

案例教程 01 数字量 IO 控制

在 Visual Studio 2022 里从新建项目开始，
写出一个读输入、写输出的 C# 程序。
每一步都有截图，每一行代码都讲清楚。

Visual Studio 2022

C# WinForms

.NET 8

软卡 SDK

适用软件	SCMotion 4.4.17
配套视频	教程01_数字量IO控制_编程篇_WinForms版.mp4
配套工程	配套工程/IoDemo
文档版本	V1.0
发布日期	2026-10

ABOUT 关于本教程

适合谁	第一次用软卡 SDK 写程序的人。会装软件、会用 Windows 就行，不需要 C# 基础：每一行代码都在这里讲清楚。
做出什么	一个 Windows 窗体程序：连接主软件、取得控制权，写一个数字量输出点（让灯亮、灭），读一个输入点，并且每 200 毫秒自动刷新输入指示灯。
开发环境	Visual Studio 2022 · .NET 8 · C# Windows 窗体应用（WinForms）
适用软件	SCMotion 4.4.17，软卡 SDK 用同一版本主软件生成
配套视频	教程01_数字量IO控制_编程篇_WinForms版.mp4 （23 分钟，18 个章节）。本文各章和视频章节一一对应，章标题右边标了视频里的时间。
配套工程	配套工程/loDemo ：代码和本文附录 A 一字不差，打开就能编译运行。
建议先看	../教程01_数字量IO控制_上篇_主软件操作.mp4 ：先在主软件里把 IO 测通，再来写程序。WinForms / WPF / Qt 三个版本共用这一集。

怎么用这份教程

- 建议一边看视频一边做：视频看「怎么点」，这份文档查「点哪里、写什么、为什么」。
- 带编号圆点的步骤照顺序做；截图里的**红框**就是要点的地方，红色圆点上的数字对应步骤编号。
- 「」里是界面上的文字，比如按钮、菜单、属性名。**等宽字**是代码里的名字。
- 代码块左边的数字是 **Form1.cs 里的行号**，和附录 A 的完整代码、配套工程里的文件完全对得上。视频里说的「第一行」「第二行」，指的是函数里面的第几行。
- 出了问题先看第 9 章「常见问题」，再按返回码查附录 C。

需要的东西

东西	说明
电脑	Windows 10/11 64 位，装好 SCSMotion 4.4.17
Visual Studio 2022	社区版就可以，安装时勾选「.NET 桌面开发」（见 2.1）
EtherCAT IO 模块	DI0 接一个按钮，DO0 接一个指示灯；网线接模块的 IN 口。 没有硬件也能学 ：把主软件切到「仿真」，连接、读写的流程都能走通，只是没有真实的输入信号。

CONTENTS 目录

第 1 章 最终效果	视频 00:22	4
第 2 章 准备工作	视频 00:52	5
2.1	Visual Studio 2022	5
2.2	生成软卡 SDK 包	5
2.3	让主软件处于运行状态	6
第 3 章 新建项目	视频 02:11	7
第 4 章 引用 SDK	视频 03:17	9
4.1	把 DLL 放进项目文件夹	9
4.2	添加引用	9
4.3	把程序设成 64 位	11
第 5 章 画界面	视频 04:46	12
5.1	改窗口标题	12
5.2	放第一个按钮	12
5.3	照表放好其余控件	13
第 6 章 挂事件	视频 06:33	16
第 7 章 写代码	视频 07:30	18
7.1	文件开头的三行 using	18
7.2	三个成员：配置文件路径、身份号、句柄	18
7.3	工具函数 Log：往日志框写一行	19
7.4	「连接」按钮	19
7.5	断开：Disconnect 函数	21
7.6	写输出：WriteOutput 函数	22
7.7	读输入：「读输入」按钮	23
7.8	定时刷新：tmrPoll_Tick	23
第 8 章 编译运行	视频 19:38	25
8.1	编译	25
8.2	运行前确认主软件在运行	25
8.3	运行和验证	25
第 9 章 常见问题	视频 21:24	28
附录 A 完整代码 Form1.cs		29
附录 B 本教程用到的 SDK 指令		32
附录 C 返回码速查		35

第 1 章 最终效果

视频 00:22

先看最后要做出来的程序。界面一共五块：

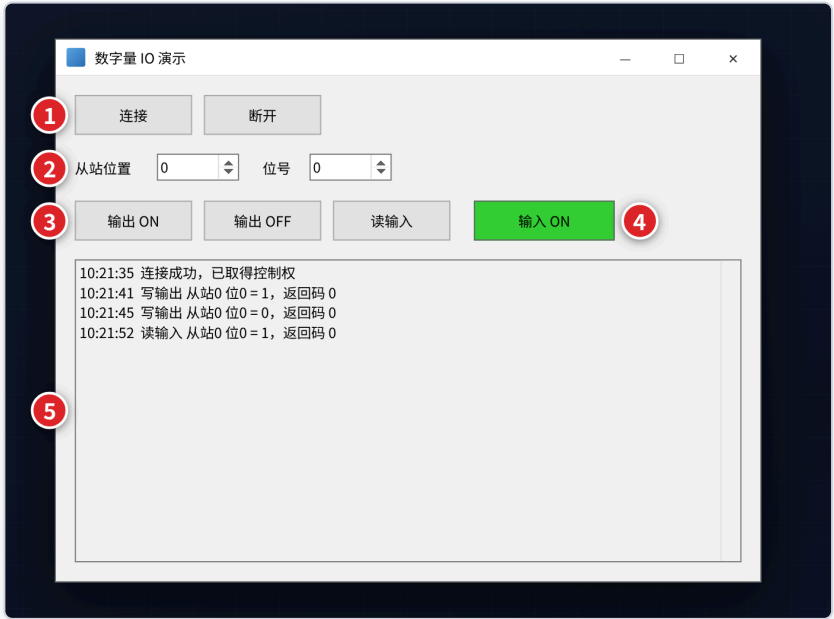


图 1-1 最后做出来的程序（运行中，已经连接）

编号	控件	作用
①	「连接」「断开」	连接主软件并取得控制权；断开时把控制权还给主软件
②	从站位置、位号	选要读写的是哪个点。从站位置是扫描到的顺序，第一台是 0；位号是这台从站上的第几个点，也从 0 开始
③	「输出 ON」「输出 OFF」「读输入」	给输出点写 1（接通）或写 0（断开）；读一次输入点
④	输入指示灯	程序每 200 毫秒自动读一次输入：有信号显示绿色「输入 ON」，没有信号显示灰色「输入 OFF」
⑤	日志框	每一步做了什么、返回码是多少，都记在这里，出问题时先看它

第 2 章 准备工作

视频 00:52

开始之前，准备好三样东西：Visual Studio 2022、软卡 SDK 包、处于运行状态的主软件。

2.1 Visual Studio 2022 视频 00:57

安装 Visual Studio 2022 时，在「工作负荷」里勾上「**.NET 桌面开发**」。不勾的话，新建项目时找不到「Windows 窗体应用」这个模板。

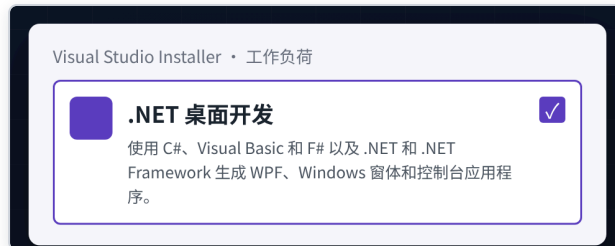


图 2-1 安装时勾选「.NET 桌面开发」

提示 已经装好、但当时没勾？打开开始菜单里的 **Visual Studio Installer**，在 Visual Studio 2022 那一栏点「**修改**」，勾上「.NET 桌面开发」，再点右下角的「**修改**」即可。

2.2 生成软卡 SDK 包 视频 01:09

- 在主软件左侧工程树的「**SYSTEM - 配置**」下面，点「**软卡设置**」(图中 ①)。
- 点「**生成 SDK 包**」(图中 ②)，选一个输出目录，就会生成一整套 SDK。

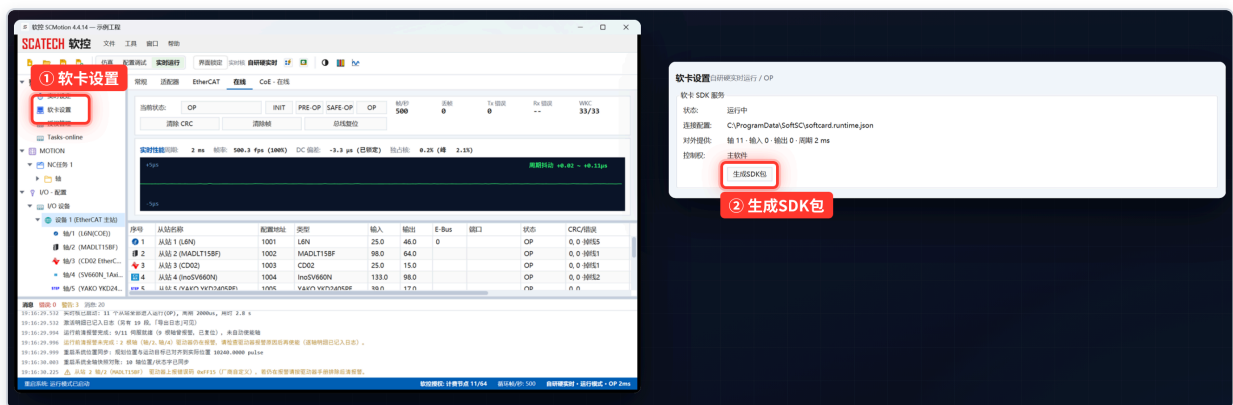


图 2-2 主软件里生成 SDK 包：① 软卡设置 ② 生成 SDK 包

生成的 SDK 包结构如下。**C# 程序只需要其中一个文件：** `sdk\dotnet\SCmotionSoftCard.Core.dll`。

```
SDK包\  
├─ sdk\  
│   └─ dotnet\  
│       ├── SCmotionSoftCard.Core.dll ← C# 只引用这一个  
│       └─ interp_core.dll (插补才用)  
│   └─ c\ (C/C++ 用)  
├─ docs\  
└─ README.txt
```

