SCATECH 软控

CASE TUTORIAL 01 · 编程篇 · WPF 版（图文版）

案例教程 01　数字量 IO 控制

在 Visual Studio 2022 里从新建项目开始，
用 XAML 画界面、用 P/Invoke 调用 C 动态库，
写出一个读输入、写输出的 WPF 程序。

Visual Studio 2022 | C# WPF | .NET 8 | P/Invoke

适用软件	SCMotion 4.4.17
配套视频	教程01_数字量IO控制_编程篇_WPF版.mp4
配套工程	配套工程/IoDemoWpf
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、第一次写 WPF 程序的人。会装软件、会用 Windows 就行：每一行代码都在这里讲清楚。
做出什么	一个 WPF 窗口程序：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。界面、按钮、日志的文字和 WinForms 版一样。
和 WinForms 版的区别	① 界面用 **XAML** 写；② SDK 不引用 .NET 库，而是用 **P/Invoke** 直接调用 C 语言的动态库 `SCmotionSoftCard.dll`。
开发环境	Visual Studio 2022 · .NET 8 · C# WPF 应用程序
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_WPF版.mp4**。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/IoDemoWpf**：代码和本文附录 A 一字不差，打开 `IoDemoWpf.sln` 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、进 OP，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、怎么敲」，这份文档查「点哪里、写什么、为什么」。
- 截图里的**红框**就是要点、要看的地方；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**文件里的行号**，和附录 A 的完整代码、配套工程里的文件完全对得上。
- 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 SCMotion 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。**没有硬件也能学**：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 00:28 5

第 2 章 准备工作 视频 00:55 6

2.1 Visual Studio 2022 6

2.2 生成软卡 SDK 包 6

2.3 让主软件处于运行状态 7

第 3 章 新建项目 视频 02:15 8

第 4 章 加入 SDK 视频 03:29 10

4.1 把 dll 放进项目文件夹 10

4.2 复制到输出目录：如果较新则复制 10

4.3 平台目标改成 x64 11

第 5 章 声明 SDK 函数 视频 05:24 12

5.1 新建 ScmSdk.cs 12

5.2 文件开头和类 12

5.3 C 的类型，在 C# 里怎么写 13

5.4 七个函数的声明 13

5.5 两个转换字符串的小函数 14

第 6 章 画界面（XAML） 视频 09:55 15

6.1 窗口和共用的按钮样式 15

6.2 用 StackPanel 一排一排摆 15

6.3 第三排和日志框 16

6.4 窗口的 Closing 事件 16

第 7 章 写代码 视频 14:03 18

7.1 两行 using 和四个成员 18

7.2 工具函数 Log 19

7.3 「连接」按钮 19

7.4 断开：Disconnect 20

7.5 写输出：ReadPoint 和 WriteOutput 20

7.6 读输入：「读输入」按钮 22

7.7 定时刷新：Timer_Tick 22

第 8 章 编译运行 视频 21:41 23

8.1 编译 23

8.2 运行和验证 23

第 9 章 常见问题 视频 22:51 25

附录 A 完整代码 26

MainWindow.xaml 26
ScmSdk.cs 26
MainWindow.xaml.cs 27
附录 B 本教程用到的 SDK 指令 31
附录 C 返回码速查 33

第 1 章 最终效果

视频 00:28

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	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，出问题先看它

第 2 章 准备工作

视频 00:55

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

安装 Visual Studio 2022 时，在「工作负荷」里勾上 **「.NET 桌面开发」**，WPF 的项目模板就在里面。

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」。

2.2 生成软卡 SDK 包

1. 在主软件左侧工程树的 **「SYSTEM - 配置」** 下面，点 **「软卡设置」**（图中 ①）。新版本的主软件里，这一页改名叫 **「SDK包」**。
2. 点 **「生成SDK包」**（图中 ②），选一个输出目录。

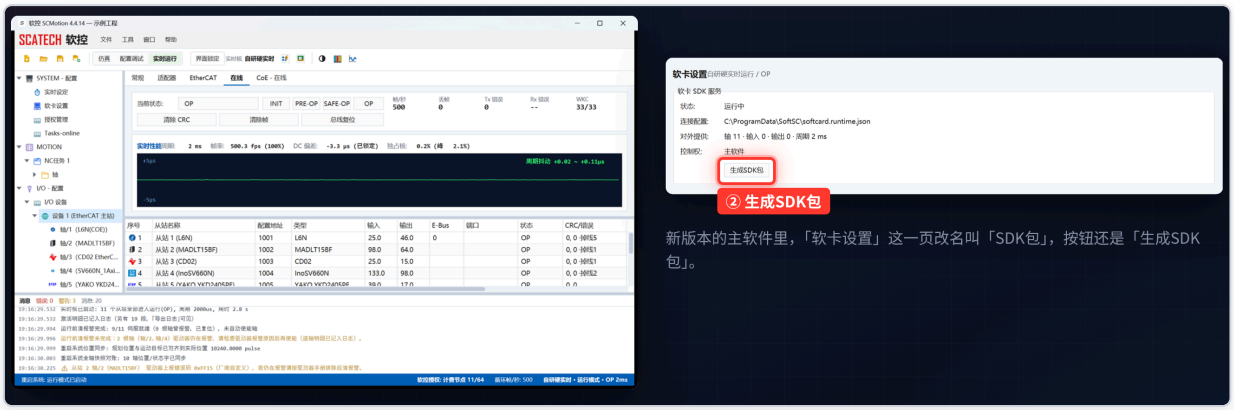

图 2-2 主软件里生成 SDK 包：① 软卡设置　② 生成SDK包

生成的 SDK 包里，**WPF 版只用其中一个文件**：`sdk\c\win-x64\SCmotionSoftCard.dll`，这是 C 语言写的 64 位动态库。旁边 `include` 文件夹里的头文件 `scmotion_card.h` 列出了每个函数的原型，第 5 章声明函数时照着它写。

```
SCmotionSoftCardSdk_工程名_时间\
├─ sdk\
│   ├─ dotnet\    （WinForms 版用）
│   └─ c\
│       ├─ include\scmotion_card.h     ← C 头文件，函数原型在这里
│       └─ win-x64\
│           ├─ SCmotionSoftCard.dll   ← WPF 版用这一个
│           └─ SCmotionSoftCard.lib
├─ docs\软卡SDK指令说明.html
└─ README.txt
```

注意 **主软件升级以后，要重新生成一次 SDK 包**，用新的 `SCmotionSoftCard.dll` 换掉项目里的旧文件。SDK 必须和主软件是同一个版本，混用时可能连不上，或者调用结果不对。

2.3 让主软件处于运行状态

程序连接的是正在运行的主软件：真机点工具栏上的「激活配置参数」进 OP；没有硬件就点模式条上的「仿真」。主软件这边怎么操作，见《上篇 主软件操作》。进了 OP（或仿真）以后，主软件会自动写出连接配置文件 `C:\ProgramData\SoftSC\softcard.runtime.json`，程序就是读它来连接的。

