

案例教程 02 轴使能与点动

在 WPF 程序里让一根轴使能，
按住按钮点动、松开就停，
并实时显示轴的状态。

Visual Studio 2022

C# WPF

.NET 8

P/Invoke

适用软件	SCMotion 4.4.17
配套视频	教程02_轴使能与点动_编程篇_WPF版.mp4
配套工程	配套工程/AxisDemoWpf
文档版本	V1.0
发布日期	2026-10

ABOUT

关于本教程

适合谁	做过教程01 WPF 版（数字量 IO）的人。没做过也能跟：新建项目、加入 SDK 的每一步这里都写了，但讲得简略些。
做出什么	一个 WPF 窗口程序：连接主软件，让一根轴使能、下使能、复位；按住按钮点动，松开就停；每 200 毫秒刷新一次轴的使能、故障、位置和速度。
这一期的重点	① 受理不等于完成：使能、复位返回 0 只表示命令收下了；② 点动必须收手；③ 用 C# 结构体接 C 的 <code>SCM_AXIS_STATUS</code> 。
开发环境	Visual Studio 2022 · .NET 8 · C# WPF 应用程序 · P/Invoke 调用 C 动态库
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02_轴使能与点动_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/AxisDemoWpf：代码和本文附录 A 一字不差，打开 <code>AxisDemoWpf.sln</code> 就能编译运行。
建议先看	../教程02_轴使能与点动_上篇_主软件操作.mp4：先在主软件里扫描建轴、进 OP、在轴的「在线」页低速点动过，再来写程序。

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看的地方；红色圆点上的数字对应表格里的编号。
- 代码块左边的数字是文件里的行号，和附录 A、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。

注意

轴是会动的。第一次上真机：电机和负载尽量脱开，急停按钮放在手边，点动速度先用小值；先在主软件的轴「在线」页里低速点动过、确认方向，再用自己的程序控制。

CONTENTS 目录

第 1 章 最终效果	视频 00:30	4
第 2 章 准备工作	视频 00:57	5
第 3 章 新建项目	视频 01:45	6
第 4 章 加入 SDK	视频 02:16	7
第 5 章 声明 SDK 函数	视频 02:57	8
5.1 轴状态结构体 SCM_AXIS_STATUS		8
5.2 连接用的五个声明		8
5.3 五个轴函数		9
第 6 章 画界面 (XAML)	视频 06:30	10
6.1 两个样式：按钮和标签		10
6.2 前三排		10
6.3 第四排：点动按钮不用 Click		11
第 7 章 写代码	视频 09:31	13
7.1 公共部分：和教程01 一样的地方		13
7.2 断开：先收手、再停轴		13
7.3 发命令前的检查：Ready		14
7.4 使能、下使能、复位		14
7.5 点动：StartJog 和 StopJog		15
7.6 停止		16
7.7 状态刷新：Timer_Tick		17
第 8 章 编译运行	视频 17:09	18
第 9 章 常见问题	视频 18:10	19
第 10 章 三个要点	视频 18:55	20
附录 A 完整代码		21
MainWindow.xaml		21
ScmSdk.cs		22
MainWindow.xaml.cs		23
附录 B 本教程用到的轴指令		28
附录 C 返回码速查		30

第 1 章 最终效果

视频 00:30



图 1-1 最后做出来的程序（运行中，已经连接并使能）

编号	控件	作用
①	「连接」「断开」	连接主软件并取得控制权；断开时先让点动收手、停轴，再归还控制权
②	轴号、点动速度	轴号从 0 开始（主软件里的「轴 1」是 0 号）；点动速度的单位是主软件轴页里的用户单位每秒，必须大于 0
③	「使能」「下使能」「复位」	发出使能、下使能、清除故障的命令
④	两个状态块	已使能显示绿色「已使能」；有故障显示橙红色「故障」
⑤	「反向点动」「正向点动」「停止」	按住点动，松开就停；「停止」是普通停止
⑥	位置和速度	实际位置、实际速度，保留三位小数
⑦	日志框	每一步的返回码都记在这里

第 2 章 准备工作

视频 00:57

让轴动起来之前，先确认三件事：

- ❶ **机械安全**：电机和负载尽量脱开，急停按钮放在手边，速度先用小值。
- ❷ **先在主软件里点动过**：在轴的「在线」页里低速点动，确认方向是对的。
- ❸ **主软件进 OP，或者切到仿真**：没有伺服就切仿真，仿真轴一样能使能、点动。

后两件都是主软件这边的操作，《上篇 主软件操作》里一步一步做过。SDK 包还是教程01 那一个：`sdk\c\win-x64\SCmotionSoftCard.dll`。Visual Studio 2022 要装好「.NET 桌面开发」。

第 3 章 新建项目

视频 01:45

和教程01 一样：

- 1 打开 Visual Studio 2022，「创建新项目」，搜索 `WPF`，选 C# 的「WPF 应用程序」。不要选带「.NET Framework」的那个。
- 2 项目名称填 `AxisDemoWpf`，勾上「将解决方案和项目放在同一目录中」。
- 3 框架选「.NET 8.0 (长期支持)」，点「创建」。

第 4 章 加入 SDK

视频 02:16

也是教程01 那三步：

- 1 把 SDK 包 `sdk\c\win-x64` 里的 `SCmotionSoftCard.dll` 复制到项目文件夹（和 `.csproj` 在一起）。
- 2 在解决方案资源管理器里选中 `SCmotionSoftCard.dll`，「复制到输出目录」改成「如果较新则复制」。
- 3 右键项目 → 「属性」 → 「生成」 → 「常规」，「平台目标」改成 `x64`，按 `Ctrl+S` 保存。

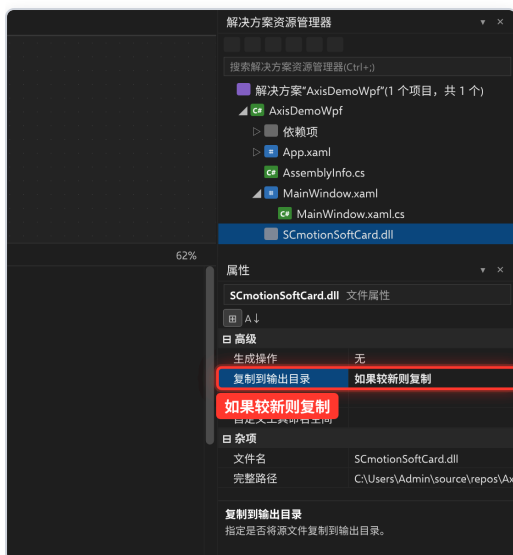


图 4-1 ② 复制到输出目录：如果较新则复制



图 4-2 ③ 平台目标：x64

第 5 章 声明 SDK 函数

视频 02:57

右键项目 → 「添加」 → 「类」，名称填 `ScmSdk.cs`。和教程 01 一样，先加 `using System.Runtime.InteropServices;`，把类改成 `internal static class`。

