 2**026年1月**

PARDISO使用手册V2.0

MKL（2025版本）

**撰 写 人: 杨洋**

**指导教师: 左文杰 教授**

**轮胎结构研究课题组**

目录

[第1章 Intel Math Kernel Library(Intel MKl)的安装及配置VS2019 1](#_Toc63661050)

[1.1 Intel MKL的安装 1](#_Toc63661051)

[1.2 Intel MKL配置VS2019 4](#_Toc63661052)

[第2章 动态链接库（DLL）的创建 12](#_Toc63661053)

[2.1 向DLL项目中添加头文件 12](#_Toc63661054)

[2.2 向DLL项目中添加源文件 16](#_Toc63661055)

[2.3 生成DLL文件 18](#_Toc63661056)

[2.4 查找已生成的DLL文件 18](#_Toc63661057)

[2.5 查找PARDISO依赖文件 20](#_Toc63661058)

[第3章 C#调用动态链接库（DLL） 23](#_Toc63661059)

[3.1 DLL文件存放位置 23](#_Toc63661060)

[3.2 DLL导入C#程序与Pardiso函数声明 25](#_Toc63661061)

[3.3 参数设置 27](#_Toc63661062)

[3.4 函数调用 38](#_Toc63661063)

[3.4.1 项目添加引用 38](#_Toc63661064)

[3.4.2 函数调用示例 40](#_Toc63661065)

# Intel Math Kernel Library(Intel MKl)的安装及配置VS2019

**安装前提：当前电脑已安装VS2019或其他版本Visual Studio，并且必须安装“使用C++的桌面开发”模块，其他模块根据需求自行选择即可，如**图 1‑1**所示。**

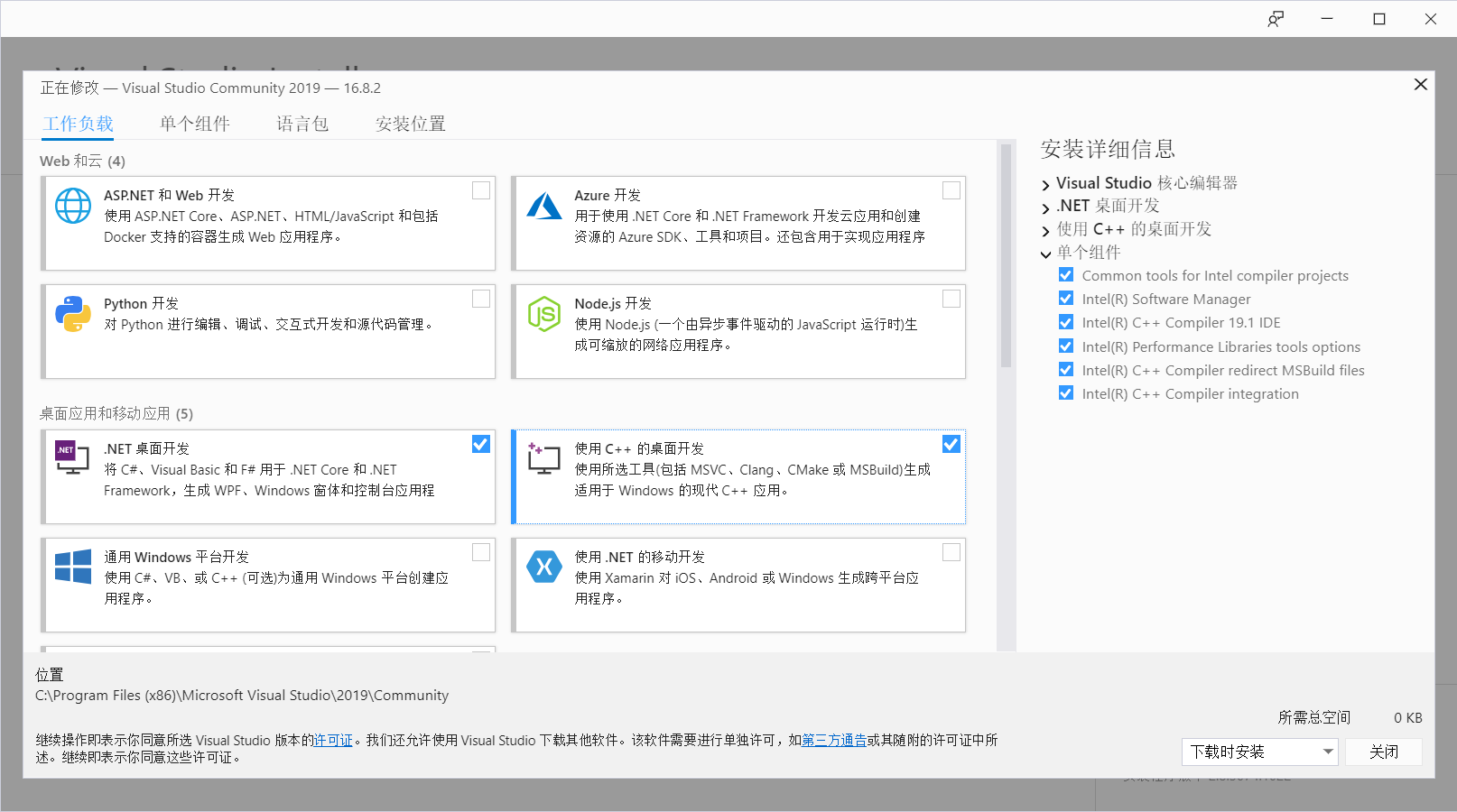


图 1‑1 VS2019模块安装

## Intel MKL的安装

**本次安装为在线安装，请时刻保持网络处于连接状态。**

（1）双击附件中的应用程序启动安装程序（如图 1‑2所示，此版本为2025版，如需下载最新版本，可以访问 Intel oneMKL 官方下载页：https://www.intel.com/content/

www/us/en/developer/tools/oneapi/onemkl-download.html )



图 1‑2 安装程序

（2）得到图 1‑3对话框，首先勾选“I accept the terms of the license agreement”复选框，软件安装路径需要通过截屏、拍照等方式保存软件安装路径，因为后面即将用到安装路径，然后我们点击“Custom Installation”右边的“Customize”，如图1-4所示。

图形用户界面, 文本, 应用程序, 电子邮件, Teams

AI 生成的内容可能不正确。

图 1‑3 勾选 I accept the terms of the license agreement以及文件安装路径

图形用户界面, 文本, 应用程序, 电子邮件, Teams

AI 生成的内容可能不正确。

图 1‑4 勾选Customize

（3）得到图 1‑5对话框，连续点击至最后一处对话框，依次点击“I consent to the collection of my information”和“install”进行MKL下载安装。

图形用户界面, 文本

AI 生成的内容可能不正确。

图 1‑5 安装MKL

（4）得到图 1‑6，请耐心等待文件下载完毕。

图形用户界面, 应用程序

AI 生成的内容可能不正确。

图 1‑6 在线下载安装文件

（5）文件下载完毕后得到图 1‑7，点击“Finish”。

图形用户界面, 应用程序

AI 生成的内容可能不正确。

图 1‑7 MKL安装完毕

至此，Intel MKL在电脑上安装完毕。

## Intel MKL配置VS2019

1. 打开VS2019，点击“创建新项目”，如图 1‑8所示。

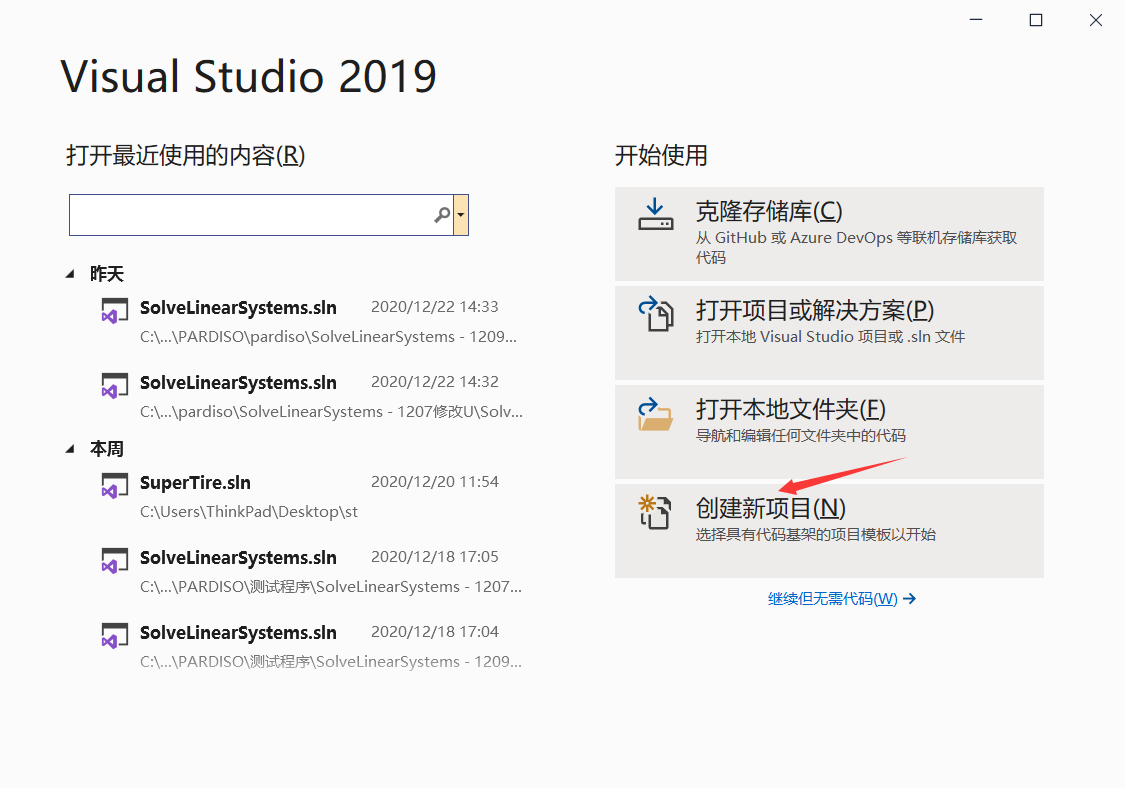


图 1‑8 VS2019启动初始界面

1. 语言选择“C++”，系统类型选择“Windows”，项目类型选择“库”，并在下方找到“动态链接库（DLL）”并单击，如图 1‑9所示，然后点击“下一步”。



图 1‑9 项目选择

1. 项目名称更改为“PardisoForCSharp”，解决方案名称会自动更新，其余默认即可，如图 1‑10所示，然后点击“创建”（建议读者将项目的“位置”复制保存到记事本中，后面会用到项目的位置信息）。