第 3 章 新建项目

视频 02:15

1. 打开 Visual Studio 2022，启动窗口右边点 **「创建新项目」**。
2. 在搜索框里输入 `WPF`，选 **「WPF 应用程序」**，语言标签是 C#。**不要选**名字后面带「.NET Framework」的那个，那是老框架。选中后点 **「下一步」**。

图 3-1 选 C# 的「WPF 应用程序」，不要选带「.NET Framework」的

3. 项目名称填 `IoDemoWpf`，勾上 **「将解决方案和项目放在同一目录中」**，点 **「下一步」**。
4. 框架选 **「.NET 8.0 (长期支持)」**，点 **「创建」**。

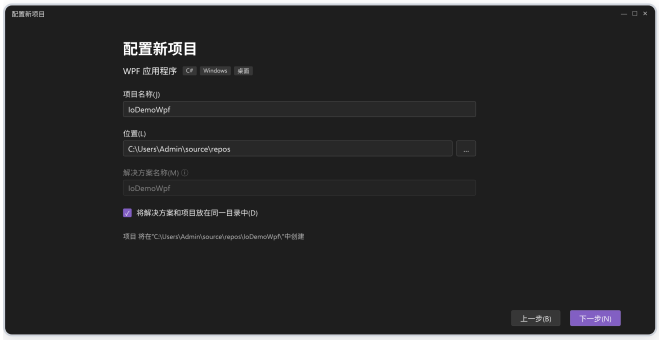

图 3-2 项目名称 IoDemoWpf

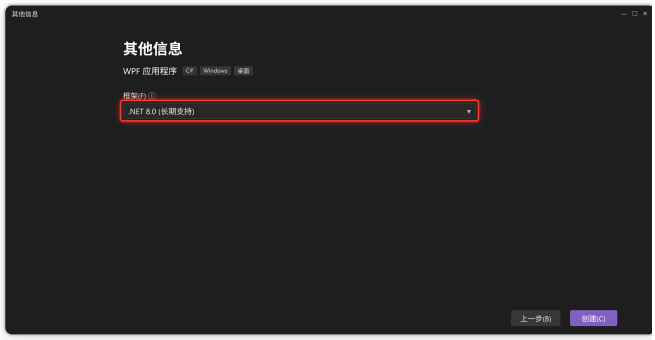

图 3-3 框架 .NET 8.0

项目建好以后，中间上半部分是**设计器**，显示窗口的样子；下半部分是 **XAML**，窗口里有什么控件、怎么排，就写在这里。两部分是同步的：改了 XAML，上面的样子马上跟着变。

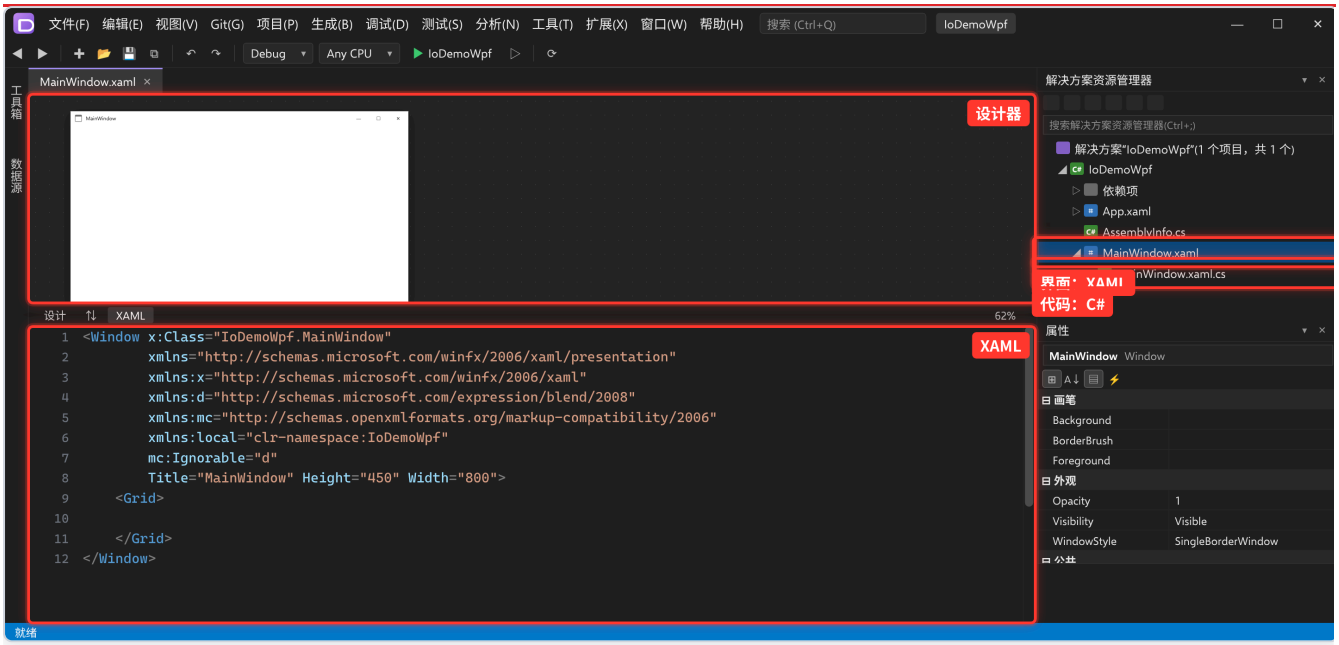

图 3-4　新建好的项目：设计器 + XAML；右上角 MainWindow.xaml 是界面，MainWindow.xaml.cs 是代码

第 4 章 加入 SDK

视频 03:29

4.1 把 dll 放进项目文件夹

1. 打开 SDK 包的 `sdk\c\win-x64` 文件夹，选中 `SCmotionSoftCard.dll`，按 `Ctrl+C` 复制。
2. 打开项目文件夹 `IoDemoWpf`（和 `IoDemoWpf.csproj` 在一起的那一层），按 `Ctrl+V` 粘贴。

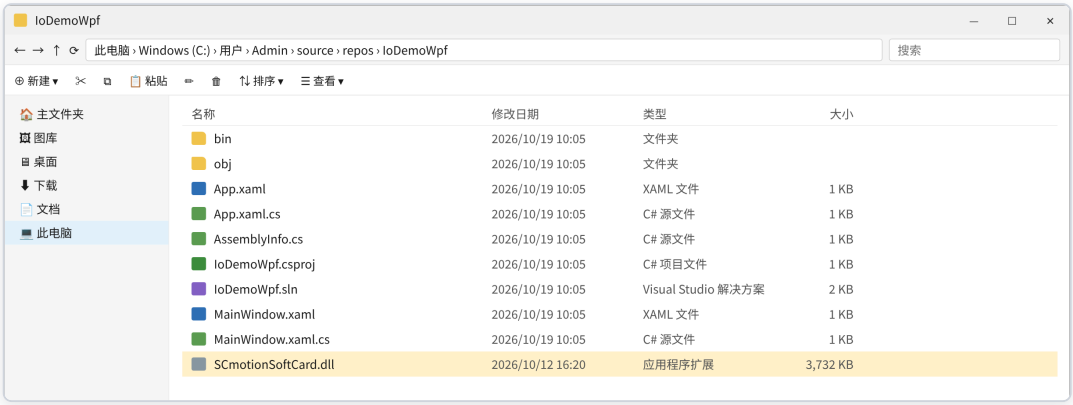

图 4-1 dll 和 .csproj 放在同一个文件夹

dll 放在项目文件夹里，整个项目拷到别的电脑上，也能直接编译运行。

4.2 复制到输出目录：如果较新则复制

回到 Visual Studio，解决方案资源管理器里自动多出了 `SCmotionSoftCard.dll`。它不是 C# 代码，编译时不会被自动带到程序旁边，要改一个属性：