5.1 轴状态结构体 SCM_AXIS_STATUS

读轴状态的 C 函数，会把一整块数据写回来。C# 里要先定义一个一模一样的结构体来接：

```
10 // 轴状态：字段的顺序和类型，必须和头文件 scmotion_card.h 里的 SCM_AXIS_STATUS 一模一样
11 [StructLayout(LayoutKind.Sequential)]
12 public struct SCM_AXIS_STATUS
13 {
14     public int axis;           // 轴号
15     public int state;         // 轴状态码
16     public int powered;       // 已使能：1 是，0 否
17     public int faulted;       // 有故障：1 是，0 否
18     public int homed;         // 已回零：1 是，0 否
19     public double actualPosition; // 实际位置（轴用户单位）
20     public double targetPosition; // 目标位置
21     public double actualVelocity; // 实际速度（轴用户单位/秒）
22     public double followingError; // 跟随误差
23     public ushort statusWord;   // 驱动器状态字 6041
24     public int errorCode;        // 驱动器错误码
25     public double commandVelocity; // 规划速度
26 }
```

- `[StructLayout(LayoutKind.Sequential)]`：字段在内存里按写的顺序一个挨一个排，和 C 语言一样。
- 名字、字段的顺序和类型都照着头文件 `scmotion_card.h` 里的 `SCM_AXIS_STATUS` 写。C 函数是按位置往里写数据的：少一个字段、顺序颠倒、类型写错，后面读到的数全都错位。

C 头文件里	C# 里写成	说明
<code>typedef struct { ... } SCM_AXIS_STATUS</code>	<code>[StructLayout(LayoutKind.Sequential)] struct</code>	字段按顺序排，和 C 一样
<code>int powered / faulted</code>	<code>int</code>	1 是，0 否；不要写成 <code>bool</code> ， <code>bool</code> 的大小和 <code>int</code> 不一样
<code>double actualPosition</code>	<code>double</code>	8 字节小数
<code>unsigned short statusWord</code>	<code>ushort</code>	2 字节无符号整数
<code>SCM_AXIS_STATUS* status</code> （出参）	<code>out SCM_AXIS_STATUS status</code>	C 函数把整块状态写回来

5.2 连接用的五个声明

文件名常量 `DLL`、返回码常量 `SCM_OK`，以及连接、取错误原因、取得 / 归还控制权、关闭这五个声明，和教程 01 一模一样（第 31–52 行，见附录 A）。

5.3 五个轴函数

```
54 // ----- 轴：使能、复位、停止、点动、读状态 -----
55 // enable 传 1 使能，传 0 下使能
56 [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
57 public static extern int SCM_AxisPower(IntPtr handle, int axis, int enable);
58
59 [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
60 public static extern int SCM_AxisReset(IntPtr handle, int axis);
61
62 [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
63 public static extern int SCM_AxisStop(IntPtr handle, int axis);
64
65 // direction: 1 正向, -1 反向, 0 收手; 加速度、减速度传 0, 用主软件轴页的默认值
66 [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
67 public static extern int SCM_AxisJog(IntPtr handle, int axis, int direction,
68     double velocity, double acceleration, double deceleration);
69
70 [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
71 public static extern int SCM_AxisGetStatus(IntPtr handle, int axis,
72     out SCM_AXIS_STATUS status);
```

函数	参数	说明
SCM_AxisPower	句柄、轴号、enable	enable 传 1 使能，传 0 下使能
SCM_AxisReset	句柄、轴号	清除轴故障
SCM_AxisStop	句柄、轴号	普通停止
SCM_AxisJog	句柄、轴号、方向、速度、加速度、减速度	方向 1 正向、-1 反向、0 收手；后三个是 double，加减速度传 0 用主软件轴页的默认值
SCM_AxisGetStatus	句柄、轴号、out SCM_AXIS_STATUS	把整块轴状态写进刚才定义的结构体

最后是转换字符串的 U8、FromU8，也和教程01 一样。按 **Ctrl+S** 保存。

第 6 章 画界面 (XAML)

视频 06:30

切到 `MainWindow.xaml`。窗口那一行：标题「轴使能与点动演示 (WPF)」，高 460、宽 640；模板自带的 `<Grid>` 删掉。

6.1 两个样式：按钮和标签

```

9 <Window.Resources>
10 <!-- 本窗口里的按钮统一：宽 90、高 30，右边空 10 -->
11 <Style TargetType="Button">
12 <Setter Property="Width" Value="90"/>
13 <Setter Property="Height" Value="30"/>
14 <Setter Property="Margin" Value="0,0,10,0"/>
15 </Style>
16 <!-- 本窗口里的标签统一：高 30，左边空 10，浅灰底、灰边框，文字居中 -->
17 <Style TargetType="Label">
18 <Setter Property="Height" Value="30"/>
19 <Setter Property="Margin" Value="10,0,0,0"/>
20 <Setter Property="Background" Value="Gainsboro"/>
21 <Setter Property="BorderBrush" Value="Gray"/>
22 <Setter Property="BorderThickness" Value="1"/>
23 <Setter Property="HorizontalContentAlignment" Value="Center"/>
24 <Setter Property="VerticalContentAlignment" Value="Center"/>
25 </Style>
26 </Window.Resources>

```

按钮样式和教程01一样。这一期界面上有三个状态块，都是 Label、样子一样，所以再加一个 **Label 的样式**：高 30、左边空 10、浅灰底、1 像素灰边、文字居中。后面写标签只要写名字、内容和宽度。

