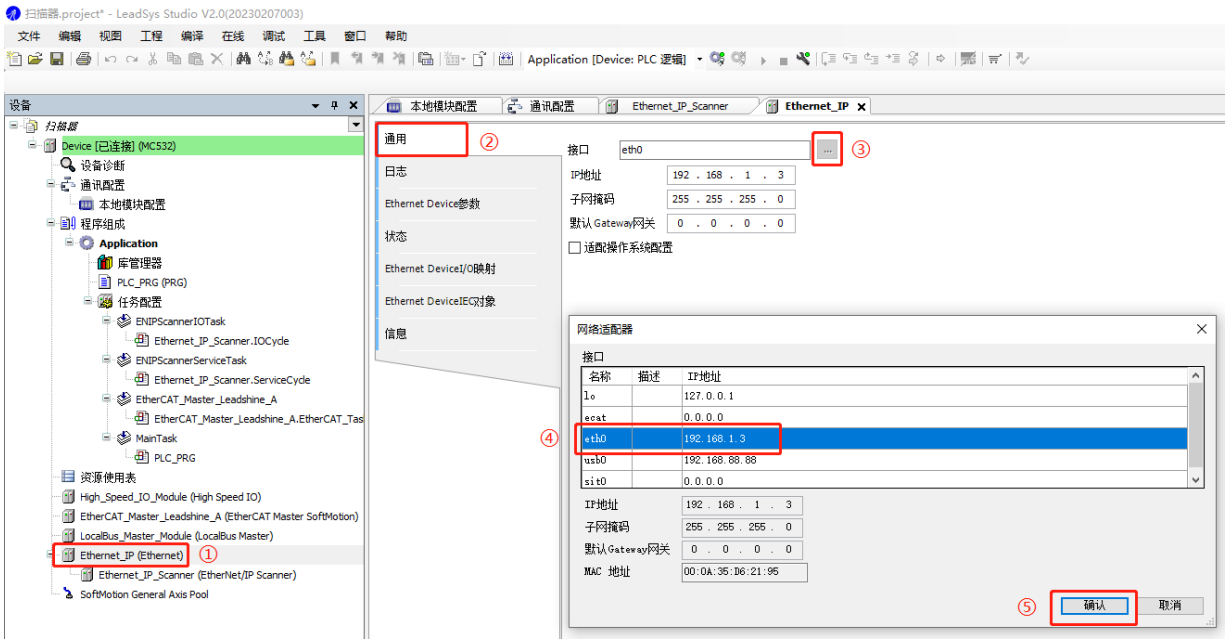


图 12.53　设置通讯的参数

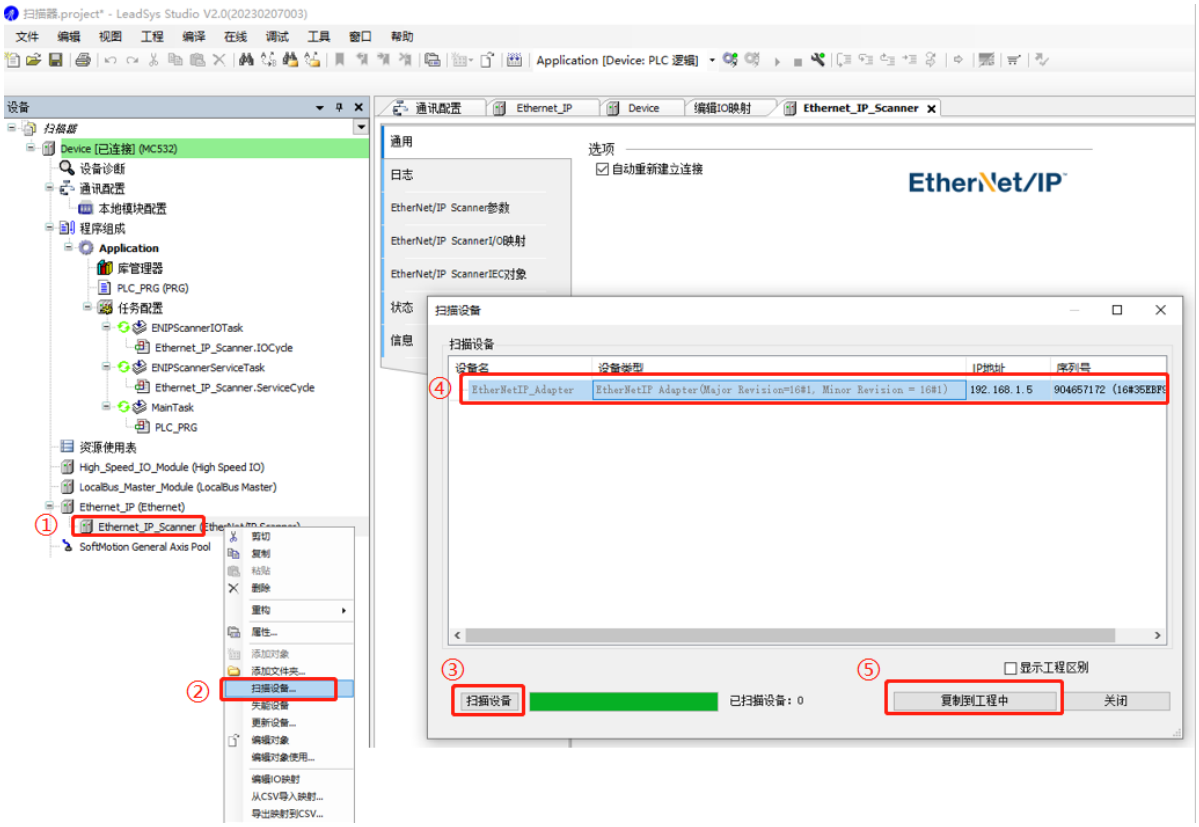


图 12.54　扫描设备添加适配器

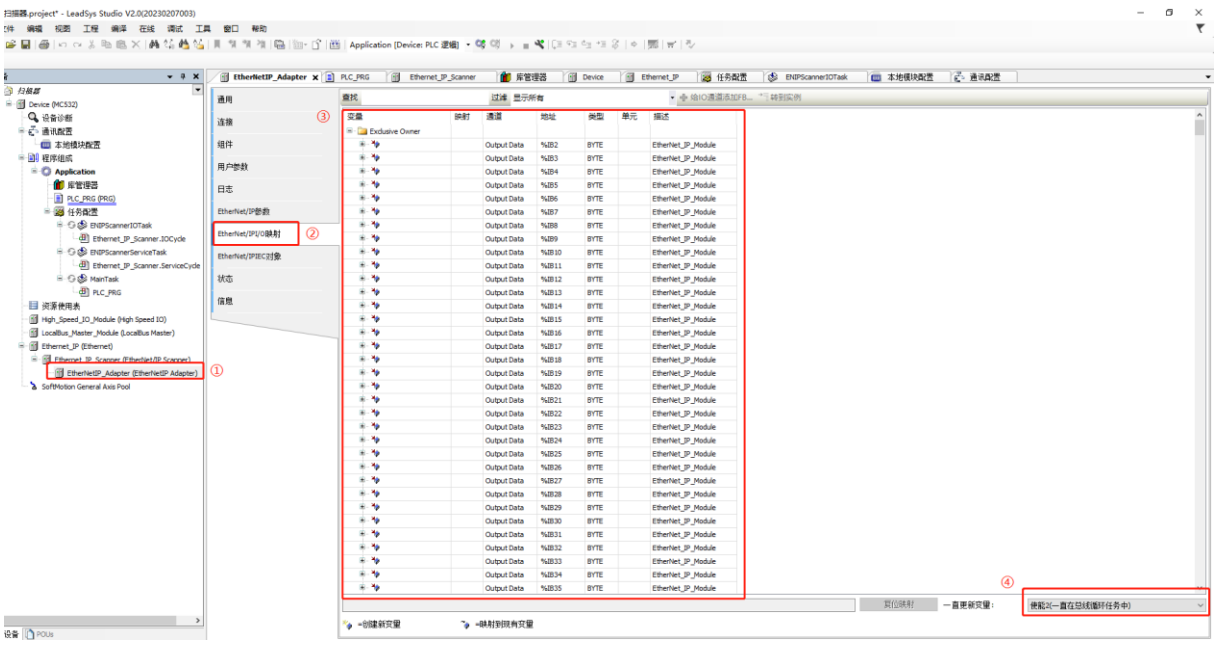


图 12.55　通讯数据的变量映射与设置

例程中的变量声明时直接使用了通讯通道的地址进行数据传输。其中适配器程序中有轴运动的指令，为了避免轴运动过程中被其他任务中断，需要将任务配置中的适配器数据传输模块的任务与 PLC_PRG 放在同一任务列表下。

例程的代码如下。

1) 扫描器程序

```
PROGRAM PLC_PRG
VAR
    Start AT %QX2.0: BOOL;              //启动信号
    State AT %IX2.0: BOOL;              //适配器状态
    Distance AT%QB3:LREAL:=10;          //相对运动距离,一个 LREAL 大小占用 8 个 BYTE
    vel AT %QB12:LREAL:=5;              // 相对运动距离
    Xpos AT %IB3:LREAL;                 // 相对运动距离
    Xspeed AT %IB12:LREAL;              // 相对运动距离
    doneCount AT %IB21:WORD;            // 运动完成次数
END_VAR

IF  Start=TRUE AND State=FALSE THEN     // 当适配器处于停止状态且启动信号为 true 时
    Distance:=12.345;                   // 运动距离
    vel:=6.789;                         // 运动速度
ELSE
    start:=FALSE; // 状态复位
END_IF
```

2) 适配器程序

```
PROGRAM PLC_PRG
VAR
    axes:dut_pulse_axis;                // 脉冲轴结构体
    MC_MoveRelative_1: MC_MoveRelative; // 相对运动实例化
    MC_Power_1: MC_Power;               // 轴使能实例化
    start AT %IX2.0: BOOL;              // 启动信号
    State  AT %QX2.0:BOOL;              // 适配器状态
    Distance AT%IB3: LREAL;             // 相对运动距离,,一个 LREAL 大小占用 8 个 BYTE
    vel AT%IB12:LREAL;                  // 相对运动速度
    Xpos AT%QB3: LREAL;                 // X 轴当前位置
    Xspeed AT%QB12: LREAL;              // X 轴当前速度
    doneCount AT%QB21: WORD;            // 完成次数
    R_TRIG_1: R_TRIG;                   // 上升沿计数
END_VAR

axes.pulaxis_0:=ADR(X);                 // 获取 X 轴的地址
LS_MotionControl_P(stAxis:=axes, xClearErr:=, fLimtAxisSpeedJump:=,
                   xDone=>, xError=>, eErrorID=>, xLimitAxisMoveFlag=> ); // 绑定脉冲轴
//轴使能:
MC_Power_1(Axis:=X ,Enable:=1 , bRegulatorOn:=1 ,bDriveStart:=1 , Status=> ,
           bRegulatorRealState=>, bDriveStartRealState=>, Busy=>, Error=>, ErrorID=> );
// 相对运动指令:
MC_MoveRelative_1(Axis:=X , Execute:=start , Distance:= Distance, Velocity:=vel ,
                  Acceleration:=100 , Deceleration:=100 , Jerk:=1000 , BufferMode:= ,
                  Done=> , Busy=>State , Active=> , CommandAborted=> , Error=> ,
                  ErrorID=> );
Xpos:=X.fActPosition;                   //X 轴当前位置
Xspeed:=X.fActVelocity;                 //X 轴当前速度
// 运动完成信号上升沿，对完成信号进行计数:
R_TRIG_1(CLK:=MC_MoveRelative_1.Done=TRUE , Q=> );
```

```
IF R_TRIG_1.Q THEN
    doneCount:=doneCount+1;                    // 更新完成次数
END_IF
```

3） 程序运行结果

当启动信号 Start 为 TRUE 时，读取到适配器的轴处于停止状态，扫描器向适配器发送运动距离、运动速度和启动信号；适配器接收到数据后，以这些运动参数执行一条相对运动指令；并向扫描器发送当前位置和速度以及轴状态。如图 12.56 所示。

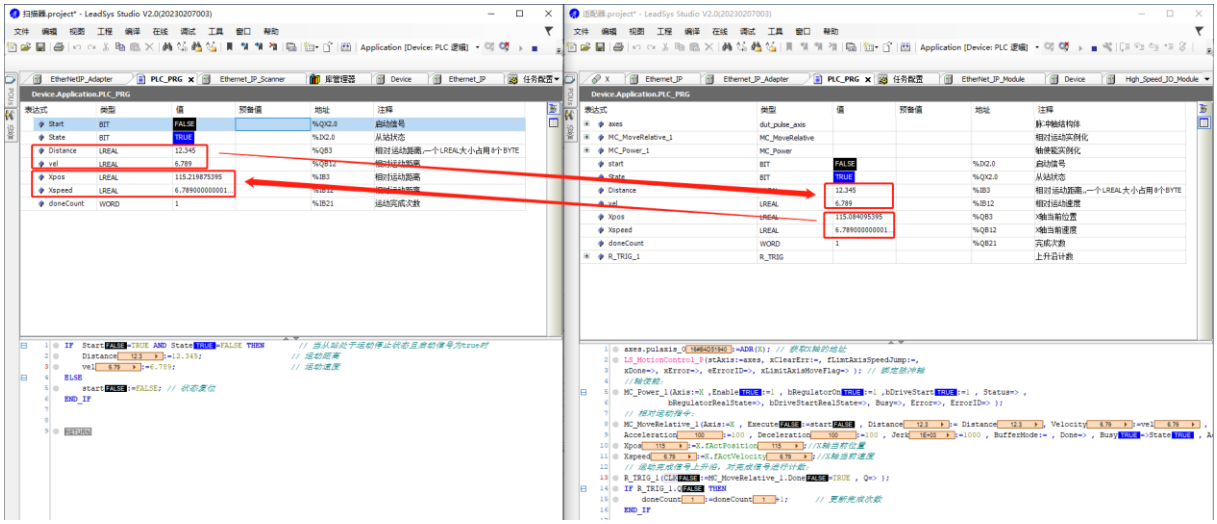


图 12.56　程序运行结果

第13章 文件处理与应用

MC300CS 系列 PLC 支持内部文件处理、SD 卡文件处理，可用于用户传输参数配置文件、用户数据以及程序下载。

MC300CS 系列 PLC 只能识别 FAT32 文件系统的 SD 卡，对于 NTFS 文件系统的 SD 卡是无法识别的。最大能处理的文件大小为 193M，支持的文件后缀名有“.data”、“.txt”、“.cnc”、“.gcode”等。

一般情况下建议用户将系统类的参数文件存储在“数据文件”文件夹内，工艺类的文件存储在“配置文件”内。文件路径如下。若操作的文件无路径仅有文件名时，则默认操作“数据文件”目录下的文件。

数据文件路径：/usr/src/CODESYSControl/UsrData/文件名称

配置文件路径：/usr/src/CODESYSControl/UsrConfig/文件名称

MC300CS 系列 PLC 的内部文件可在“IEC 控制器文件管理”中查看，打开方式如图 13.1、图 13.2 所示。

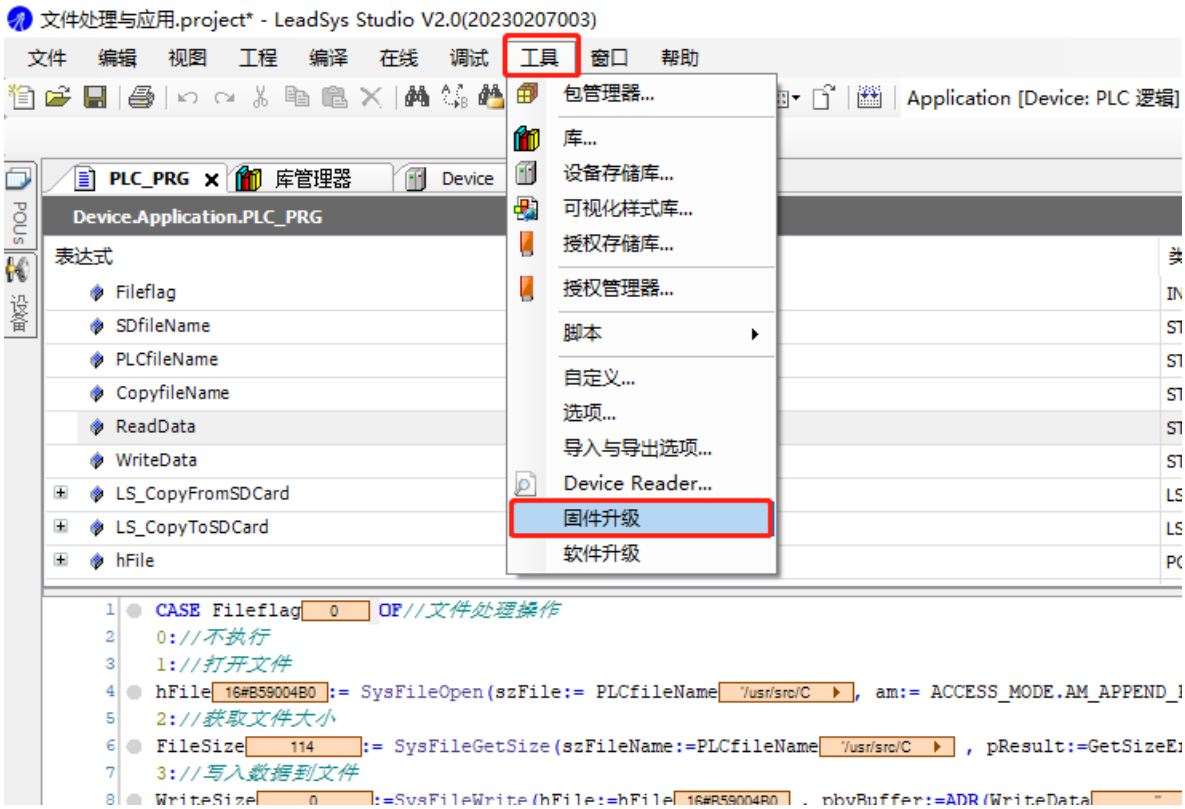


图 13.1　PLC 的内部文件查看方式

13.1. 文件处理指令

MC300CS 系列 PLC 的文件处理指令如表 13.1 所示。其中指令参数说明如表 13.2 至表 13.11 所示，详细说明请参考《雷赛大中型 PLC 指令手册》。