图形用户界面, 文本, 应用程序, 电子邮件

AI 生成的内容可能不正确。

图 1‑10 配置新项目

1. 把程序改为“x64”（这一步非常关键，需要提前设置，顺序不可颠倒），如图 1‑11所示，修改完毕后的窗口如图 1‑12所示。

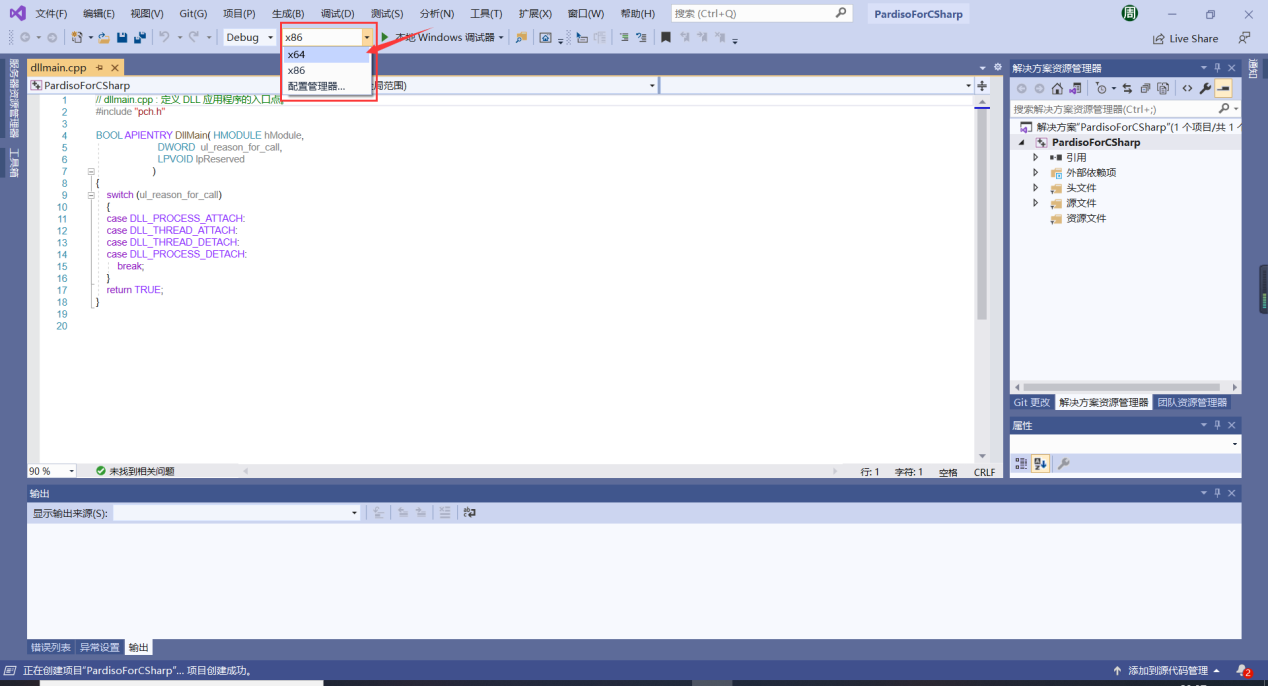


图 1‑11 程序类型改为x64

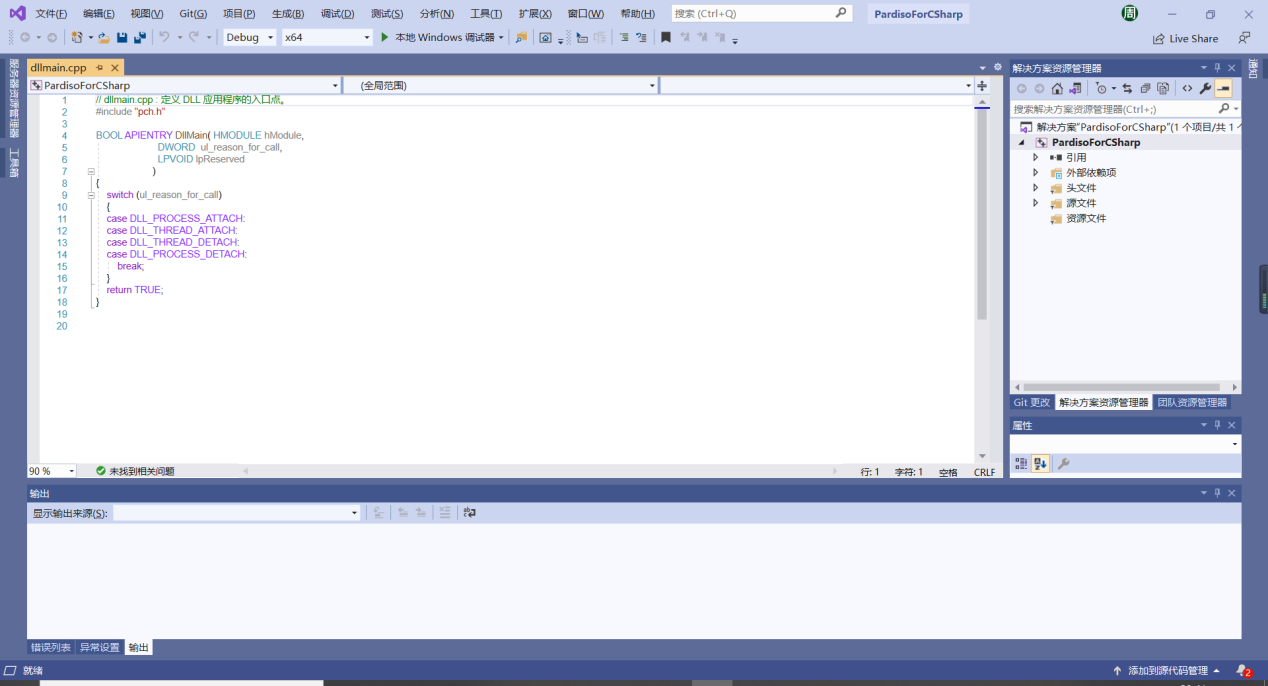


图 1‑12 x64程序窗口

1. 点击菜单栏中的“项目选项卡”，然后点击下拉菜单中的“PardisoForCSharp属性（P）”，如图 1‑13所示。

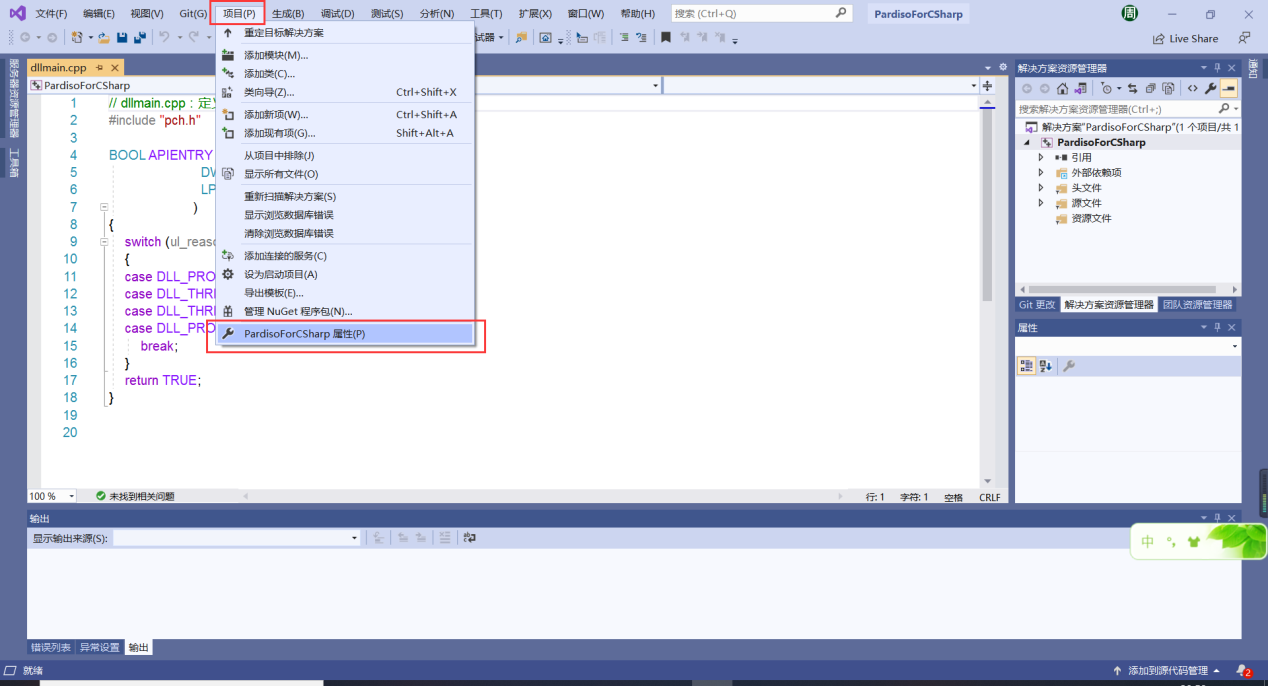


图 1‑13 查看PardisoForCSharp项目属性

1. 进入到“PardisoForCSharp属性页”对话框，按图 1‑14所示顺序依次点击。

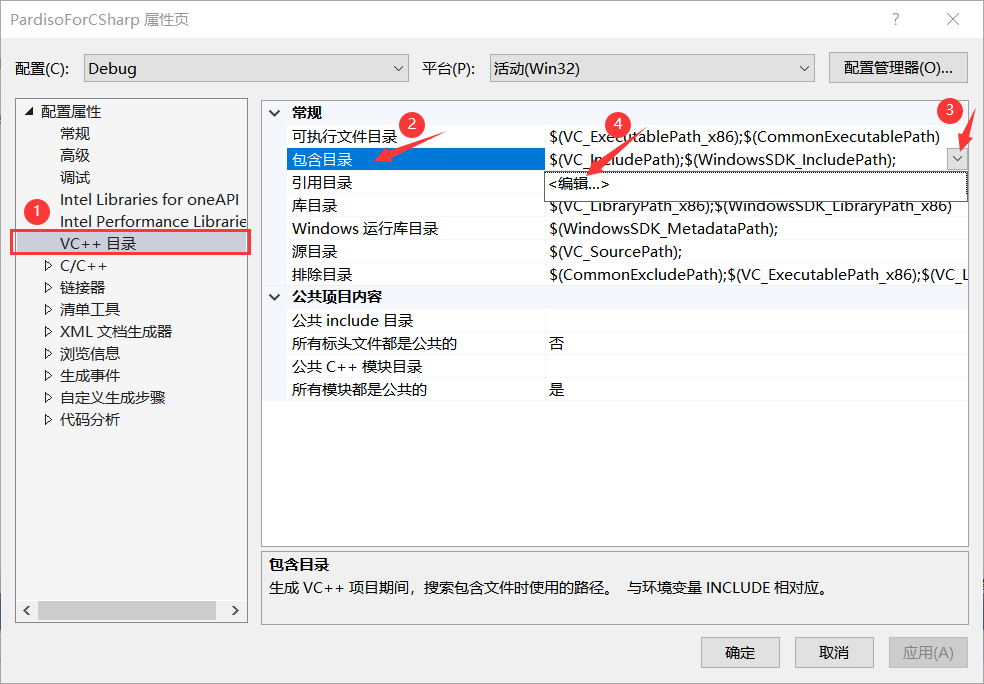


图 1‑14 为PardisoForCSharp项目添加包含目录

1. 进入到“包含目录”对话框，按图 1‑15所示顺序依次点击。

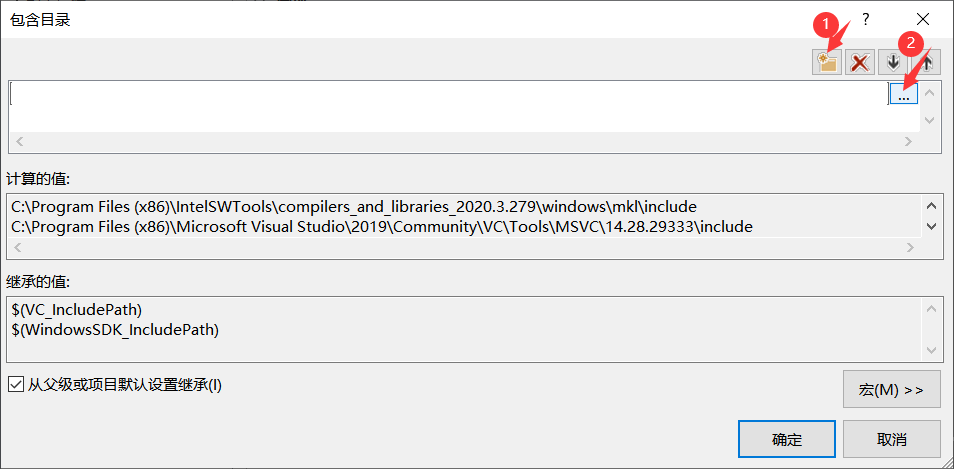


图 1‑15 为PardisoForCSharp项目添加包含目录

1. 找到“include”文件夹，教程中的“include”文件夹位置为：

C:\Program Files (x86)\Intel\oneAPI\mkl\2025.3\include

找到文件夹后点击“选择文件夹”，如图 1‑16所示，然后点击图 1‑17中的“确定”。

表格

AI 生成的内容可能不正确。

图 1‑16 为PardisoForCSharp项目添加包含目录

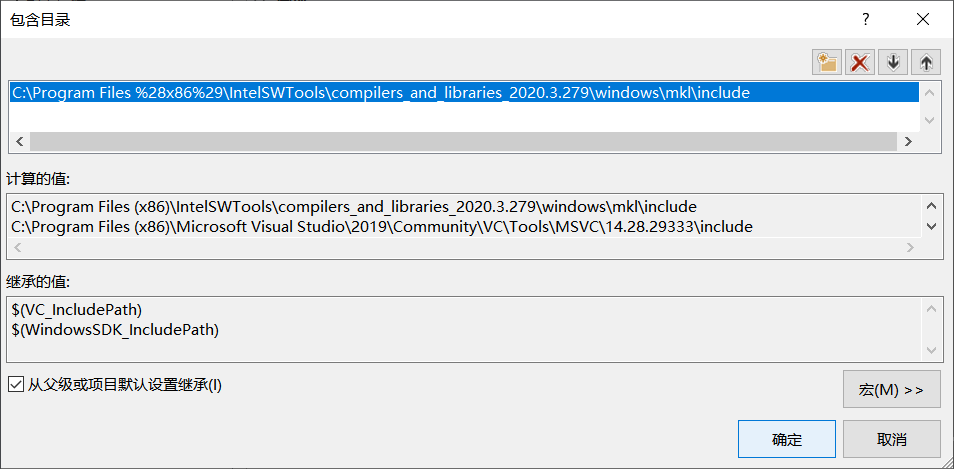


图 1‑17 为PardisoForCSharp项目添加包含目录

1. 然后按照上述同样的步骤添加两个“库目录”，教程中的“库目录”位置为：

C:\Program Files (x86)\Intel\oneAPI\mkl\2025.3\lib

C:\Program Files(x86) \Intel\oneAPI\compiler\2025.3\lib

1. 现在，我们打开“链接器”，点击“输入”，添加“附加依赖项”，如图 1‑18所示。

添加内容为：mkl\_intel\_lp64.lib;mkl\_intel\_thread.lib;mkl\_core.lib;libiomp5md.lib;

如图 1‑19所示，然后点击“确定”得到图 1‑20，再点击“确定”。最后，我们需要项目中配置 MKL 的库路径，在PardisoForCSharp项目的**链接器设置**中，添加 MKL 的库目录：打开项目属性 → 链接器 → 常规 → 附加库目录，如图1-21所示。新增路径为：