注意 主软件升级以后,要**重新生成一次 SDK 包**,并把项目里的 DLL 换成新的。SDK 和主软件必须是同一个版本,版本不一致时连接会被拒绝。

2.3 让主软件处于运行状态 视频 01:50

- **接了真机**:在主软件里点工具栏的「**激活配置参数**」,让从站进入 OP (上篇视频里演示过)。
- **手边没有硬件**:点模式条上的「**仿真**」。



图 2-3 没有硬件时,点模式条上的「仿真」

这时主软件会自动写出一个**连接配置文件**,路径固定是 `C:\ProgramData\SoftSC\softcard.runtime.json`。我们的程序就是读这个文件来连接主软件的。在「软卡设置」页能看到它:

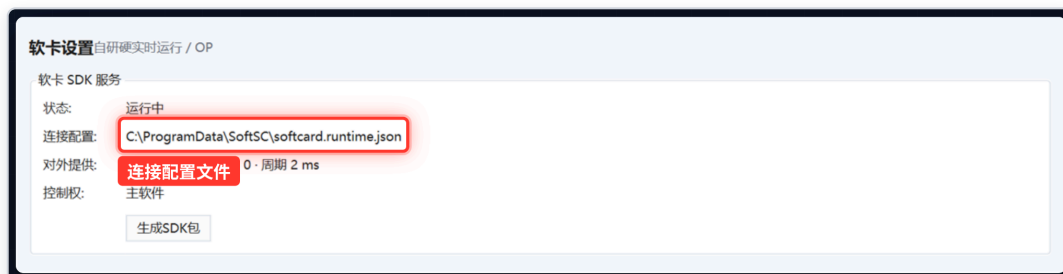


图 2-4 「软卡设置」页: SDK 服务状态和连接配置文件的路径

说明 主软件停在「配置调试」、没有进 OP 也没有切仿真时,这个文件不存在,程序会连接失败 (返回码 10)。

第 3 章 新建项目

视频 02:11

- 1 打开 Visual Studio 2022，在启动窗口右边点「创建新项目」。

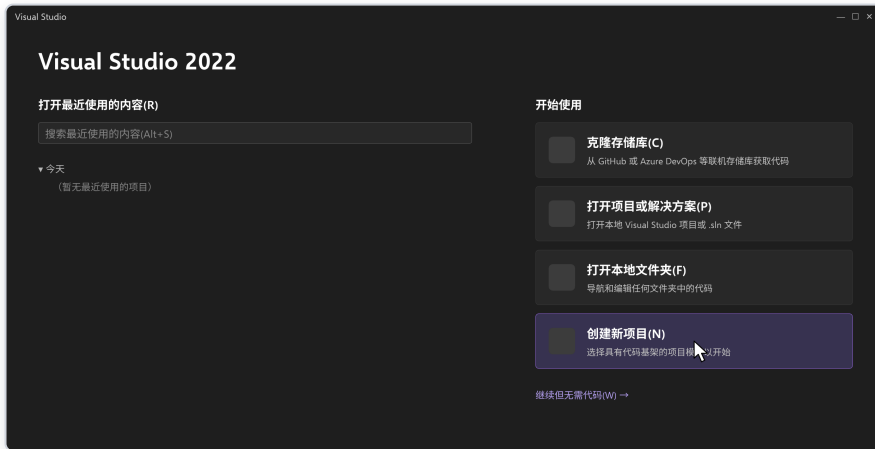


图 3-1 启动窗口：点「创建新项目」

- 2 在上方搜索框里输入「窗体」两个字。
- 3 在列表里选「Windows 窗体应用」：语言标签是 C#，名字后面不带「(.NET Framework)」。



图 3-2 选 C# 的「Windows 窗体应用」

注意 列表里有几个长得很像的模板：标签是 Visual Basic 的不要选；名字后面带「(.NET Framework)」的也不要选，那是旧框架。

- 4 「项目名称」填 **IoDemo**；「位置」选一个自己找得到的文件夹。
- 5 勾选「将解决方案和项目放在同一目录中」，这样文件夹结构最简单。点「下一步」。
- 6 「框架」选 **.NET 8.0（长期支持）**，点「创建」。

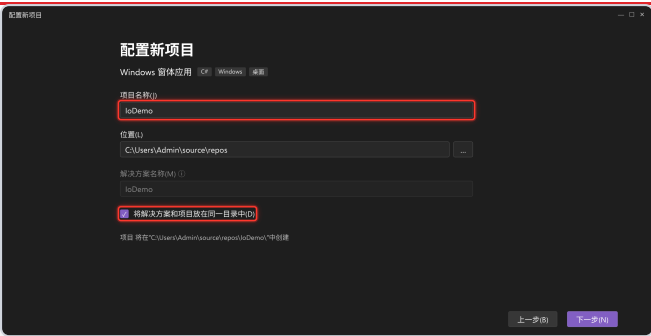


图 3-3 项目名称 IoDemo，勾选「放在同一目录中」



图 3-4 框架选 .NET 8.0，① 点「创建」

等几秒，项目就建好了。认识一下三块区域，后面会反复用到：

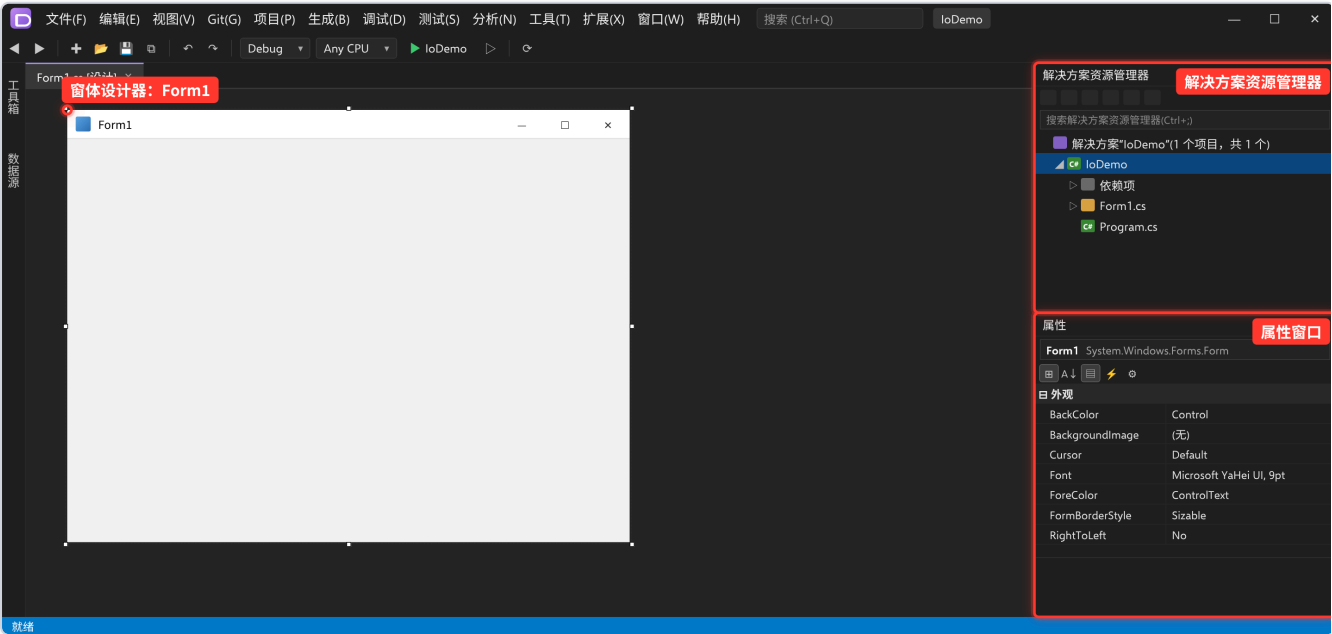


图 3-5 新建好的项目：窗体设计器、解决方案资源管理器、属性窗口

区域	是什么
窗体设计器（中间）	显示程序的主窗口 Form1。往上面拖控件，就是在画界面
解决方案资源管理器（右上）	列出项目里的文件和引用
属性窗口（右下）	选中什么，就显示什么的属性：窗体、按钮、定时器都在这里改

第 4 章 引用 SDK

视频 03:17

4.1 把 DLL 放进项目文件夹 视频 03:17

- 1 打开项目文件夹 IoDemo，在里面新建一个文件夹，命名为 **sdk**。
- 2 把 SDK 包里的 **SCmotionSoftCard.Core.dll** 复制到这个 **sdk** 文件夹里。