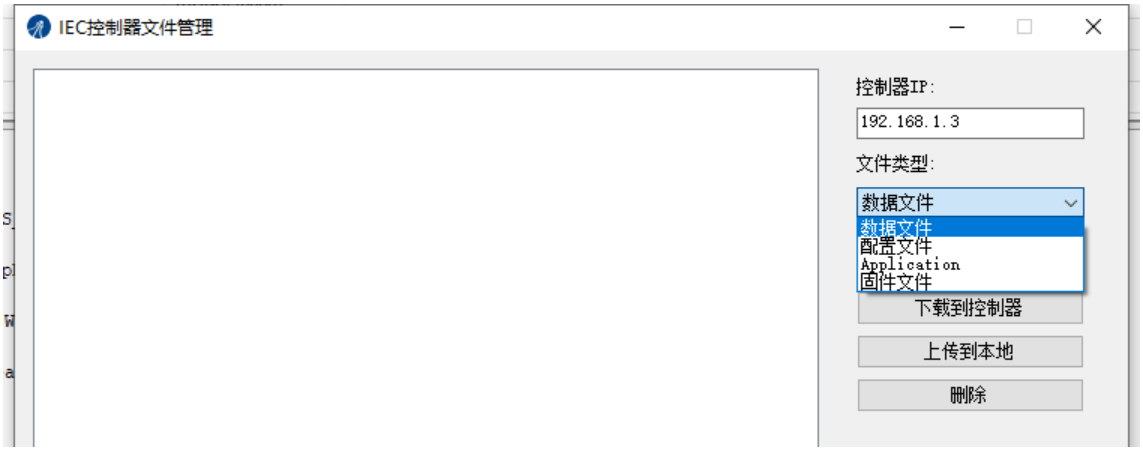


图 13.2 IEC 控制器文件管理

使用文件处理指令前需要在工程中添加“SysFile,3.5.15.0(System)”和“SysTypes Interfaces,3.5.2.0(System)”库文件，文件处理指令的错误码可在库文件“CmpErrors,3.3.1.40(System)”中查看。

表 13.1 文件处理指令列表

指令类别	名称	功能
文件处理指令	SysFileOpen	打开指定文件
	SysFileClose	关闭指定文件
	SysFileCopy	复制指定文件
	SysFileDelete	删除指定文件
	SysFileRename	重命名指定文件
	SysFileWrite	将数据内容写入文件
	SysFileRead	读取文件的数据内容
	SysFileSetPos	设置文件指针到指定位置
	SysFileGetPos	获取文件指针的位置
	SysFileGetSize	获取文件大小
	SysFileGetTime	获取文件时间

1. 打开指定文件 SysFileOpen

该指令用于打开或创建指定路径下文件，指令返回为文件句柄，可作为读写文件等指令操作的输入。其格式如下。

```
Return:=SysFileOpen(szFile:=文件名,am:=文件打开模式,pResult:=文件打开错误码指针);
```

其中：

文件打开模式类型：

AM_READ:以只读权限打开现有文件。若文件不存在，则打开失败；

AM_WRITE—创建具有写权限的新文件。若文件存在，则取消创建新文件；

AM_APPEND—以追加(仅写)权限打开现有文件。若文件不存在，则打开失败；

AM_READ_PLUS—以读/写访问权限打开现有文件。若文件不存在，则打开失败；

AM_WRITE_PLUS—创建具有读/写权限的新文件。若文件存在，则取消创建新文件；
AM_APPEND_PLUS—以追加(读/写)访问权限打开现有文件。若文件不存在，将创建新文件。

文件打开模式类型中_PLUS 模式，在文件读取、写入后都需要调用 SysFileGetPos 或 SysFileSetPos 获取指针位置，否则文件指针可能在一个无效的位置，会导致读取、写入失败。

将 SD 卡内文件数据复制到 PLC 内部的文件时只具有只读权限,若需要更多操作权限建议使用 AM_WRITE_PLUS 模式在 PLC 内部创建具有读/写权限的新文件，再将 SD 卡内文件数据复制到创建的内部文件中。

2. 关闭指定文件 SysFileClose

SysFileClose 指令用于文件读写操作结束后关闭文件，指令的返回值为关闭文件错误码。文件处理完成后必须要关闭文件，否则会导致内存溢出等问题，严重可能导致 PLC 系统崩溃。其格式如下。

```
Return:=SysFileClose(hFile:= 文件句柄);
```

3. 复制指定文件 SysFileCopy

SysFileCopy 指令可以将 szSourceFileName 的文件的内容复制到 szDestFileName 的文件中，指令的返回值为复制文件错误码。其格式如下。

```
Return:=SysFileCopy(szDestFileName:=目标文件路径及文件名,szSourceFileName:=源文件路径
                    及文件名,pulCopied:=操作的字节数);
```

其中 szDestFileName 中的文件夹路径必须是已经存在的，否则将会报错。其中若文件存在，则复制之后会覆盖原文件，若文件不存在则复制之后会创建文件。
CoSourceFileName 中的文件必须是存在的，否则将会报错。

4. 删除指定文件 SysFileDelete

SysFileDelete 指令用于删除指定文件，指令的返回值为错误码。其格式如下。

```
SysFileDelete(szFileName:= 指定文件路径及文件名);
```

5. 将数据内容写入文件 SysFileWrite

SysFileWrite 指令用于在文件打开后，写入数据内容，指令的返回值为实际写入文件字节个数。其格式如下。

```
Return:=SysFileWrite(hFile:= 文件句柄, pbyBuffer:= 写入数据变量的指针, ulSize:= 写入的
                     字节数, pResult:= 错误码指针);
```

6. 读取文件的数据内容 SysFileRead

SysFileRead 指令用于在文件打开后，读取文件的数据内容，指令的返回值为实际读取文件字节个数。其格式如下。

```
Return:=SysFileRead(hFile:= 文件句柄,pbyBuffer:=读取数据变量的指针,ulSize:=读取的字节
                    数, pResult:= 错误码指针);
```

7. 设置文件指针到指定位置 SysFileSetPos

SysFileSetPos 指令用于设置文件指针到指定位置，指令的返回值为错误码。其格式如下。

```
Return:=SysFileSetPos(hFile:= 文件句柄, ulOffset:= 偏移位置);
```

8. 获取文件指针的位置 SysFileGetPos

SysFileGetPos 指令用于读取文件指针当前位置，指令的返回值为错误码。其格式如下。

```
Return:=SysFileGetPos(hFile:= 文件句柄 , pulPos:=指针位置 );
```

9. 获取文件大小 SysFileGetSize

SysFileGetSize 指令用于获取指定文件的大小，指令的返回值为获取指定文件的字节个数。其格式如下。

```
Return:=SysFileGetSize(szFileName:=文件名, pResult:= 错误代码的指针);
```

10. 获取文件时间 SysFileGetTime

SysFileGetTime 指令用于获取指定文件的文件时间，指令的返回值为错误码。其格式如下。

```
Return:=SysFileGetTime(szFileName:= 文件句柄, ptFileTime:= 文件时间变量的指针);
```

13.2. SD 卡文件处理指令

MC300CS 系列 PLC 的 SD 卡文件处理指令如表 13.2 所示。详细说明请参考《雷赛大中型 PLC 指令手册》。

SD 卡文件处理指令中操作的文件名类型为 STRING(32)，若在操作的文件名中加入文件路径，则会因为文件名过长导致文件操作失败。因此在使用 SD 卡文件处理指令时操作的文件仅输入文件名即可，文件保存在“数据文件”的目录文件中。

表 13.2 SD 卡文件处理指令列表

指令类别	名称	功能
SD 卡文件处理指令	LS_CopyFromSDCard	从 SD 卡复制文件至 PLC
	LS_CopyToSDCard	将 PLC 文件复制至 SD 卡
	LS_GetSDCardInformation	获取 SD 卡中的文件信息

1. 从 SD 卡复制文件至 PLC LS_CopyFromSDCard

LS_CopyFromSDCard 指令能从 SD 卡复制文件至 PLC。其格式如下。

```
LS_CopyFromSDCard(xExecute:=启动信号, strSourceFileName:= 源文件名, strDestFileName:=
                  目标文件名, xDone=>执行成功, xError=>错误标志, diErrorID=>错误 ID,
                  dwCopyedSizeBytes=>拷贝的字节长度);
```

2. 将 PLC 文件复制至 SD 卡 LS_CopyToSDCard

LS_CopyToSDCard 指令能从 PLC 复制文件至 SD 卡。其格式如下。

```
LS_CopyToSDCard(xExecute:= 启动信号, strSourceFileName:= 源文件名, strDestFileName:=
```

目标文件名, xDone=>执行成功, xError=>错误标志, diErrorID=>错误 ID, dwCopyedSizeBytes=>拷贝的字节长度);

3. 获取 SD 卡中的文件信息 LS_GetSDCardInformation

LS_GetSDCardInformation 指令能获取 SD 卡中文件信息，包括文件名、文件创建时间、文件最近修改时间、文件大小。其格式如下。

LS_GetSDCardInformation(xExecute:=启动信号, strFileExtension:= 源文件名,iMaxFileNums:=允许读取最大文件数, pstFile_Information_Point:= 文件信息变量指针, xDone=>执行成功, xError=>错误标志, diErrorID=>错误 ID, iNumberOfFiles=>拷贝的字节长度);

其中错误码 diErrorID：0—无错误；

1—找不到 SD 卡；

2—创建本地文件错误；

3—打开 SD 卡文件错误；

4—文件名异常；

16—文件不存在。

13.3. 文件处理应用例程

该例程使用“样条插值指令例程”小节中的程序，将 PT 运动参数的赋值方式替换为以文件处理的方式将 SD 卡内文件名为“Point.data”的坐标点数据进行写入，坐标点数据如表 13.3 中黑体部分所示。在运动完成后输出文件名为 “State.data”，内容为“文件名+轨迹完成次数”的文件到 SD 卡。

在数据操作时，因字符串操作指令中最大数据量为 String(255)，为保证文件内数据不丢失且便于文件数据操作，建议在进行数据操作时每次操作的数据长度不超过 255 个 String,分多次进行操作。

表 13.3 “Point.data”坐标点数据

序号	时间（s）	X 轴位移	Y 轴位移	序号	时间（s）	X 轴位移	Y 轴位移
1	0.2	**25.78**	**2.26**	19	3.8	**8.42**	**-96.23**
2	0.4	**49.24**	**8.68**	20	4	**15.04**	**-85.29**
3	0.6	**68.3**	**18.3**	21	4.2	**18.3**	**-68.3**
4	0.8	**81.38**	**29.62**	22	4.4	**17.1**	**-46.99**
5	1	**87.54**	**40.82**	23	4.6	**10.94**	**-23.46**
6	1.2	**86.6**	**50**	24	4.8	**0**	**0**
7	1.4	**79.12**	**55.4**	25	5	**-14.84**	**21.2**
8	1.6	**66.34**	**55.67**	26	5.2	**-32.14**	**38.3**
9	1.8	**50**	**50**	27	5.4	**-50**	**50**
10	2	**32.14**	**38.3**	28	5.6	**-66.34**	**55.67**
11	2.2	**14.85**	**21.2**	29	5.8	**-79.12**	**55.4**
12	2.4	**0**	**0**	30	6	**-86.6**	**50**
13	2.6	**-10.94**	**-23.46**	31	6.2	**-87.54**	**40.82**
14	2.8	**-17.1**	**-46.98**	32	6.4	**-81.38**	**29.62**
15	3	**-18.3**	**-68.3**	33	6.6	**-68.3**	**18.3**
16	3.2	**-15.04**	**-85.29**	34	6.8	**-49.24**	**8.68**
17	3.4	**-8.42**	**-96.22**	35	7	**-25.78**	**2.26**
18	3.6	**0**	**-100**	36	7.2	**0**	**0**

1) 变量声明

```
PROGRAM PLC_PRG
  VAR
    kx: INT:=1;
    ky: INT:=1;
    Count: INT;                 //字符计数
    xPOS: STRING;               //X 轴坐标数据
    yPOS: STRING;               //Y 轴坐标数据
    Pointflag: BOOL;            //数据读取信号
    doneCount:INT;              //完成计数
    Fileflag: INT;              //文件操作事件类型
    SDfileName:STRING;
    SDfileName_IN:STRING:='Point.data';              //读取的 SD 卡文件名
    SDfileName_OUT:STRING:='State.data';             //写入的 SD 卡文件名
    PLCfileName:STRING;
    //数据文件路径和文件名
    PLCfileName_IN:STRING:='/usr/src/CODESYSControl/UsrData/PLC_Point.txt';
    //状态文件路径和文件名
    PLCfileName_OUT:STRING:='/usr/src/CODESYSControl/UsrData/PLC_state.txt';
    ReadData:ARRAY[0..10] OF STRING(300);            //从 PLC_Point.txt 中读取数据
    WriteData:STRING;//写入的数据
    LS_CopyFromSDCard:LS_CopyFromSDCard;             //从 SD 卡中拷贝文件指令实例化
    LS_CopyToSDCard:LS_CopyToSDCard;                 //拷贝文件到 SD 卡指令实例化
    hFile:RTS_IEC_HANDLE;                               //文件句柄
    FileSize: __XWORD;                                  //文件大小
    ReadSize: __XWORD;                                  //读取到的数据大小
    WriteSize: __XWORD;                                 //写入的数据大小
    OpenErrorID: POINTER TO SysFile.RTS_IEC_RESULT;     //打开文件的错误码 ID
    ReadErrorID: POINTER TO SysFile.RTS_IEC_RESULT;     //读取文件数据的错误码 ID
    GetSizeErrorID: POINTER TO SysFile.RTS_IEC_RESULT;  //获取文件大小的错误码 ID
    WriteErrorID: POINTER TO SysFile.RTS_IEC_RESULT;    //获取文件大小的错误码 ID
    CloseErrorID: RTS_IEC_RESULT;                       //关闭文件的错误码 ID
END_VAR
```

2) 程序内容

```
IF Pointflag = TRUE THEN                                  // PT 运动参数的赋值
    PLCfileName:=PLCfileName_IN;
    PTdataX.IsAbsolute := TRUE;                           // 绝对坐标
    PTdataX.Number_of_pairs :=36;                         // PT 数据点的个数
    PTdataY.IsAbsolute := TRUE;
    PTdataY.Number_of_pairs :=36;
    CASE Fileflag OF
    0://从 SD 卡中拷贝文件到 PLC
    LS_CopyFromSDCard(xExecute:=TRUE ,strSourceFileName:=SDfileName_IN ,
                      strDestFileName:='PLC_Point.txt',xDone=> ,xError=> ,diErrorID=>,
                      dwCopyedSizeBytes=>);
```

```
    IF LS_CopyFromSDCard.xDone=TRUE THEN
        Fileflag:=1;
    END_IF
    1://打开文件
hFile:=SysFileOpen(szFile:=PLCfileName,am:=ACCESS_MODE.AM_READ,pResult:=OpenErrorID );
    IF hFile<>16#00000000 AND hFile<>16#FFFFFFFF THEN
        Fileflag:=2;
    END_IF
    2://获取文件大小
    FileSize:= SysFileGetSize(szFileName:=PLCfileName , pResult:=GetSizeErrorID );
    IF FileSize>0   THEN
        Fileflag:=3;
    END_IF
    3://读取文件数据
    FOR i:=0 TO  TO_INT(FileSize/255) DO
//因字符串操作指令最大数据量为 String(255)，为便于后续操作，故将对数据分多次读取
    ReadSize:=SysFileRead(hFile:=hFile , pbyBuffer:=ADR(ReadData[i]) , ulSize:=255 ,
                          pResult:= ReadErrorID);
    IF i>0 AND LEN(STR:= ReadData[i-1])>0 THEN      //判断数组元素中剩余的数据
        ReadData[i]:=CONCAT(STR1:=ReadData[i-1] , STR2:=ReadData[i] );
    END_IF
    IF ReadSize>0 AND kx<PTdataX.Number_of_pairs THEN
WHILE FIND(STR1:=ReadData[i],STR2:='$T')>0 OR FIND(STR1:=ReadData[i],STR2:='$R$N')>0DO
//$T 为空格符，$R$N 为回车符
    IF kx <=ky THEN
    Count:=FIND(STR1:=ReadData[i] , STR2:='$T');    //$T 符前为 X 轴坐标数据
    PTdataX.MC_TP_Array[kx].position :=STRING_TO_LREAL(LEFT(STR:=ReadData[i] ,
    SIZE:=Count-1 ));
    PTdataX.MC_TP_Array[kx].delta_time :=T#200MS;
    ReadData[i]:=DELETE(STR:=ReadData[i] , LEN:=Count , POS:=0 );
        IF Count>0 THEN
            kx:=kx+1;
        END_IF
     ELSE
    Count:=FIND(STR1:=ReadData[i] , STR2:='$R$N');  //$R$N 符前为 Y 轴坐标数据
    PTdataY.MC_TP_Array[ky].position :=STRING_TO_LREAL(LEFT(STR:=ReadData[i] ,
    SIZE:=Count-1 ));
    PTdataY.MC_TP_Array[ky].delta_time :=T#200MS;
    ReadData[i]:=DELETE(STR:=ReadData[i] , LEN:=Count+2 , POS:=0 );
        IF Count>0 THEN
            ky:=ky+1;
        END_IF
     END_IF
    END_WHILE
    ELSE
```

```
        CloseErrorID:=SysFileClose(hFile:=hFile);//关闭文件
    END_IF
        Pointflag:=FALSE;
    END_FOR
    END_CASE
END_IF
IF donePTx AND donePTy  THEN                          //轨迹完成信号
    startPT := FALSE;
    doneCount:=doneCount+1;                           //轨迹完成次数
    PLCfileName:=PLCfileName_OUT;
    WriteData:=CONCAT(SDfileName_IN ,':The trajectory file was successfully executed.');
    WriteData:=CONCAT(WriteData ,TO_STRING(doneCount));
    WriteData:=CONCAT(WriteData, '$R$N');
    hFile:= SysFileOpen(szFile:= PLCfileName, am:= ACCESS_MODE.AM_APPEND,
                        pResult:=OpenErrorID );        //打开文件
    IF hFile=16#FFFFFFFF THEN
    hFile:= SysFileOpen(szFile:= PLCfileName, am:= ACCESS_MODE.AM_WRITE_PLUS,
                        pResult:=OpenErrorID );        //打开文件
    END_IF
    IF hFile<>16#00000000 AND hFile<>16#FFFFFFFF THEN
    //数据写入
        WriteSize:=SysFileWrite(hFile:=hFile , pbyBuffer:=ADR(WriteData),ulSize:=
                            INT_TO_DWORD(LEN(STR:= WriteData)),pResult:=WriteErrorID
        CloseErrorID:=SysFileClose(hFile:=hFile);       //关闭文件
        IF WriteSize>0 AND CloseErrorID=16#0000000 THEN
            //将 PLC 内部文件复制到 SD 卡
        LS_CopyToSDCard(xExecute:=TRUE, strSourceFileName:= 'PLC_state.txt',
                        strDestFileName:=SDfileName_OUT , xDone=> , xError=> ,
                        diErrorID=> , dwCopyedSizeBytes=> );
            IF LS_CopyToSDCard.xDone=TRUE   THEN
                donePTx:=FALSE;
                donePTy:=FALSE;
            END_IF
        END_IF
    END_IF
END_IF
```