C:\Program Files (x86)\Intel\oneAPI\mkl\2025.3\lib

C:\Program Files (x86)\Intel\oneAPI\compiler\2025.3\lib

至此，Intel MKL与VS2019配置完成。

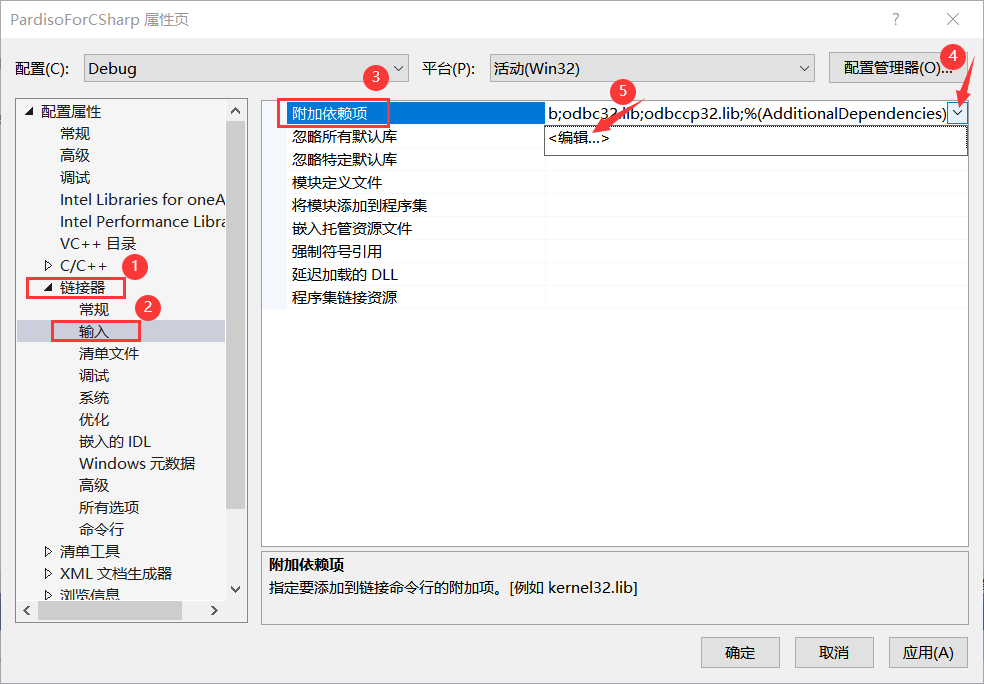


图 1‑18 编辑PardisoForCSharp项目附加依赖项



图 1‑19编辑PardisoForCSharp项目附加依赖项

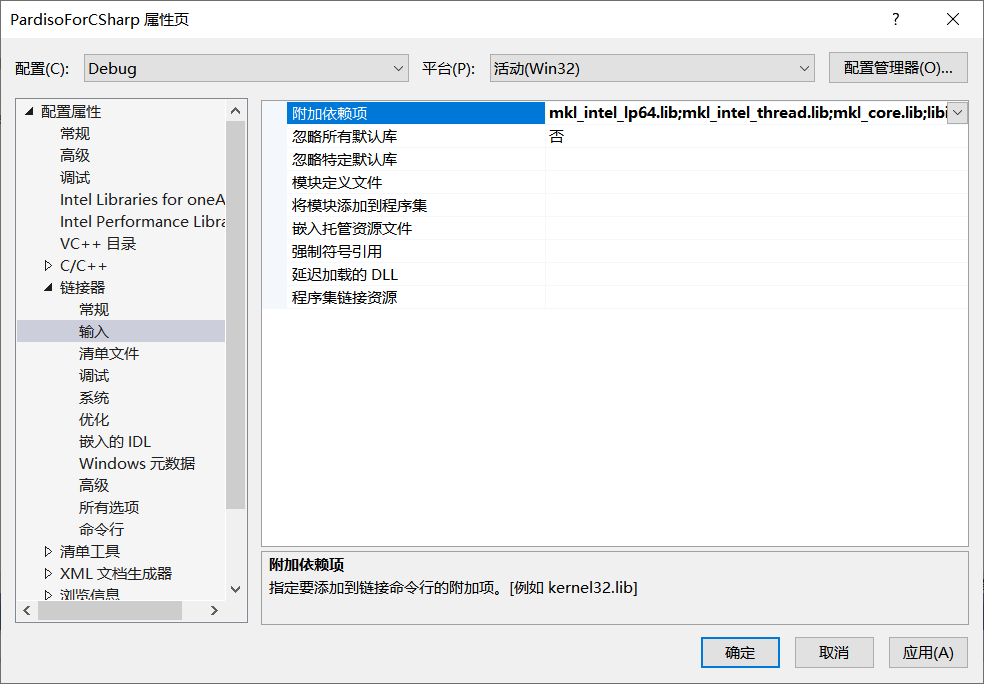


图 1‑20编辑PardisoForCSharp项目附加依赖项

图形用户界面, 文本, 应用程序

AI 生成的内容可能不正确。

图 1‑21在项目中配置 MKL 的附加库路径

# 动态链接库（DLL）的创建

在第1章中我们已经创建好了动态链接库（DLL）项目，并且完成了项目与Intel MKL的配置，这一章我们将正式开始动态链接库（DLL）的创建与生成。

## 向DLL项目中添加头文件

头文件名称：PardisoForCSharp.h

1. 右击解决方案资源管理器中的“头文件”，依次选择“添加（D）”“新建项（W）…”，如图 2‑1所示。

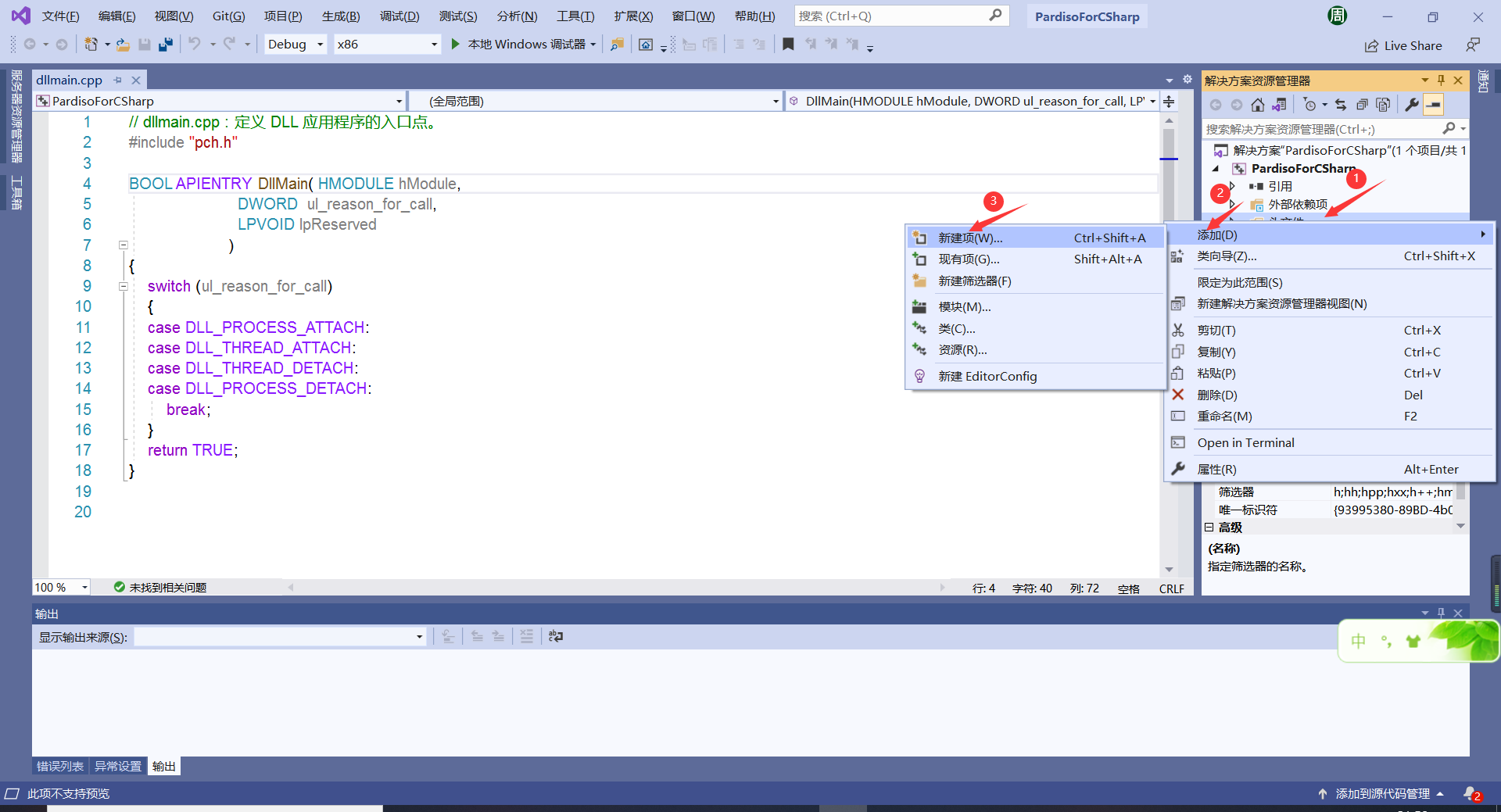


图 2‑1 添加头文件

1. 得到图 2‑2对话框，点击“头文件（.h）”，输入头文件名称“PardisoForCSharp.h”，然后点击“添加（A）”。现在，我们已经在项目中新添加了一个空的头文件。

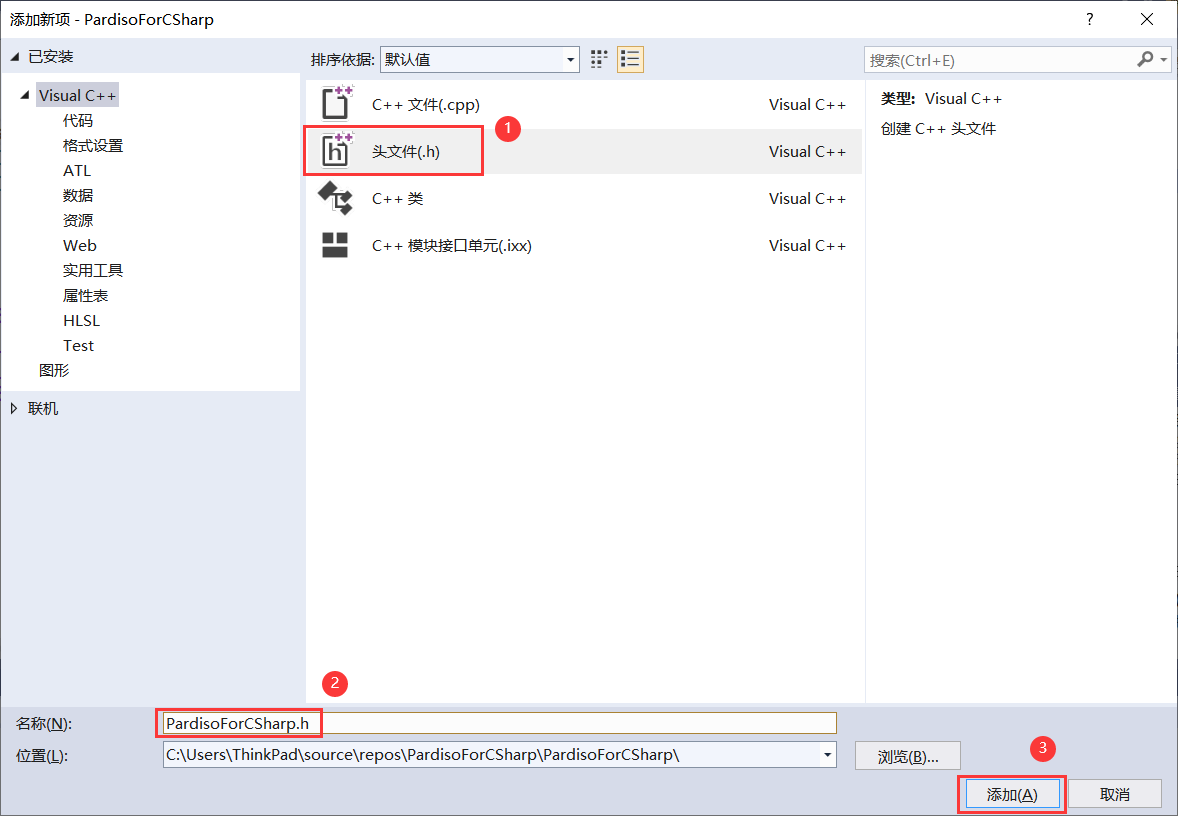


图 2‑2 添加头文件

1. 向头文件中写入以下代码，用作函数的声明，结果如图 2‑3所示。

|  |
| --- |
| #pragma once  #ifdef MATHLIBRARY\_EXPORTS  #define MATHLIBRARY\_API \_\_declspec(dllexport)  #else  #define MATHLIBRARY\_API \_\_declspec(dllimport)  #endif  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*函数声明\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  extern "C" MATHLIBRARY\_API void Pardiso(double\* pt, int\* maxfct, int\* mnum  , int\* mtype, int\* phase, int\* n, double\* a, int\* ia, int\* ja , int\* perm, int\* nrhs  , int\* iparm, int\* msglvl, double\* b, double\* x, int\* error); |
|  |