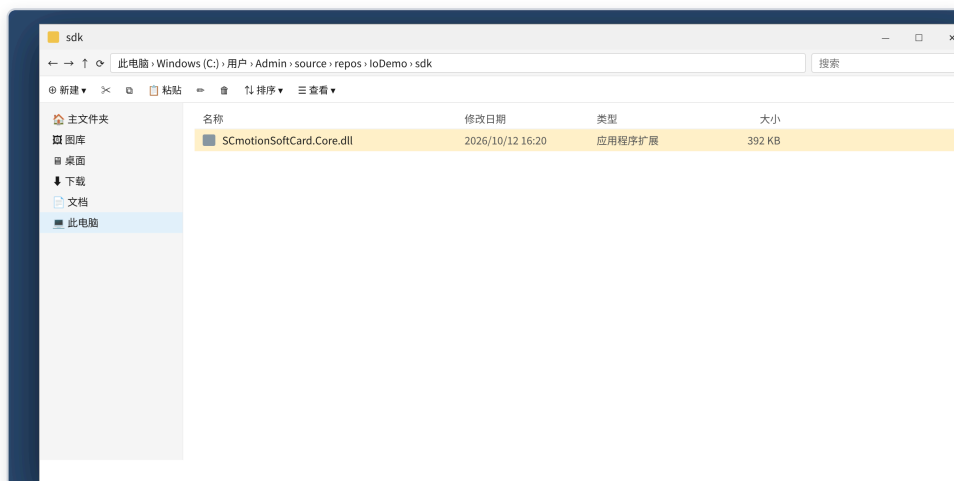


图 4-1 项目文件夹 IoDemo\sdk 里放好 SCmotionSoftCard.Core.dll

提示 SDK 放在项目文件夹里，整个项目拷到别的电脑上也能直接编译。

不知道项目文件夹在哪？在解决方案资源管理器里右键项目 IoDemo，点「在文件资源管理器中打开文件夹」。

4.2 添加引用 视频 03:41

- 1 回到 Visual Studio，在解决方案资源管理器里右键「依赖项」，选「添加项目引用」。

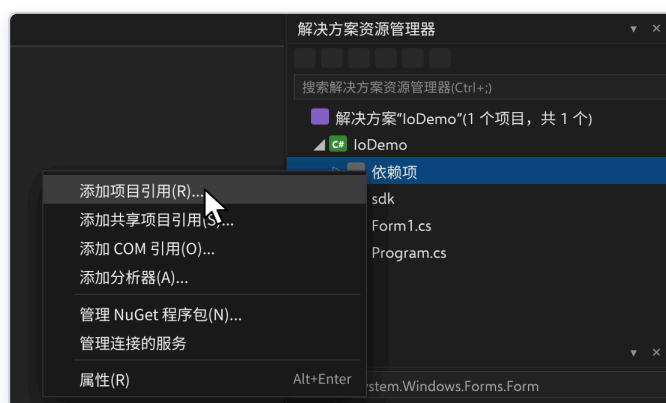


图 4-2 右键「依赖项」→「添加项目引用」

- 2 在弹出的引用管理器里，先点左边的「浏览」(①)，再点右下角的「浏览(B)...」按钮 (②)。
- 3 找到刚才复制到 **sdk** 文件夹里的 **SCmotionSoftCard.Core.dll**，选中，点「添加」(③)。