3) 程序运行结果

当 Pointflag 为 TRUE 时，执行读取 SD 卡名称为“Point.data”的文件并在 PLC 内部创建文件名为“PLC_Point.txt”的程序，将 PT 运动参数赋值到对应的参数中，Input0 为 TRUE 时启动样条插值轨迹，运行结果如图 13.3 所示。

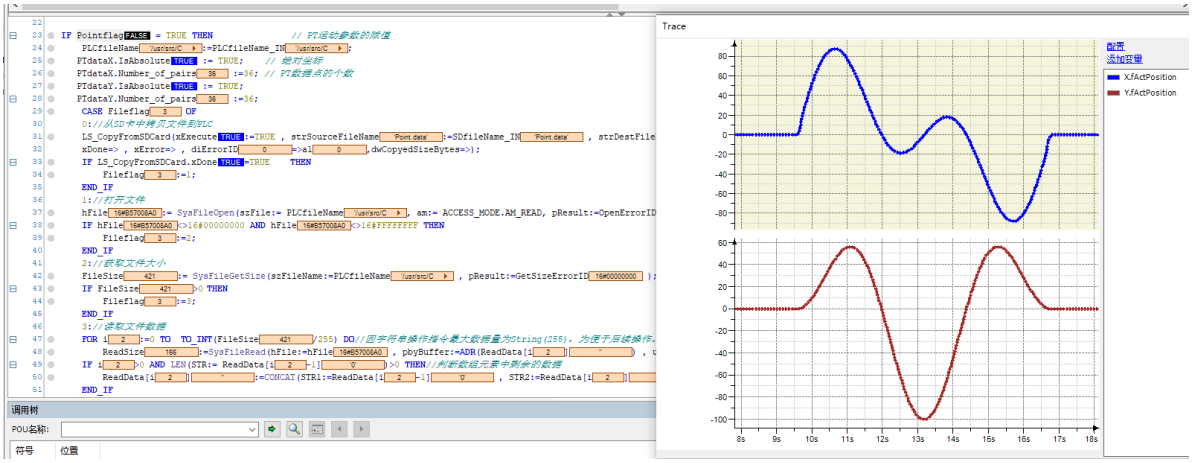


图 13.3　从 SD 卡读取数据文件并运行

当运动结束后，执行将 PLC 内部文件“PLC_ State.txt”复制到 SD 卡并将文件命名为“State.data”的程序，使用读卡器读取 SD 卡找到并将“State.data”文件后缀名改为“.txt”即可通过记事本打开查看，运行结果如图 13.4 所示。

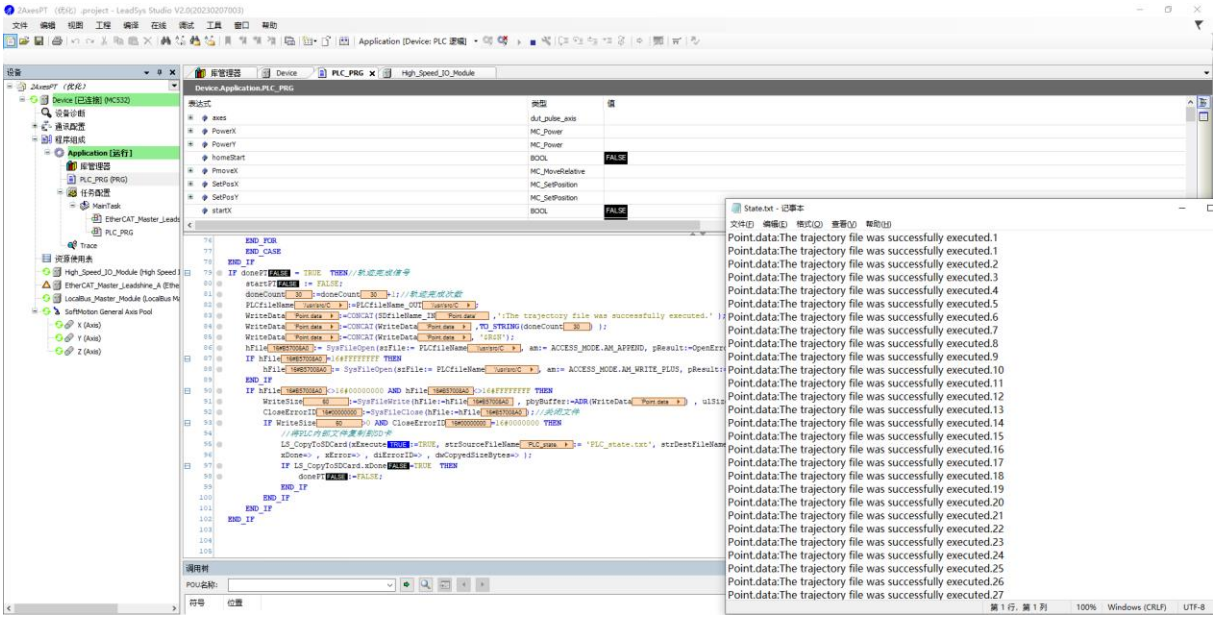


图 13.4　将 PLC 内部文件复制到 SD 卡

13.4.　SD 卡下载程序

MC300CS 系列 PLC 可使用 SD 卡中保存的应用程序包下载并替换 PLC 中的应用程序。具体操作如下：

1）　生成 PLC 应用程序包

编写完程序且编译通过后，通过“创建启动应用程序”生成应用程序包 Application.app 和 Application.crc 这两个文件，操作如图 13.5 所示。

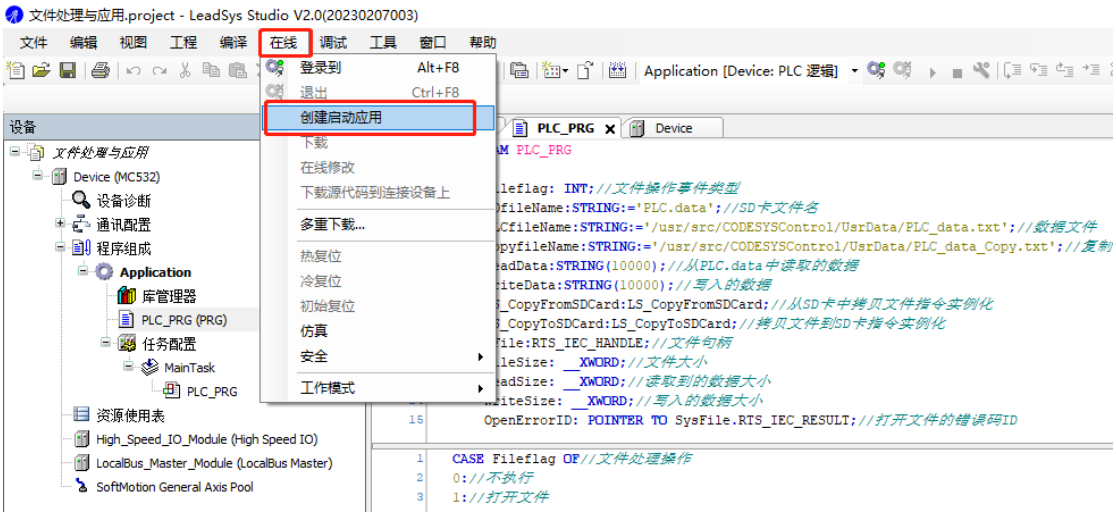


图 13.5　生成应用程序包

2)　将应用程序包存放在 SD 卡的根目录下，如图 13.6 所示。

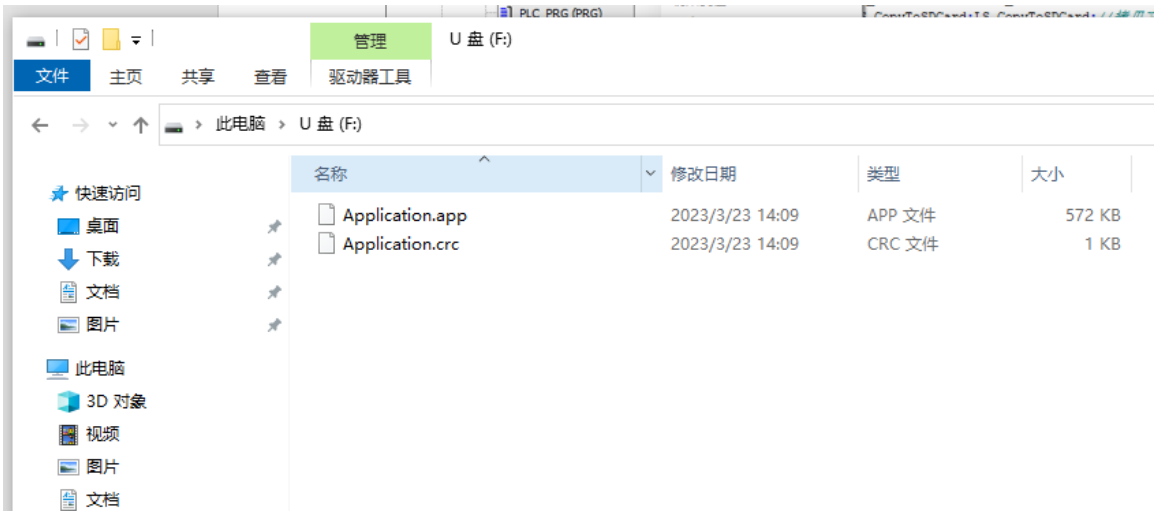


图 13.6　应用程序包存放位置

3)　插入 SD 卡下载程序

在 PLC 的 SD 卡槽中插入包含应用程序包的 SD 卡，将控制器断电重启后自动进行程序下载。程序下载完成后，程序运行中 RUN 指示灯常亮。若下载失败或程序未运行，RUN 指示灯熄灭。

第14章 故障诊断与修复

LeadSys Studio 的编程语言具有执行效率高、编程灵活等特点，但是对用户的编程能力也有较高要求。

用户在编写程序时需要避免指针非法访问、除数为 0、数组越界、类型隐式转换、死循环等异常操作，否则将可能导致 PLC 系统出现执行异常甚至崩溃的情况。

本章介绍 PLC 出现异常情况时的一些处理方法，用户可根据实际情况参考使用。

14.1. 程序编译错误的定位

1） 编译错误的定位

程序编译后，LeadSys 软件默认显示输出编译错误信息。在编译输出窗口，可以查看编译错误，双击错误显示行，可以定位到出错的代码，如图 14.1 所示。

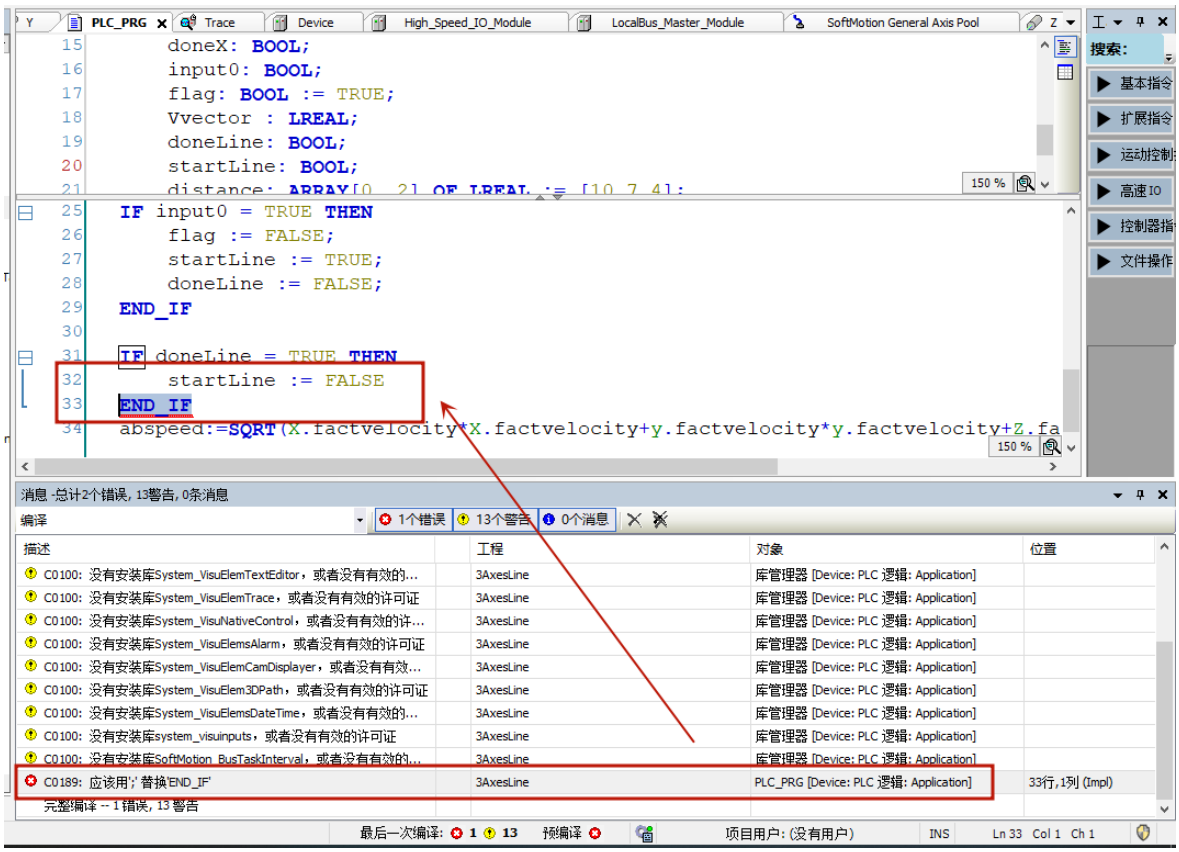


图 14.1　定位到出错代码位置

根据提示，在代码后添加“；”后重新编译通过，没有报错，如图 14.2 所示。

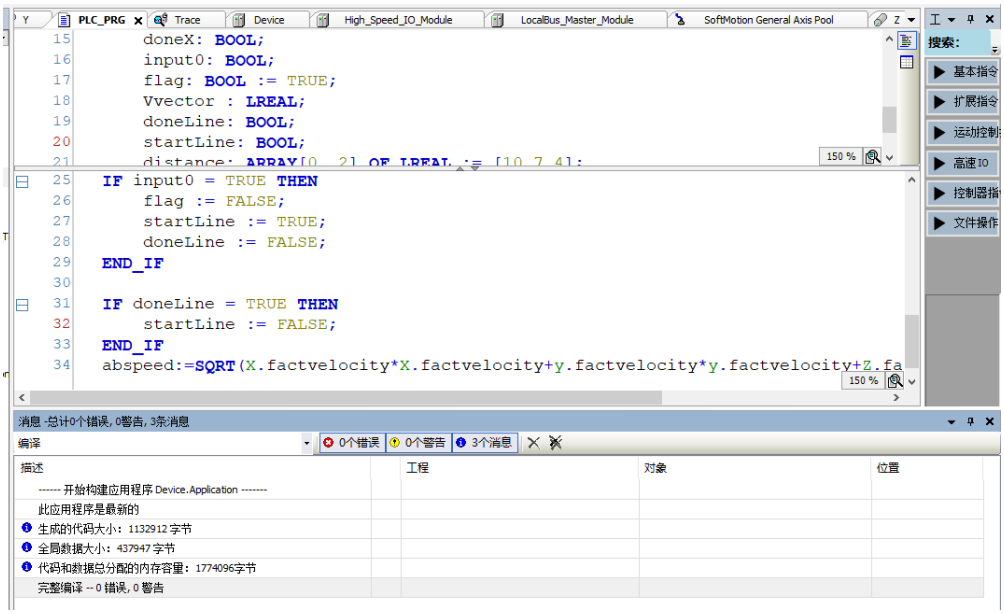


图 14.2　重新编译后 0 错误

2） 预编译错误的定位

在显示编译错误信息窗口，可切换到“预编译”，查看预编译的错误，如图 14.3 所示。

预编译错误，是在编译之前进行代码语法分析后得到的错误信息。“预编译”输出显示，如图 14.4 所示，有 4 个错误。

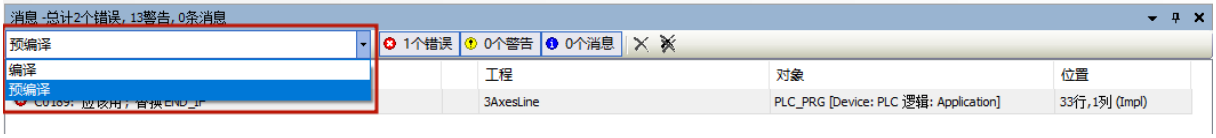


图 14.3　切换至“预编译”

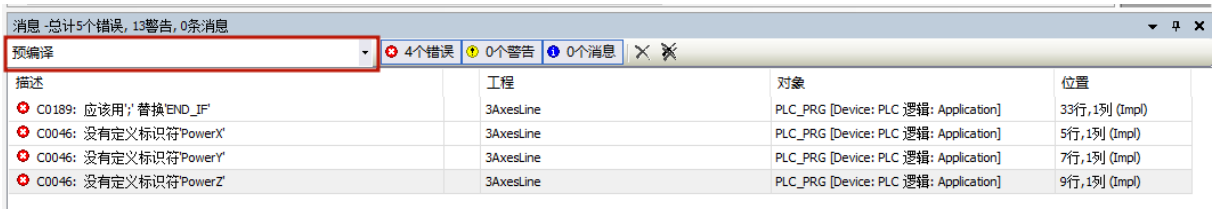


图 14.4　“预编译”显示的提示

3） 算法库错误信息

用同样的方法可切换到“库管理器”，查看算法库错误提示信息，如图 14.5 所示。

打开库管理器，如图 14.6 所示，如果不用到此库，可删除。用到了此库，则手动安装算法库。

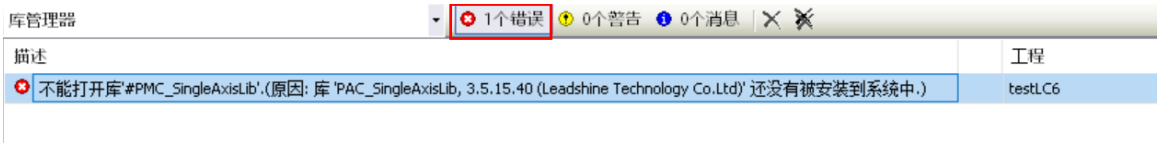


图 14.5　“库管理器”显示的提示

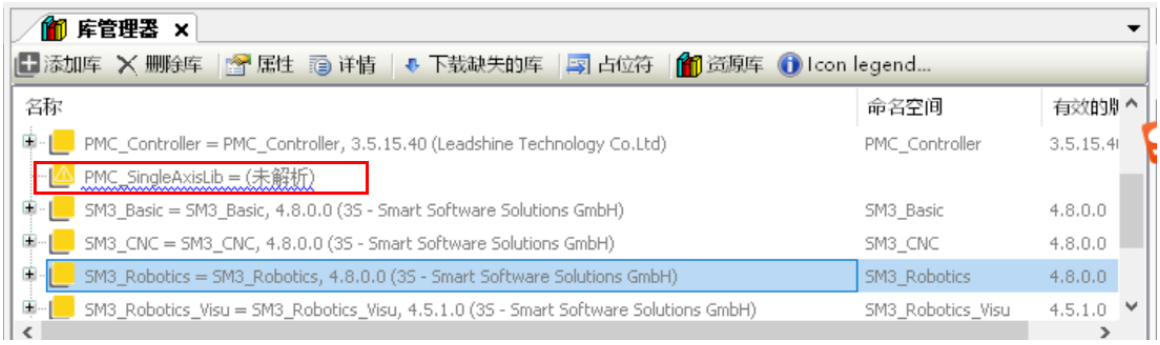


图 14.6　打开库管理器，添加或删除某个库

14.2. PLC 程序运行异常处理

在编写、调试程序的过程中可能遇见以下 4 种异常现象：

1. 下载程序后程序无法扫描到相应的 PLC 设备，PLC 运行拨码置为 stop 之后再重新上电，可正常扫描。
2. 下载程序时或者运行一段时间后，PLC 信息显示栏提示错误内容“程序下载-异常”，如图 14.7 所示。