6.2 前三排

```

27 <StackPanel Margin="12">
28 <!-- 第一排：连接、断开 -->
29 <StackPanel Orientation="Horizontal" Margin="0,0,0,10">
30 <Button x:Name="btnConnect" Content="连接" Click="btnConnect_Click"/>
31 <Button x:Name="btnDisconnect" Content="断开" Click="btnDisconnect_Click"/>
32 </StackPanel>
33 <!-- 第二排：轴号、点动速度 -->
34 <StackPanel Orientation="Horizontal" Margin="0,0,0,10">
35 <TextBlock Text="轴号" VerticalAlignment="Center"/>
36 <TextBox x:Name="txtAxis" Text="0" Width="60" Margin="8,0,20,0"/>
37 <TextBlock Text="点动速度" VerticalAlignment="Center"/>
38 <TextBox x:Name="txtSpeed" Text="10" Width="80" Margin="8,0,0,0"/>
39 </StackPanel>
40 <!-- 第三排：使能、下使能、复位，右边两个状态块 -->
41 <StackPanel Orientation="Horizontal" Margin="0,0,0,10">
42 <Button x:Name="btnPowerOn" Content="使能" Click="btnPowerOn_Click"/>
43 <Button x:Name="btnPowerOff" Content="下使能" Click="btnPowerOff_Click"/>
44 <Button x:Name="btnReset" Content="复位" Click="btnReset_Click"/>
45 <Label x:Name="lblPowered" Content="未使能" Width="90"/>
46 <Label x:Name="lblFaulted" Content="无故障" Width="90"/>

```

47 `</StackPanel>`

- 第二排：轴号输入框 `txtAxis` 默认 0；点动速度输入框 `txtSpeed` 默认 10。
- 第三排：使能、下使能、复位三个按钮各挂一个 `Click`；`lblPowered` 显示使能状态，`lblFaulted` 显示故障状态。

6.3 第四排：点动按钮不用 Click

```

48 <!-- 第四排：反向点动、正向点动、停止，右边显示位置和速度 -->
49 <StackPanel Orientation="Horizontal" Margin="0,0,0,10">
50     <Button x:Name="btnJogNeg" Content="反向点动"
51         PreviewMouseLeftButtonDown="btnJogNeg_PreviewMouseLeftButtonDown"
52         PreviewMouseLeftButtonUp="btnJogNeg_PreviewMouseLeftButtonUp"
53         LostMouseCapture="btnJogNeg_LostMouseCapture" />
54     <Button x:Name="btnJogPos" Content="正向点动"
55         PreviewMouseLeftButtonDown="btnJogPos_PreviewMouseLeftButtonDown"
56         PreviewMouseLeftButtonUp="btnJogPos_PreviewMouseLeftButtonUp"
57         LostMouseCapture="btnJogPos_LostMouseCapture" />
58     <Button x:Name="btnStop" Content="停止" Click="btnStop_Click" />
59     <Label x:Name="lblInfo" Content="--" Width="230" />
60 </StackPanel>

```

点动要「按住才走、松开就停」，所以不能用 `Click`：Click 是松开鼠标以后才触发的，而且一次点击只触发一次。要分开处理「按下」和「松开」：

事件	什么时候触发	为什么用它
<code>PreviewMouseLeftButtonDown</code>	鼠标左键按下	WPF 的按钮自己处理鼠标按下，普通的 <code>MouseLeftButtonDown</code> 到不了我们这里，要用带 <code>Preview</code> 的
<code>PreviewMouseLeftButtonUp</code>	鼠标左键松开	松开就收手
<code>LostMouseCapture</code>	按住时按钮丢了鼠标	比如中途切换了窗口，收不到松开了，用它兜底收手

挂事件的方法和教程 01 一样：敲 事件名=，选「新建事件处理程序」。最后在窗口那一行末尾挂上 `Closing`：关窗口之前要先停轴、还控制权。

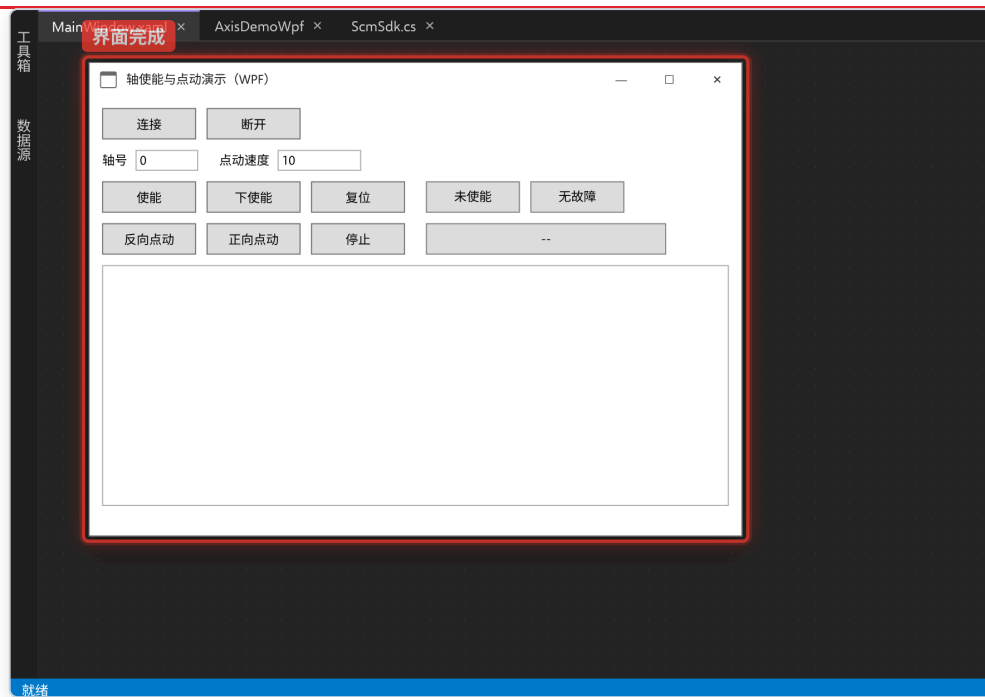


图 6-1 写好的界面 (设计器)

第 7 章 写代码

视频 09:31

打开 `MainWindow.xaml.cs`，十三个事件的空函数都生成好了。

7.1 公共部分：和教程01 一样的地方 视频 09:31

- 第 11–12 行：两行 using（定时器 `DispatcherTimer` 的命名空间、`using static AxisDemoWpf.ScmSdk`）。
- 第 21–31 行：配置文件路径、身份号、句柄 `card`、定时器 `timer`。
- 第 36–41 行：构造函数里设定定时器：200 毫秒，Tick 挂到 `Timer_Tick`。
- 第 43–69 行：「连接」按钮；第 221–226 行：`Log` 函数。