1. 在解决方案资源管理器里选中 `SCmotionSoftCard.dll`。
2. 在属性窗口找到 **「复制到输出目录」**，从「不复制」改成 **「如果较新则复制」**。

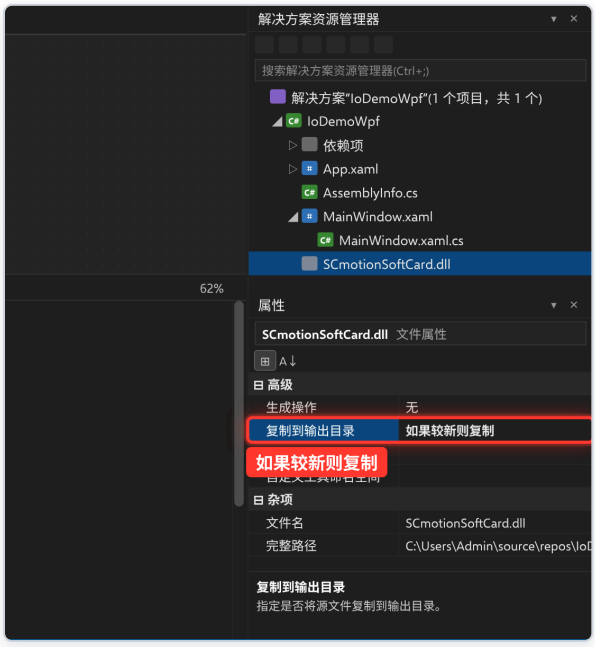

图 4-2 选中 dll，「复制到输出目录」改成「如果较新则复制」

这样每次生成，它都会被拷到 exe 所在的文件夹，程序运行时才找得到它。这一步忘了做，程序一运行到调用 SDK 的地方，就会报 `DllNotFoundException`（找不到 SCmotionSoftCard.dll）。

4.3 平台目标改成 x64

这个 dll 是 64 位的，程序也必须是 64 位：

1. 右键项目 `IoDemoWpf`，选 **「属性」**。
2. 在 **「生成」** → **「常规」** 里，把 **「平台目标」** 从 `Any CPU` 改成 `x64`，按 `Ctrl+S` 保存。

图 4-3 「平台目标」改成 x64

不改的话，程序一调用 SDK 就会报 `BadImageFormatException`（试图加载格式不正确的程序），也就是 64 位的 dll 装不进 32 位的进程。

第 5 章 声明 SDK 函数

视频 05:24

C 语言的函数，C# 不能直接调用，要先**声明**一遍：告诉 C#「这个函数在哪个 dll 里、叫什么、参数是什么类型」。我们把声明集中放在一个新文件 `ScmSdk.cs` 里。

5.1 新建 ScmSdk.cs

1. 右键项目 → **「添加」** → **「类」**。
2. 名称改成 `ScmSdk.cs`，点 **「添加」**。

图 5-1　右键项目 → 添加 → 类

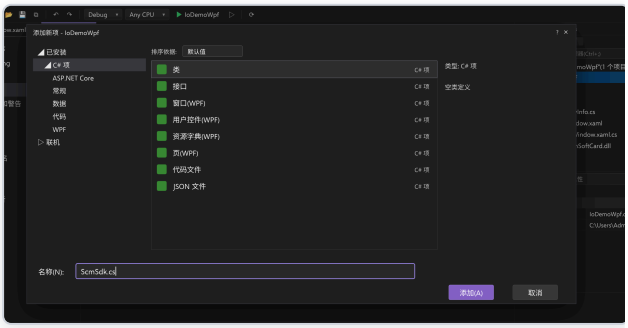

图 5-2　名称 ScmSdk.cs

5.2 文件开头和类

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;

namespace IoDemoWpf
{
    // 软卡 SDK 的函数声明：用 DllImport 把 SCmotionSoftCard.dll 里的 C 函数接到 C# 里来
    internal static class ScmSdk
    {
        // C 动态库的文件名，运行时它要和 exe 在同一个文件夹
        const string Dll = "SCmotionSoftCard.dll";

        // 返回码 0 表示成功
        public const int SCM_OK = 0;
```

行	写了什么	为什么
4	`using System.Runtime.InteropServices;`	声明 C 函数要用的 `DllImport` 在这个命名空间里
10–11	`internal static class ScmSdk`	`static` 表示静态类：只用来放函数，不需要先 new 一个对象
14	`const string Dll = "SCmotionSoftCard.dll";`	动态库的文件名。每个声明都要写它，定义成常量就不用重复写
17	`public const int SCM_OK = 0;`	所有 SDK 函数都返回整数返回码，0 表示成功

5.3 C 的类型，在 C# 里怎么写

照着头文件 `scmotion_card.h` 写声明，参数类型要换成 C# 里对应的写法：

C 头文件里	C# 里写成	为什么
int（返回码、普通整数参数）	`int`	一样，都是 32 位整数
const char* configPath（字符串）	`byte[] path`	传 UTF-8 字节，结尾补一个 0；中文路径不会乱码
SCM_CARD_HANDLE handle（句柄）	`IntPtr handle`	句柄是指针，64 位下 8 字节，**不能用 int**
SCM_CARD_HANDLE* handle（出参）	`out IntPtr handle`	C 函数把句柄写回来
int* value（出参）	`out int value`	C 函数把读到的值写回来
char* buffer, int bufferSize	`byte[] buffer, int bufferSize`	C 函数往缓冲区里写字

5.4 七个函数的声明

```
19 [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
20 public static extern int SCM_OpenProjectConfig(byte[] path, int clientId,
21     out IntPtr handle);
22
23 [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
24 public static extern int SCM_GetOpenErrorText(byte[] buffer, int bufferSize);
25
26 [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
27 public static extern int SCM_RequestOwner(IntPtr handle);
28
29 [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
30 public static extern int SCM_ReleaseOwner(IntPtr handle);
31
32 [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
33 public static extern int SCM_Close(IntPtr handle);
34
35 [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
36 public static extern int SCM_WriteOutputBit(IntPtr handle, int slave, int bit, int value);
37
38 [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
```

```
39 public static extern int SCM_ReadInputBit(IntPtr handle, int slave, int bit,
40     out int value);
```

每个声明两行：

- 方括号里的 `DllImport(Dll, CallingConvention = CallingConvention.Cdecl)`：这个函数在 `Dll` 这个文件里，按 C 语言的调用约定 Cdecl 调用。
- 下一行 `public static extern int 函数名(参数);`：`public` 别的类能用，`static` 静态，`extern` 表示函数体不在这里、在外部的 dll 里，返回值 `int` 就是返回码。
- 函数名、参数顺序必须和头文件里一模一样。只声明程序用得到的函数，以后用到别的再照着头文件加。

5.5 两个转换字符串的小函数