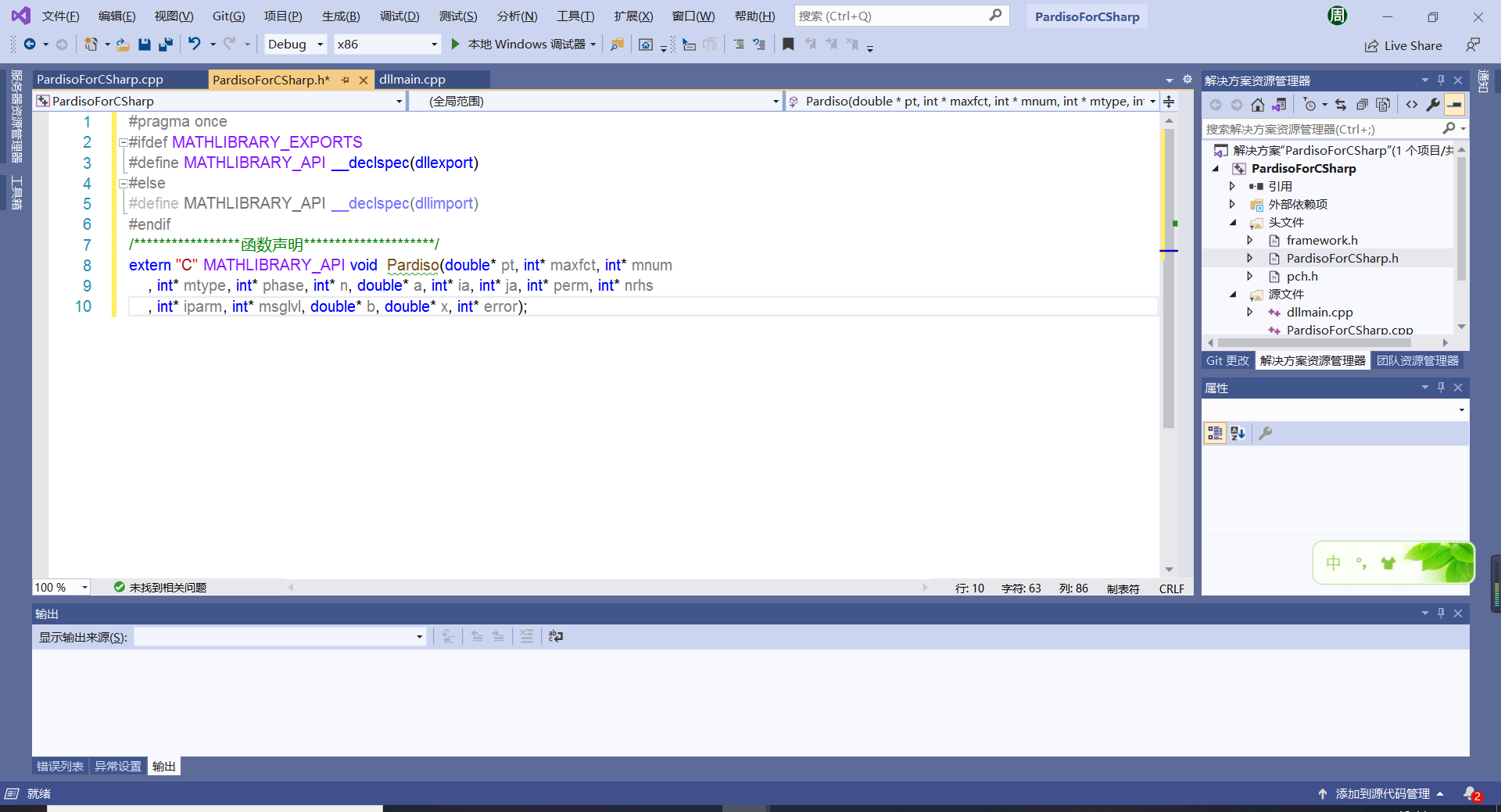


图 2‑3 头文件代码

1. 现在，我们需要按以下步骤添加一个预定义：MATHLIBRARY\_EXPORTS

**第一步**：点击菜单栏中的“项目（P）”选项卡，然后点击下拉菜单中的“PardisoForCSharp属性（P）”，如图 2‑4所示。

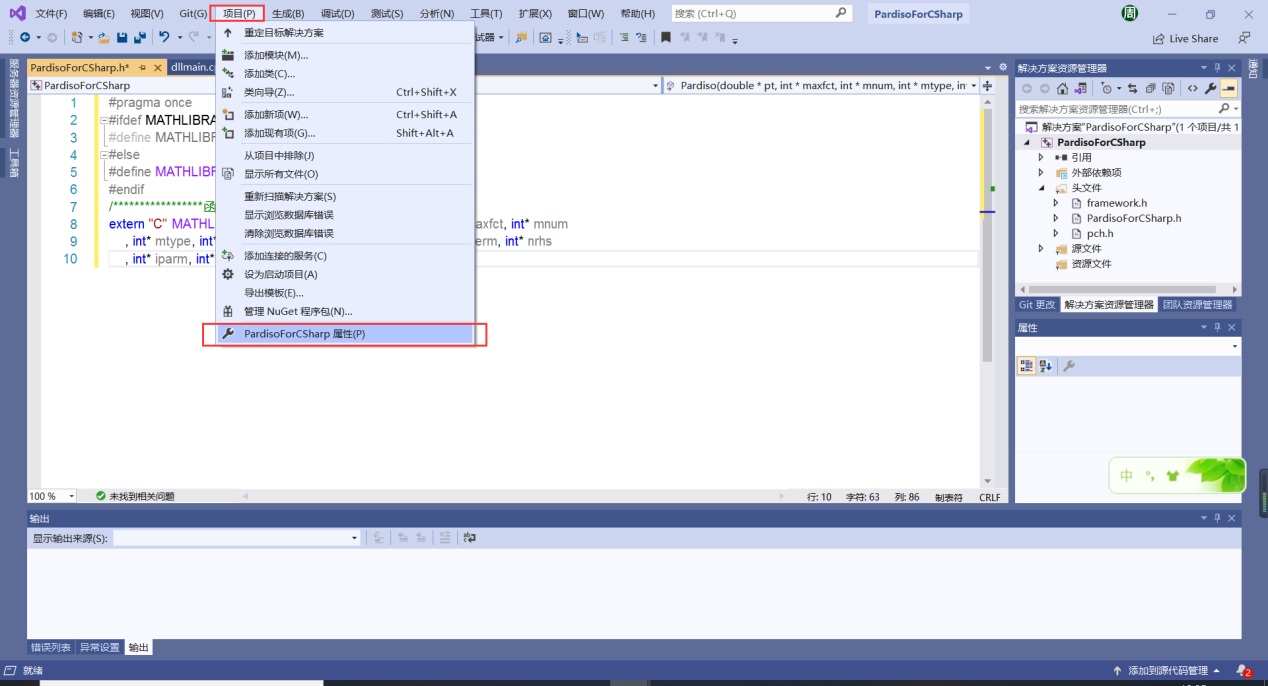


图 2‑4 查看PardisoForCSharp项目属性

**第二步**：进入到“PardisoForCSharp属性页”对话框，打开“C/C++”，选择“预处理器”，点击“预处理器定义”，然后点击右侧下三角符号，最后点击“编辑”，如图 2‑5所示。

**第三步**：进入到“预处理定义”对话框，按图 2‑6所示添加内容，然后点击确定，返回到“PardisoForCSharp属性页”对话框，再次点击确定。

至此，头文件部分我们已经创建完毕。