新的只有一个成员 `jogAxis`：

```
33 // 正在点动的轴号，-1 表示现在没有点动
34 int jogAxis = -1;
```

它记住现在正在点动的是哪根轴，初值 -1 表示没在点动。为什么要它，7.5 节写点动时再讲。

7.2 断开：先收手、再停轴 视频 10:43

```
207 // 断开：停定时器 → 点动收手、停轴 → 归还控制权 → 关闭句柄
208 private void Disconnect()
209 {
210     if (card == IntPtr.Zero) return;
211
212     timer.Stop();
213     StopJog();
214     if (int.TryParse(txtAxis.Text, out int axis)) SCM_AxisStop(card, axis);
215     SCM_ReleaseOwner(card);
216     SCM_Close(card);
217     card = IntPtr.Zero;
218     Log("已断开");
219 }
```

和教程01 比多了两步（第 213–214 行）：先让点动收手（`StopJog`，7.5 节写），再用 `SCM_AxisStop` 停轴，然后才归还控制权、关闭句柄。因为 `SCM_ReleaseOwner` 和 `SCM_Close` 都不会停轴：点动还在走的时候直接断开，轴会一直走下去。

「断开」按钮（第 71–75 行）和窗口的 `Closing` 事件（第 150–154 行）里，都调用 `Disconnect`。

7.3 发命令前的检查：Ready 视频 11:34

```

194 // 发轴命令之前的检查：连上了没有、轴号对不对，再取一次控制权
195 // 使能走完以后主软件会把控制权收回去；控制权本来就在自己手上，这里直接返回 0
196 private bool Ready(out int axis)
197 {
198     axis = 0;
199     if (card == IntPtr.Zero) { Log("请先连接"); return false; }
200     if (!int.TryParse(txtAxis.Text, out axis)) { Log("轴号要填整数"); return false; }
201
202     int rc = SCM_RequestOwner(card);
203     if (rc != SCM_OK) { Log($"取控制权失败, 返回码 {rc}"); return false; }
204     return true;
205 }

```

- 返回 `bool`：检查都通过返回 `true`；读出来的轴号从 `out` 参数 `axis` 带出去。开头 `axis = 0`：`out` 参数在返回之前必须有值。
- 检查有没有连接，再把轴号输入框里的文字转成整数。
- 最后再调用一次 `SCM_RequestOwner`。帮助里写着：使能流程走完以后，主软件会把控制权收回去；调试时程序停在断点上超过 5 秒、主软件收不到心跳，也会收回。所以每次发轴命令之前都再取一次。控制权本来就在自己手上的话，它直接返回 0，没有副作用。

7.4 使能、下使能、复位 视频 12:43

```

77 // 「使能」按钮：返回 0 只表示命令收下了，使能完成要看 powered
78 private void btnPowerOn_Click(object sender, RoutedEventArgs e)
79 {
80     if (!Ready(out int axis)) return;
81
82     int rc = SCM_AxisPower(card, axis, 1);
83     Log($"使能 轴{axis}, 返回码 {rc}");
84 }
85
86 // 「下使能」按钮
87 private void btnPowerOff_Click(object sender, RoutedEventArgs e)
88 {
89     if (!Ready(out int axis)) return;
90
91     int rc = SCM_AxisPower(card, axis, 0);
92     Log($"下使能 轴{axis}, 返回码 {rc}");
93 }
94
95 // 「复位」按钮：清除轴故障，完成要看 faulted 变回 0
96 private void btnReset_Click(object sender, RoutedEventArgs e)
97 {
98     if (!Ready(out int axis)) return;
99
100     int rc = SCM_AxisReset(card, axis);
101     Log($"复位 轴{axis}, 返回码 {rc}");
102 }

```

`SCM_AxisPower` 三个参数：句柄、轴号，最后一个 1 表示使能、0 表示下使能。`SCM_AxisReset` 清除轴故障。

注意 受理不等于完成。SCM_AxisPower 返回 0，只表示主软件「收下了」这条命令，并不表示轴已经使能好了。使能是主软件在后台一步一步做完的：顺利的话一百多毫秒到几秒，最坏要十二秒多，等待的超时不要短于 15 秒。完成没有，看轴状态里的 powered（7.7 节在定时器里读）。复位也一样：完成要看 faulted 变回 0，至少要 0.45 秒。

7.5 点动：StartJog 和 StopJog 视频 13:54

```

169 // 开始点动: direction 为 1 正向, 为 -1 反向
170 private void StartJog(int direction)
171 {
172     if (!Ready(out int axis)) return;
173     if (!double.TryParse(txtSpeed.Text, out double speed) || speed <= 0)
174     {
175         Log("点动速度要填大于 0 的数");
176         return;
177     }
178
179     int rc = SCM_AxisJog(card, axis, direction, speed, 0, 0);
180     Log($"点动 轴{axis} 方向{direction} 速度{speed}, 返回码 {rc}");
181     if (rc == SCM_OK) jogAxis = axis;
182 }
183
184 // 点动收手: direction 传 0, 轴按减速度停下; 没在点动就什么也不做
185 private void StopJog()
186 {
187     if (card == IntPtr.Zero || jogAxis < 0) return;
188
189     int rc = SCM_AxisJog(card, jogAxis, 0, 0, 0, 0);
190     Log($"点动收手 轴{jogAxis}, 返回码 {rc}");
191     jogAxis = -1;
192 }

```

行	做什么
172-177	先调用 Ready; 再用 double.TryParse 读点动速度: 转不了、或者不大于 0, 就提示并返回。方向不是 0 的时候, 速度必须大于 0
179	SCM_AxisJog: 句柄、轴号、方向、速度、加速度、减速度。加减速度传 0, 用主软件轴页里的默认值
181	返回 0 的话, 把轴号记进 jogAxis: 这根轴正在点动
187	没连接, 或者 jogAxis 小于 0 (现在没在点动), 就什么也不做
189	收手还是 SCM_AxisJog, 方向传 0: 轴按减速度停下来, 这不是急停。轴号用 jogAxis: 就算点动中途改了轴号输入框, 收手也发给正在走的那根轴
191	jogAxis 设回 -1。松开和丢鼠标两个事件可能先后都来, 第二次进来 jogAxis 已经是 -1, 就不会重复收手

六个点动事件: 按下调用 StartJog(-1) / StartJog(1), 松开、丢了鼠标都调用 StopJog:

```

104 // 「反向点动」按下: 开始反向走
105 private void btnJogNeg_PreviewMouseLeftButtonDown(object sender, MouseButtonEventArgs e)
106 {
107     StartJog(-1);