```
42 // 字符串转成以 0 结尾的 UTF-8 字节，传给 C 函数
43 public static byte[] U8(string s)
44 {
45     return Encoding.UTF8.GetBytes(s + "\0");
46 }
47
48 // C 函数写回来的 UTF-8 字节转成字符串，到第一个 0 为止
49 public static string FromU8(byte[] buf)
50 {
51     int n = Array.IndexOf(buf, (byte)0);
52     return Encoding.UTF8.GetString(buf, 0, n < 0 ? buf.Length : n);
53 }
```

- `U8`：把 C# 的字符串转成 UTF-8 字节，结尾补一个 0，C 语言就靠这个 0 知道字符串到哪里结束。
- `FromU8`：反过来，C 函数写回来的字节，找到第一个 0，把它前面的部分转成字符串；没找到 0 就整段转换。

第 6 章 画界面（XAML）

视频 09:55

切到 `MainWindow.xaml`。最上面一段是窗口本身的设置，前几行是固定的命名空间，不用动。完整的 XAML 只有 43 行：

6.1 窗口和共用的按钮样式

```
    Title="数字量 IO 演示（WPF）" Height="420" Width="560" Closing="Window_Closing">
<Window.Resources>
    <!-- 本窗口里的按钮统一：宽 90、高 30，右边空 10 -->
    <Style TargetType="Button">
        <Setter Property="Width" Value="90"/>
        <Setter Property="Height" Value="30"/>
        <Setter Property="Margin" Value="0,0,10,0"/>
    </Style>
</Window.Resources>
```

- 第 8 行：标题 `Title` 改成「数字量 IO 演示（WPF）」，高 `Height` 420、宽 `Width` 560（像素）；最后的 `Closing` 在窗口关闭之前触发，第 6.4 节再说。
- 模板自带的 `<Grid>` 用不到，删掉。
- `Window.Resources` 里的 `Style TargetType="Button"`：这个窗口里所有的按钮都套用它——宽 90、高 30，`Margin` 外边距 0,0,10,0（左、上、右、下），也就是每个按钮右边空出 10。

> **提示** 敲完右尖括号，Visual Studio 会自动补上结束标签；敲等号时引号也会成对出现，接着在中间写就行。

6.2 用 StackPanel 一排一排摆

```
<StackPanel Margin="12">
    <!-- 第一排：连接、断开 -->
    <StackPanel Orientation="Horizontal" Margin="0,0,0,10">
        <Button x:Name="btnConnect" Content="连接" Click="btnConnect_Click"/>
        <Button x:Name="btnDisconnect" Content="断开" Click="btnDisconnect_Click"/>
    </StackPanel>
    <!-- 第二排：从站位置、位号 -->
    <StackPanel Orientation="Horizontal" Margin="0,0,0,10">
        <TextBlock Text="从站位置" VerticalAlignment="Center"/>
        <TextBox x:Name="txtSlave" Text="0" Width="60" Margin="8,0,20,0"/>
        <TextBlock Text="位号" VerticalAlignment="Center"/>
        <TextBox x:Name="txtBit" Text="0" Width="60" Margin="8,0,0,0"/>
    </StackPanel>
```

- `StackPanel` 把里面的东西一个接一个排起来，默认从上往下；`Margin="12"` 四周留 12 像素。
- 每一排又是一个 `Orientation="Horizontal"` 的 StackPanel：里面的东西横着排，`Margin="0,0,0,10"` 让这一排下边空出 10。
- `x:Name` 是控件在代码里的名字，`Content` 是按钮上显示的文字。

- 敲 `Click=` 时 Visual Studio 弹出「新建事件处理程序」，回车选中，处理函数名自动填好，同时在 MainWindow.xaml.cs 里生成一个空函数。
- `TextBlock` 显示一段文字；`TextBox` 是输入框，`Text="0"` 是默认内容。`Margin="8,0,20,0"`：左边空 8 和文字隔开，右边空 20 和后面的「位号」隔开。

6.3 第三排和日志框

```
30          <!-- 第三排：输出 ON、输出 OFF、读输入、输入指示 -->
31          <StackPanel Orientation="Horizontal" Margin="0,0,0,10">
32              <Button x:Name="btnOn" Content="输出 ON" Click="btnOn_Click"/>
33              <Button x:Name="btnOff" Content="输出 OFF" Click="btnOff_Click"/>
34              <Button x:Name="btnRead" Content="读输入" Click="btnRead_Click"/>
35              <Label x:Name="lblDi" Content="输入 --" Width="110" Margin="10,0,0,0"
36                     Background="Gainsboro" BorderBrush="Gray" BorderThickness="1"
37                     HorizontalContentAlignment="Center" VerticalContentAlignment="Center"/>
38          </StackPanel>
39          <!-- 日志框 -->
40          <TextBox x:Name="txtLog" Height="230" IsReadOnly="True" TextWrapping="Wrap"
41                   VerticalScrollBarVisibility="Auto"/>
42      </StackPanel>
43  </Window>
```

- 三个按钮同样挂上 `Click` 事件。
- `Label` 当输入指示灯：`Background="Gainsboro"` 浅灰色背景，`BorderBrush`、`BorderThickness` 加一圈 1 像素的灰边框，最后两项让文字左右、上下都居中。
- 日志框也是 `TextBox`：`IsReadOnly` 只读，`TextWrapping` 长行自动换行，内容多了出现竖向滚动条。

6.4 窗口的 Closing 事件

回到第 8 行 `Window` 那一行，在末尾加上 `Closing=`，同样选「新建事件处理程序」，生成 `Window_Closing`。直接点右上角的叉关掉程序时，它会被调用，我们在里面断开连接。

界面写完了，按 `Ctrl+S` 保存。设计器里的样子：

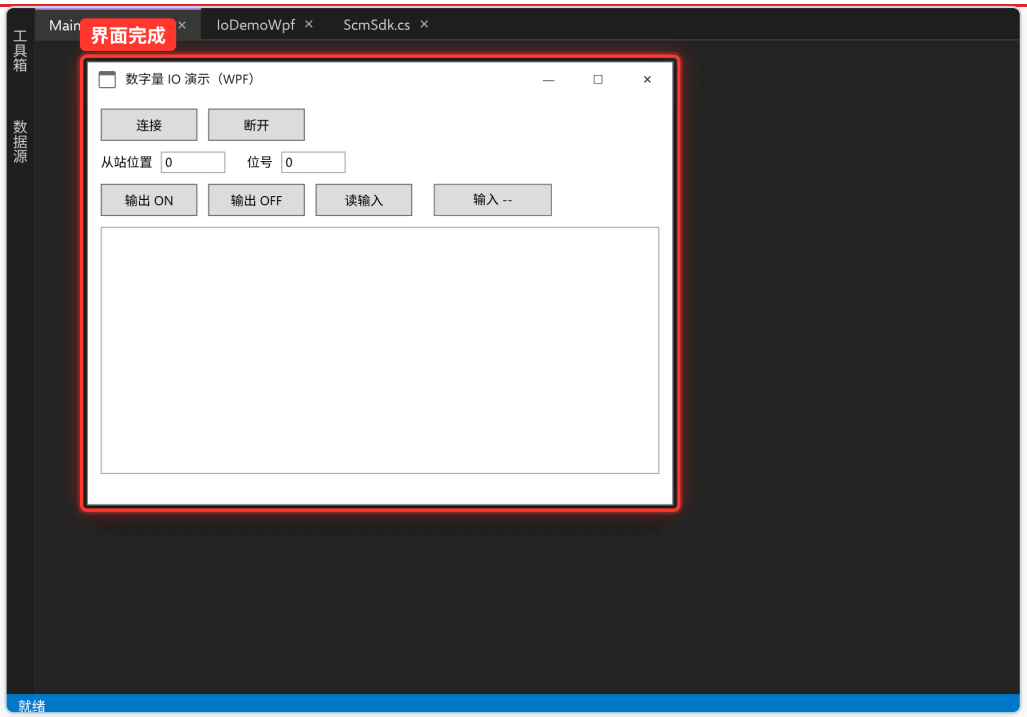

图 6-1　写好的界面（设计器）

第 7 章 写代码

视频 14:03

打开 `MainWindow.xaml.cs`。刚才挂的六个事件，Visual Studio 已经生成了六个空函数。下面按视频的顺序，一段一段写。

7.1 两行 using 和四个成员 视频 14:03