图 14.7　程序下载异常

此时在日志界面可以看到有 E 类型错误，程序无法正常运行，如图 14.8 所示。

图 14.8 日志界面

3. 下载程序后或者运行一段时间，登录 PLC 后，弹出“没有可用于此对象的源代码…”提示框，如图 14.9 所示。

图 14.9　提示框，“没有源代码使用于这个项目…”

4. 下载程序时，弹出“下载失败：PLC 异常.”提示框，如图 14.10 所示。

图 14.10　提示框，“程序下载失败”

排查上述异常现象的措施有 3 种：

1. 隐含检查的 POU — 指针检查

该方法适用于排查空指针（即指针为 0）的情况。具体操作步骤如下：

1）鼠标右键点击“Application”，选择“添加对象”→“用于隐式检查的 POU”，如图 14.11 所示；

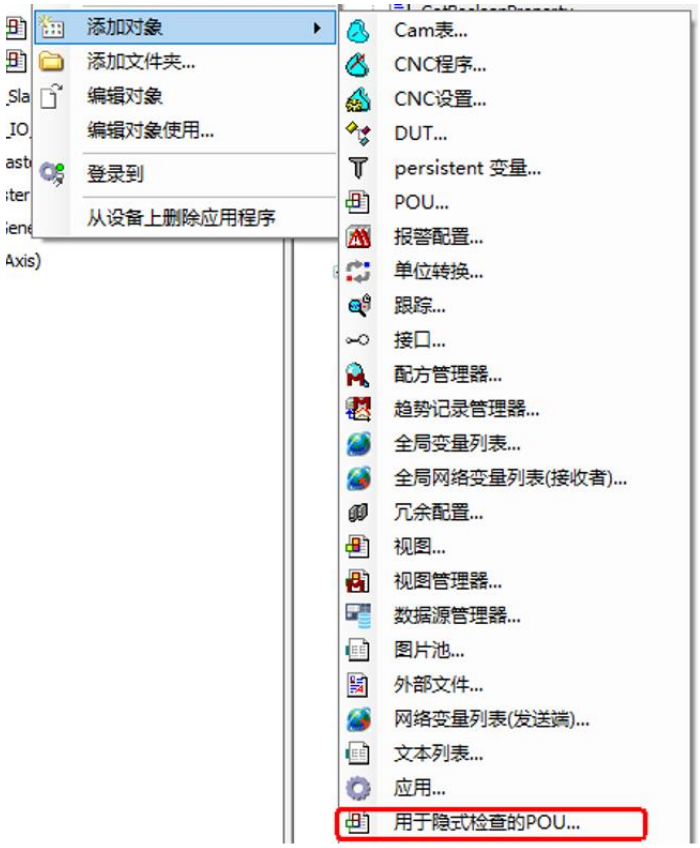


图 14.11　添加“用于隐式检测的 POU”

2）在弹出界面勾选“指针检查”，点击“打开”，如图 14.12 所示；

3）如图 14.13 所示，在添加的函数“CheckPointer”里面手动输入以下代码：

```
IF ptToTest = 0 THEN
  CheckPointer := ptToTest;
END_IF
```

4）登录 PLC，将程序运行状态切换至“停止”状态；

5）在新增代码处添加断点，如图 14.14 所示，并激活（快捷键 F9）；

图 14.12　勾选“指针检查”

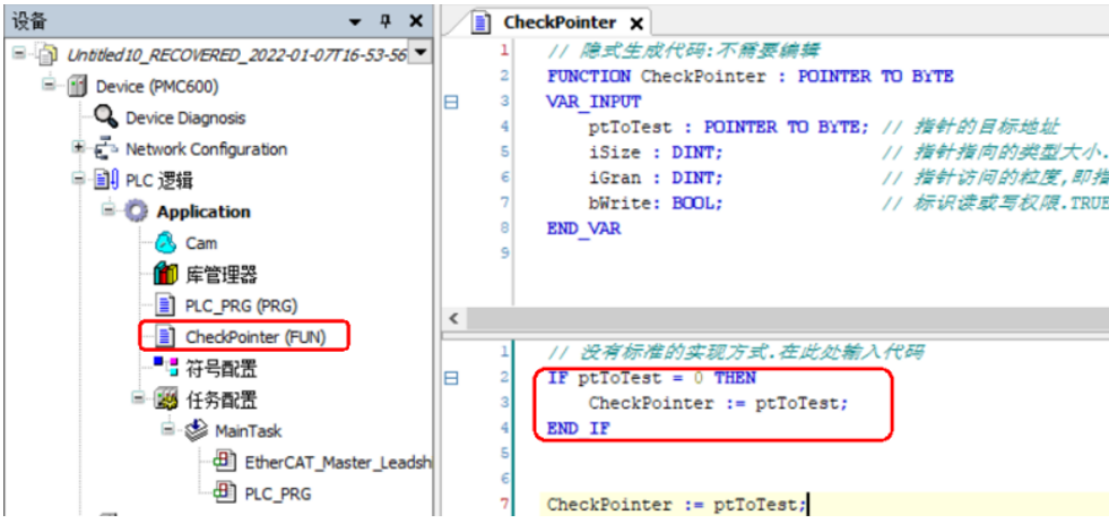


图 14.13　“指针检查”的代码

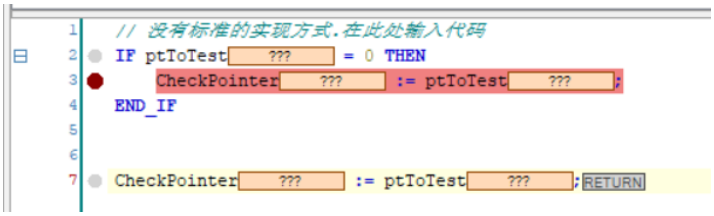


图 14.14　添加断点

6）运行程序，当检查到空指针时，进入断点，如图 14.15 所示；

7）当进入断点后，按 F10，直到定位到程序中出现问题的位置，如图 14.16 所示；

8）定位到问题程序段后，用户需要根据实际情况修改程序，避免异常出现。

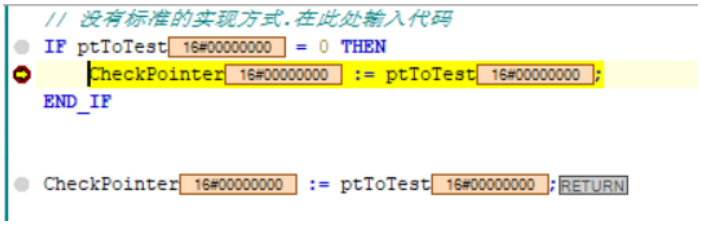


图 14.15　程序运行，进入断点

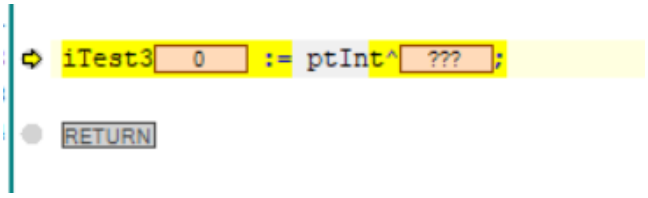


图 14.16　按 F10 键，定位到程序出错处

2. 隐含检查的 POU — 绑定检查

该方法适用于排查数组越界的情况。具体操作步骤如下：

1）右键点击“Application”，选择“添加对象”→“用于隐式检查的 POU”；

2）在弹出界面勾选“绑定检查”，点击“打开”，如图 14.17 所示；

3）登录 PLC，停止程序运行，在新增的绑定函数“CheckBounds”里面的代码“CheckBounds := lower;”和“CheckBounds := upper;”处添加断点并激活（快捷键 F9），如图 14.18 所示；

4）　运行程序，当检测到数组越界后，进入断点，如图 14.19 所示；

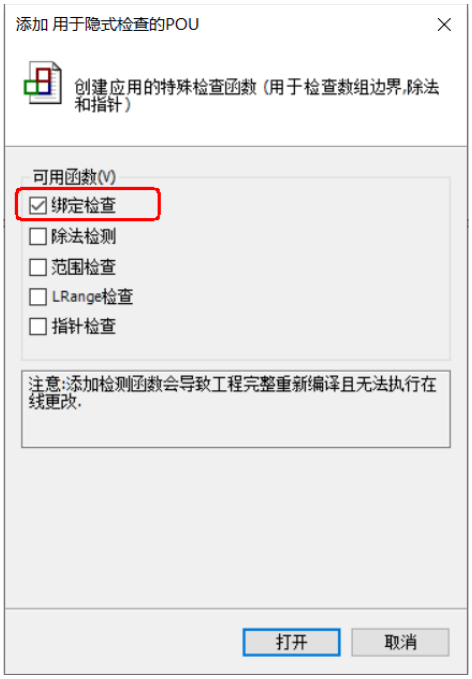


图 14.17　勾选“绑定检查”

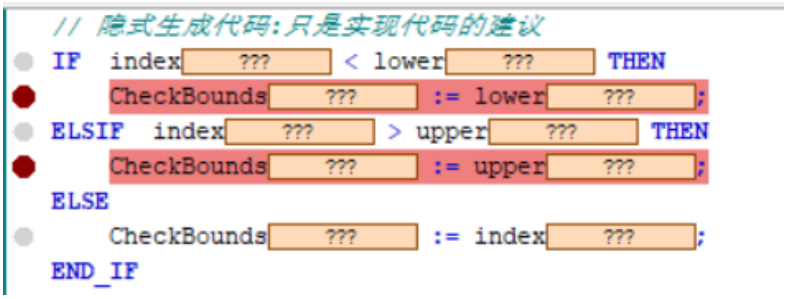


图 14.18 在程序中添加断点并激活

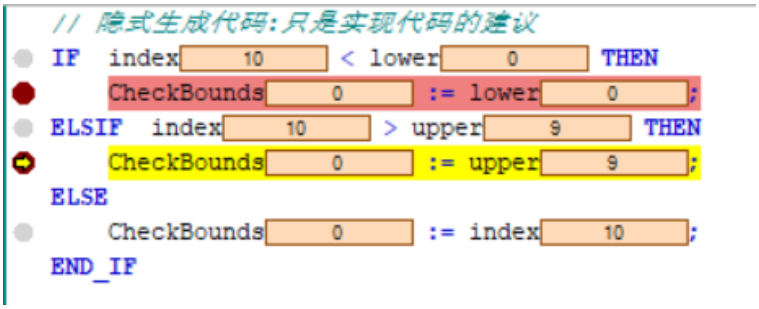


图 14.19 数组越界，程序跳至断点处

5）使用快捷键 F10 直到定位到程序中异常的位置，如图 14.20 所示；

6）定位到问题程序段后，用户需要根据实际情况修改程序，避免异常出现。

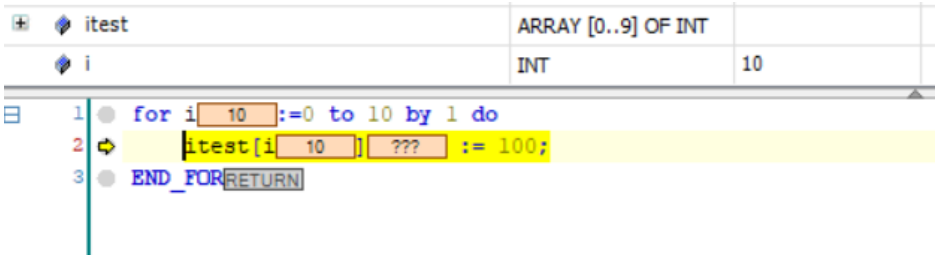


图 14.20 定位至程序异常处

3. 隐含检查的 POU — 除法检测

该方法适用于排查除数为 0 的情况具体操作步骤如下：

1）右键点击“Application”，选择“添加对象”→“用于隐式检查的 POU”；

2）在弹出界面勾选“除法检测”，点击“打开”，成功添加 4 个函数，如图 14.21 和图 14.22 所示；

3）登录 PLC，停止程序运行，在新增函数里的代码“CheckDivDInt:=1;”、“CheckDivLInt:=1;”、“CheckDivLReal:=1;”、“CheckDivReal:=1;”位置增加断点并激活（快捷键 F9），如图 14.23 所示。

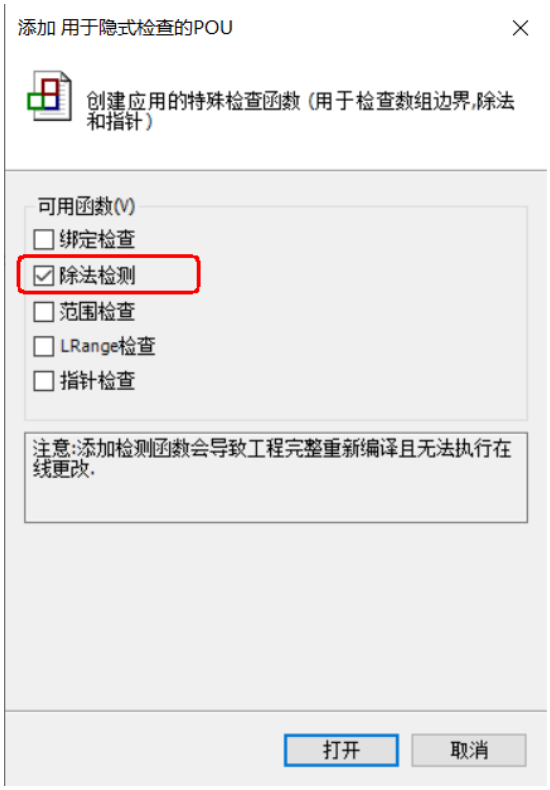


图 14.21　勾选“除法检测”

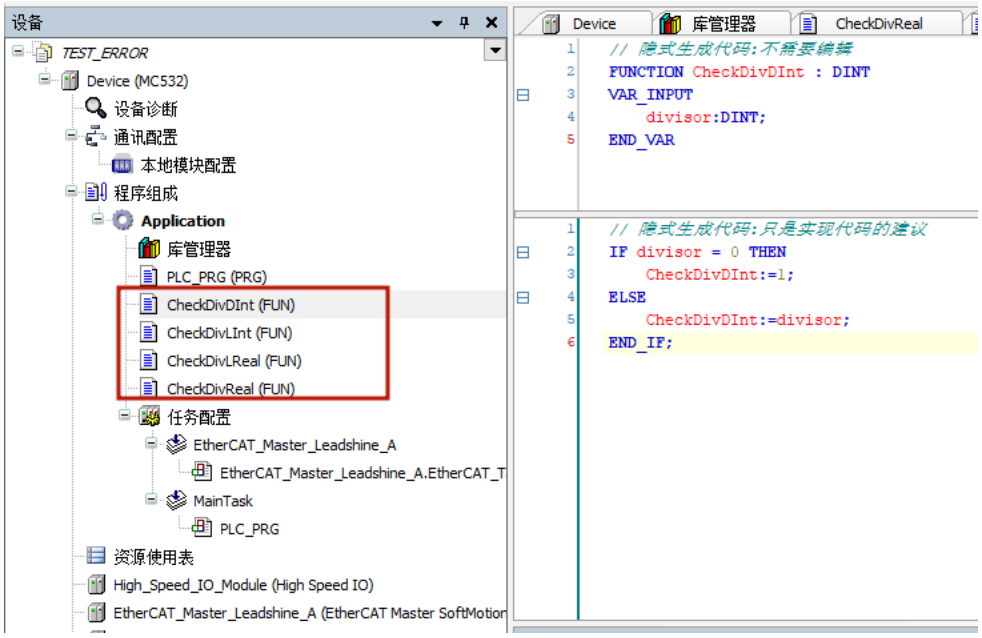


图 14.22　添加的 4 个“除法检测”函数

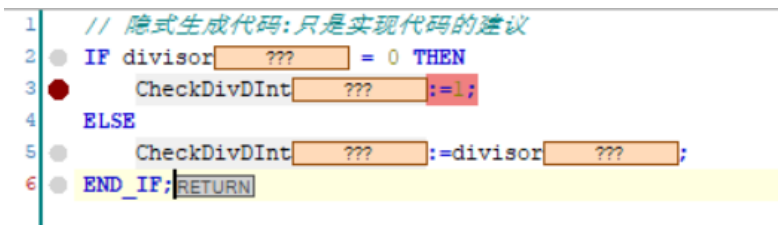


图 14.23　在程序中添加断点并激活

4）运行程序，直到检测到除数为 0 的情况，进入断点，如图 14.24 所示；
5）使用快捷键 F10 直到定位到程序中异常的位置，如图 14.25 所示；
6）定位到问题程序段后，用户需要根据实际情况修改程序，避免异常出现。

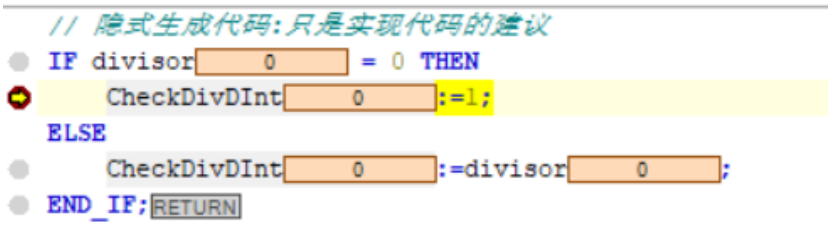


图 14.24　检测到除数为 0 的情况，进入断点

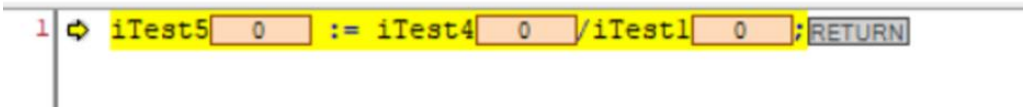


图 14.25　定位至除数为 0 处

14.3. 常见故障码

错误码 SMC_ERROR（Enumeration）

这个数据结构包括了 SoftMotion 功能块可能返回的错误编号，可以通过错误编号了解发生了什么错误，方便及时处理。调用 SMC_ErrorString 指令可以根据错误号生成错误字符串输出量。其格式如下：