```

```

108 }
109
110 // 「反向点动」松开: 收手
111 private void btnJogNeg_PreviewMouseLeftButtonUp(object sender, MouseButtonEventArgs e)
112 {
113     StopJog();
114 }
115
116 // 「反向点动」按住时丢了鼠标 (比如中途切换了窗口), 收不到松开: 同样收手
117 private void btnJogNeg_LostMouseCapture(object sender, MouseEventArgs e)
118 {
119     StopJog();
120 }
121
122 // 「正向点动」按下: 开始正向走
123 private void btnJogPos_PreviewMouseLeftButtonDown(object sender, MouseButtonEventArgs e)
124 {
125     StartJog(1);
126 }
127
128 // 「正向点动」松开: 收手
129 private void btnJogPos_PreviewMouseLeftButtonUp(object sender, MouseButtonEventArgs e)
130 {
131     StopJog();
132 }
133
134 // 「正向点动」按住时丢了鼠标: 同样收手
135 private void btnJogPos_LostMouseCapture(object sender, MouseEventArgs e)
136 {
137     StopJog();
138 }

```

注意 点动必须收手。如果只在按下时发命令、松开不发, 轴会一直走下去。

7.6 停止 视频 15:53

```

140 // 「停止」按钮: 普通停止, 停下后要重新发运动指令才会再动
141 private void btnStop_Click(object sender, RoutedEventArgs e)
142 {
143     if (!Ready(out int axis)) return;
144
145     jogAxis = -1;
146     int rc = SCM_AxisStop(card, axis);
147     Log($"停止 轴{axis}, 返回码 {rc}");
148 }

```

SCM_AxisStop 是普通停止, 也是「最终停」: 轴停下来以后, 要重新发运动指令才会再动。停止之前把 jogAxis 设回 -1, 后面松开鼠标就不会再多发一次收手。

7.7 状态刷新: Timer_Tick 视频 16:13

```
156 // 定时器: 每 200 毫秒读一次轴状态, 刷新右边的状态块
157 private void Timer_Tick(object? sender, EventArgs e)
158 {
159     if (!int.TryParse(txtAxis.Text, out int axis)) return;
160     if (SCM_AxisGetStatus(card, axis, out SCM_AXIS_STATUS st) != SCM_OK) return;
161
162     lblPowered.Content = st.powered != 0 ? "已使能" : "未使能";
163     lblPowered.Background = st.powered != 0 ? Brushes.LimeGreen : Brushes.Gainsboro;
164     lblFaulted.Content = st.faulted != 0 ? "故障" : "无故障";
165     lblFaulted.Background = st.faulted != 0 ? Brushes.OrangeRed : Brushes.Gainsboro;
166     lblInfo.Content = $"位置 {st.actualPosition:F3}    速度 {st.actualVelocity:F3}";
167 }
```

- `SCM_AxisGetStatus` 的最后一个参数 `out SCM_AXIS_STATUS st`: 声明一个结构体变量 `st` 接状态。返回码不是 0 (比如轴号不存在) 就不刷新。
- `st.powered` 不等于 0: 使能完成, 显示「已使能」、背景变绿。`st.faulted` 不等于 0: 有故障, 显示「故障」、背景橙红。
- 最后一行显示实际位置和实际速度, 冒号后面的 `F3` 表示保留三位小数。

第 8 章 编译运行

视频 17:09

按 **Ctrl+Shift+B** 生成，输出窗口显示 1 成功、0 失败。确认主软件已经进了 OP，或者切到了仿真，按 **F5** 运行。

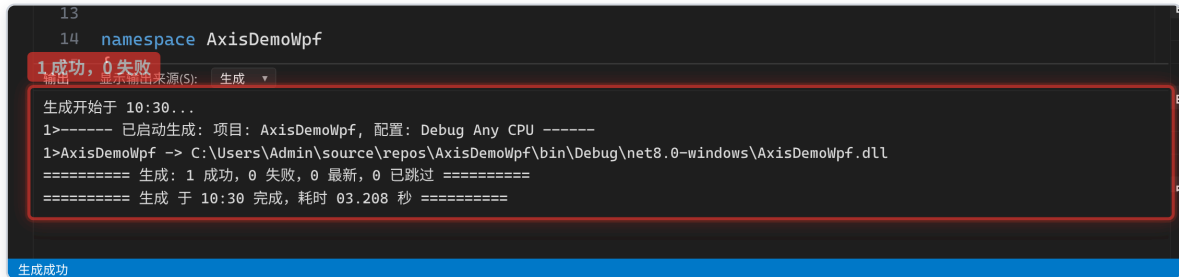


图 8-1 输出窗口：1 成功，0 失败

- 1 点「连接」，日志里出现「连接成功，已取得控制权」。
- 2 点「使能」。日志里马上出现返回码 0，但这时状态块还是「未使能」——命令只是被收下了。
- 3 过了一会，状态块变成绿色的「已使能」，这才是使能真正完成。
- 4 按住「正向点动」，轴开始正向走，位置一直在增加；松开，轴减速停下，速度回到 0。
- 5 按住「反向点动」，位置开始减少；松开，同样停下。
- 6 最后点「下使能」，再点「断开」，控制权还给主软件。



图 8-2 ② 返回码 0，但还是「未使能」



图 8-3 ③ 过一会儿才「已使能」



图 8-4 ④ 按住正向点动



图 8-5 ④ 松开：减速停下

第 9 章 常见问题

视频 18:10

现象	原因	怎么办
点了「使能」一直是「未使能」	使能还没走完；轴号不存在（读状态返回 4）；主软件没进 OP	等够 15 秒；核对轴号（主软件里的「轴 1」是 0 号）；看主软件状态
点动返回 7	控制权不在自己手上	发命令前先 SCM_RequestOwner（本例的 Ready）
点动返回 10	方向不是 0 时速度填了 0	点动速度填大于 0 的数
松开按钮轴还在走	没挂 PreviewMouseLeftButtonUp，或者用了 Click	按下、松开、丢鼠标三个事件都要挂
位置、速度读数离谱	结构体字段顺序或类型和头文件不一致	逐个对照 scmotion_card.h 里的 SCM_AXIS_STATUS
DllNotFoundException / BadImageFormatException	dll 没拷到 exe 旁边 / 平台不是 x64	同教程01 WPF 版：「如果较新则复制」、平台目标 x64

第 10 章 三个要点

视频 18:55

- ❶ **受理不等于完成**：使能和复位返回 0，只表示命令收下了。使能完成看 `powered`，复位完成看 `faulted`；使能等待的超时不短于 15 秒。
- ❷ **点动必须收手**：`PreviewMouseLeftButtonDown` 开始，`PreviewMouseLeftButtonUp` 和 `LostMouseCapture` 都要收手（方向传 0）。
- ❸ **结构体对齐头文件**：`SCM_AXIS_STATUS` 的字段顺序、类型和头文件一模一样；发轴命令之前再取一次控制权。