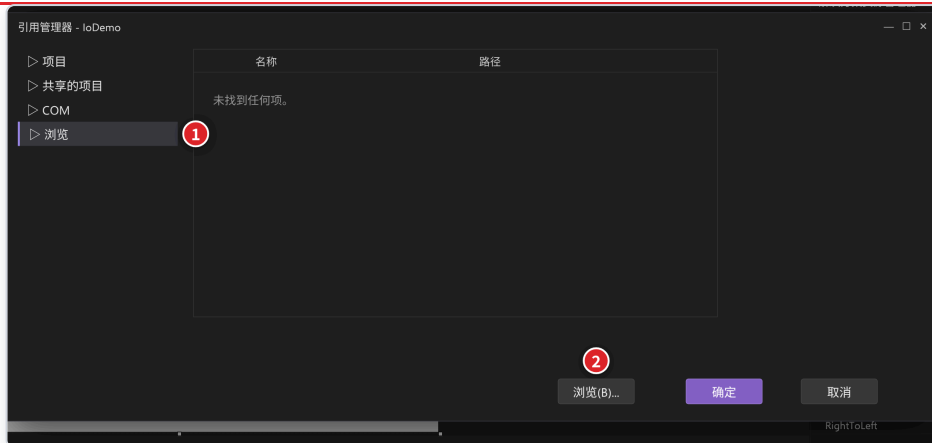


图 4-3 引用管理器：① 左边点「浏览」 ② 右下角点「浏览(B)...」

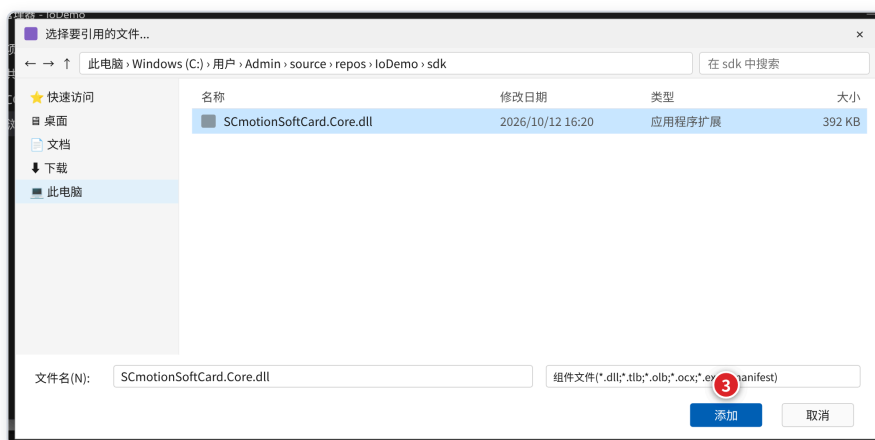


图 4-4 ③ 选中 sdk 文件夹里的 SCmotionSoftCard.Core.dll，点「添加」

4 回到引用管理器，确认它前面打了勾，点「确定」(④)。

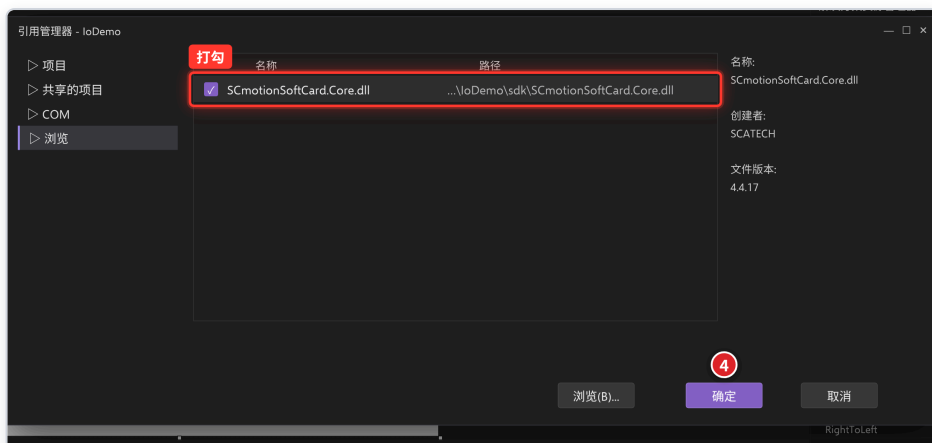


图 4-5 确认打勾，④ 点「确定」

5 在解决方案资源管理器里展开「依赖项」→「程序集」，看到 SCmotionSoftCard.Core，引用就成功了。

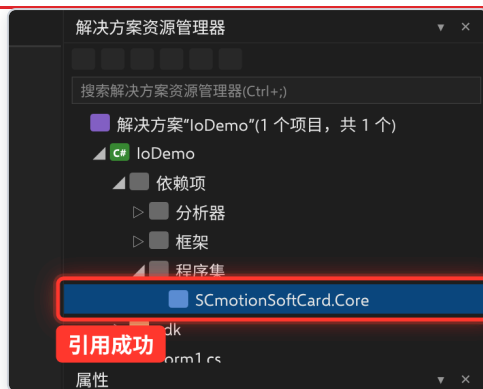


图 4-6 「程序集」下面出现 SCmotionSoftCard.Core

4.3 把程序设成 64 位 视频 04:15

这个 SDK 是 64 位的，所以我们的程序也要设成 64 位。

- 1 右键项目「IoDemo」，选「属性」。

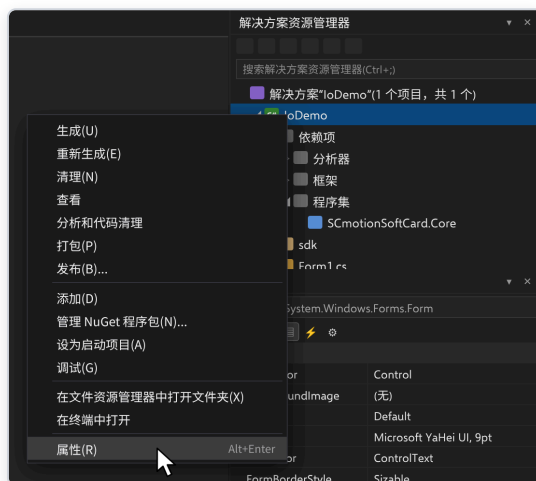


图 4-7 右键项目 IoDemo → 「属性」

- 2 在「生成」→「常规」里找到「平台目标」，从 Any CPU 改成 x64。

- 3 按 **Ctrl** + **S** 保存。



图 4-8 「平台目标」改成 x64

注意 不改的话，编译时会出现「处理器架构不匹配」的警告，程序运行时还可能出错。

第 5 章 画界面

视频 04:46

5.1 改窗口标题 视频 04:46

- 1 点一下设计器里的窗体，在属性窗口找到「Text」，改成 **数字量 IO 演示**。这就是程序窗口的标题。

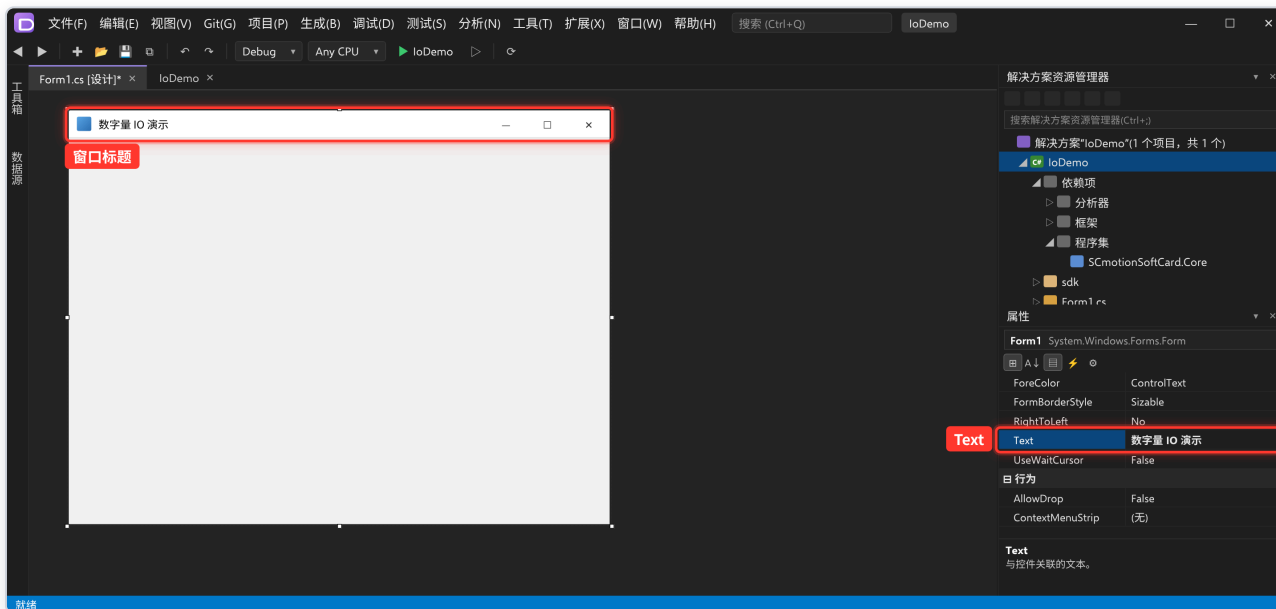


图 5-1 选中窗体，属性 Text 改成「数字量 IO 演示」

5.2 放第一个按钮 视频 04:58

- 2 点开左边的「工具箱」，在「公共控件」里找到 **Button**。
- 3 按住它拖到窗体上，松开，就放好了一个按钮。

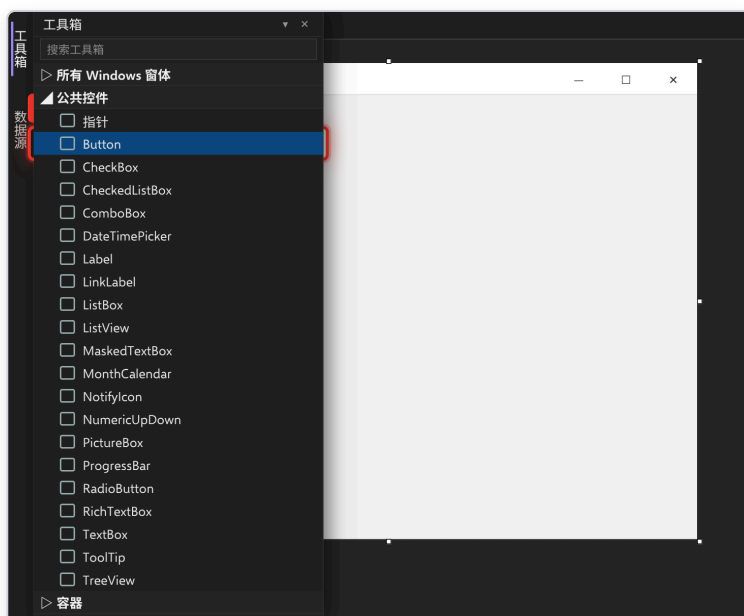


图 5-2 工具箱 → 公共控件 → Button，拖到窗体上

4 选中这个按钮，在属性窗口里把「(Name)」改成 btnConnect，把「Text」改成 连接。

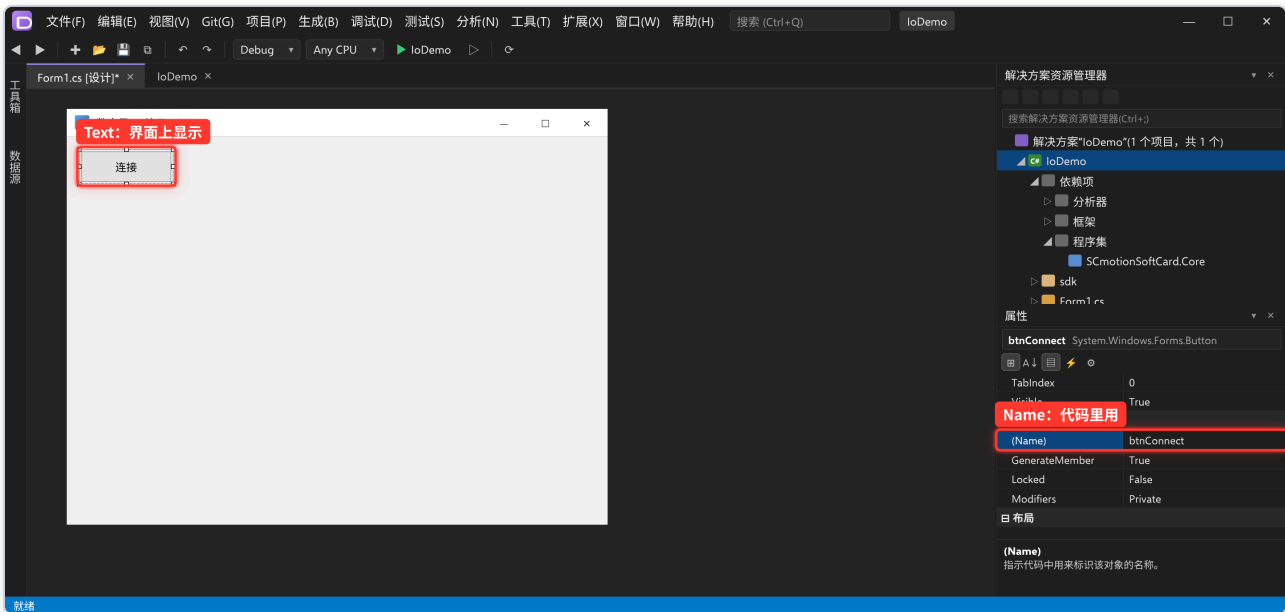


图 5-3 按钮的 Name 改成 btnConnect，Text 改成「连接」

说明 Name 和 Text 的区别：Name 是代码里用的名字，Text 是界面上显示的文字。Name 起得有意义（btn 开头表示按钮），后面写代码才不会乱。

5.3 照表放好其余控件 视频 05:27

用同样的方法放好下面所有控件。**Name 必须一字不差**（代码就是按名字找控件的），位置和大小照着图 5-4 摆个大概就行。

控件类型	Name	Text	还要改的属性
窗体	Form1	数字量 IO 演示	—
Button	btnConnect	连接	
Button	btnDisconnect	断开	
Label	label1	从站位置	名字不用改
NumericUpDown	numSlave		Maximum = 255
Label	label2	位号	名字不用改
NumericUpDown	numBit		Maximum = 255
Button	btnOn	输出 ON	
Button	btnOff	输出 OFF	
Button	btnRead	读输入	
Label	lblDi	输入 --	AutoSize = False, BorderStyle = FixedSingle, TextAlign = MiddleCenter, BackColor = Gainsboro
TextBox	txtLog		Multiline = True, ReadOnly = True, ScrollBars = Vertical
Timer (在「组件」里)	tmrPoll		Interval = 200

几个属性是什么意思：

属性	改成	为什么
Maximum (数字框)	255	数字框默认最大只能到 100，改大一点，从站多、位号大时也能选
AutoSize (标签)	False	关掉自动大小，标签才能拉成一个方块，当指示灯用
BorderStyle (标签)	FixedSingle	给标签加一圈边框
TextAlign (标签)	MiddleCenter	文字在方块里居中
Multiline (文本框)	True	能显示多行，日志才能一行一行往下记
ReadOnly (文本框)	True	只读，防止误改日志
ScrollBars (文本框)	Vertical	日志多了可以往下翻
Interval (定时器)	200	每 200 毫秒触发一次，单位是毫秒

提示 Timer 不在「公共控件」里，在工具箱的「组件」分组里。它拖到窗体上以后不显示在窗体上，而是出现在设计器下方的灰色区域里。

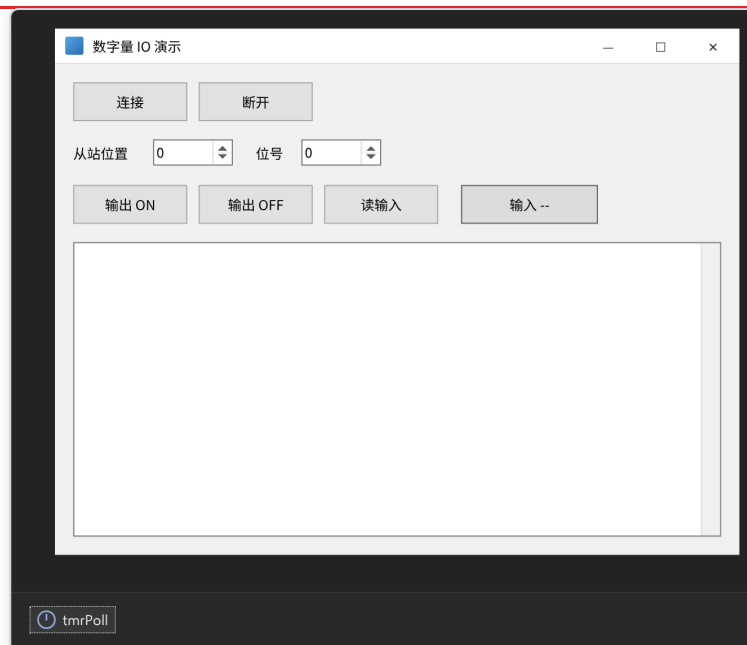


图 5-4 画好的界面（下方是定时器 tmrPoll）

第 6 章 挂事件

视频 06:33

界面画好了，接下来告诉程序：点了哪个按钮，就执行哪段代码。这一步叫「挂事件」。

- 1 在设计器里双击「连接」按钮。Visual Studio 会自动切到代码，生成一个空函数 `btnConnect_Click`。程序运行时点这个按钮，就会执行这个函数里的代码。