```
11 using System.Windows.Threading;
12 using static IoDemoWpf.ScmSdk;
```

- `System.Windows.Threading`：定时器 `DispatcherTimer` 在这里。
- `using static IoDemoWpf.ScmSdk;`：写了它，`SCM_` 开头的函数、`SCM_OK` 和 `U8` 可以直接用，不用每次写 `ScmSdk.`。

```
21 // 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
22 const string ConfigPath = @"C:\ProgramData\SoftSC\softcard.runtime.json";
23
24 // 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
25 const int ClientId = 88;
26
27 // 软卡句柄：连接成功后才有值，IntPtr.Zero 表示还没连接
28 IntPtr card = IntPtr.Zero;
29
30 // 定时器：连接以后每 200 毫秒读一次输入
31 DispatcherTimer timer = new DispatcherTimer();
32
33 public MainWindow()
34 {
35     InitializeComponent();
36     timer.Interval = TimeSpan.FromMilliseconds(200);
37     timer.Tick += Timer_Tick;
38 }
```

行	成员	说明
22	`ConfigPath`	连接配置文件的路径。`const` 是常量；字符串前面的 `@` 让反斜杠按原样使用
25	`ClientId`	本程序的身份号，自己定一个数。几个程序同时连主软件时，各自的身份号要不一样
28	`card`	软卡句柄，`IntPtr`。初值 `IntPtr.Zero` 表示还没连接，后面靠它是不是 0 来判断有没有连接
31	`timer`	WPF 没有可以拖的 Timer 控件，在代码里 new 一个 `DispatcherTimer`。它的 Tick 在界面线程里执行，可以直接改界面上的控件
36–37	构造函数	间隔设成 200 毫秒；`+=` 把 Tick 事件挂到 `Timer_Tick`（7.7 节写）

7.2 工具函数 Log 视频 15:52

```
147 // 在日志框末尾加一行，前面带上时间
148 private void Log(string text)
149 {
150     txtLog.AppendText($"{DateTime.Now:HH:mm:ss}  {text}\r\n");
151     txtLog.ScrollToEnd();
152 }
```

`AppendText` 在日志框末尾追加文字。`$` 表示字符串插值：花括号里的当前时间按「时:分:秒」格式化，再加上内容和回车换行 `\r\n`。WPF 的文本框追加以后不会自动滚到底，所以再调用 `ScrollToEnd`。

7.3 「连接」按钮 视频 16:28

```
40 // 「连接」按钮：打开连接 → 取得控制权 → 启动定时器
41 private void btnConnect_Click(object sender, RoutedEventArgs e)
42 {
43     if (card != IntPtr.Zero) { Log("已经连接了"); return; }
44
45     int rc = SCM_OpenProjectConfig(U8(ConfigPath), ClientId, out card);
46     if (rc != SCM_OK)
47     {
48         byte[] why = new byte[1024];
49         SCM_GetOpenErrorText(why, why.Length);
50         Log($"连接失败，返回码 {rc}：{FromU8(why)}");
51         card = IntPtr.Zero;
52         return;
53     }
54
55     rc = SCM_RequestOwner(card);
56     if (rc != SCM_OK)
57     {
58         Log($"取控制权失败，返回码 {rc}");
59         SCM_Close(card);
60         card = IntPtr.Zero;
61         return;
62     }
63
64     timer.Start();
65     Log("连接成功，已取得控制权");
66 }
```

行	做什么
43	防重复：card 不是 0，说明已经连上了，写一条日志就返回
45	`SCM_OpenProjectConfig` 打开连接：路径先用 `U8` 转成 UTF-8 字节；第二个参数是身份号；`out card` 接收句柄。返回码存进 `rc`，0 表示成功
46–53	连接失败：准备 1024 字节的数组，用 `SCM_GetOpenErrorText` 取回失败原因，`FromU8` 转成文字，和返回码一起写进日志；card 设回 0，返回
55	`SCM_RequestOwner` 取得控制权：刚连上只能读，拿到控制权才能写输出。同一时刻只有一个程序能拿到控制权
56–62	取不到：最常见返回 3（主软件没进 OP）或 7（控制权在别的程序手上）。关掉句柄，返回
64–65	两步都成功：启动定时器，写日志

7.4 断开：Disconnect 视频 18:12

```
135 // 断开：停定时器 → 归还控制权 → 关闭句柄
136 private void Disconnect()
137 {
138     if (card == IntPtr.Zero) return;
139
140     timer.Stop();
141     SCM_ReleaseOwner(card);
142     SCM_Close(card);
143     card = IntPtr.Zero;
144     Log("已断开");
145 }
```

停定时器 → `SCM_ReleaseOwner` 归还控制权 → `SCM_Close` 关闭句柄 → card 设回 0。如果不还控制权就直接退出，主软件要等大约 5 秒，才会判定程序失联、把控制权收回去。

「断开」按钮和窗口的 Closing 事件里，都调用它：

```
68 // 「断开」按钮
69 private void btnDisconnect_Click(object sender, RoutedEventArgs e)
70 {
71     Disconnect();
72 }
```

```
96 // 关窗口前也断开，把控制权还给主软件
97 private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
98 {
99     Disconnect();
100 }
```

7.5 写输出：ReadPoint 和 WriteOutput 视频 18:57

WPF 的输入框里是文字，要先转成整数。公用函数 `ReadPoint` 从两个输入框读出从站位置和位号：