附录 A 完整代码

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

MainWindow.xaml

```

1 <Window x:Class="AxisDemoWpf.MainWindow"
2     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5     xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6     xmlns:local="clr-namespace:AxisDemoWpf"
7     mc:Ignorable="d"
8     Title="轴使能与点动演示 (WPF)" Height="460" Width="640" Closing="Window_Closing">
9     <Window.Resources>
10         <!-- 本窗口里的按钮统一: 宽 90、高 30, 右边空 10 -->
11         <Style TargetType="Button">
12             <Setter Property="Width" Value="90"/>
13             <Setter Property="Height" Value="30"/>
14             <Setter Property="Margin" Value="0,0,10,0"/>
15         </Style>
16         <!-- 本窗口里的标签统一: 高 30, 左边空 10, 浅灰底、灰边框, 文字居中 -->
17         <Style TargetType="Label">
18             <Setter Property="Height" Value="30"/>
19             <Setter Property="Margin" Value="10,0,0,0"/>
20             <Setter Property="Background" Value="Gainsboro"/>
21             <Setter Property="BorderBrush" Value="Gray"/>
22             <Setter Property="BorderThickness" Value="1"/>
23             <Setter Property="HorizontalContentAlignment" Value="Center"/>
24             <Setter Property="VerticalContentAlignment" Value="Center"/>
25         </Style>
26     </Window.Resources>
27     <StackPanel Margin="12">
28         <!-- 第一排: 连接、断开 -->
29         <StackPanel Orientation="Horizontal" Margin="0,0,0,10">
30             <Button x:Name="btnConnect" Content="连接" Click="btnConnect_Click"/>
31             <Button x:Name="btnDisconnect" Content="断开" Click="btnDisconnect_Click"/>
32         </StackPanel>
33         <!-- 第二排: 轴号、点动速度 -->
34         <StackPanel Orientation="Horizontal" Margin="0,0,0,10">
35             <TextBlock Text="轴号" VerticalAlignment="Center"/>
36             <TextBox x:Name="txtAxis" Text="0" Width="60" Margin="8,0,20,0"/>
37             <TextBlock Text="点动速度" VerticalAlignment="Center"/>
38             <TextBox x:Name="txtSpeed" Text="10" Width="80" Margin="8,0,0,0"/>
39         </StackPanel>
40         <!-- 第三排: 使能、下使能、复位, 右边两个状态块 -->
41         <StackPanel Orientation="Horizontal" Margin="0,0,0,10">
42             <Button x:Name="btnPowerOn" Content="使能" Click="btnPowerOn_Click"/>
43             <Button x:Name="btnPowerOff" Content="下使能" Click="btnPowerOff_Click"/>
44             <Button x:Name="btnReset" Content="复位" Click="btnReset_Click"/>
45             <Label x:Name="lblPowered" Content="未使能" Width="90"/>
46             <Label x:Name="lblFaulted" Content="无故障" Width="90"/>
47         </StackPanel>
48         <!-- 第四排: 反向点动、正向点动、停止, 右边显示位置和速度 -->
49         <StackPanel Orientation="Horizontal" Margin="0,0,0,10">
50             <Button x:Name="btnJogNeg" Content="反向点动"
51                 PreviewMouseLeftButtonDown="btnJogNeg_PreviewMouseLeftButtonDown"
52                 PreviewMouseLeftButtonUp="btnJogNeg_PreviewMouseLeftButtonUp"
53                 LostMouseCapture="btnJogNeg_LostMouseCapture"/>

```

```

54         <Button x:Name="btnJogPos" Content="正向点动"
55             PreviewMouseLeftButtonDown="btnJogPos_PreviewMouseLeftButtonDown"
56             PreviewMouseLeftButtonUp="btnJogPos_PreviewMouseLeftButtonUp"
57             LostMouseCapture="btnJogPos_LostMouseCapture"/>
58         <Button x:Name="btnStop" Content="停止" Click="btnStop_Click"/>
59         <Label x:Name="lblInfo" Content="--" Width="230"/>
60     </StackPanel>
61     <!-- 日志框 -->
62     <TextBox x:Name="txtLog" Height="230" IsReadOnly="True" TextWrapping="Wrap"
63         VerticalScrollBarVisibility="Auto"/>
64 </StackPanel>
65 </Window>

```

ScmSdk.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Runtime.InteropServices;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace AxisDemoWpf
9  {
10     // 轴状态: 字段的顺序和类型, 必须和头文件 scmotion_card.h 里的 SCM_AXIS_STATUS 一模一样
11     [StructLayout(LayoutKind.Sequential)]
12     public struct SCM_AXIS_STATUS
13     {
14         public int axis;           // 轴号
15         public int state;          // 轴状态码
16         public int powered;        // 已使能: 1 是, 0 否
17         public int faulted;        // 有故障: 1 是, 0 否
18         public int homed;          // 已回零: 1 是, 0 否
19         public double actualPosition; // 实际位置 (轴用户单位)
20         public double targetPosition; // 目标位置
21         public double actualVelocity; // 实际速度 (轴用户单位/秒)
22         public double followingError; // 跟随误差
23         public ushort statusWord;   // 驱动器状态字 6041
24         public int errorCode;        // 驱动器错误码
25         public double commandVelocity; // 规划速度
26     }
27
28     // 软卡 SDK 的函数声明: 用 DllImport 把 SCmotionSoftCard.dll 里的 C 函数接到 C# 里来
29     internal static class ScmSdk
30     {
31         // C 动态库的文件名, 运行时它要和 exe 在同一个文件夹
32         const string Dll = "SCmotionSoftCard.dll";
33
34         // 返回码 0 表示成功
35         public const int SCM_OK = 0;
36
37         // ----- 连接、控制权、关闭: 和教程 01 一样 -----
38         [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
39         public static extern int SCM_OpenProjectConfig(byte[] path, int clientId,
40             out IntPtr handle);
41
42         [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
43         public static extern int SCM_GetOpenErrorText(byte[] buffer, int bufferSize);
44
45         [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
46         public static extern int SCM_RequestOwner(IntPtr handle);

```