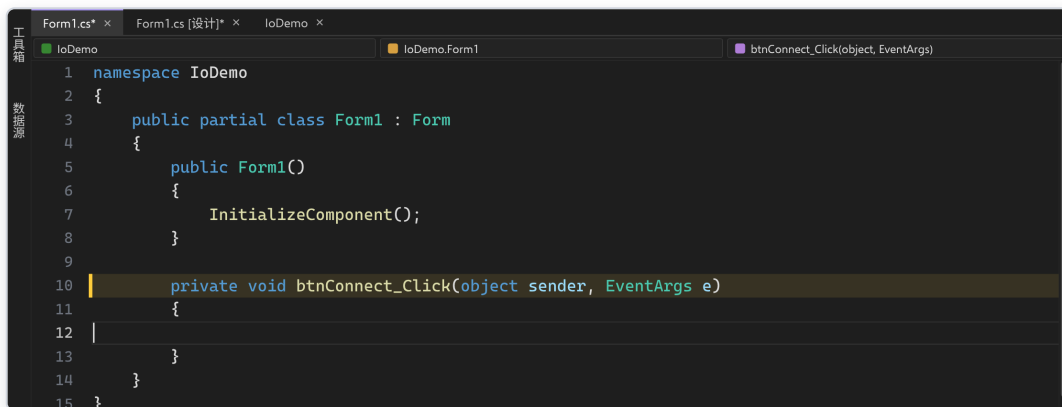


图 6-1 双击「连接」按钮，自动生成 `btnConnect_Click`

- 2 回到设计器（点上方的「Form1.cs [设计]」标签），对「断开」「输出 ON」「输出 OFF」「读输入」四个按钮也各双击一次。
- 3 再双击设计器下方的 `tmrPoll`，生成定时器每次触发时执行的 `tmrPoll_Tick`。
- 4 最后选中窗体，在属性窗口点闪电图标（①）切到事件列表，找到「FormClosing」，双击它。这个事件在窗口关闭之前触发。

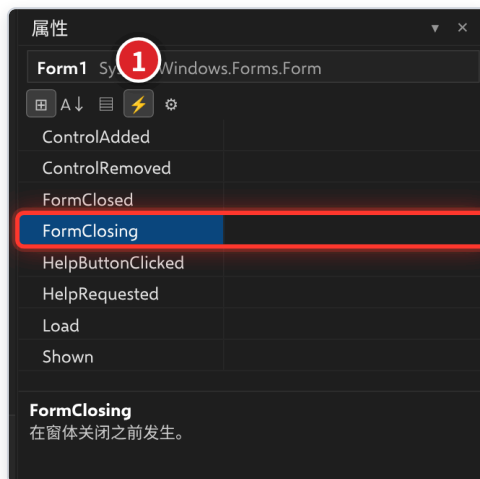


图 6-2 ① 点闪电图标切到事件列表，双击 `FormClosing`

- 5 切回代码，一共生成了七个空函数，接下来就往里面填代码。

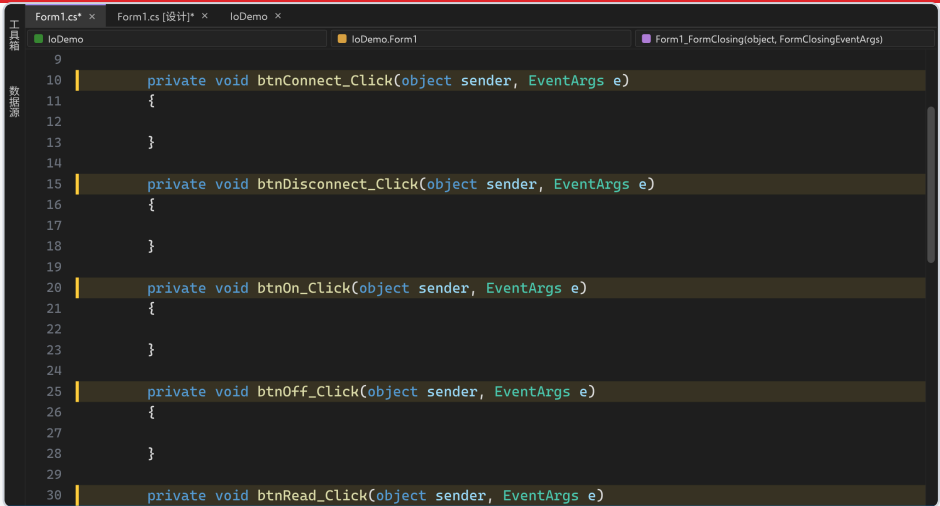


图 6-3 代码里多了七个空函数

自动生成的函数	什么时候执行
btnConnect_Click	点「连接」
btnDisconnect_Click	点「断开」
btnOn_Click	点「输出 ON」
btnOff_Click	点「输出 OFF」
btnRead_Click	点「读输入」
tmrPoll_Tick	定时器启动以后，每 200 毫秒一次
Form1_FormClosing	窗口关闭之前

注意 这几个函数一定要双击生成，不要自己手敲。双击时 Visual Studio 会同时在设计器文件里把按钮和函数「挂」在一起；手敲一个同名函数，点按钮不会有任何反应。

第 7 章 写代码

视频 07:30

下面按视频的顺序，一段一段往 Form1.cs 里填代码。每段代码下面是逐行说明。代码块左边是最终文件里的行号，写完以后整个文件和附录 A 一模一样，可以对照着检查每段放的位置。

整个程序只用到下面 7 条 SDK 指令，这一章会逐条讲到，完整说明见附录 B。

指令	作用	在哪用
SCM_OpenProjectConfig	读连接配置文件，连上主软件，拿到句柄	7.4 连接
SCM_GetOpenErrorText	连接失败时取出具体原因	7.4 连接
SCM_RequestOwner	取得控制权，之后才能写输出	7.4 连接
SCM_ReleaseOwner	把控制权还给主软件	7.5 断开
SCM_Close	关闭句柄，断开连接	7.4、7.5
SCM_WriteOutputBit	按「从站位置 + 位号」写一个输出位	7.6 写输出
SCM_ReadInputBit	按「从站位置 + 位号」读一个输入位	7.7、7.8

提示 注释（// 开头的绿色文字）不影响运行，可以不写，但建议照着写上，以后回来看一眼就知道这段是干什么的。

7.1 文件开头的三行 using 视频 07:30

```
1 using System.Text;
2 using SCmotion.SoftCard.Sdk;
3 using static SCmotion.SoftCard.Sdk.ScMotionApi;
```

using 的意思是：下面要用到这个命名空间里的东西，写的时候可以省掉前缀。

行	说明
1	using System.Text;：后面接收错误信息要用的 StringBuilder 在这个命名空间里。
2	using SCmotion.SoftCard.Sdk;：软卡 SDK 的命名空间，句柄类型 ScMotionCard 在里面。
3	多了一个 static：using static 后面跟的是 ScMotionApi 这个类。写了它，调用 SDK 指令时就能直接写 SCM_ 开头的函数名，不用每次都在前面加 ScMotionApi.。这样写出来的指令名和参数顺序，和手册里、和 C 语言版本完全一样。

7.2 三个成员：配置文件路径、身份号、句柄 视频 08:34

在类里面、构造函数 public Form1() 的上方，定义三个成员：

```

9 // 连接配置文件：主软件进 OP（或切到仿真）时自动生成，路径固定
10 const string ConfigPath = @"C:\ProgramData\SoftSC\softcard.runtime.json";
11
12 // 本程序的身份证号：自己定一个，多个程序同时连接时要各不相同
13 const int ClientId = 88;
14
15 // 软卡句柄：连接成功后才有值，null 表示还没连接
16 ScMotionCard? card;

```

行	说明
10	连接配置文件的路径。 <code>const</code> 表示常量，定义好就不能再改； <code>string</code> 表示字符串。路径前面的 <code>@</code> 让里面的反斜杠按原样使用，不用写成两个 <code>\\</code> 。
	这个路径是固定的：主软件进入 OP 或者切到仿真时，会自动在这里写出这个文件。连接时说找不到它，多半是主软件还没进 OP，而不是路径写错了。
13	<code>ClientId</code> 是本程序的身份证号， <code>int</code> 表示整数。这个数自己定，88 只是举例。好几个程序同时连主软件时，每个程序的身份证号要不一样。配置文件里也有一个 <code>clientId</code> ，那是主软件自己的， 不要照抄 。
16	软卡句柄 <code>card</code> ，类型是 <code>ScMotionCard</code> 。连接成功以后 SDK 会把句柄交给它，之后每一条 SDK 指令的第一个参数都是它。
	类型后面的 <code>?</code> 表示它可以是空值 <code>null</code> 。没连接的时候它就是 <code>null</code> ，后面我们就靠判断它是不是 <code>null</code> ，来知道现在有没有连接。

7.3 工具函数 Log：往日志框写一行 视频 10:06

先写一个小工具函数 `Log`，后面每个按钮都用它往日志框里写一行字。把它放在类的**最后面**：

```

119 // 在日志框末尾追加一行，前面带上时间
120 private void Log(string text)
121 {
122     txtLog.AppendText($"{DateTime.Now:HH:mm:ss} {text}\r\n");
123 }

```

行	说明
120	<code>private void</code> 表示这是窗体内部用的函数，不返回任何值。括号里的 <code>string text</code> 是它的参数，也就是要写的那段文字。
122	<code>txtLog.AppendText(...)</code> ：在日志框的末尾追加文字。
	字符串前面的 <code>\$</code> 表示字符串插值：花括号里可以直接放变量。 <code>DateTime.Now</code> 是当前时间，后面的 <code>HH:mm:ss</code> 把它格式化成「时:分:秒」。
	最后的 <code>\r\n</code> 是回车换行，这样每条日志各占一行。

7.4 「连接」按钮 视频 11:00

这是整个程序最重要的一段。连接要做三件事：**打开连接 → 取得控制权 → 启动定时刷新**。在 `btnConnect_Click` 里写：

```
23 // 「连接」按钮：打开连接 → 取得控制权 → 启动定时刷新
24 private void btnConnect_Click(object sender, EventArgs e)
25 {
26     if (card != null) { Log("已经连接了"); return; }
27
28     int rc = SCM_OpenProjectConfig(ConfigPath, ClientId, out card);
29     if (rc != SCM_OK)
30     {
31         var why = new StringBuilder(1024);
32         SCM_GetOpenErrorText(why, why.Capacity);
33         Log($"连接失败，返回码 {rc}: {why}");
34         card = null;
35         return;
36     }
37
38     rc = SCM_RequestOwner(card);
39     if (rc != SCM_OK)
40     {
41         Log($"取控制权失败，返回码 {rc}");
42         SCM_Close(card);
43         card = null;
44         return;
45     }
46
47     tmrPoll.Start();
48     Log("连接成功，已取得控制权");
49 }
```