```
// 从两个输入框读出从站位置和位号；填的不是整数就提示，返回 false
private bool ReadPoint(out int slave, out int bit)
{
    bit = 0;
    if (!int.TryParse(txtSlave.Text, out slave) || !int.TryParse(txtBit.Text, out bit))
    {
        Log("从站位置和位号要填整数");
        return false;
    }
    return true;
}
```

- `int.TryParse` 尝试把文字转成整数，成功返回 true，结果从 out 参数带出来；两个里只要有一个失败，就提示并返回 false。
- 开头的 `bit = 0;`：out 参数在返回之前必须有值。

```
// 写一个数字量输出位：value 为 1 接通，为 0 断开
private void WriteOutput(int value)
{
    if (card == IntPtr.Zero) { Log("请先连接"); return; }
    if (!ReadPoint(out int slave, out int bit)) return;

    int rc = SCM_WriteOutputBit(card, slave, bit, value);
    Log($"写输出 从站{slave} 位{bit} = {value}，返回码 {rc}");
}
```

`SCM_WriteOutputBit` 写一个输出位，四个参数：句柄、从站位置、位号、值。从站位置是扫描到的顺序，从 0 开始；位号是这台从站自己的第几个点，也从 0 开始。

> **注意** 没映射的点，写它也会返回 0，只是什么都不会发生。第一次接线一定到现场看灯有没有亮。

「输出 ON」「输出 OFF」分别传 1 和 0：

```
// 「输出 ON」按钮
private void btnOn_Click(object sender, RoutedEventArgs e)
{
    WriteOutput(1);
}

// 「输出 OFF」按钮
private void btnOff_Click(object sender, RoutedEventArgs e)
{
    WriteOutput(0);
}
```

7.6 读输入：「读输入」按钮 视频 20:17

```
86  // 「读输入」按钮
87  private void btnRead_Click(object sender, RoutedEventArgs e)
88  {
89      if (card == IntPtr.Zero) { Log("请先连接"); return; }
90      if (!ReadPoint(out int slave, out int bit)) return;
91
92      int rc = SCM_ReadInputBit(card, slave, bit, out int value);
93      Log($"读输入 从站{slave} 位{bit} = {value}，返回码 {rc}");
94  }
```

`SCM_ReadInputBit` 读一个输入位，最后一个参数 `out int value` 带回读到的值。读输入不需要控制权，连上就能读。没映射的点读回来也是 0，返回码也是 0：读到 0 不一定是没有信号，也可能是这个点没配置。

7.7 定时刷新：Timer_Tick 视频 20:48

```
102 // 定时器：每 200 毫秒读一次输入，刷新右边的指示块
103 private void Timer_Tick(object? sender, EventArgs e)
104 {
105     if (!int.TryParse(txtSlave.Text, out int slave)) return;
106     if (!int.TryParse(txtBit.Text, out int bit)) return;
107
108     SCM_ReadInputBit(card, slave, bit, out int value);
109     lblDi.Content = value != 0 ? "输入 ON" : "输入 OFF";
110     lblDi.Background = value != 0 ? Brushes.LimeGreen : Brushes.Gainsboro;
111 }
```

- 参数 `sender` 后面的 `?` 表示它可以是空值。
- 输入框里不是整数就什么也不做，不写日志，免得每 200 毫秒刷一条提示；定时读取的返回码不处理。
- 条件运算符 `value != 0 ? A : B`：有信号显示「输入 ON」、背景绿色 `Brushes.LimeGreen`；否则「输入 OFF」、浅灰 `Brushes.Gainsboro`。WPF 的 Label 显示的内容叫 `Content`。

代码全部写完了，按 Ctrl+S 保存。

第 8 章 编译运行

视频 21:41

8.1 编译

按 `Ctrl+Shift+B` 生成解决方案。输出窗口显示 **1 成功，0 失败**，就是编译通过了。程序在 `bin\Debug\net8.0-windows` 文件夹里，`SCmotionSoftCard.dll` 也被拷到了那里。

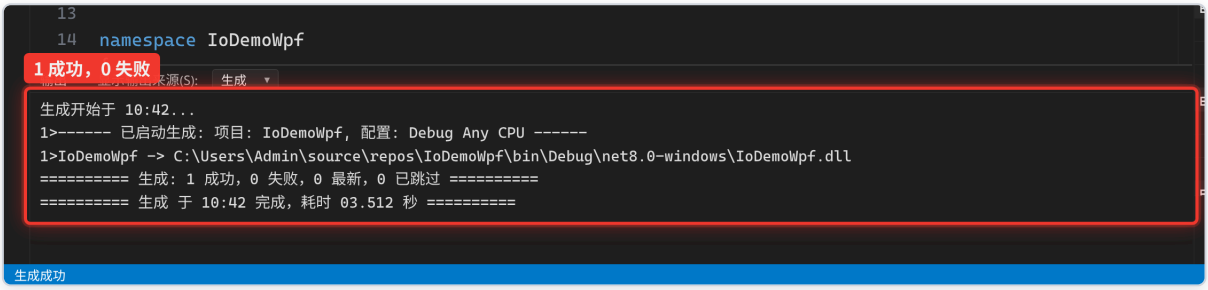

图 8-1 输出窗口：1 成功，0 失败

8.2 运行和验证

确认主软件已经进了 OP，或者切到了仿真，然后按 `F5`。

1. 点 **「连接」**，日志里出现「连接成功，已取得控制权」。
2. 从站位置和位号都是 0。点 **「输出 ON」**，返回码 0，接在 DO0 上的灯亮了；点 **「输出 OFF」**，灯灭了。
3. 按住接在 DI0 上的按钮，右边的指示灯马上变绿；点 **「读输入」**，日志里读到 1。松开按钮，指示灯变回灰色。
4. 最后点 **「断开」**，控制权还给主软件。

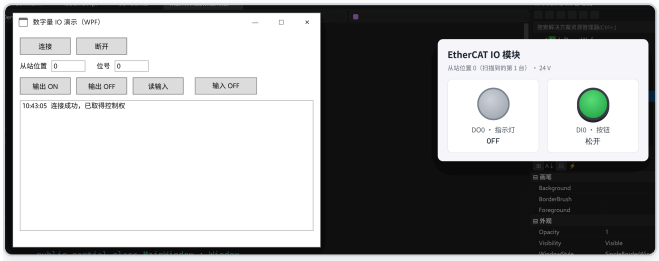

图 8-2 ① 连接成功

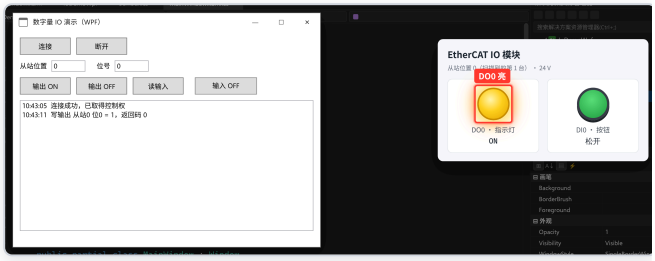

图 8-3 ② 输出 ON：DO0 的灯亮

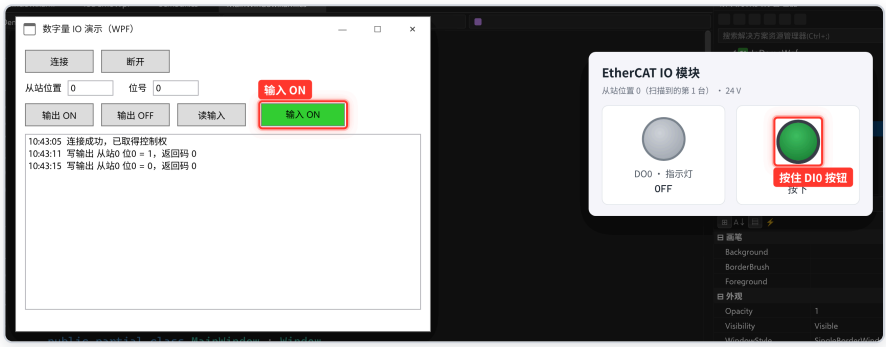

图 8-4 ③ 按住 DI0 上的按钮：指示灯变绿，「读输入」读到 1

提示 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 22:51

前三个是 P/Invoke 特有的；返回码的完整含义见附录 C。

现象	原因	怎么办
DllNotFoundException：找不到 SCmotionSoftCard.dll	dll 没有拷到 exe 旁边	dll 的「复制到输出目录」改成「如果较新则复制」
BadImageFormatException：试图加载格式不正确的程序	程序是 32 位或 Any CPU，dll 是 64 位	项目属性「平台目标」改成 x64
EntryPointNotFoundException：找不到入口点	声明里的函数名写错，或 dll 不是同一个版本	照头文件核对函数名；重新生成 SDK 包
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、SDK 服务没起来，或者 SDK 和主软件版本不一致	看「软卡设置」（SDK包）页的状态；必要时重新生成 SDK 包，换掉项目里的 dll
取控制权失败，返回码 3 / 7	3：主软件没进 OP；7：控制权在别的程序手上，或刚强行停止过程序	进 OP 或切仿真；先断开另一个程序，或等 5 秒再连
写输出返回 0，灯却不亮	这个点没映射，或从站位置、位号填错了	到主软件的「I/O 测试」页写同一个点核对
点按钮没有反应	XAML 里的 Click 没挂上，或处理函数名对不上	检查 Click="…" 和代码里的函数名一字不差

附录 A 完整代码

三个文件都和配套工程 **配套工程/IoDemoWpf** 一字不差。左边是行号。

MainWindow.xaml