```

47
48     [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
49     public static extern int SCM_ReleaseOwner(IntPtr handle);
50
51     [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
52     public static extern int SCM_Close(IntPtr handle);
53
54     // ----- 轴: 使能、复位、停止、点动、读状态 -----
55     // enable 传 1 使能, 传 0 下使能
56     [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
57     public static extern int SCM_AxisPower(IntPtr handle, int axis, int enable);
58
59     [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
60     public static extern int SCM_AxisReset(IntPtr handle, int axis);
61
62     [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
63     public static extern int SCM_AxisStop(IntPtr handle, int axis);
64
65     // direction: 1 正向, -1 反向, 0 收手; 加速度、减速度传 0, 用主软件轴页的默认值
66     [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
67     public static extern int SCM_AxisJog(IntPtr handle, int axis, int direction,
68         double velocity, double acceleration, double deceleration);
69
70     [DllImport(Dll, CallingConvention = CallingConvention.Cdecl)]
71     public static extern int SCM_AxisGetStatus(IntPtr handle, int axis,
72         out SCM_AXIS_STATUS status);
73
74     // 字符串转成以 0 结尾的 UTF-8 字节, 传给 C 函数
75     public static byte[] U8(string s)
76     {
77         return Encoding.UTF8.GetBytes(s + "\0");
78     }
79
80     // C 函数写回来的 UTF-8 字节转成字符串, 到第一个 0 为止
81     public static string FromU8(byte[] buf)
82     {
83         int n = Array.IndexOf(buf, (byte)0);
84         return Encoding.UTF8.GetString(buf, 0, n < 0 ? buf.Length : n);
85     }
86 }
87 }

```

MainWindow.xaml.cs

```

1  using System.Text;
2  using System.Windows;
3  using System.Windows.Controls;
4  using System.Windows.Data;
5  using System.Windows.Documents;
6  using System.Windows.Input;
7  using System.Windows.Media;
8  using System.Windows.Media.Imaging;
9  using System.Windows.Navigation;
10 using System.Windows.Shapes;
11 using System.Windows.Threading;
12 using static AxisDemoWpf.ScmSdk;
13
14 namespace AxisDemoWpf
15 {
16     /// <summary>
17     /// Interaction logic for MainWindow.xaml

```

```

18     /// </summary>
19     public partial class MainWindow : Window
20     {
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         // 本程序的身份证号: 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 毫秒读一次轴状态
31         DispatcherTimer timer = new DispatcherTimer();
32
33         // 正在点动的轴号, -1 表示现在没有点动
34         int jogAxis = -1;
35
36         public MainWindow()
37         {
38             InitializeComponent();
39             timer.Interval = TimeSpan.FromMilliseconds(200);
40             timer.Tick += Timer_Tick;
41         }
42
43         // 「连接」按钮: 打开连接 → 取得控制权 → 启动定时器
44         private void btnConnect_Click(object sender, RoutedEventArgs e)
45         {
46             if (card != IntPtr.Zero) { Log("已经连接了"); return; }
47
48             int rc = SCM_OpenProjectConfig(U8(ConfigPath), ClientId, out card);
49             if (rc != SCM_OK)
50             {
51                 byte[] why = new byte[1024];
52                 SCM_GetOpenErrorText(why, why.Length);
53                 Log($"连接失败, 返回码 {rc}: {FromU8(why)}");
54                 card = IntPtr.Zero;
55                 return;
56             }
57
58             rc = SCM_RequestOwner(card);
59             if (rc != SCM_OK)
60             {
61                 Log($"取控制权失败, 返回码 {rc}");
62                 SCM_Close(card);
63                 card = IntPtr.Zero;
64                 return;
65             }
66
67             timer.Start();
68             Log("连接成功, 已取得控制权");
69         }
70
71         // 「断开」按钮
72         private void btnDisconnect_Click(object sender, RoutedEventArgs e)
73         {
74             Disconnect();
75         }
76
77         // 「使能」按钮: 返回 0 只表示命令收下了, 使能完成要看 powered
78         private void btnPowerOn_Click(object sender, RoutedEventArgs e)
79         {
80             if (!Ready(out int axis)) return;
81

```

```

82         int rc = SCM_AxisPower(card, axis, 1);
83         Log($"使能 轴{axis}, 返回码 {rc}");
84     }
85
86     // 「下使能」按钮
87     private void btnPowerOff_Click(object sender, RoutedEventArgs e)
88     {
89         if (!Ready(out int axis)) return;
90
91         int rc = SCM_AxisPower(card, axis, 0);
92         Log($"下使能 轴{axis}, 返回码 {rc}");
93     }
94
95     // 「复位」按钮: 清除轴故障, 完成要看 faulted 变回 0
96     private void btnReset_Click(object sender, RoutedEventArgs e)
97     {
98         if (!Ready(out int axis)) return;
99
100        int rc = SCM_AxisReset(card, axis);
101        Log($"复位 轴{axis}, 返回码 {rc}");
102    }
103
104    // 「反向点动」按下: 开始反向走
105    private void btnJogNeg_PreviewMouseLeftButtonDown(object sender, MouseButtonEventArgs e)
106    {
107        StartJog(-1);
108    }
109
110    // 「反向点动」松开: 收手
111    private void btnJogNeg_PreviewMouseLeftButtonUp(object sender, MouseButtonEventArgs e)
112    {
113        StopJog();
114    }
115
116    // 「反向点动」按住时丢了鼠标 (比如中途切换了窗口), 收不到松开: 同样收手
117    private void btnJogNeg_LostMouseCapture(object sender, MouseEventArgs e)
118    {
119        StopJog();
120    }
121
122    // 「正向点动」按下: 开始正向走
123    private void btnJogPos_PreviewMouseLeftButtonDown(object sender, MouseButtonEventArgs e)
124    {
125        StartJog(1);
126    }
127
128    // 「正向点动」松开: 收手
129    private void btnJogPos_PreviewMouseLeftButtonUp(object sender, MouseButtonEventArgs e)
130    {
131        StopJog();
132    }
133
134    // 「正向点动」按住时丢了鼠标: 同样收手
135    private void btnJogPos_LostMouseCapture(object sender, MouseEventArgs e)
136    {
137        StopJog();
138    }
139
140    // 「停止」按钮: 普通停止, 停下后要重新发运动指令才会再动
141    private void btnStop_Click(object sender, RoutedEventArgs e)
142    {
143        if (!Ready(out int axis)) return;
144
145        jogAxis = -1;

```