SMC_ErrorString(ErrorID:= ,Language:= ,SMC_ErrorString=>)

其中：

ErrorID：错误码，数据类型为 SMC_Error
Language：所需语言，数据类型为 SMC_LANGUAGE_TYPE
SMC_ErrorString：错误描述，数据类型为 STRING(100)

表 14.1 是错误码及其描述。

表 14.1　错误码及其描述

错误编号	模块	枚举量	描述
0	所有	SMC_NO_ERROR	无错误
1	DriveInterface	SMC_DI_GENERAL_COMMUNICATION_ERROR	通信错误。现场总线从站不再处于运行状态。
2	DriveInterface	SMC_DI_AXIS_ERROR	轴错误
3	DriveInterface	SMC_DI_FIELDBUS_LOST_SYNCHRONITY	总线周期同步失败
10	DriveInterface	SMC_DI_SWLIMITS_EXCEEDED	超出了软件限位
11	DriveInterface	SMC_DI_HWLIMITS_EXCEEDED	超出了硬件限位
12	DriveInterface	SMC_DI_LINEAR_AXIS_OUTOFRANGE	超过轴的位置限制

13	DriveInterface	SMC_DI_HALS_OR_QUICKSTOP_NOT_SUPPORTED	不支持暂停和快速停止
14	DriveInterface	SMC_DI_VOLTAGE_DISABLED	轴未使能
15	DriveInterface	SMC_DI_IRREGULAR_ACTPOSITION	从驱动获取的实际位置为非法值，请检查通信。
16	DriveInterface	SMC_DI_POSITIONLAGERROR	跟随误差超出限制错误。设置位置和实际位置之间的误差超过了给定的限制。
17	DriveInterface	SMC_DI_HOMING_ERROR	回零过程错误
18	DriveInterface	SMC_DI_LICENSING_ERROR	软件授权失败
20	all motion generating modules	SMC_REGULATOR_OR_START_NOT_SET	控制器不能执行或者正在刹车
21	All motion generating modules	SMC_WRONG_CONTROLLER_MODE	轴处于错误的控制模式。
25		SMC_INVALID_ACTION_FOR_LOGICAL	无效的逻辑动作
30	DriveInterface	SMC_FB_WASNT_CALLED_DURING_MOTION	在运动没有结束前，运动模块没有被调用。
31	All modules	SMC_AXIS_IS_NO_AXIS_REF	所给的 AXIS_REF 变量不是 AXIS_REF 类型的。
32	All motion generating modules	SMC_AXIS_REF_CHANGED_DURING_OPERATION	在运动过程中，输入的轴 AXIS_REF 变量被更换。
33	DriveInterface	SMC_FB_ACTIVE_AXIS_DIABLED	在运动过程中，轴失效（MC_Power.bRegulatorOn）
34	All motion generating modules	SMC_AXIS_NOT_READY_FOR_MOTION	轴没有处于准备好的状态，不能执行运动
35	All motion generating function blocks	SMC_AXIS_ERROR_DURING_MOTION	轴在运动过程中报错
40	VirtualDrive	SMC_VD_MAX_VELOCITY_EXCEEDED	超过最大速度（fMaxVelocity）
41	VirtualDrive	SMC_VD_MAX_ACCELERATION_EXCEEDED	超过最大加速度（fMaxAcceleration）
42	VirtualDrive	SMC_VD_MAX_DECELERATION_EXCEEDED	超过最大减速度（fMaxDeceleration）
50	SMC_Homing	SMC_3SH_INVALID_VELACC_VALUES	无效的速度或加速度值
51	SMC_Homing	SMC_3SH_MODE_NEEDS_HWLIMIT	模式要求（可能基于安全原因）关闭限位开关
60		SMC_FRC_NO_FREE_HANDLE	无空闲句柄用于打开文件
70	SMC_SetControllerMode	SMC_SCM_NOT_SUPPORTED	模式不支持
71	SMC_SetControllerMode	SMC_SCM_AXIS_IN_WRONG_STATE	在当前模式下，不能改变控制器模式。
72	SMC_SetControllerMode	SMC_SCM_INTERRUPTED	设置控制器模式被 MC_Stop 指令或者 errorstop 打断
75	SMC_SetTorque	SMC_ST_WRONG_CONTROLLER_MODE	轴不在正确的模式下
90	SMC_ChangeGearingRatio	SMC_CGR_ZERO_VALUES	无效值
91	SMC_ChangeGearingRatio	SMC_CGR_DRIVE_POWERED	电子齿轮参数在轴受控时不可被修改

92	SMC_ChangeGearingRatio	SMC_CGR_INVALID_POSPERIOD	旋转轴无效的模值（<=0）
93	SMC_ChangeGearingRatio	SMC_CGR_POSPERIOD_NOT_INTEGRAL	参数值非整数
110	MC_Reset	SMC_P_FTASKCYCLE_EMPTY	设置的轴运动循环周期错误（fTaskCycle = 0）
120	MC_Reset	SMC_R_NO_ERROR_TO_RESET	轴无错误
121	MC_Reset	SMC_R_DRIVE_DOESNT_ANSWER	轴不执行“错误复位”
122	MC_Reset	SMC_R_ERROR_NOT_RESETTABLE	错误不能被复位
123	MC_Reset	SMC_R_DRIVE_DOESNT_ANSWER_IN_TIME	轴的通讯状态不响应
130	MC_ReadParameter, MC_ReadBoolParameter	SMC_RP_PARAM_UNKNOWN	参数个数不明确
131	MC_ReadParameter, MC_ReadBoolParameter	SMC_RP_REQUESTING_ERROR	向驱动设备传递时出错;可通过见功能块 ReadDriveParameter 获取错误号。
132	MC_ReadParameter, MC_ReadBoolParameter	SMC_RP_DRIVE_PARAMETER_NOT_MAPPED	驱动参数没有映射
133	MC_ReadParameter, MC_ReadBoolParameter	SMC_RP_PARAM_CONVERSION_ERROR	参数值转换失败
140	MC_WriteParameter, MC_WriteBoolParameter	SMC_WP_PARAM_INVALID	不可识别的参数编号,或写错误
141	MC_WriteParameter, MC_WriteBoolParameter	SMC_WP_SENDING_ERROR	可通过功能块 WriteDriveParameter 获取错误号。
142	MC_WriteParameter, MC_WriteBoolParameter	SMC_WP_DRIVE_PARAMETER_NOT_MAPPED	驱动参数没有映射
143	MC_WriteParameter, MC_WriteBoolParameter	SMC_WP_PARAM_CONVERSION_ERROR	参数值转换失败
170	MC_Home	SMC_H_AXIS_WASNT_STANDSTILL	轴没有处于 standstill 状态
171	MC_Home	SMC_H_AXIS_DIDNT_START_HOMING	启动回零运动时出错。
172	MC_Home	SMC_H_AXIS_DIDNT_ANSWER	通讯错误。
173	MC_Home	SMC_H_ERROR_WHEN_STOPPING	回零运动错误（没有设置减速）
174	MC_Home	SMC_H_AXIS_IN_ERRORSTOP	回零运动无法执行(轴处于错误停止状态)
180	MC_Stop	SMC_MS_UNKNOWN_STOPPING_ERROR	停止时出现未知错误
181	MC_Stop	SMC_MS_INVALID_ACCDEC_VALUES	无效速度或加速度值
182	MC_Stop	SMC_MS_DIRECTION_NOT_APPLICABLE	方向信号不能使用 shortest
183	MC_Stop	SMC_MS_AXIS_IN_ERRORSTOP	停止无法执行(轴处于错误停止状态)
184	MC_Stop	SMC_BLOCKING_MC_STOP_WASNT_CALLED	MC_Stop 功能块未调用

185		SMC_MS_AXIS_ALREADY_STOPPING	正在执行停止运动,功能块无法中断
200		SMC_UNKNOWN_TASK_INTERVAL	不确定的总线任务时间
201	MC_MoveAbsolute	SMC_MA_INVALID_VELACC_VALUES	速度或加速度值无效
202	MC_MoveAbsolute	SMC_MA_INVALID_DIRECTION	方向错误
226	MC_MoveRelative	SMC_MR_INVALID_VELACC_VALUES	速度或加速度值无效
227	MC_MoveRelative	SMC_MR_INVALID_DIRECTION	方向错误
251	MC_MoveAdditive	SMC_MAD_INVALID_VELACC_VALUES	速度或加速度值无效
252	MC_MoveAdditive	SMC_MAD_INVALID_DIRECTION	方向错误
276	MC_MoveSuperImposed	SMC_MSI_INVALID_VELACC_VALUES	速度或加速度值无效
277	MC_MoveSuperImposed	SMC_MSI_INVALID_DIRECTION	方向错误
301	MC_MoveVelocity	SMC_MV_INVALID_ACCDEC_VALUES	速度或加速度值无效
302	MC_MoveVelocity	SMC_MV_DIRECTION_NOT_APPLICABLE	方向错误，不能使用 shortest/fastest
325	MC_PositionProfile	SMC_PP_ARRAYSIZE	数组大小错误
326	MC_PositionProfile	SMC_PP_STEPOMS	每段间隔时间= t#0s
350	MC_VelocityProfile	SMC_VP_ARRAYSIZE	数组大小错误
351	MC_VelocityProfile	SMC_VP_STEPOMS	每段间隔时间= t#0s
375	MC_AccelerationProfile	SMC_AP_ARRAYSIZE	数组大小错误
376	MC_AccelerationProfile	SMC_AP_STEPOMS	每段间隔时间= t#0s
400	MC_TouchProbe	SMC_TP_TRIGGEROCCUPIED	触发已经激活
401	MC_TouchProbe	SMC_TP_COULDNT_SET_WINDOW	DriveInterface 不支持窗口函数
402	MC_TouchProbe	SMC_TP_COMM_ERROR	通讯错误
410	MC_AbortTrigger	SMC_AT_TRIGGERNOTOCCUPIED	触发已重新分配
426	SMC_MoveContinuousRelative	SMC_MCR_INVALID_VELACC_VALUES	速度或加速度值无效
427	SMC_MoveContinuousRelative	SMC_MCR_INVALID_DIRECTION	无效的方向
451	SMC_MoveContinuousAbsolute	SMC_MCA_INVALID_VELACC_VALUES	速度或加速度值无效
452	SMC_MoveContinuousAbsolute	SMC_MCA_INVALID_DIRECTION	无效的方向
453	SMC_MoveContinuousAbsolute	SMC_MCA_DIRECTION_NOT_APPLICABLE	Direction= fastest 不能使用
475	SMC_ChangeDynamicLimits	SMC_SDL_INVALID_AXIS_STATE	轴状态不正确，应该为 standshill 或 power_off 才能调用该模块
476	SMC_ChangeDynamicLimits	SMC_SDL_INVALID_VELACC_VALUES	无效的速度，加减速度，加加速度
600	SMC_CamRegister	SMC_CR_NO_TAPPETS_IN_CAM	CAM 不包含任何 tappets
601	SMC_CamRegister	SMC_CR_TOO_MANY_TAPPETS	Tappet 组 ID 超过 MAX_NUM_TAPPETS
602	SMC_CamRegister	SMC_CR_MORE_THAN_32_ACCESSES	在一个 CAM_REF 上有超过 32 个

			访问连接
625	MC_CamIN	SMC_CI_NO_CAM_SELECTED	没有 CAM 被选择
626	MC_CamIN	SMC_CI_MASTER_OUT_OF_SCALE	主轴在超出有效值范围
627	MC_CamIN	SMC_CI_RAMPIN_NEEDS_VELACC_VALUES	对 ramp_in 函数,速度和加速度值必须被指定。
628	MC_CamIN	SMC_CI_SCALING_INCORRECT	缩放变量 fEditor / TableMasterMin / Max 不正确
629	MC_CamIn	SMC_CI_TOO_MANY_TAPPETS_PER_CYCLE	在同一周期内有太多的 Tappet 被激活使用
640	SMC_CAMBounds, SMC_CamBounds_Pos	SMC_CB_NOT_IMPLEMENTED	调用 CAM 数据表的功能块没有执行
675	MC_GearIn	SMC_GI_RATIO_DENOM	分母为 0
676	MC_GearIn	SMC_GI_INVALID_ACC	加速度无效
677	MC_GearIn	SMC_GI_INVALID_DEC	减加速度无效
678	MC_GearIn, MC_CamIn	SMC_GI_MASTER_REGULATOR_CHANGED	在不允许的情形下，主轴的 Enable/Disable 状态切换
679	MC_GearIn	SMC_GI_INVALID_JERK	无效的加加速度
725	MC_Phase	SMC_PH_INVALID_VELACCDEC	速度、减加速度或加速度无效
726	MC_Phase	SMC_PH_ROTARYAXIS_PERIOD0	旋转轴 fPositionPeriod = 0
750	All modules using MC_CAM_REF as input	SMC_NO_CAM_REF_TYPE	所给的 CAM 不是 MC_CAM_REF 类型的
751	MC_CamTableSelect	SMC_CAM_TABLE_DOES_NOT_COVER_MASTER_SCALE	CamTable 的数据没有覆盖 Master area 从 xStart 到 xEnd 的范围
752	MC_CamTableSelect	SMC_CAM_TABLE_EMPTY_MASTER_RANGE	凸轮表的主轴范围为空
753	MC_CamTableSelect	SMC_CAM_TABLE_INVALID_MASTER_MINMAX	凸轮表的主轴范围最大最小值无效
754	MC_CamTableSelect	SMC_CAM_TABLE_INVALID_SLAVE_MINMAX	凸轮表的从轴范围最大最小值无效
775	MC_GearInPos	SMC_GIP_MASTER_DIRECTION_CHANGE	在与从轴连接期间,主轴改变了旋转方向
776	MC_GearInPos	SMC_GIP_SLAVE_REVERSAL_CANNOT_BE_AVOIDED	VAR_INPUT“AvoidReversal”被设置，但是从轴反转无法避免
777	MC_GearInPos	SMC_GIP_AVOID_REVERSAL_FOR_FINITE_AXIS	如果是直线轴，VAR_INPUT “AvoidReversal”不需要设置
800	SMC_BacklashCompensation	SMC_BC_BL_TOO_BIG	齿轮间隙（fBacklash）太大（>位置周期/2）
825	All motion generating function blocks	SMC_QPROF_DIVERGES	内部计算错误
826	All motion generating function blocks	SMC_QPROF_DISTANCE_TOO_SHORT	内部计算错误
827	All motion generating function blocks	SMC_QPROF_NO_RESULT	内部计算错误
728	All motion generating function blocks	SMC_QPROF_INVALID_NEW_LBD	内部轨迹二次方计算错误

729	All motion generating function blocks	SMC_QPROF_BAD_NEGOTIATION	内部轨迹二次方计算错误
730	All motion generating function blocks	SMC_QPROF_INVALID_INTERVAL	内部轨迹二次方计算错误
731	All motion generating function blocks	SMC_QPROF_NOT_ENOUGH_PHASES	内部轨迹二次方计算错误
850	SMC_SetRampType	SMC_SRT_NOT_STANDSTILL_OR_POWEROFF	只允许轴在 STANDSTILL 和 POWER_OFF 状态下设置
851	SMC_SetRampType	SMC_SRT_INVALID_RAMPTYPE	无效的类型
852	SMC_SetRampType	SMC_SMT_NOT_STANDSTILL_OR_POWEROFF	只允许轴在 STANDSTILL 和 POWER_OFF 状态下设置
853	SMC_SetRampType	SMC_SMT_INVALID_MOVEMENTTYPE_OR_POSITIONPERIOD	无效的运动类型或位置周期
854	SMC_SetRampType	SMC_SMT_AXIS_NOT_VIRTUAL	只允许在虚轴上使用的功能块
1000	CNC function blocks which are supervising the licensing	SMC_NO_LICENSE	无 CNC 许可
1001	SMC_Interpolator	SMC_INT_VEL_ZERO	轨迹未被处理,因为设置速度为 0
1002	SMC_Interpolator	SMC_INT_NO_STOP_AT_END	上一个轨迹对象最终的速度 Vel_End > 0。
1003	SMC_Interpolator	SMC_INT_DATA_UNDERRUN	警告：在 DataIn 中处理了 GEOINFO 表，但是列表最后速度未到达。 原因：忘记设置在 DataIn 中的队列的 EndOfList，或者 SMC_Interpolator 速度比轨迹预处理模块要快。
1004	SMC_Interpolator	SMC_INT_VEL_NONZERO_AT_STOP	停止时速度>0。
1005	SMC_Interpolator	SMC_INT_TOO_MANY_RECURSIONS	多个 SMC_Interpolator 使用同一个轴
1006	SMC_Interpolator	SMC_INT_NO_CHECKVELOCITIES	在数据预处理阶段，SMC_CHeckVelocities 不是最后一个模块。
1007	SMC_Interpolator	SMC_INT_PATH_EXCEEDED	内部/数字错误。
1008	SMC_Interpolator	SMC_INT_VEL_ACC_DEC_ZERO	速度、加速度或减加速度值为空或太小。
1009	SMC_Interpolator	SMC_INT_DWIPOTIME_ZERO	dwIpoTime 为 0
1010	SMC_Interpolator	SMC_INT_JERK_NONPOSITIVE	加加速度必须是正值
1011	SMC_Interpolator	SMC_INT_QPROF_DIVERGES	内部计算错误
1012	SMC_Interpolator	SMC_INT_INVLALID_VELOCITY_MODE	无效的速度模式
1013	SMC_Interpolator	SMC_INT_TOO_MANY_AXES_INTERPOLATED	太多轴插补
1050	SMC_Interpolator2Dir	SMC_INT2DIR_BUFFER_TOO_SMALL	数据缓冲区太小
1051	SMC_Interpolator2Dir	SMC_INT2DIR_PATH_FITS_NOT_IN_QUEUE	轨迹在队列中没有运行完。
1070		SMC_XINT_INVALID_DIRECTION	无效的方向输入值
1071		SMC_XINT_NOINTERSECTION	根据给定的 CNC 路径中的 X 轴位置，无法确定终止位置
1080	SMC_Interpolator	SMC_WAR_INT_OUTQUEUE_TOO_SMALL	OutQueueDataIn 太小
1081	SMC_Interpolator	SMC_WAR_END_VELOCITIES_INCORRECT	终止速度不一致
1100	SMC_CheckVelocities	SMC_CV_ACC_DEC_VEL_NONPOSITIVE	速度、加速度或减速度值不为正值