```
<Window x:Class="IoDemoWpf.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:IoDemoWpf"
        mc:Ignorable="d"
        Title="数字量 IO 演示（WPF）" Height="420" Width="560" Closing="Window_Closing">
    <Window.Resources>
        <!-- 本窗口里的按钮统一：宽 90、高 30，右边空 10 -->
        <Style TargetType="Button">
            <Setter Property="Width" Value="90"/>
            <Setter Property="Height" Value="30"/>
            <Setter Property="Margin" Value="0,0,10,0"/>
        </Style>
    </Window.Resources>
    <StackPanel Margin="12">
        <!-- 第一排：连接、断开 -->
        <StackPanel Orientation="Horizontal" Margin="0,0,0,10">
            <Button x:Name="btnConnect" Content="连接" Click="btnConnect_Click"/>
            <Button x:Name="btnDisconnect" Content="断开" Click="btnDisconnect_Click"/>
        </StackPanel>
        <!-- 第二排：从站位置、位号 -->
        <StackPanel Orientation="Horizontal" Margin="0,0,0,10">
            <TextBlock Text="从站位置" VerticalAlignment="Center"/>
            <TextBox x:Name="txtSlave" Text="0" Width="60" Margin="8,0,20,0"/>
            <TextBlock Text="位号" VerticalAlignment="Center"/>
            <TextBox x:Name="txtBit" Text="0" Width="60" Margin="8,0,0,0"/>
        </StackPanel>
        <!-- 第三排：输出 ON、输出 OFF、读输入、输入指示 -->
        <StackPanel Orientation="Horizontal" Margin="0,0,0,10">
            <Button x:Name="btnOn" Content="输出 ON" Click="btnOn_Click"/>
            <Button x:Name="btnOff" Content="输出 OFF" Click="btnOff_Click"/>
            <Button x:Name="btnRead" Content="读输入" Click="btnRead_Click"/>
            <Label x:Name="lblDi" Content="输入 --" Width="110" Margin="10,0,0,0"
                   Background="Gainsboro" BorderBrush="Gray" BorderThickness="1"
                   HorizontalContentAlignment="Center" VerticalContentAlignment="Center"/>
        </StackPanel>
        <!-- 日志框 -->
        <TextBox x:Name="txtLog" Height="230" IsReadOnly="True" TextWrapping="Wrap"
                 VerticalScrollBarVisibility="Auto"/>
    </StackPanel>
</Window>
```

ScmSdk.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
```

```
using System.Threading.Tasks;

namespace IoDemoWpf
{
    // 软卡 SDK 的函数声明：用 DllImport 把 SCmotionSoftCard.dll 里的 C 函数接到 C# 里来
    internal static class ScmSdk
    {
        // C 动态库的文件名，运行时它要和 exe 在同一个文件夹
        const string Dll = "SCmotionSoftCard.dll";

        // 返回码 0 表示成功
        public const int SCM_OK = 0;

        [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
        public static extern int SCM_OpenProjectConfig(byte[] path, int clientId,
            out IntPtr handle);

        [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
        public static extern int SCM_GetOpenErrorText(byte[] buffer, int bufferSize);

        [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
        public static extern int SCM_RequestOwner(IntPtr handle);

        [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
        public static extern int SCM_ReleaseOwner(IntPtr handle);

        [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
        public static extern int SCM_Close(IntPtr handle);

        [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
        public static extern int SCM_WriteOutputBit(IntPtr handle, int slave, int bit, int value);

        [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
        public static extern int SCM_ReadInputBit(IntPtr handle, int slave, int bit,
            out int value);

        // 字符串转成以 0 结尾的 UTF-8 字节，传给 C 函数
        public static byte[] U8(string s)
        {
            return Encoding.UTF8.GetBytes(s + "\0");
        }

        // C 函数写回来的 UTF-8 字节转成字符串，到第一个 0 为止
        public static string FromU8(byte[] buf)
        {
            int n = Array.IndexOf(buf, (byte)0);
            return Encoding.UTF8.GetString(buf, 0, n < 0 ? buf.Length : n);
        }
    }
}
```