行	说明
23	注释，写清楚这个函数要做的三件事。
26	先做防护： <code>card</code> 不等于 <code>null</code> ，说明已经连上了，就写一条日志，然后 <code>return</code> 直接返回，避免重复连接。
28	连接的核心：调用 <code>SCM_OpenProjectConfig</code> 。三个参数依次是配置文件路径、身份号；第三个前面有 <code>out</code> ，表示出参，连接成功后 SDK 会把句柄写回 <code>card</code> 。
	函数返回一个整数返回码，存进变量 <code>rc</code> 。所有 SDK 指令都返回这样一个码： 0 表示成功 ，不是 0 就表示出了问题。
29	<code>SCM_OK</code> 是 SDK 定义好的常量，值就是 0。 <code>rc != SCM_OK</code> 就说明连接失败了。
31	失败的时候光看返回码还不够。先准备一个 <code>StringBuilder</code> ，可以装 1024 个字符，用来接收失败原因。
32	调用 <code>SCM_GetOpenErrorText</code> ，把失败的具体原因写进 <code>why</code> 。如果是双方参数对不上，它会逐项列出双方的轴数、IO 点数和周期，照着对就知道哪里不一致。除了打开连接的指令，这是 SDK 里唯一一条不需要句柄的指令，因为连接失败的时候还没有句柄。
33	把返回码和原因一起写进日志。
34-35	把 <code>card</code> 重新设成 <code>null</code> ，然后 <code>return</code> ，下面的代码就不再执行。
38	第二步，取得控制权：调用 <code>SCM_RequestOwner</code> ，参数只有一个句柄。返回码同样存进 <code>rc</code> （前面已经声明过 <code>rc</code> ，这里直接赋值，不再写 <code>int</code> ）。

	刚连上的时候，程序只有读的权限；拿到控制权，才能写输出、发运动指令。同一时刻只有一个程序能拿到控制权。取得控制权的同时 SDK 会自动完成武装，所以不需要再调用 <code>SCM_EtherCATStart</code> 。
39-41	同样判断返回码。最常见的两种失败：返回 3 ，主软件没进 OP 或当前模式不允许下发；返回 7 ，控制权在别的程序手上。
42-44	拿不到控制权就没法写输出，所以调用 <code>SCM_Close</code> 关掉句柄，把 card 设回 null，然后返回。
47	两步都成功了，启动定时器 <code>tmrPoll</code> 。从现在开始，每 200 毫秒会自动执行一次 <code>tmrPoll_Tick</code> 。
48	写一条日志，告诉自己连接成功。

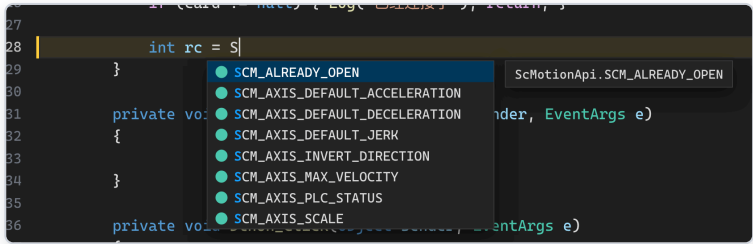


图 7-1 输入 SCM 下划线时，智能提示会列出所有 SDK 指令

说明 从头捋一遍：防重复连接 → 打开连接，失败就记原因 → 取控制权，失败就关句柄 → 都成功了，启动定时器。

7.5 断开：Disconnect 函数 视频 14:45

有连接就要有断开。在 `Log` 函数的上面，再写一个 `Disconnect` 函数：

```
107 // 断开：停定时器 → 归还控制权 → 关闭句柄
108 private void Disconnect()
109 {
110     if (card == null) return;
111
112     tmrPoll.Stop();
113     SCM_ReleaseOwner(card);
114     SCM_Close(card);
115     card = null;
116     Log("已断开");
117 }
```

行	说明
110	如果本来就没有连接，直接返回。
112	先停掉定时器，免得断开以后它还去读输入。
113	<code>SCM_ReleaseOwner</code> ：把控制权还给主软件，还了以后主软件马上就能恢复操作。程序不还就直接退出的话，主软件要等大约 5 秒，才会判定它失联、把控制权收回去。
114	<code>SCM_Close</code> ：关闭句柄，释放这个连接。
115-116	把 card 设回 null，再写一条日志。

然后在「断开」按钮和窗口关闭事件里，各调用一句 `Disconnect()`：

```
51 // 「断开」按钮
52 private void btnDisconnect_Click(object sender, EventArgs e)
53 {
54     Disconnect();
55 }
```

```
90 // 关闭窗口前也断开，把控制权还给主软件
91 private void Form1_FormClosing(object sender, FormClosingEventArgs e)
92 {
93     Disconnect();
94 }
```

提示 窗口关闭事件里也调用 Disconnect，这样就算用户直接点右上角的叉关掉程序，控制权也会及时还给主软件。

7.6 写输出：WriteOutput 函数 视频 15:54

「输出 ON」和「输出 OFF」只差一个值，所以写一个公用函数 WriteOutput，放在 Disconnect 的上面：

```
96 // 写一个数字量输出位：value 为 1 接通，为 0 断开
97 private void WriteOutput(int value)
98 {
99     if (card == null) { Log("请先连接"); return; }
100
101     int slave = (int)numSlave.Value;
102     int bit = (int)numBit.Value;
103     int rc = SCM_WriteOutputBit(card, slave, bit, value);
104     Log($"写输出 从站{slave} 位{bit} = {value}, 返回码 {rc}");
105 }
```

行	说明
97	参数 value 是要写的值：写 1 表示接通，写 0 表示断开。
99	先检查有没有连接，没连接就提示「请先连接」并返回。
101	从界面读出从站位置。numSlave.Value 是数字框里的值，类型是 decimal（小数）；前面括号里的 (int) 把它转换成整数。
	从站位置就是扫描时的顺序，从 0 开始编号：扫描到的第一台从站是 0，第二台是 1。
102	位号也一样读出来。位号是这台从站自己的第几个输出点（读输入时就是第几个输入点），也从 0 开始。
103	真正写输出：SCM_WriteOutputBit，四个参数依次是句柄、从站位置、位号和值。
	这种「从站 + 位号」的写法是推荐用法：别的从站增减 IO 点数，不会影响这个点的地址；只有在它前面插入或拿掉从站，从站位置才会跟着变。
104	把写了什么、返回码是多少，记到日志里。

注意 如果这个点在主软件里根本没有映射，写它也会返回 0，只是什么都不会发生。所以第一次接线，一定要到现场看指示灯有没有亮。

最后在两个按钮里调用它，「输出 ON」传 1，「输出 OFF」传 0：

```

57 // 「输出 ON」按钮
58 private void btnOn_Click(object sender, EventArgs e)
59 {
60     WriteOutput(1);
61 }
62
63 // 「输出 OFF」按钮
64 private void btnOff_Click(object sender, EventArgs e)
65 {
66     WriteOutput(0);
67 }

```

7.7 读输入：「读输入」按钮 视频 17:48

```

69 // 「读输入」按钮
70 private void btnRead_Click(object sender, EventArgs e)
71 {
72     if (card == null) { Log("请先连接"); return; }
73
74     int slave = (int)numSlave.Value;
75     int bit = (int)numBit.Value;
76     int rc = SCM_ReadInputBit(card, slave, bit, out int value);
77     Log($"读输入 从站{slave} 位{bit} = {value}, 返回码 {rc}");
78 }

```

行	说明
72-75	前三行和写输出一样：检查连接，读出从站位置和位号。
76	SCM_ReadInputBit 读一个输入位。前三个参数和写输出一样；第四个参数是出参：out int value 在这里顺手声明了一个整数变量 value，读到的值就放进它。
	读输入是只读操作，其实不需要控制权，只要连上就能读。
77	把读到的值写进日志：value 等于 1 表示有信号，等于 0 表示没有信号。

注意 没有映射的点，读回来也是 0，返回码照样是 0。所以读到 0 不一定是没有信号，也可能是这个点没有配置。

7.8 定时刷新：tmrPoll_Tick 视频 18:43

定时器的 Tick 函数每 200 毫秒执行一次，用来实时刷新右边的输入指示灯：

```
80 // 定时器：每 200 毫秒读一次输入，刷新右边的指示灯
81 private void tmrPoll_Tick(object sender, EventArgs e)
82 {
83     if (card == null) return;
84
85     SCM_ReadInputBit(card, (int)numSlave.Value, (int)numBit.Value, out int value);
86     lblDi.Text = value != 0 ? "输入 ON" : "输入 OFF";
87     lblDi.BackColor = value != 0 ? Color.LimeGreen : Color.Gainsboro;
88 }
```

行	说明
83	还是先判断：没连接就什么也不做。
85	读一次输入位。这次把数字框的值直接写在参数里，效果和先存进变量一样。定时读取的返回码我们不处理，所以没有接收它。
86	条件运算符：? 前面是条件。value != 0 成立时取冒号前面的「输入 ON」，否则取冒号后面的「输入 OFF」，结果显示在标签上。
87	用同样的写法改背景色：有信号是绿色 LimeGreen，没信号是浅灰色 Gainsboro。这样不用点按钮，就能实时看到输入的变化。