1120	SMC_Controlaxisbypos	SMC_CA_INVALID_ACCDEC_VALUES	fGapVelocity / fGapAcceleration / fGapDeceleration 不为正值
1200	SMC_NCDecoder	SMC_DEC_ACC_TOO_LITTLE	不允许的加速度值
1201	SMC_NCDecoder	SMC_DEC_RET_TOO_LITTLE	不允许的减速度值
1202	SMC_NCDecoder	SMC_DEC_OUTQUEUE_RAN_EMPTY	数据队列为空
1203	SMC_NCDecoder	SMC_DEC_JUMP_TO_UNKNOWN_LINE	由于行号未知,跳转指令不能执行
1204	SMC_NCDecoder	SMC_DEC_INVALID_SYNTAX	语法无效
1205	SMC_NCDecoder	SMC_DEC_3DMODE_OBJECT_NOT_SUPPORTED	对象不支持 3D 模式。
1206	SMC_NCDecoder	SMC_DEC_NEGATIVE_PERIOD	辅助轴周期为负值不正确
1207	SMC_NCDecoder	SMC_DEC_DIMENSIONS_EXCLUSIVE_AU	插补中,轴 A 与轴 U 不能同时执行
1208	SMC_NCDecoder	SMC_DEC_DIMENSIONS_EXCLUSIVE_BV	插补中,轴 B 与轴 V 不能同时执行
1209	SMC_NCDecoder	SMC_DEC_DIMENSIONS_EXCLUSIVE_CW	插补中,轴 C 与轴 W 不能同时执行
1300	SMC_GCodeViewer	SMC_GCV_BUFFER_TOO_SMALL	缓冲区过小
1301	SMC_GCodeViewer	SMC_GCV_BUFFER_WRONG_TYPE	缓冲区元素类型错误
1302	SMC_GCodeViewer	SMC_GCV_UNKNOWN_IPO_LINE	当前行不能从插补器中找到
1500	All function blocks using SMC_CNC_REF	SMC_NO_CNC_REF_TYPE	给出的 CNC 程序不是 SMC_CNC_REF 类型的
1501	All function blocks using SMC_OUTQUEUE	SMC_NO_OUTQUEUE_TYPE	给出的 OutQueue 不是 SMC_OUTQUEUE 类型的
1502	All function blocks using pbyBuffer	SMC_GEOINFO_BUFFER_MISALIGNED	在 pbyBuffer 中，没有使用 4 字节对齐
1600	CNC function blocks	SMC_3D_MODE_NOT_SUPPORTED	功能块只支持 2D 模式
1700	SMC_SmoothAddAxes	SMC_SAA_SMOOTHAREA_TOO_LARGE	平滑范围过大
1701	SMC_SmoothAddAxes	SMC_SAA_SP_INVALID_INPUT	dSmoothingPart 输入参数无效，有效范围[0..1]
1800	SMC_SegmentAnalyzer	SMC_SA_QUEUE_NOT_IN_BUFFER	功能块检测到 OutQueue 缓存已经满，但是数据没有结束
1801	SMC_SegmentAnalyzer	SMC_SA_QUEUE_CHANGED_DURING_OP	当功能块操作 OutQueue 时，OutQueue 缓存发生改变
1820		SMC_OS_INVALID_PARAMETER	无效参数
1830	SMC_BlockSearchSavePos	SMC_BSSP_IPO_NOT_ACTIVE	位置不能保存,插补器是无效的
1831	SMC_BlockSearch	SMC_BS_SAVEDPOS_NOT_REACHED	保存的位置没有找到,可能是一个不同的路径
1832	SMC_BlockSearch	SMC_BS_NO_POS_STORED	SMC_BlockSearchSavePos 不能执行或者为一个错误的状态
1900		SMC_INVALID_FEATURE_FLAG	特征标记有效值范围为[1..31]
1901		SMC_SMB_HFUN_NOT_SUPPORTED	功能块不支持 H 代码
1902		SMC_SMB_ONLY_3DMODE	功能块只能工作在 3D 模式
1903		SMC_SMB_ERROR_COMPUTING_SPLINE	计算样条错误
1910		SMC_SMM_INVALID_PARAM_NUMBER	辅助参数值太大
2000	SMC_ReadNCFile	SMC_RNCF_FILE_DOESNT_EXIST	文件不存在
2001	SMC_ReadNCFile	SMC_RNCF_NO_BUFFER	无缓冲区被分配

2002	SMC_ReadNCFile	SMC_RNCF_BUFFER_TOO_SMALL	缓冲区太小
2003	SMC_ReadNCFile	SMC_RNCF_DATA_UNDERRUN	缓冲区为空。
2004	SMC_ReadNCFile	SMC_RNCF_VAR_COULDNT_BE_REPLACED	占位符变量不能被替换。
2005	SMC_ReadNCFile	SMC_RNCF_NOT_VARLIST	输入 pvl 不指向 SMC_VARLIST 对象。
2006	SMC_ReadNCFile	SMC_RNCF_NO_STRINGBUFFER	读 NC 文件没有定义缓存
2007	SMC_ReadNCFile	SMC_RNCF_STRINGBUFFER_OVERRUN	读 NC 文件的缓存溢出
2050	SMC_ReadNCQueue	SMC_RNCQ_FILE_DOESNT_EXIST	文件不存在。
2051	SMC_ReadNCQueue	SMC_RNCQ_NO_BUFFER	无缓冲区被定义
2052	SMC_ReadNCQueue	SMC_RNCQ_BUFFER_TOO_SMALL	缓冲区太小
2053	SMC_ReadNCQueue	SMC_RNCQ_UNEXPECTED_EOF	文件异常结束
2100	SMC_AxisDiagnosticLog	SMC_ADL_FILE_CANNOT_BE_OPENED	文件不能打开
2101	SMC_AxisDiagnosticLog	SMC_ADL_BUFFER_OVERRUN	缓冲区过载；WriteToFile 一定是被过于频繁地调用
2200	SMC_ReadCAM	SMC_RCAM_FILE_DOESNT_EXIST	文件不能打开
2201	SMC_ReadCAM	SMC_RCAM_TOO_MUCH_DATA	要存储地 CAM 过大
2202	SMC_ReadCAM	SMC_RCAM_WRONG_COMPILE_TYPE	错误地编译模式
2203	SMC_ReadCAM	SMC_RCAM_WRONG_VERSION	文件版本错误
2204	SMC_ReadCAM	SMC_RCAM_UNEXPECTED_EOF	文件异常结束
3001	SMC_WriteDriveParamsToFile	SMC_WDPF_CHANNEL_OCCUPIED	此错误已过时，仅出于兼容性而保留。SMC_WDPF_TIMEOUT_PREPARING_LIST
3002	SMC_WriteDriveParamsToFile	SMC_WDPF_CANNOT_CREATE_FILE	文件不能被创建
3003	SMC_WriteDriveParamsToFile	SMC_WDPF_ERROR_WHEN_READING_PARAMS	读取参数出错
3004	SMC_WriteDriveParamsToFile	SMC_WDPF_TIMEOUT_PREPARING_LIST	准备参数表时超时
5000	SMC_Encoder	SMC_ENC_DENOM_ZERO	编码参考的转变因数（dwRatioTechUnitsDenom）分母为 0。
5001	SMC_Encoder	SMC_ENC_AXISUSEDBYOTHERFB	其他模块尝试在编码器轴上处理运动。
5002	DriveInterface	SMC_ENC_FILTER_DEPTH_INVALID	无效的过滤器深度。

第15章 LeadSys Studio 使用技巧

15.1. 在线帮助与联网帮助配置

LeadSys Studio 软件自带的帮助信息很丰富，默认用软件自带的在线帮助，打开 LeadSys Studio 软件，在菜单栏中找到工具→选项，去掉下图的勾选。如图 15.1 所示。

如果需要联网帮助，再勾选下面的选项。

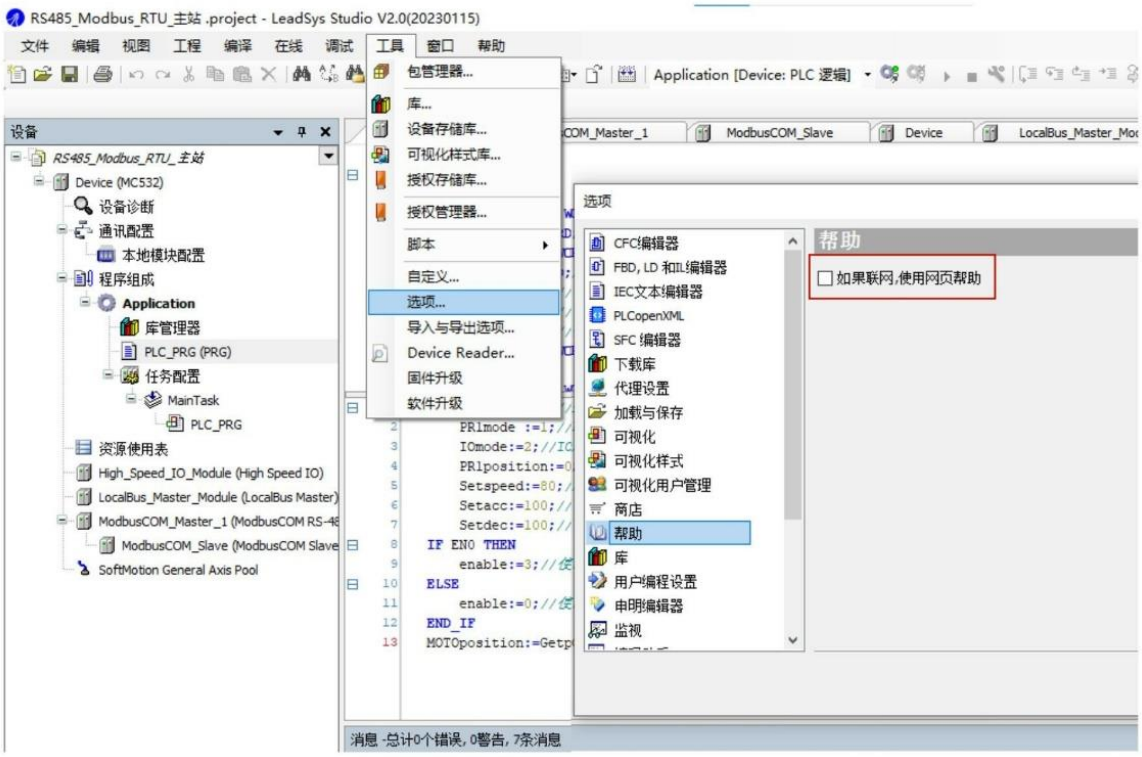


图 15.1　联网帮助选项

在菜单栏的“帮助”可打开在线帮助，如图 15.2 所示。

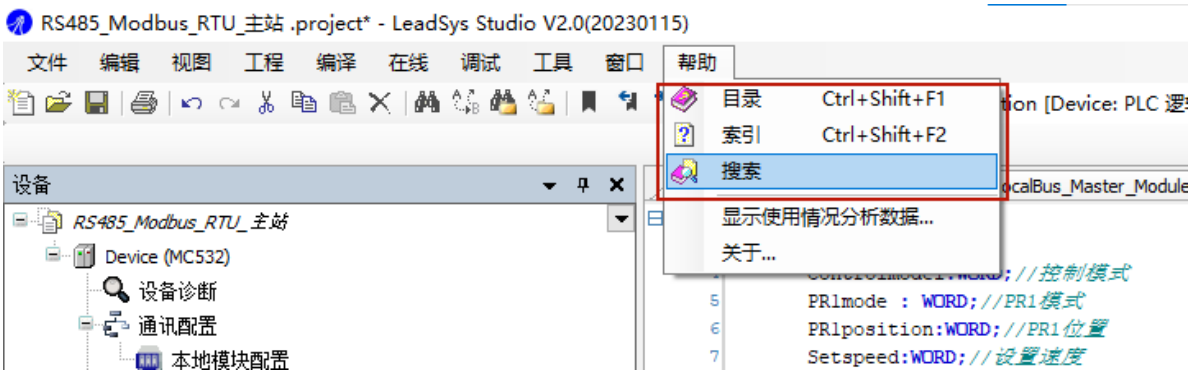


图 15.2　“帮助”菜单

示例：搜索 CheckLRangeSigned 的帮助信息。如图 15.3 所示。

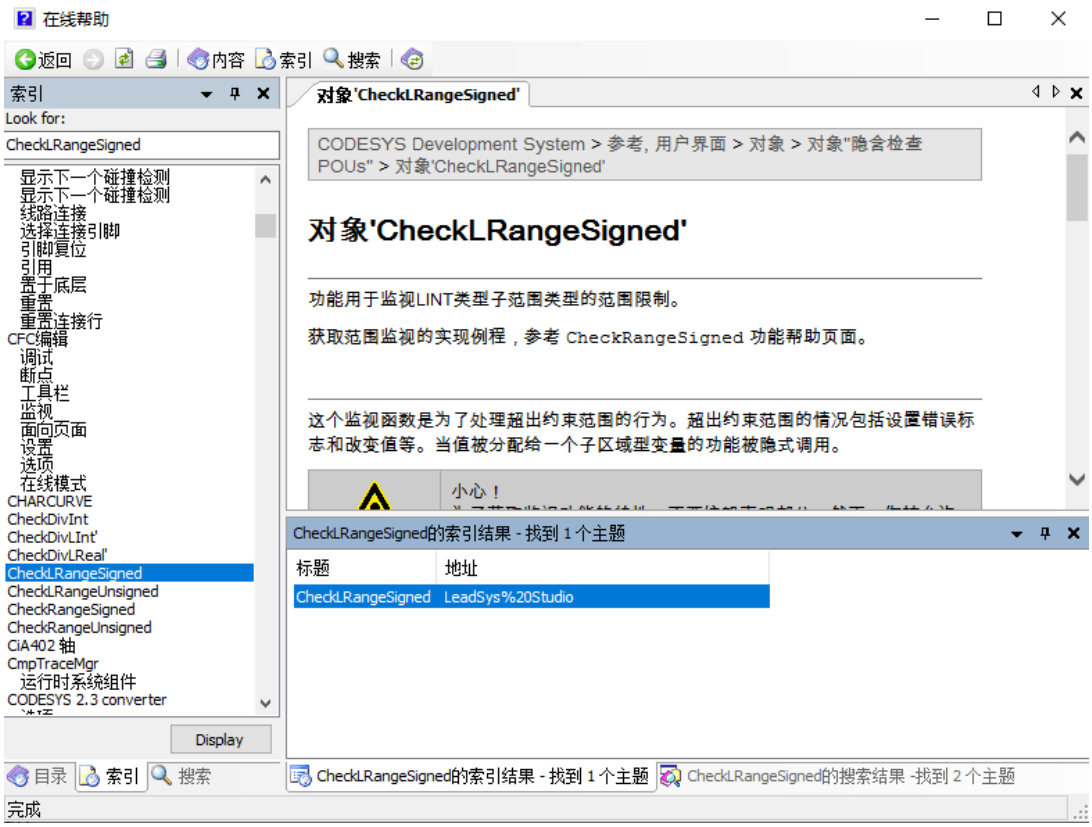


图 15.3　CheckLRangeSigned 的帮助信息

在编程过程中，当对某指令不清楚时，可将光标移至该指令上，再按 F1 键，即可弹出相关帮助信息。如图 15.4 所示，查询 CASE 指令的帮助信息。

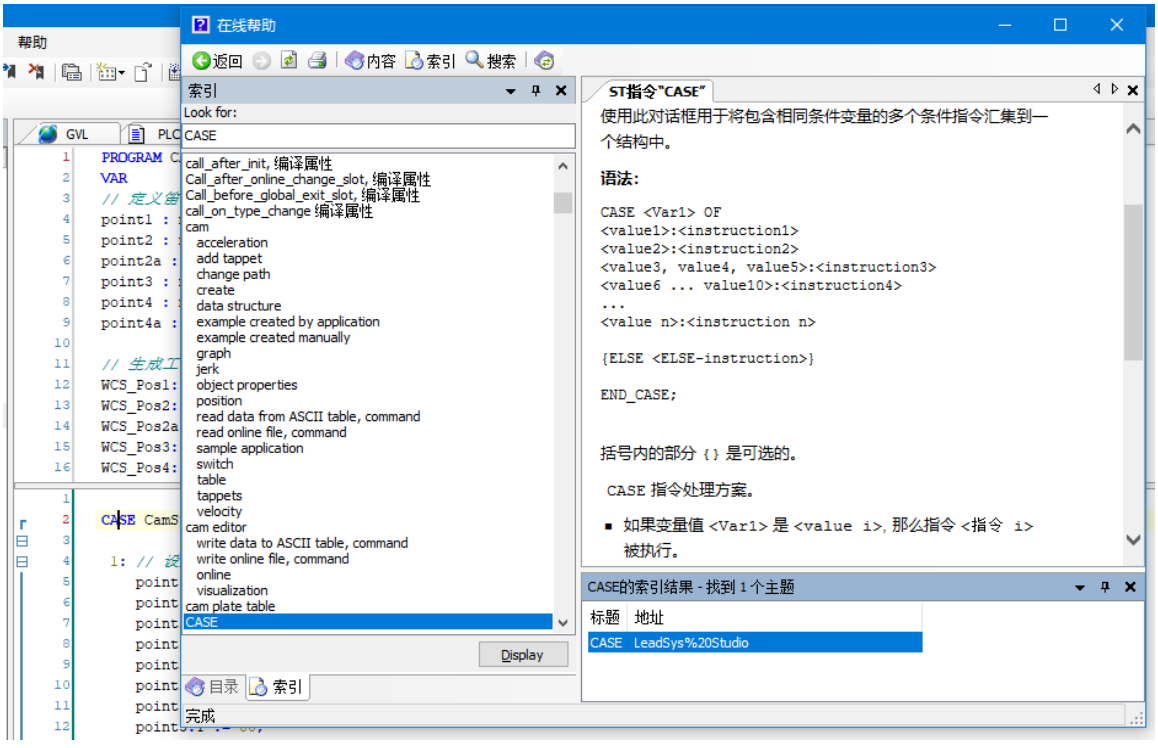


图 15.4　按 F1 键获取帮助信息

15.2. 编码助手的设置

LeadSys Studio 软件默认已经使能了编程助手，当变量未定义时，需要提示自动定义变量。编程助手的设置方法为：从菜单→工具→选项中选“编码助手”，可根据自己的喜好选择项目，如图 15.5 所示。