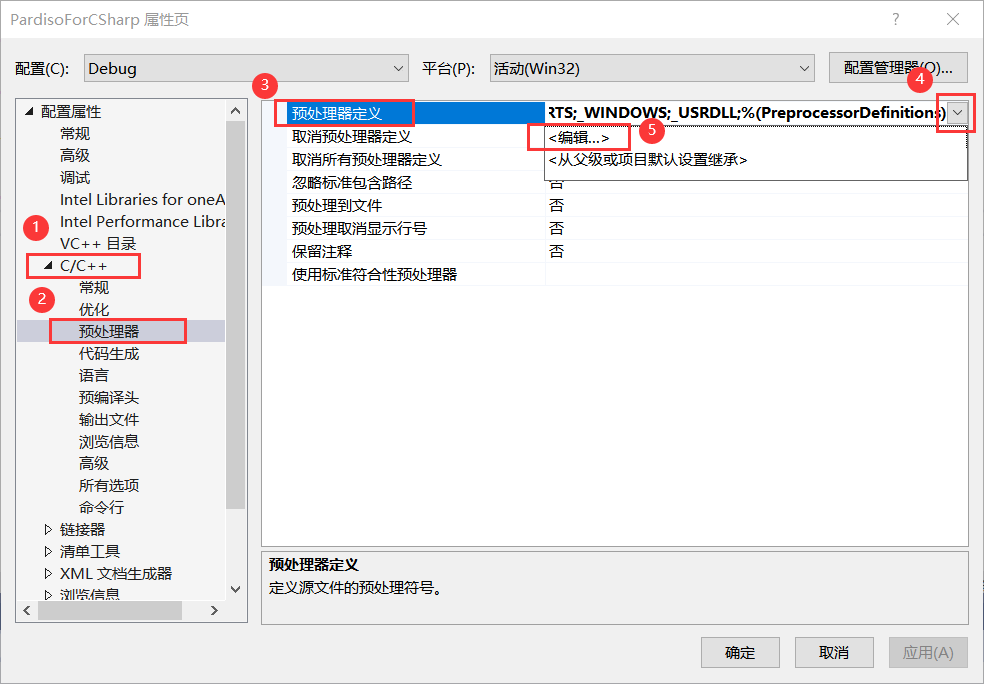


图 2‑5 添加预处理定义

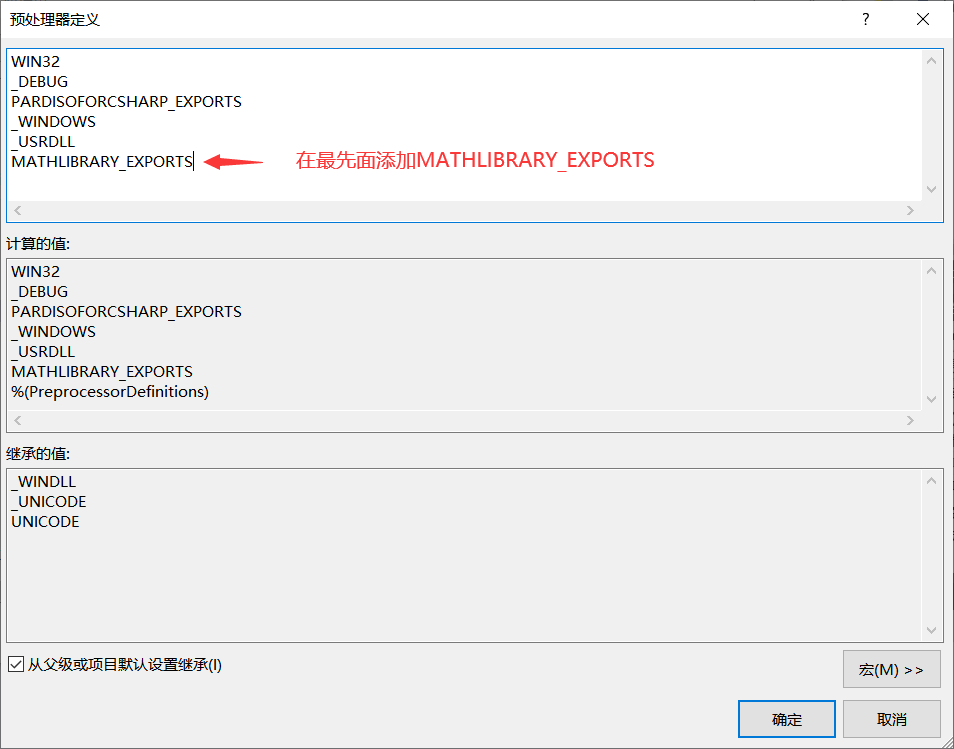


图 2‑6 添加预处理定义

## 向DLL项目中添加源文件

源文件名称：PardisoForCSharp.cpp

1. 右击解决方案资源管理器中的“源文件”，依次选择“添加（D）”“新建项（W）…”，如图 2‑7所示。

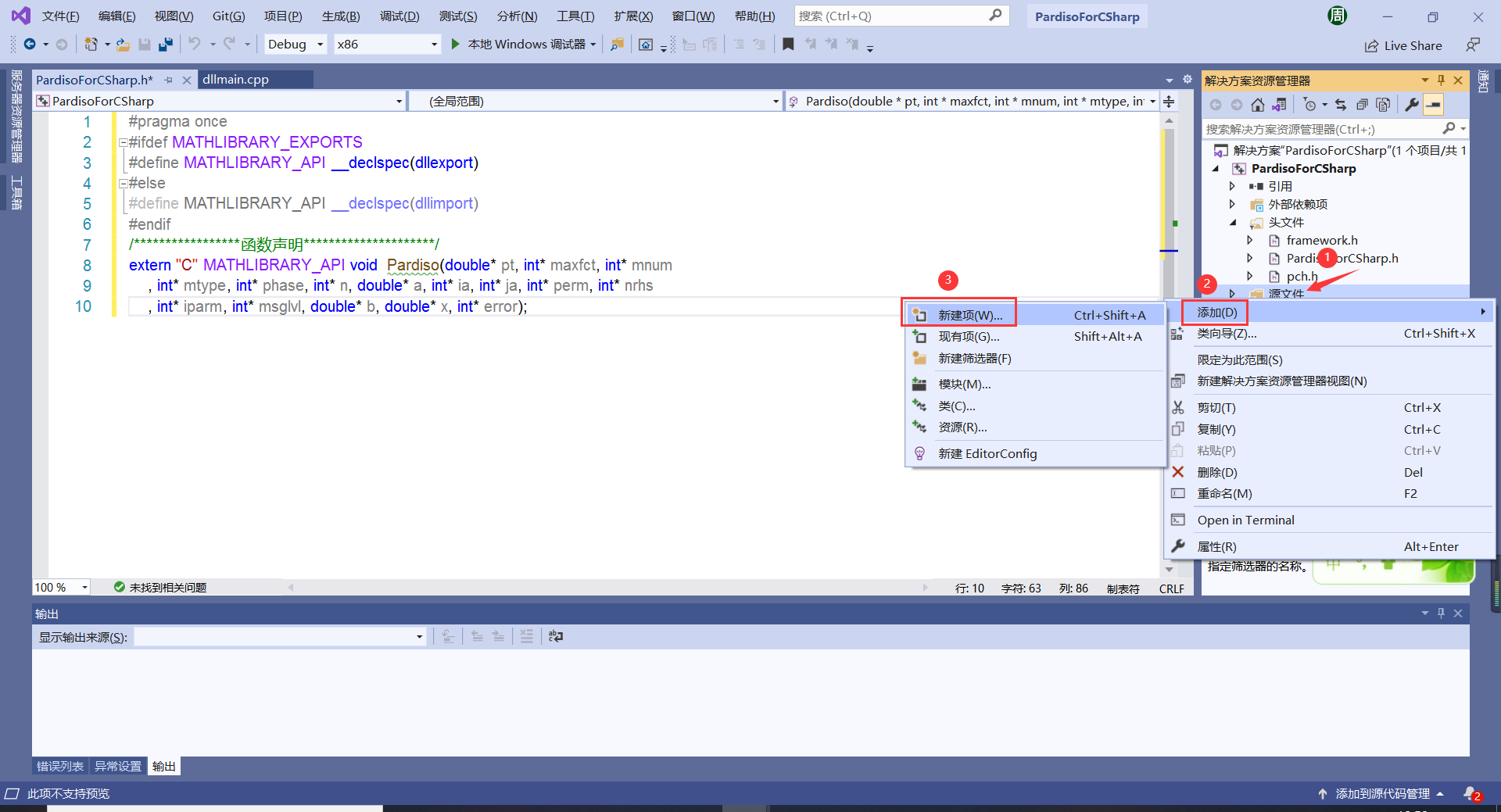


图 2‑7 添加源文件

1. 得到图 2‑8对话框，点击“C++文件（.cpp）”，输入源文件名称“PardisoForCSharp.cpp”，然后点击“添加（A）”。现在，我们已经在项目中新添加了一个空的源文件。