到这里代码就全部写完了，按 Ctrl + S 保存。

第 8 章 编译运行

视频 19:38

8.1 编译 视频 19:39

- 1 按 **Ctrl** + **Shift** + **B** 生成解决方案。下面的输出窗口显示「1 成功, 0 失败」, 说明编译通过了。

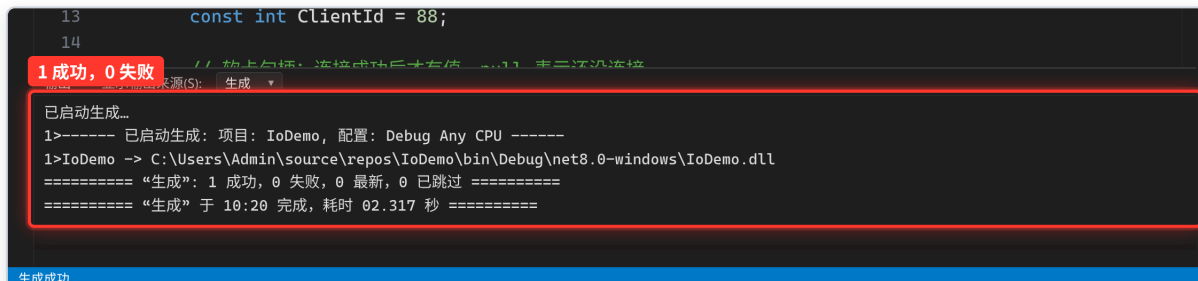


图 8-1 输出窗口: 1 成功, 0 失败

说明 如果报错, 先看「错误列表」窗口。最常见的两个原因: 代码里的控件名和设计器里起的 Name 不一致; 忘了添加 SDK 引用 (第 4 章)。

8.2 运行前确认主软件在运行 视频 20:01

确认主软件已经进了 OP, 或者切到了仿真。打开「软卡设置」页, SDK 服务的状态显示「运行中」, 我们的程序就能连上。



图 8-2 「软卡设置」页: 状态「运行中」

8.3 运行和验证 视频 20:14

- 1 回到 Visual Studio, 按 **F5** 启动程序。
- 2 点「连接」, 日志里出现「连接成功, 已取得控制权」。这时主软件那边的输出操作变成只读, 由我们的程序来控制。

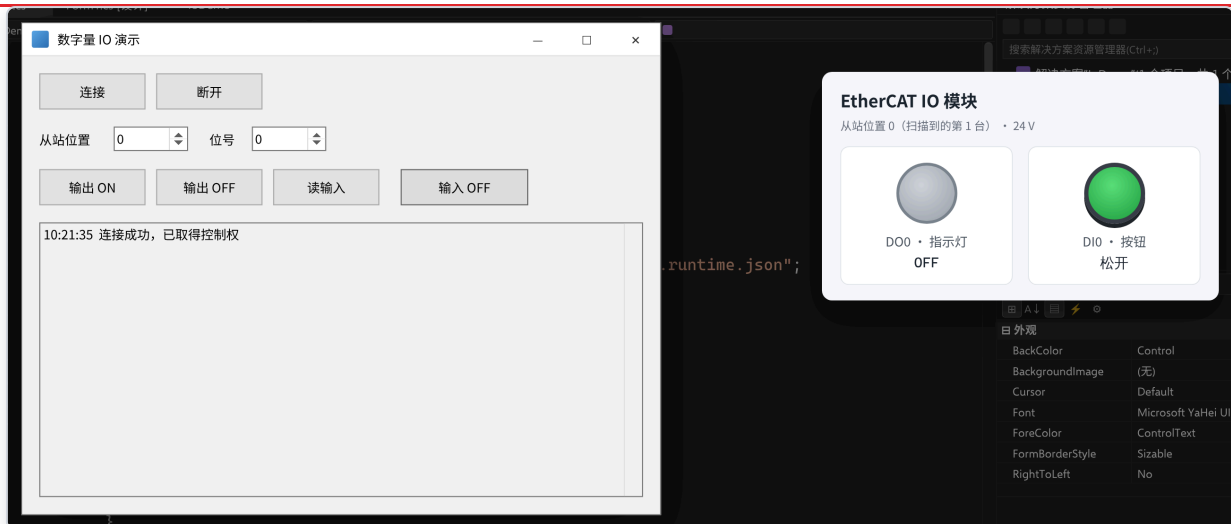


图 8-3 连接成功 (右边是 IO 模块的示意: DO0 接指示灯、DI0 接按钮)

- 3 从站位置和位号都是 0。点「输出 ON」, 返回码 0, 接在 DO0 上的指示灯亮了; 点「输出 OFF」, 灯灭。

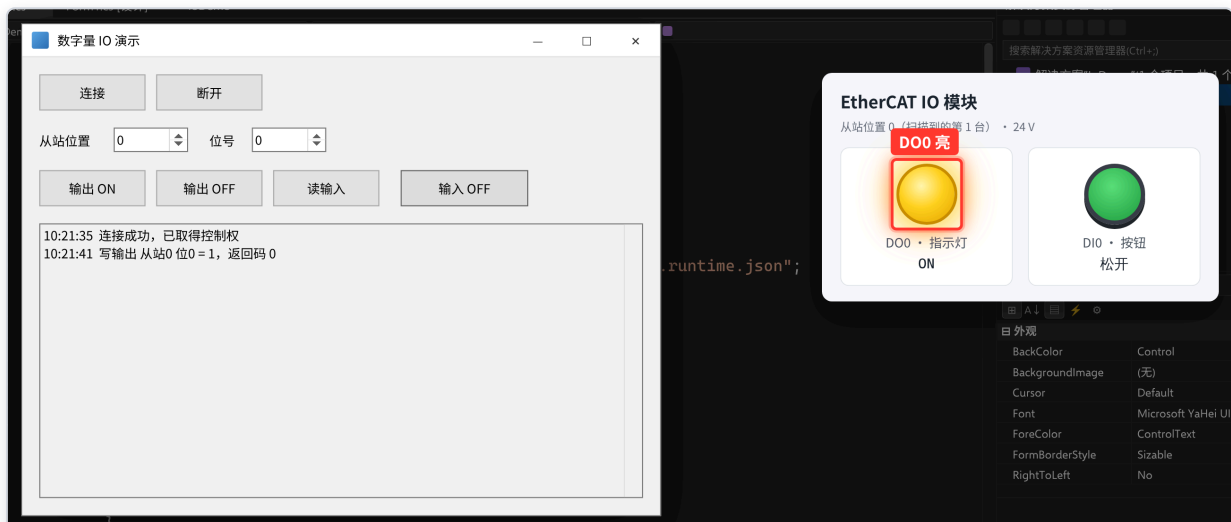


图 8-4 点「输出 ON」: DO0 的灯亮

- 4 按住接在 DI0 上的按钮, 右边的指示灯马上变绿; 这时点「读输入」, 日志里读到 1。松开按钮, 指示灯又变回灰色。
- 5 最后点「断开」, 控制权还给主软件。

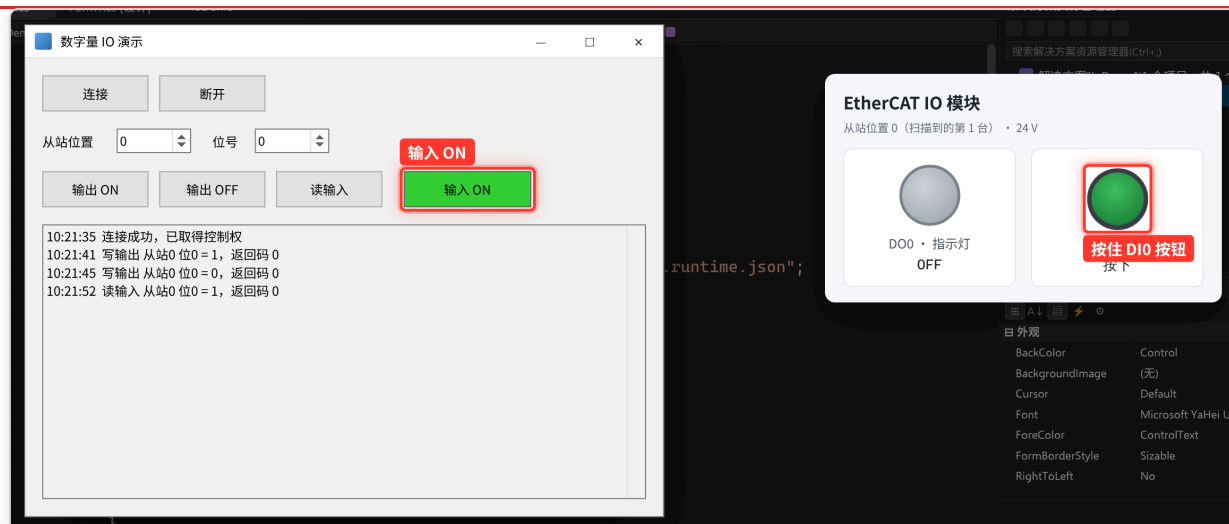


图 8-5 按住 DI0 上的按钮：指示灯变绿，「读输入」读到 1

说明 这里用的是一块同时带输入和输出的 IO 模块，它正好是扫描到的第一台从站，所以从站位置填 0。如果输入和输出在不同的模块上，就要分别填各自的从站位置。
在仿真模式下没有真实的输入信号，可以先用它练习连接和读写的流程。

第 9 章 常见问题

视频 21:24

出错时先看日志里的**返回码**，再对照下表。返回码的完整含义见附录 C。

现象	原因	怎么办
连接失败，返回码 10	连接配置文件不存在或读不了：多半是主软件没进 OP，也没切仿真	先让主软件进 OP，或者切到仿真
连接失败，返回码 5	主软件没运行、软卡 SDK 服务没起来，或者 SDK 和主软件版本不一致	看「软卡设置」页的状态；状态是「运行中」还连不上，就重新生成一次 SDK 包，换掉项目里的 DLL
取控制权失败，返回码 3	主软件停在「配置调试」，或者没进 OP	点「激活配置参数」进 OP，或者切到仿真
取控制权失败，返回码 7	控制权在别的程序手上；或者刚在 Visual Studio 里强行停止过程序，旧的控制权还没超时	先把另一个程序断开；强行停止过的，等 5 秒左右再连
写输出返回 7	控制权已经不在本程序手上	点「断开」，再点「连接」重新取得控制权
写输出返回 0，灯却不亮	这个点没有映射，或者从站位置、位号填错了	先在程序里点「断开」，再到主软件的「I/O 测试」页写同一个点核对
读输入一直是 0	没有信号，或者这个点没有映射（没映射也返回 0）	到主软件的「I/O 测试」页看同一个点有没有跳变
编译报错：找不到 ScMotionCard / SCM_...	没添加 SDK 引用，或者 using 没写	按第 4 章重新添加引用；检查文件开头的三行 using
编译报错：当前上下文中不存在名称 btnXxx / tmrPoll	设计器里控件的 Name 和代码里不一致	回设计器改 Name，必须和第 5 章的表一字不差
点按钮没有反应	事件函数是手敲的，没有和按钮挂上	删掉手敲的函数，回设计器双击按钮重新生成，再把代码放进去