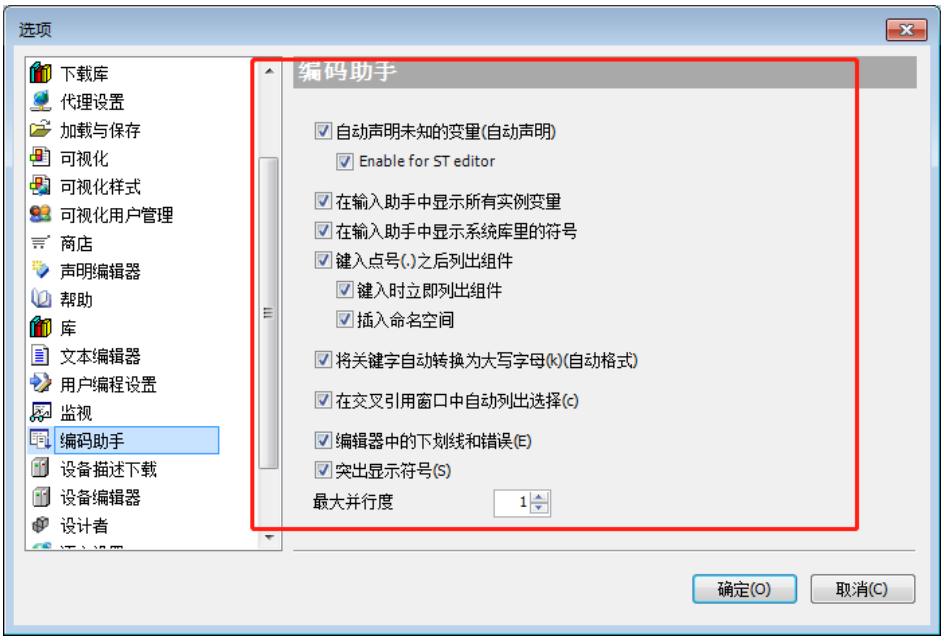


图 15.5　编码助手的配置

15.3. 电机功能码

在 LeadSys Studio 软件可以直接读写总线型伺服电机驱动器和步进电机驱动器的参数，如图 15.6 所示。

但目前只能支持雷赛公司的电机，型号为：L7EC 系列、L7EC-S 系列、CL3C 系列、DM3C 和 DM3E 系列。

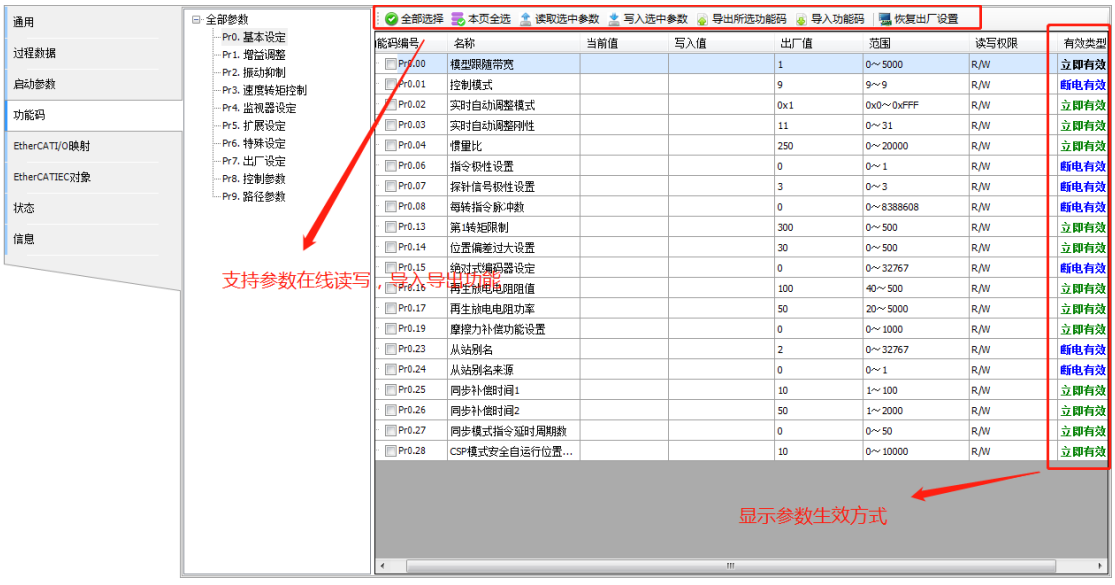


图 15.6　读、写电机驱动器参数

15.4. 添加库文件

以添加自由通讯协议库文件为例，鼠标双击左侧列表中的“库管理器”，打开后点击“添加库”，选择“高级”，搜索“CAA Types Extern”，选中确定后添加该库。如图 15.7、图 15.8 所示。

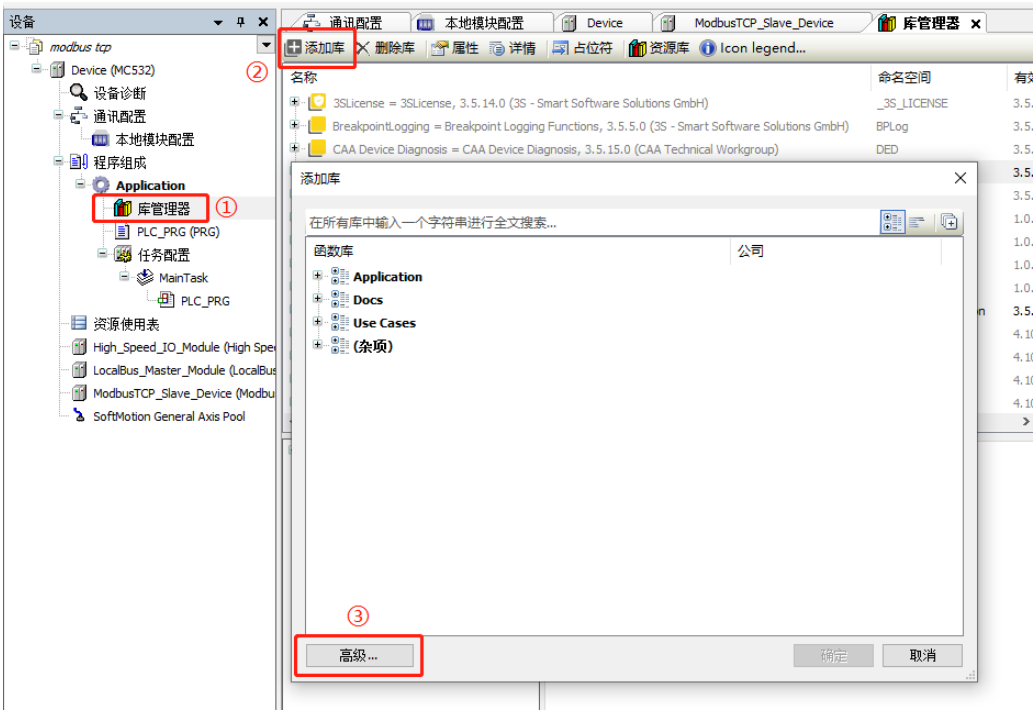


图 15.7　添加 CAA 与 TCP 库

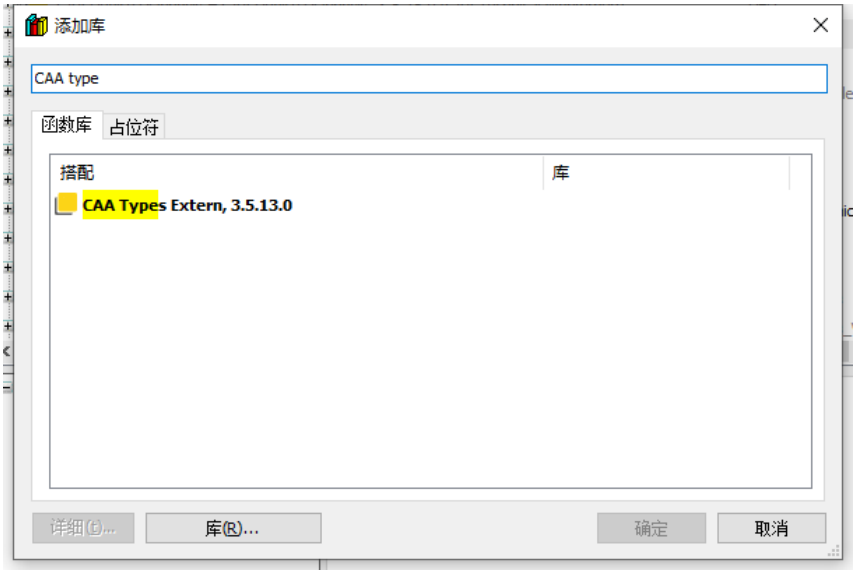


图 15.8　添加 CAA Types Extern 库

重复以上步骤，添加“CAA Net Base Services”库，安装库后，在库管理器中可看到已添加的库文件，如图 15.9 所示。

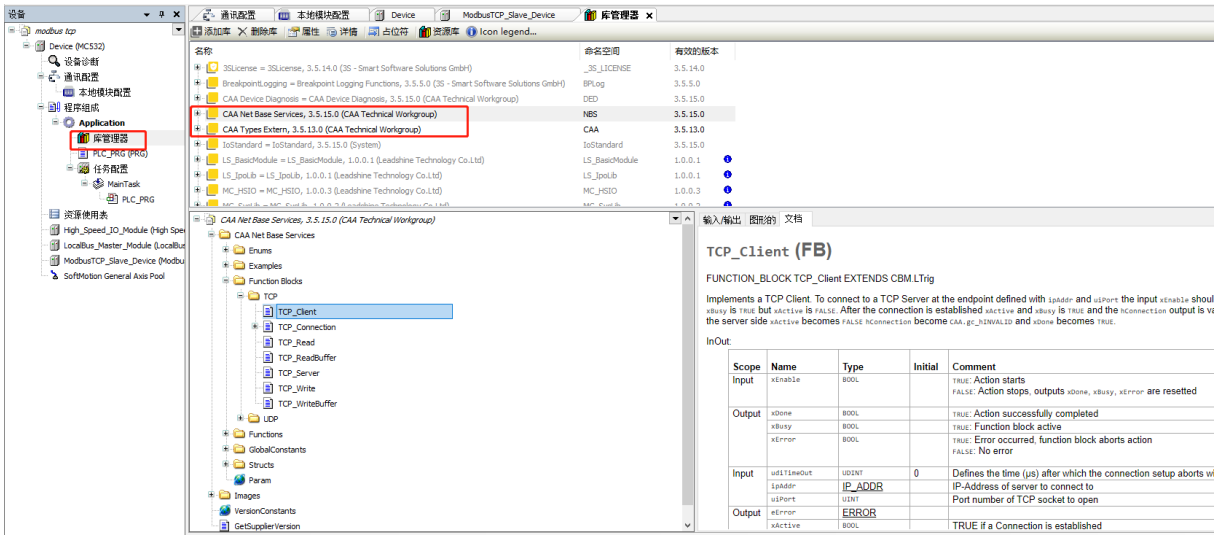


图 15.9　添加 CAA Types Extern 库与 CAA Net Base Services 库

15.5. 描述文件添加

LeadSys Studio 中已经预先添加了雷赛公司的 EtherCAT 从站设备，用户使用其他公司的设备时，需要添加设备描述文件。

以添加松下伺服驱动器描述文件为例，具体的操作步骤如下：

1、打开 LeadSys 软件，选择菜单栏中的“工具”选项卡，在二级菜单中选择“设备储存库”，如图 15.10 所示。

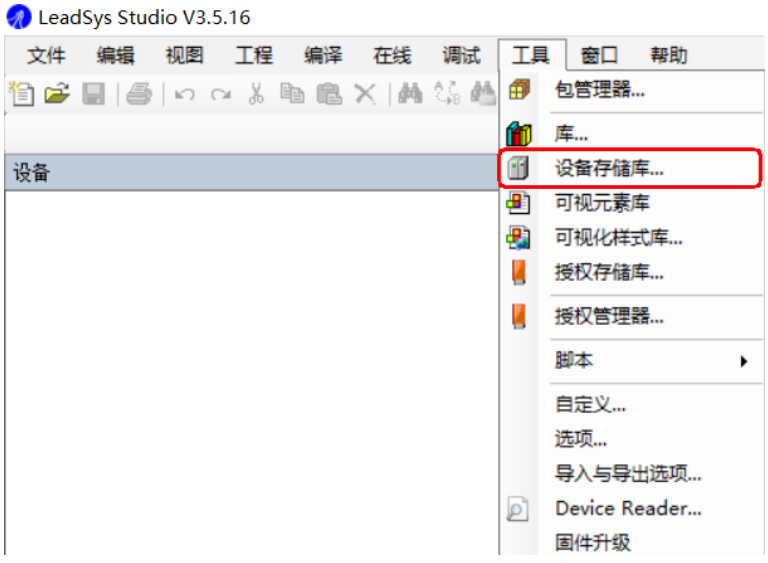


图 15.10　选择“设备存储库”

2、在弹出的窗口中点击“安装”按钮，如图 15.11 所示。

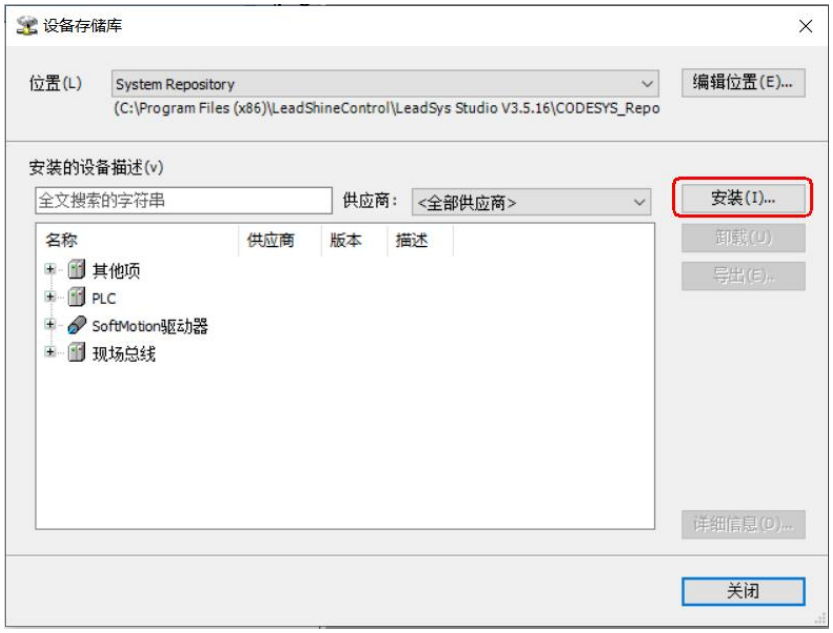


图 15.11　安装“设备存储库”

3、在弹出窗口中找到存放设备描述文件的路径。

注意：此次要将文件类型选择为 EtherCAT XML 设备描述配置文件，如果选择为其他文件类型，会导致无法正常选取 XML 文件，找到对应的 XML 后，点击“打开”按钮，如图 15.12 所示。