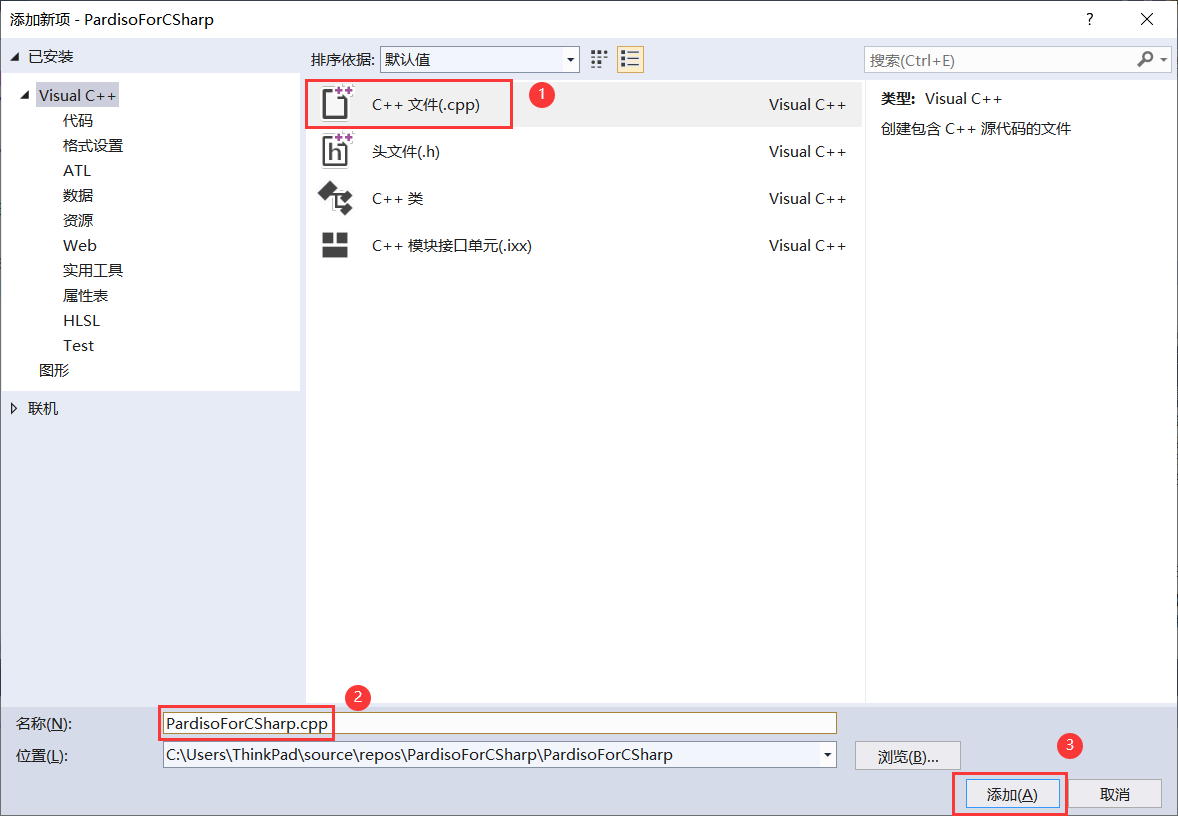


图 2‑8 添加源文件

1. 向源文件中写入以下代码，用作函数的定义，结果如图 2‑9所示。

至此，我们完成了动态链接库（DLL）项目的创建、与MKL的配置、头文件源文件的创建和代码的编写。

|  |
| --- |
| #include "pch.h" // use stdafx.h in Visual Studio 2017 and earlier  #include <utility>//包含标准库文件  #include <limits.h>//包含标准库文件  #include "PardisoForCSharp.h"//包含用户自定义头文件  #include"mkl.h"//包含MKL自定义头文件  #include <stdio.h>//包含标准库文件  #include "mkl\_pardiso.h"//包含MKL自定义头文件  #include "mkl\_types.h"//包含MKL自定义头文件  #include "mkl\_spblas.h"//包含MKL自定义头文件  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*函数定义\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void Pardiso(double\* pt, int\* maxfct, int\* mnum, int\* mtype, int\* phase, int\* n, double\* a, int\* ia, int\* ja  , int\* perm, int\* nrhs, int\* iparm, int\* msglvl, double\* b, double\* x, int\* error)  {  pardiso(pt, maxfct, mnum, mtype, phase, n, a, ia, ja, perm, nrhs, iparm, msglvl, b, x, error);  } |