MainWindow.xaml.cs

```
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
```

```
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Windows.Threading;
using static IoDemoWpf.ScmSdk;

namespace IoDemoWpf
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        // 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
        const string ConfigPath = @"C:\ProgramData\SoftSC\softcard.runtime.json";

        // 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
        const int ClientId = 88;

        // 软卡句柄：连接成功后才有值，IntPtr.Zero 表示还没连接
        IntPtr card = IntPtr.Zero;

        // 定时器：连接以后每 200 毫秒读一次输入
        DispatcherTimer timer = new DispatcherTimer();

        public MainWindow()
        {
            InitializeComponent();
            timer.Interval = TimeSpan.FromMilliseconds(200);
            timer.Tick += Timer_Tick;
        }

        // 「连接」按钮：打开连接 → 取得控制权 → 启动定时器
        private void btnConnect_Click(object sender, RoutedEventArgs e)
        {
            if (card != IntPtr.Zero) { Log("已经连接了"); return; }

            int rc = SCM_OpenProjectConfig(U8(ConfigPath), ClientId, out card);
            if (rc != SCM_OK)
            {
                byte[] why = new byte[1024];
                SCM_GetOpenErrorText(why, why.Length);
                Log($"连接失败，返回码 {rc}: {FromU8(why)}");
                card = IntPtr.Zero;
                return;
            }

            rc = SCM_RequestOwner(card);
            if (rc != SCM_OK)
            {
                Log($"取控制权失败，返回码 {rc}");
                SCM_Close(card);
                card = IntPtr.Zero;
                return;
            }

            timer.Start();
            Log("连接成功，已取得控制权");
        }

        // 「断开」按钮
        private void btnDisconnect_Click(object sender, RoutedEventArgs e)
        {
            Disconnect();
        }
```

```
73
74          // 「输出 ON」按钮
75          private void btnOn_Click(object sender, RoutedEventArgs e)
76          {
77              WriteOutput(1);
78          }
79
80          // 「输出 OFF」按钮
81          private void btnOff_Click(object sender, RoutedEventArgs e)
82          {
83              WriteOutput(0);
84          }
85
86          // 「读输入」按钮
87          private void btnRead_Click(object sender, RoutedEventArgs e)
88          {
89              if (card == IntPtr.Zero) { Log("请先连接"); return; }
90              if (!ReadPoint(out int slave, out int bit)) return;
91
92              int rc = SCM_ReadInputBit(card, slave, bit, out int value);
93              Log($"读输入 从站{slave} 位{bit} = {value}，返回码 {rc}");
94          }
95
96          // 关窗口前也断开，把控制权还给主软件
97          private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
98          {
99              Disconnect();
100         }
101
102         // 定时器：每 200 毫秒读一次输入，刷新右边的指示块
103         private void Timer_Tick(object? sender, EventArgs e)
104         {
105             if (!int.TryParse(txtSlave.Text, out int slave)) return;
106             if (!int.TryParse(txtBit.Text, out int bit)) return;
107
108             SCM_ReadInputBit(card, slave, bit, out int value);
109             lblDi.Content = value != 0 ? "输入 ON" : "输入 OFF";
110             lblDi.Background = value != 0 ? Brushes.LimeGreen : Brushes.Gainsboro;
111         }
112
113         // 写一个数字量输出位：value 为 1 接通，为 0 断开
114         private void WriteOutput(int value)
115         {
116             if (card == IntPtr.Zero) { Log("请先连接"); return; }
117             if (!ReadPoint(out int slave, out int bit)) return;
118
119             int rc = SCM_WriteOutputBit(card, slave, bit, value);
120             Log($"写输出 从站{slave} 位{bit} = {value}，返回码 {rc}");
121         }
122
123         // 从两个输入框读出从站位置和位号；填的不是整数就提示，返回 false
124         private bool ReadPoint(out int slave, out int bit)
125         {
126             bit = 0;
127             if (!int.TryParse(txtSlave.Text, out slave) || !int.TryParse(txtBit.Text, out bit))
128             {
129                 Log("从站位置和位号要填整数");
130                 return false;
131             }
132             return true;
133         }
134
135         // 断开：停定时器 → 归还控制权 → 关闭句柄
136         private void Disconnect()
```

```
137         {
138             if (card == IntPtr.Zero) return;
139 
140             timer.Stop();
141             SCM_ReleaseOwner(card);
142             SCM_Close(card);
143             card = IntPtr.Zero;
144             Log("已断开");
145         }
146 
147         // 在日志框末尾加一行，前面带上时间
148         private void Log(string text)
149         {
150             txtLog.AppendText($"{DateTime.Now:HH:mm:ss}  {text}\r\n");
151             txtLog.ScrollToEnd();
152         }
153     }
154 }
```

附录 B 本教程用到的 SDK 指令

每条先列出头文件 `scmotion_card.h` 里的 C 原型，再列出 `ScmSdk.cs` 里对应的 C# 声明。所有指令都返回 `int` 返回码，`SCM_OK`（0）表示成功。

SCM_OpenProjectConfig

```
 C  int SCM_OpenProjectConfig(const char* configPath, int clientId, SCM_CARD_HANDLE* handle);
C#  public static extern int SCM_OpenProjectConfig(byte[] path, int clientId, out IntPtr handle);
```

打开连接。configPath 是连接配置文件路径（UTF-8，结尾 0）；clientId 是本程序的身份号；成功时 handle 给回句柄。

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

SCM_GetOpenErrorText

```
 C  int SCM_GetOpenErrorText(char* buffer, int bufferSize);
C#  public static extern int SCM_GetOpenErrorText(byte[] buffer, int bufferSize);
```

读最近一次连接失败的详细原因（UTF-8 文字），不需要句柄。

返回：SCM_OK = 取到

SCM_RequestOwner

```
 C  int SCM_RequestOwner(SCM_CARD_HANDLE handle);
C#  public static extern int SCM_RequestOwner(IntPtr handle);
```

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

返回：SCM_OK = 已取得；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

SCM_ReleaseOwner

```
 C  int SCM_ReleaseOwner(SCM_CARD_HANDLE handle);
C#  public static extern int SCM_ReleaseOwner(IntPtr handle);
```

归还控制权，主软件立即恢复控制。程序退出前应显式调用。

返回：SCM_OK = 已归还；7 = 你不是当前的控制权持有者

SCM_Close

```
 C  int SCM_Close(SCM_CARD_HANDLE handle);
C#  public static extern int SCM_Close(IntPtr handle);
```

关闭句柄。关闭前如果还持有控制权，先调用 SCM_ReleaseOwner。

返回：SCM_OK（已经关过的句柄再关一次也返回 SCM_OK）

SCM_WriteOutputBit

```
C   int SCM_WriteOutputBit(SCM_CARD_HANDLE handle, int slavePosition, int bit, int value);
C#  public static extern int SCM_WriteOutputBit(IntPtr handle, int slave, int bit, int value);
```

写一个数字量输出位。slavePosition 是从站拓扑位置（扫描顺序，从 0 开始），bit 是这台从站上的输出位号（从 0 开始），value 非 0 接通、0 断开。需要控制权。

返回：SCM_OK = 已下发（没映射时同样返回 SCM_OK）；7 = 没有控制权；5 = 主软件不可用

SCM_ReadInputBit

```
C   int SCM_ReadInputBit(SCM_CARD_HANDLE handle, int slavePosition, int bit, int* value);
C#  public static extern int SCM_ReadInputBit(IntPtr handle, int slave, int bit, out int value);
```

读一个数字量输入位，value 给回 0 或 1（没映射或失败时给 0）。只读，不需要控制权。

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

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_card.h 里的 `SCM_RESULT` 和《软卡 SDK 指令说明》。