附录 A 完整代码 Form1.cs

和配套工程 配套工程/IoDemo/Form1.cs 一字不差。左边是行号。

```
1 using System.Text;
2 using SCmotion.SoftCard.Sdk;
3 using static SCmotion.SoftCard.Sdk.ScMotionApi;
4
5 namespace IoDemo
6 {
7     public partial class Form1 : Form
8     {
9         // 连接配置文件: 主软件进 OP (或切换到仿真) 时自动生成, 路径固定
10         const string ConfigPath = @"C:\ProgramData\SoftSC\softcard.runtime.json";
11
12         // 本程序的身份号: 自己定一个, 多个程序同时连接时要各不相同
13         const int ClientId = 88;
14
15         // 软卡句柄: 连接成功后才有值, null 表示还没连接
16         ScMotionCard? card;
17
18         public Form1()
19         {
20             InitializeComponent();
21         }
22
23         // 「连接」按钮: 打开连接 → 取得控制权 → 启动定时刷新
24         private void btnConnect_Click(object sender, EventArgs e)
25         {
26             if (card != null) { Log("已经连接了"); return; }
27
28             int rc = SCM_OpenProjectConfig(ConfigPath, ClientId, out card);
29             if (rc != SCM_OK)
30             {
31                 var why = new StringBuilder(1024);
32                 SCM_GetOpenErrorText(why, why.Capacity);
33                 Log($"连接失败, 返回码 {rc}: {why}");
34                 card = null;
35                 return;
36             }
37
38             rc = SCM_RequestOwner(card);
39             if (rc != SCM_OK)
40             {
41                 Log($"取控制权失败, 返回码 {rc}");
42                 SCM_Close(card);
43                 card = null;
44                 return;
45             }
46
47             tmrPoll.Start();
48             Log("连接成功, 已取得控制权");
49         }
50
51         // 「断开」按钮
52         private void btnDisconnect_Click(object sender, EventArgs e)
53         {
54             Disconnect();
55         }
56
57         // 「输出 ON」按钮
```

```

58     private void btnOn_Click(object sender, EventArgs e)
59     {
60         WriteOutput(1);
61     }
62
63     // 「输出 OFF」按钮
64     private void btnOff_Click(object sender, EventArgs e)
65     {
66         WriteOutput(0);
67     }
68
69     // 「读输入」按钮
70     private void btnRead_Click(object sender, EventArgs e)
71     {
72         if (card == null) { Log("请先连接"); return; }
73
74         int slave = (int)numSlave.Value;
75         int bit = (int)numBit.Value;
76         int rc = SCM_ReadInputBit(card, slave, bit, out int value);
77         Log($"读输入 从站{slave} 位{bit} = {value}, 返回码 {rc}");
78     }
79
80     // 定时器: 每 200 毫秒读一次输入, 刷新右边的指示灯
81     private void tmrPoll_Tick(object sender, EventArgs e)
82     {
83         if (card == null) return;
84
85         SCM_ReadInputBit(card, (int)numSlave.Value, (int)numBit.Value, out int value);
86         lblDi.Text = value != 0 ? "输入 ON" : "输入 OFF";
87         lblDi.BackColor = value != 0 ? Color.LimeGreen : Color.Gainsboro;
88     }
89
90     // 关闭窗口前也断开, 把控制权还给主软件
91     private void Form1_FormClosing(object sender, FormClosingEventArgs e)
92     {
93         Disconnect();
94     }
95
96     // 写一个数字量输出位: value 为 1 接通, 为 0 断开
97     private void WriteOutput(int value)
98     {
99         if (card == null) { Log("请先连接"); return; }
100
101         int slave = (int)numSlave.Value;
102         int bit = (int)numBit.Value;
103         int rc = SCM_WriteOutputBit(card, slave, bit, value);
104         Log($"写输出 从站{slave} 位{bit} = {value}, 返回码 {rc}");
105     }
106
107     // 断开: 停定时器 → 归还控制权 → 关闭句柄
108     private void Disconnect()
109     {
110         if (card == null) return;
111
112         tmrPoll.Stop();
113         SCM_ReleaseOwner(card);
114         SCM_Close(card);
115         card = null;
116         Log("已断开");
117     }
118
119     // 在日志框末尾追加一行, 前面带上时间
120     private void Log(string text)
121     {

```

```
122         txtLog.AppendText($"{DateTime.Now:HH:mm:ss} {text}\r\n");  
123     }  
124 }  
125 }
```

附录 B 本教程用到的 SDK 指令

所有指令都返回 `int` 返回码，`SCM_OK`（0）表示成功。下面的签名是 C# 写法（文件开头写了 `using static ScMotionApi`，所以可以省掉前缀直接调用）。

SCM_OpenProjectConfig

```
int SCM_OpenProjectConfig(string configPath, int clientId, out ScMotionCard handle)
```

参数	方向	说明
<code>configPath</code>	入	连接配置文件路径，通常是 C:\ProgramData\SoftSC\softcard.runtime.json
<code>clientId</code>	入	本程序的身份号。多个程序同时连接时必须各不相同；不要照抄配置文件里的 <code>clientId</code>
<code>handle</code>	出	成功时给回句柄；失败时为空

返回：SCM_OK = 已连接；10 = 配置文件不存在、读不通或内容非法；5 = 主软件没起，或运行时不可用

不需要控制权，连上以后默认只有读的权限。

SCM_GetOpenErrorText

```
int SCM_GetOpenErrorText(StringBuilder buffer, int bufferSize)
```

参数	方向	说明
<code>buffer</code>	出	接收失败原因的文字
<code>bufferSize</code>	入	缓冲区大小，一般传 <code>buffer.Capacity</code>

返回：SCM_OK = 取到

读最近一次连接失败的详细原因，不需要句柄。参数对不上时会逐项列出双方的轴数、I/O 点数和周期。

SCM_RequestOwner

```
int SCM_RequestOwner(ScMotionCard handle)
```

参数	方向	说明
<code>handle</code>	入	连接得到的句柄

返回：SCM_OK = 已取得（同时已自动武装）；7 = 控制权在别人手上；3 = 主软件没进 OP 或当前模式不允许下发；5 = 主软件不可用

取得后主软件界面上的输出和运动操作变成只读。程序崩溃或没归还就退出时，主软件大约 5 秒后自动收回。

SCM_ReleaseOwner

```
int SCM_ReleaseOwner(ScMotionCard handle)
```

参数	方向	说明
handle	入	连接得到的句柄

返回：SCM_OK = 已归还；7 = 你不是当前的控制权持有者

归还后主软件立即恢复控制。程序退出前应显式调用。

SCM_Close

```
int SCM_Close(ScMotionCard handle)
```

参数	方向	说明
handle	入	连接得到的句柄

返回：SCM_OK（已经关过的句柄再关一次也返回 SCM_OK）

关闭前如果还持有控制权，先调用 SCM_ReleaseOwner。关闭以后再调用读写指令不会崩溃，但会返回 1（卡未打开）。

SCM_WriteOutputBit

```
int SCM_WriteOutputBit(ScMotionCard handle, int slavePosition, int bit, int value)
```

参数	方向	说明
slavePosition	入	从站拓扑位置，从 0 开始（等于扫描顺序）
bit	入	这台从站上的输出位号，从 0 开始
value	入	非 0 接通，0 断开

返回：SCM_OK = 已下发（这个点没映射时同样返回 SCM_OK，只是什么都没写）；7 = 没有控制权；5 = 主软件不可用
需要控制权。地址不受其他从站增删 IO 点的影响，是接线固定后最稳的写法。

SCM_ReadInputBit

```
int SCM_ReadInputBit(ScMotionCard handle, int slavePosition, int bit, out int value)
```

参数	方向	说明
slavePosition	入	从站拓扑位置，从 0 开始
bit	入	这台从站上的输入位号，从 0 开始
value	出	0 或 1；没映射或失败时给 0

返回：SCM_OK = 正常（包括「这个点没映射」）；1 = 句柄已关闭；5 = 主软件不可用

只读，不需要控制权。判断时用 value != 0。读到 0 且返回 SCM_OK 有两种可能：真的没信号，或者这个点没配置。

附录 C

返回码速查

值	常量	含义	本教程里什么时候会遇到
0	SCM_OK	成功	一切正常
1	SCM_NOT_OPEN	卡未打开	句柄已经关闭后还在读写
2	SCM_ALREADY_OPEN	卡已打开	—
3	SCM_INVALID_STATE	当前状态下该操作无效	取控制权时主软件没进 OP，或当前模式不允许下发
4	SCM_INVALID_AXIS	轴号非法	—（教程02 会用到轴）
5	SCM_BACKEND_UNAVAILABLE	运行时后端不可用（主软件没起 / 断连）	连接时主软件没运行、SDK 服务没起来，或版本不一致
6	SCM_TIMEOUT	超时	—
7	SCM_REJECTED	被拒绝	控制权在别的程序手上；写输出时没有控制权
8	SCM_FAULTED	轴或运行时处于故障态	—
9	SCM_NOT_SUPPORTED	该运行时不支持此操作	—
10	SCM_INVALID_ARGUMENT	参数非法	连接时配置文件不存在或读不了

来源：《SCMotion 4.0 软卡 SDK 编程手册》5.18.8「错误码 SCM_RESULT」。被拒绝（7）时，可以再调用 `SCM_GetLastReason` 拿到更细的原因。