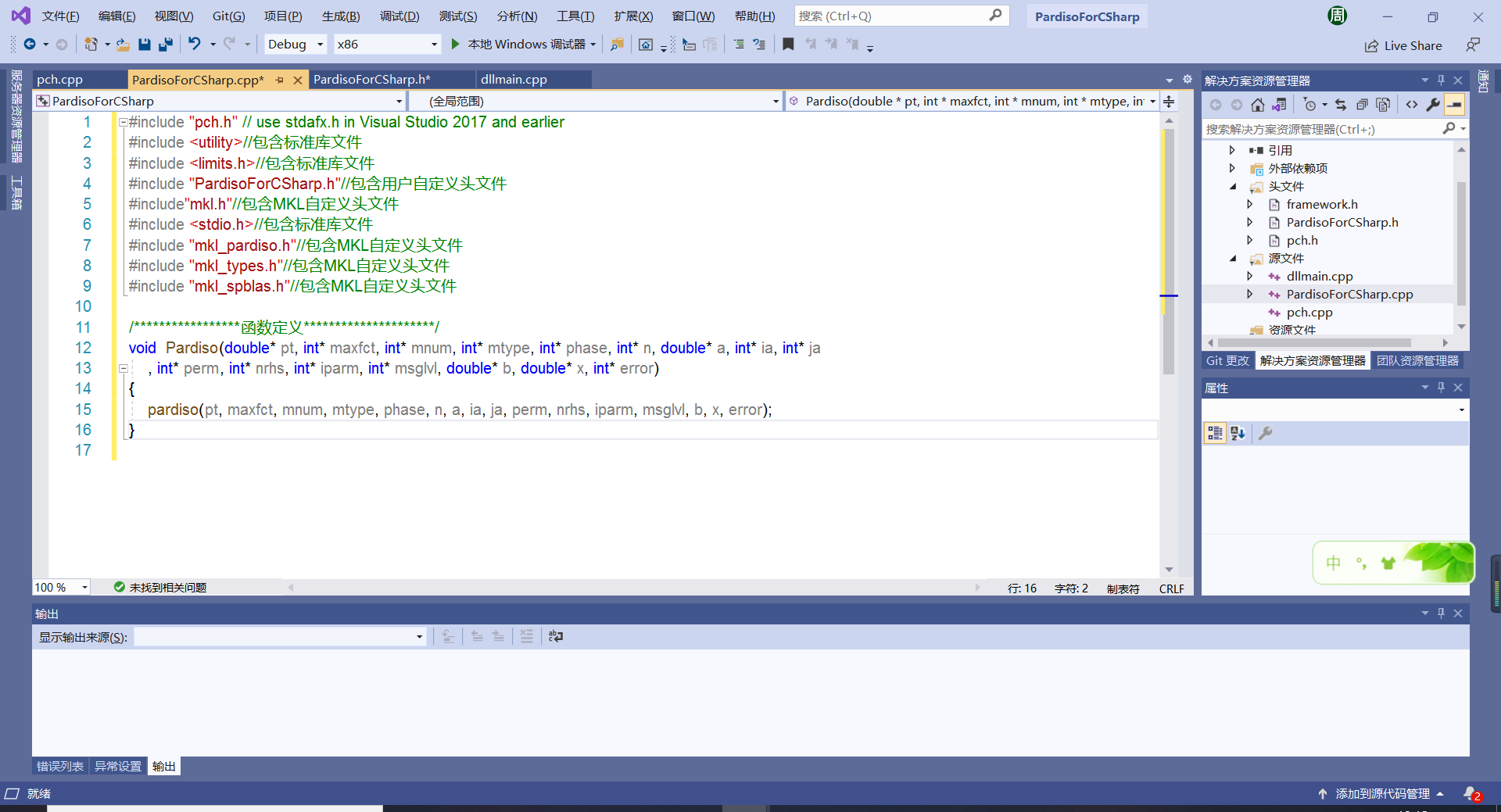


图 2‑9 源文件代码

## 生成DLL文件

现在，我们只差最后一步：生成DLL文件。

点击菜单栏上的“生成（B）”选项卡，然后点击下拉菜单中的“生成PardisoForCSharp（U）”即可，如图 2‑10所示。

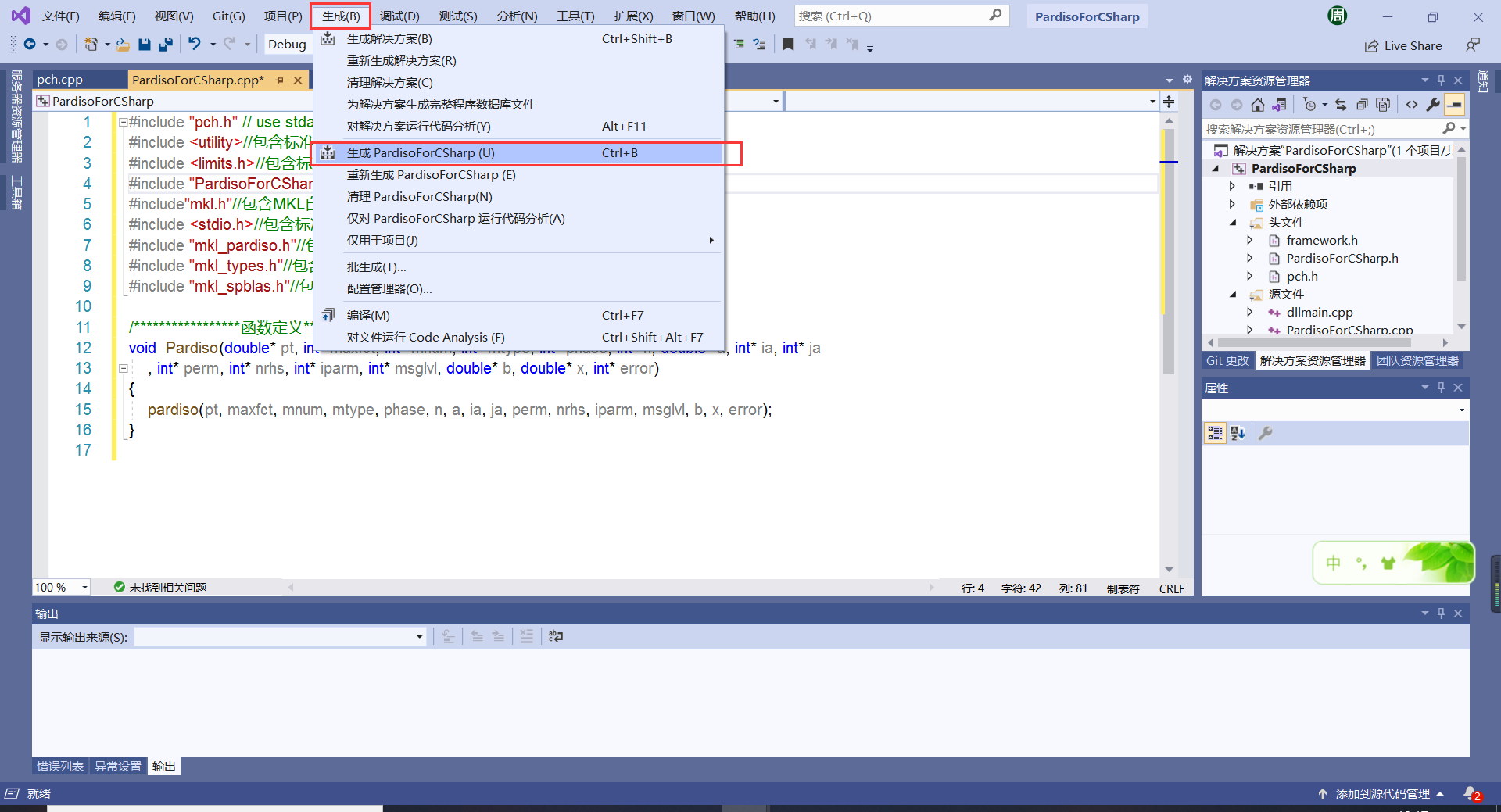


图 2‑10 DLL文件生成

## 查找已生成的DLL文件

生成的DLL文件保存在PardisoForCSharp项目的Debug文件夹中。

之前我们在创建项目时，已经保存了项目的“位置”信息，如图 2‑11所示。

现在，我们按照路径找到“repos”文件夹，并找到PardisoForCSharp文件夹。

如图 2‑12所示。

现在，我们按照以下顺序依次打开文件。

* 打开“PardisoForCSharp”文件夹
* 打开“x64”文件夹
* 打开“Debug”文件夹

Debug文件夹内容如图 2‑13所示。其中，文件“PardisoForCSharp.dll”为动态链接库文件，也是我们在SuperTire软件中需要用到的文件。

图形用户界面, 文本, 应用程序, 电子邮件

AI 生成的内容可能不正确。

图 2‑11 查看PardisoForCSharp项目的位置信息

图形用户界面

AI 生成的内容可能不正确。

图 2‑12 PardisoForCSharp项目文件夹

图形用户界面, 文本, 应用程序, 电子邮件

AI 生成的内容可能不正确。

图 2‑13 PardisoForCSharp.dll文件

## 查找PARDISO依赖文件

按照上述方法生成的PardisoForCSharp.dll不能单独使用，还需从Intel MKL安装文件夹中查找到PARDISO依赖文件：libiomp5md\_db.dll。即PardisoForCSharp.dll和libiomp5md\_db.dll这两个文件必须同时导入到SuperTire软件中才能发挥作用。在第3章中将介绍文件如何导入到SuperTire软件中。

在第1章1.1节我们介绍了“Intel MKL的安装”，期间我们保存了软件安装位置，如图 2‑14所示（实际安装位置为用户最初保存的软件安装位置）。

图形用户界面, 文本, 应用程序, 电子邮件, Teams

AI 生成的内容可能不正确。

图 2‑14 查看MKL软件安装的位置信息

首先，我们找到“oneAPI”文件夹，本教程的“oneAPI”文件夹路径为： C:\Program Files (x86)\Intel\oneAPI

然后我们依次打开文件夹：compiler →2025.3 → lib

本教程打开路径最终显示为：

C:\Program Files(x86) \Intel\oneAPI\compiler\2025.3\lib

在右侧搜索框中直接搜索“libiomp5md\_db.dll”，即可找到依赖文件。如图2-15所示。

图形用户界面, 应用程序, 表格

AI 生成的内容可能不正确。

图 2‑15 查找PARDISO依赖文件

# C#调用动态链接库（DLL）

注意，此动态链接库（DLL）无论在哪台电脑，哪个版本的Visual Studio上使用，都要保证Visual Studio 安装了“**使用C++的桌面开发**”模块！

## DLL文件存放位置

在Visual Studio 平台上开发的软件，解决方案资源管理器下一般有许多个项目，但是一般也只有一个启动项目。对于SuperTire软件来说，启动项目为“PreProcess”项目，如图 3‑1所示。