```

146         int rc = SCM_AxisStop(card, axis);
147         Log($"停止 轴{axis}, 返回码 {rc}");
148     }
149
150     // 关闭窗口前也断开, 把控制权还给主软件
151     private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
152     {
153         Disconnect();
154     }
155
156     // 定时器: 每 200 毫秒读一次轴状态, 刷新右边的状态块
157     private void Timer_Tick(object? sender, EventArgs e)
158     {
159         if (!int.TryParse(txtAxis.Text, out int axis)) return;
160         if (SCM_AxisGetStatus(card, axis, out SCM_AXIS_STATUS st) != SCM_OK) return;
161
162         lblPowered.Content = st.powered != 0 ? "已使能" : "未使能";
163         lblPowered.Background = st.powered != 0 ? Brushes.LimeGreen : Brushes.Gainsboro;
164         lblFaulted.Content = st.faulted != 0 ? "故障" : "无故障";
165         lblFaulted.Background = st.faulted != 0 ? Brushes.OrangeRed : Brushes.Gainsboro;
166         lblInfo.Content = $"位置 {st.actualPosition:F3}    速度 {st.actualVelocity:F3}";
167     }
168
169     // 开始点动: direction 为 1 正向, 为 -1 反向
170     private void StartJog(int direction)
171     {
172         if (!Ready(out int axis)) return;
173         if (!double.TryParse(txtSpeed.Text, out double speed) || speed <= 0)
174         {
175             Log("点动速度要填大于 0 的数");
176             return;
177         }
178
179         int rc = SCM_AxisJog(card, axis, direction, speed, 0, 0);
180         Log($"点动 轴{axis} 方向{direction} 速度{speed}, 返回码 {rc}");
181         if (rc == SCM_OK) jogAxis = axis;
182     }
183
184     // 点动收手: direction 传 0, 轴按减速度停下; 没在点动就什么也不做
185     private void StopJog()
186     {
187         if (card == IntPtr.Zero || jogAxis < 0) return;
188
189         int rc = SCM_AxisJog(card, jogAxis, 0, 0, 0, 0);
190         Log($"点动收手 轴{jogAxis}, 返回码 {rc}");
191         jogAxis = -1;
192     }
193
194     // 发轴命令之前的检查: 连上了没有、轴号对不对, 再取一次控制权
195     // 使能走完以后主软件会把控制权收回去; 控制权本来就在自己手上, 这里直接返回 0
196     private bool Ready(out int axis)
197     {
198         axis = 0;
199         if (card == IntPtr.Zero) { Log("请先连接"); return false; }
200         if (!int.TryParse(txtAxis.Text, out axis)) { Log("轴号要填整数"); return false; }
201
202         int rc = SCM_RequestOwner(card);
203         if (rc != SCM_OK) { Log($"取控制权失败, 返回码 {rc}"); return false; }
204         return true;
205     }
206
207     // 断开: 停定时器 → 点动收手、停轴 → 归还控制权 → 关闭句柄
208     private void Disconnect()
209     {

```

```
210         if (card == IntPtr.Zero) return;
211
212         timer.Stop();
213         StopJog();
214         if (int.TryParse(txtAxis.Text, out int axis)) SCM_AxisStop(card, axis);
215         SCM_ReleaseOwner(card);
216         SCM_Close(card);
217         card = IntPtr.Zero;
218         Log("已断开");
219     }
220
221     // 在日志框末尾加一行, 前面带上时间
222     private void Log(string text)
223     {
224         txtLog.AppendText($"{DateTime.Now:HH:mm:ss} {text}\r\n");
225         txtLog.ScrollToEnd();
226     }
227 }
228 }
```

附录 B 本教程用到的轴指令

连接、控制权、关闭五条指令见教程01 WPF 版附录 B。下面每条先列 C 原型 (scmotion_card.h)，再列 ScmSdk.cs 里的 C# 声明。

SCM_AxisPower

```
C int SCM_AxisPower(SCM_CARD_HANDLE handle, int axis, int enable);
C# public static extern int SCM_AxisPower(IntPtr handle, int axis, int enable);
```

enable 非 0 使能、0 下使能。**异步受理**：返回码只表示收下了，完成看 SCM_AxisGetStatus 的 powered；使能最坏要十二秒多。需要控制权。

返回：SCM_OK = 已受理；4 = 轴号非法；7 = 没有控制权；3 = 当前状态不允许

SCM_AxisReset

```
C int SCM_AxisReset(SCM_CARD_HANDLE handle, int axis);
C# public static extern int SCM_AxisReset(IntPtr handle, int axis);
```

复位轴故障。异步受理，完成看 faulted 转 0，耗时不少于 0.45 秒。需要控制权。

返回：SCM_OK = 已受理；4 / 7 / 3 同上

SCM_AxisStop

```
C int SCM_AxisStop(SCM_CARD_HANDLE handle, int axis);
C# public static extern int SCM_AxisStop(IntPtr handle, int axis);
```

普通停止（最终停）：停下后要重新发运动指令才会再动。需要控制权。

返回：SCM_OK = 已下发

SCM_AxisJog

```
C int SCM_AxisJog(SCM_CARD_HANDLE handle, int axis, int direction, double velocity, double acceleration, double deceleration);
C# public static extern int SCM_AxisJog(IntPtr handle, int axis, int direction, double velocity, double acceleration, double deceleration);
```

direction 大于 0 正向、小于 0 反向、0 收手（按减速度停）。velocity 是点动速度（轴用户单位/秒），direction 不是 0 时必须大于 0。需要控制权。

返回：SCM_OK = 已下发；10 = 参数非法（比如速度为 0）；7 = 没有控制权

SCM_AxisGetStatus

```
C  int SCM_AxisGetStatus(SCM_CARD_HANDLE handle, int axis, SCM_AXIS_STATUS* status);  
C# public static extern int SCM_AxisGetStatus(IntPtr handle, int axis, out SCM_AXIS_STATUS status);
```

读轴状态：powered 已使能、faulted 有故障、actualPosition 实际位置、actualVelocity 实际速度等。只读，不需要控制权。

返回：SCM_OK = 正常；4 = 轴号非法（返回非 0 时 status 里的数无意义）

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（使能、复位：只表示已受理）
1	SCM_NOT_OPEN	卡未打开	句柄已经关闭后还在调用
3	SCM_INVALID_STATE	当前状态下该操作无效	主软件没进 OP，或当前模式不允许下发
4	SCM_INVALID_AXIS	轴号非法	轴号不存在：读状态、使能都会返回 4
5	SCM_BACKEND_UNAVAILABLE	运行时后端不可用	主软件没运行、断连，或 SDK 版本不一致
6	SCM_TIMEOUT	超时	—
7	SCM_REJECTED	被拒绝	控制权不在自己手上
8	SCM_FAULTED	轴或运行时处于故障态	轴有故障时发运动指令；先复位
10	SCM_INVALID_ARGUMENT	参数非法	点动方向不是 0 时速度为 0