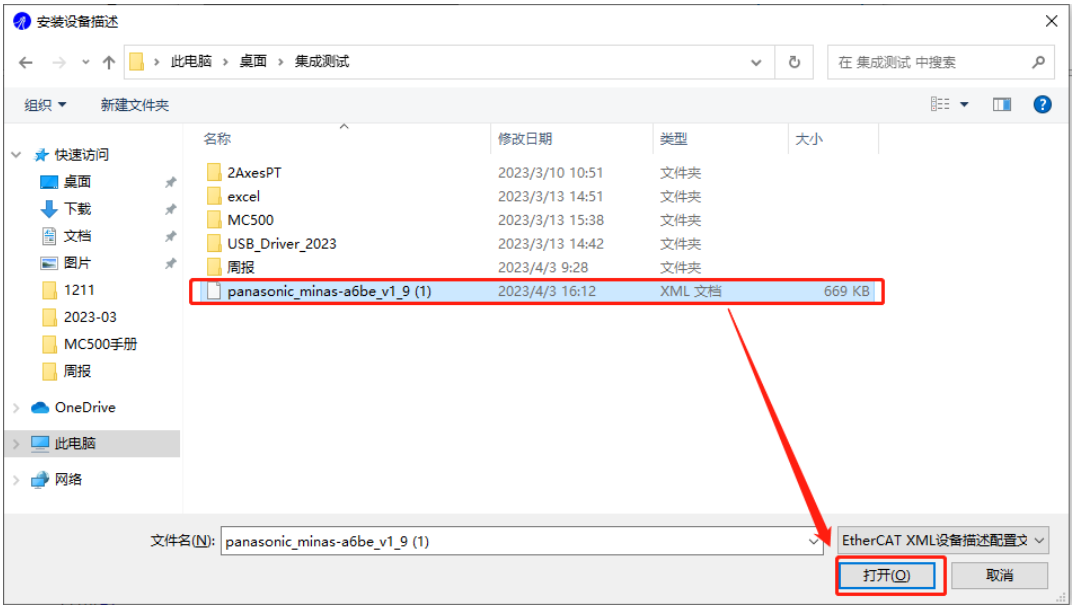


图 15.12　选择“设备描述文件”的路径

4、打开文件之后，软件会自动导入设备，此时可观察信息输出框，确认安装完成后，点击“关闭”按钮即可，如图 15.13 所示。

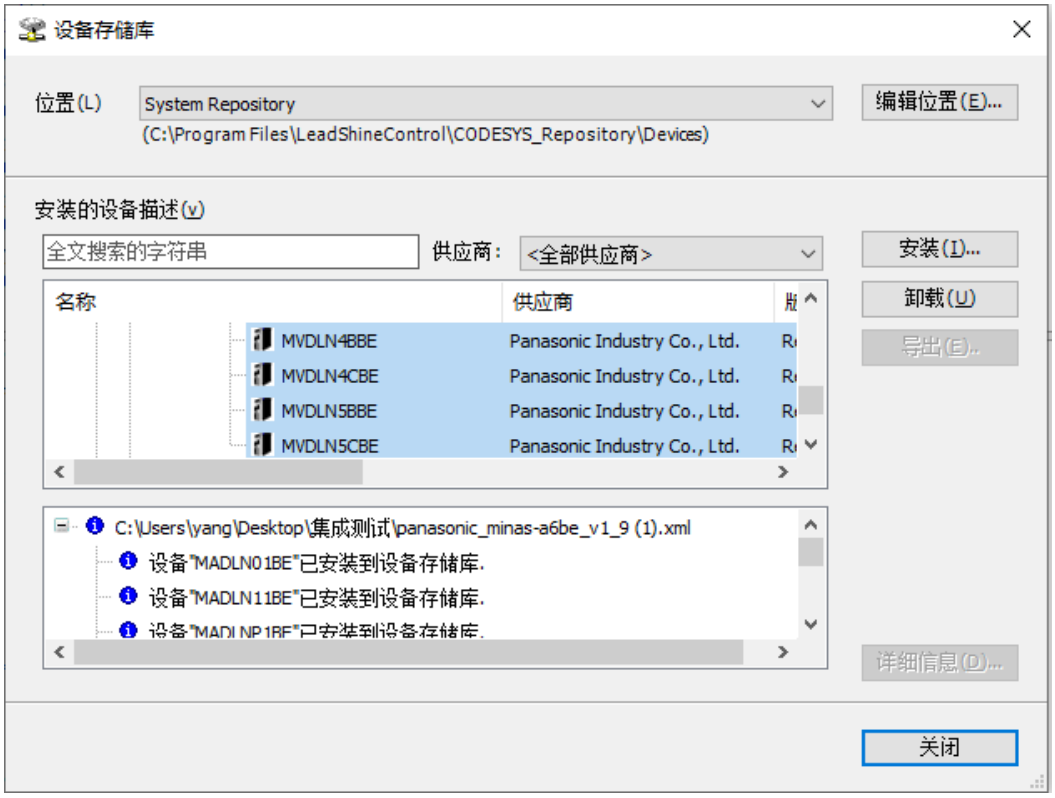


图 15.13　将“设备描述文件”导入设备

在设备存储库可以导出已经安装好的“设备描述文件”。选择要导出描述文件的设备，点击“导出”后，保存即可。导出过程如图 15.14 所示。

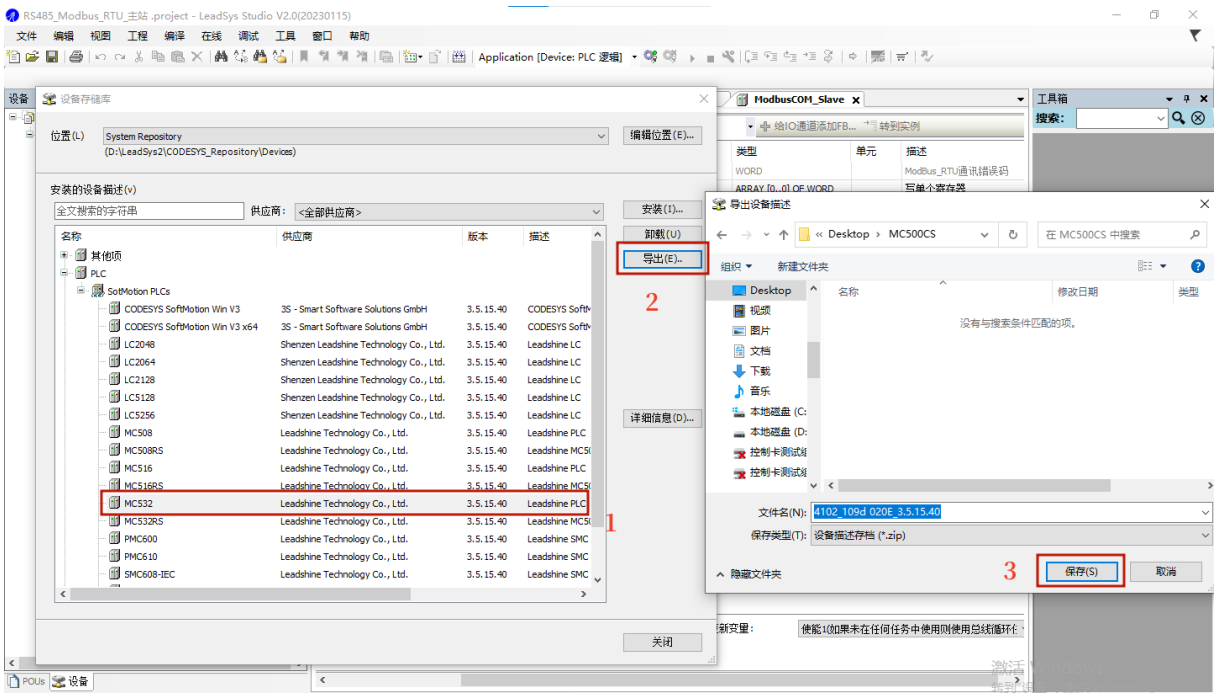


图 15.14　导出“设备描述文件”

15.6. 窗口布局

鼠标点击“窗口”选项，如图 15.15 所示，图中标注 1 的选项可以对窗口进行一系列操作，“下一个编辑器”和“上一个编辑器”可以切换编辑窗口；“重置窗口布局”窗口布局恢复到默认窗口布局。点击窗口选择“新建水平序列组”或者“新建垂直序列组”当前选择的窗口与其它窗口以水平或垂直方式布局，如图 15.16 点击“新建垂直序列组”后所示。

图 15.15 中的标注 2 处显示是已经打开的窗口。

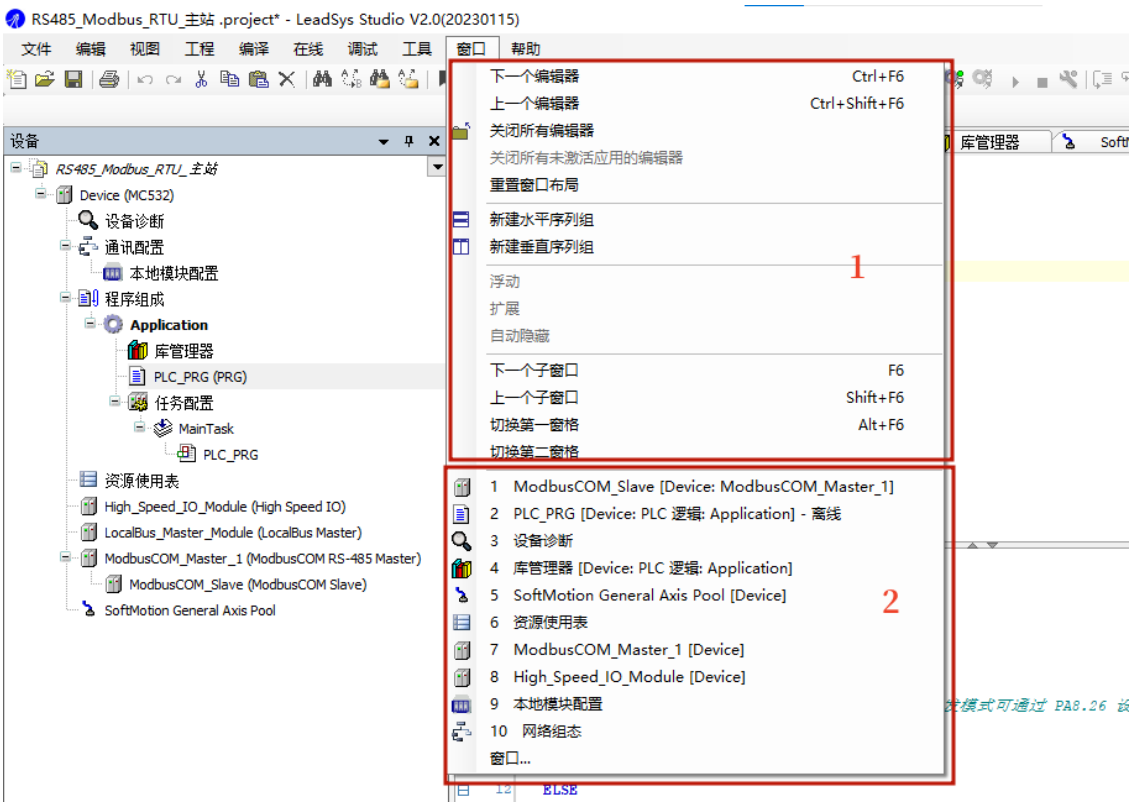


图 15.15　窗口设置选项

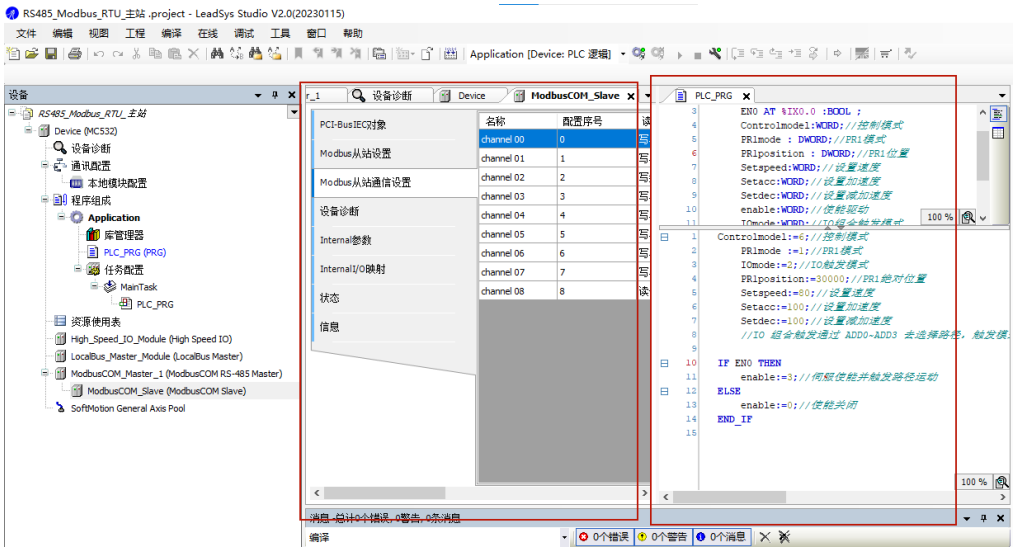


图 15.16　新建垂直序列组

通过直接拖拽窗口将变成浮动窗口状态，浮动窗口置于软件界面最顶层，可以自由拖动。也可以选中“窗口”→“浮动”选项，让窗口变成浮动状态。如图 15.18 所示。

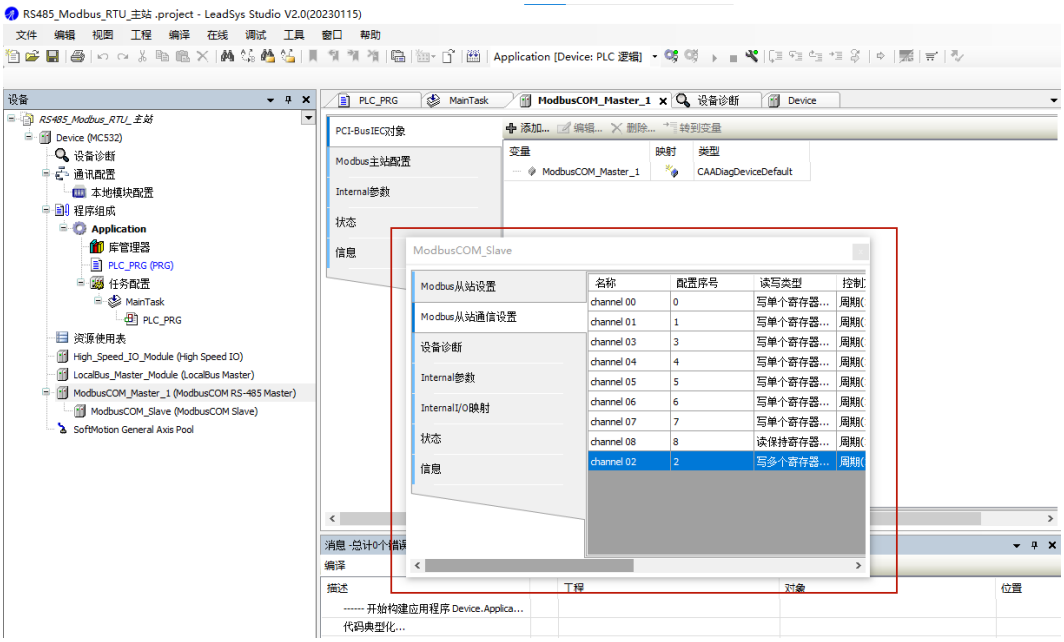


图 15.17　浮动窗口

15.7. 资源使用表

在“资源选择表”中可以看 PLC 资源使用情况和 Q 区、I 区和 M 区三个区的寄存器使用情况，如图 15.18 所示其中 PLC 默认 Q 区中地址%QB0～%QB1 已经被“Hight_Speed_IO_Modbus”输入变量定义使用，I 区中地址%IB0～%IB1 默认已经被“Hight_Speed_IO_Modbus”输出变量定义使用。

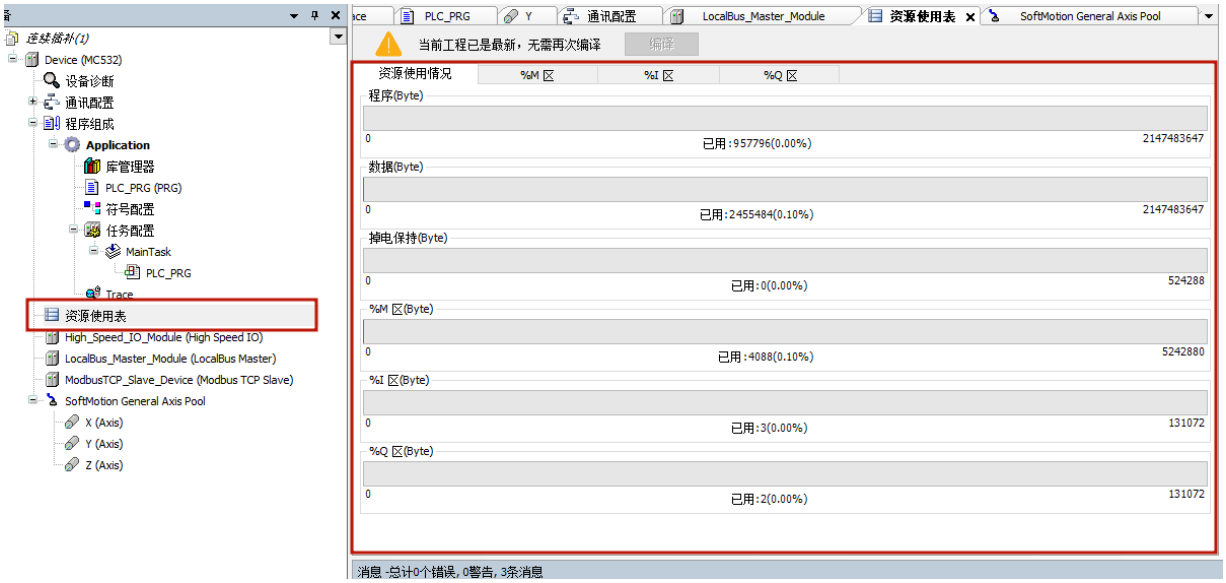


图 15.18　资源使用表

当同一个地址被重复定义使用时，就会显示为“冲突”状态。如定义两个变量并且地址一样。如图 15.19 所示。

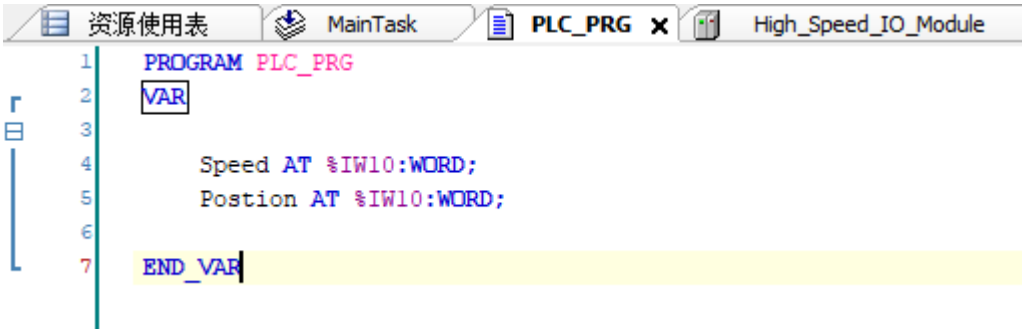


图 15.19　两个变量定义到同一地址

在“资源使用表”中点击“编译”，显示变量“Speed”和“Postion”映射的地址重复冲突。如图 15.20 所示。

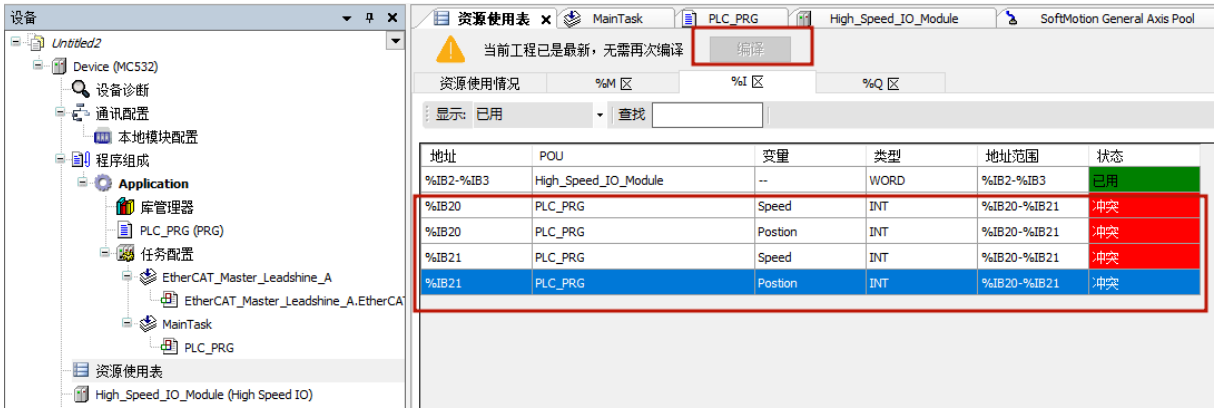


图 15.20　%I 区状态显示

客户咨询中心
目录索取·技术咨询·产品解惑
400-885-5521 销售热线
400-885-5501 技术热线

雷赛智能官方公众号

雷赛智能

成 就 客 户 ， 共 创 共 赢

深圳市雷赛智能控制股份有限公司

地址：深圳市南山区沙河西路 3185 号南山智谷产业园 B 栋 15-20 层

邮编：518000

电话：400-885-5521　　传真：0755－26402718

网址：www.leisai.com　　E_mail：marketing@leisai.com

上海分公司

上海市嘉定区江桥镇金园五路 601 号

电话：021-37829639　　传真：021-37829680

北京办事处

北京市大兴区天华大街 5 号院绿地启航国际 3 号楼 1109

电话：010-50846953　　传真：010-50846952

合肥办事处

合肥市蜀山区潜山路与高河东路交口绿地蓝海大厦 A 座 1209 室

电话：18110930188

山东办事处

济南市天桥区滨河商务中心 D 座 2003 室

电　话：　0531-55569943　　传　真：　0531-55569944

华中办事处

武汉市洪山区关山大道中建康城二期 17 栋一单元 1303

电话：13212778809

温州办事处

浙江省温州市瓯海区中汇路与振社路交叉口德信·泊林公馆 6 幢 1602 室

电话：18602163165