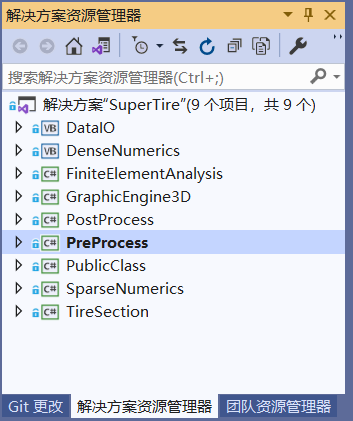


图 3‑1 解决方案资源管理器

而之前生成的PardisoForCSharp.dll文件应当放在启动项目文件夹（PreProcess）下的Debug文件夹中。

**注意：启动项目文件夹（PreProcess）下有两个Debug文件夹，分别对应32位和64位操作系统，由于我们的SuperTire软件为64位程序，以下操作将仅针对64位文件夹。**

首先，我们找到“PreProcess”文件夹，如图 3‑2



图 3‑2 SuperTire软件的项目文件夹

然后我们依次打开文件夹PreProcess→bin→x64→Debug，如图 3‑3所示的文件打开路径（文件夹内容可能不同）。

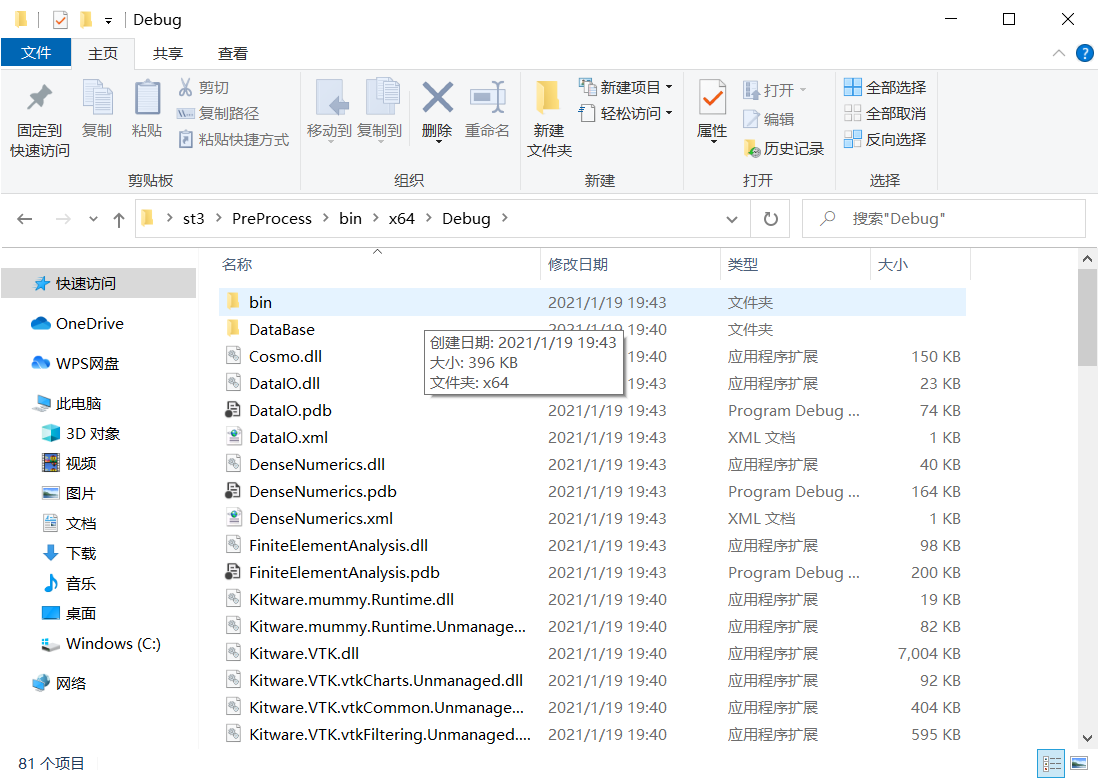


图 3‑3 DLL文件存放位置

现在，我们把“PardisoForCSharp.dll”和“libiomp5md\_db.dll”两个文件复制到上述Debug文件夹中，如图 3‑4所示。

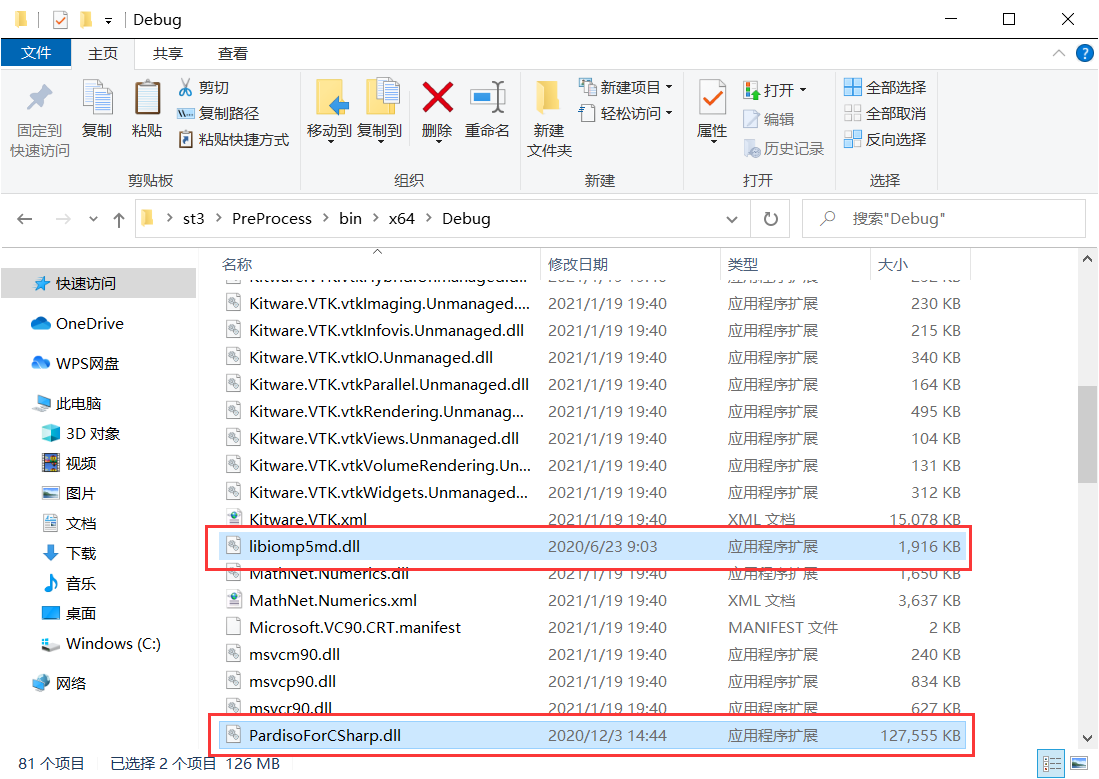


图 3‑4 复制DLL文件到SuperTire

## DLL导入C#程序与Pardiso函数声明

在3.1节中，我们已经将PardisoForCSharp.dll及其依赖文件libiomp5md\_db.dll存放到SuperTire软件的Debug文件夹中。在3.2节，我们将学习如何把PardisoForCSharp.dll文件导入到程序中并使用DLL中封装的Pardiso函数。

首先，我们启动SuperTire软件，启动之后的窗口如图 3‑5所示。需要注意的是，应首先将程序设置为“x64”（如果为x86），如图 3‑5红框所示。

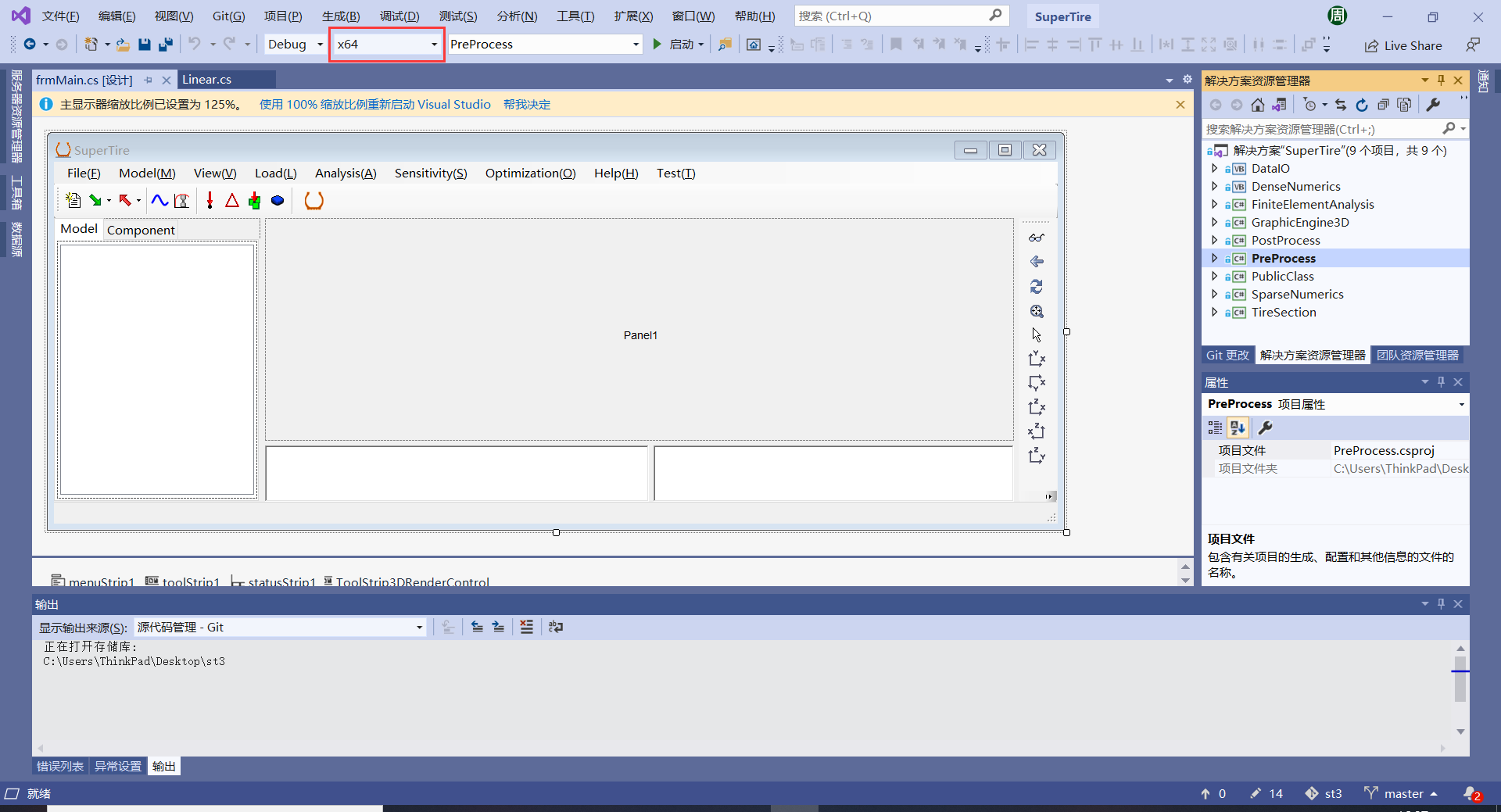


图 3‑5 SuperTire软件设计界面

然后，我们依次点击解决方案资源管理器下的“SparseNumerics”、“Equation”前面的小三角符号。找到“Linear.cs”并单击。

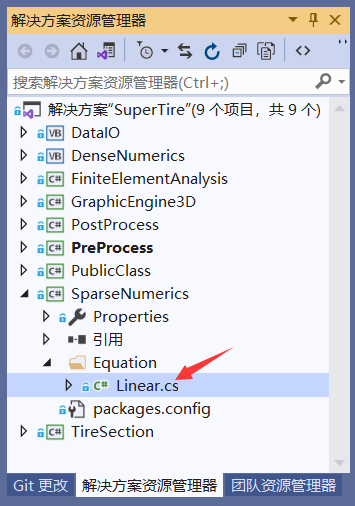


图 3‑6 解决方案资源管理器

我们首先在Linear库中写入以下代码用来导入动态链接库文件和声明动态链接库中的导出函数Pardiso，如图 3‑7所示。

|  |
| --- |
| //导入动态链接库文件  [DllImport(@"\PardisoForCSharp.dll", CallingConvention = CallingConvention.Cdecl)]  //声明动态链接库中的导出函数  private static extern void Pardiso(double[] pt, ref int maxfct, ref int mnum, ref int mtype, ref int phase, ref int n, double[] a, int[] ia, int[] ja  , int[] perm, ref int nrhs, int[] iparm, ref int msglvl, double[] b, double[] x, ref int error); |

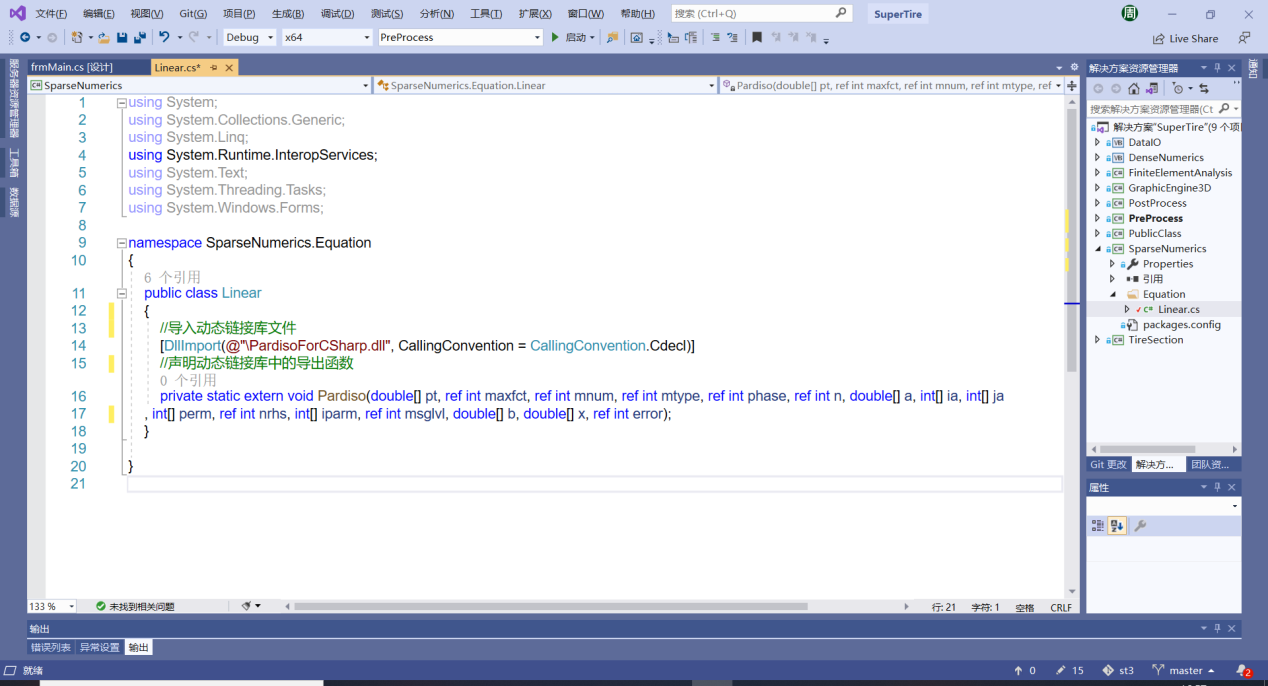


图 3‑7 DLL导入与Pardiso函数声明代码

## 参数设置

对于线性系统

我们需要用Pardiso来求解为对称矩阵和非对称矩阵的线性系统。而Pardiso又分为“in-core(IC)”和“out-of-core(OOC)”两种模式。两两组合我们需要编写四个函数来使用，如图 3‑8所示。

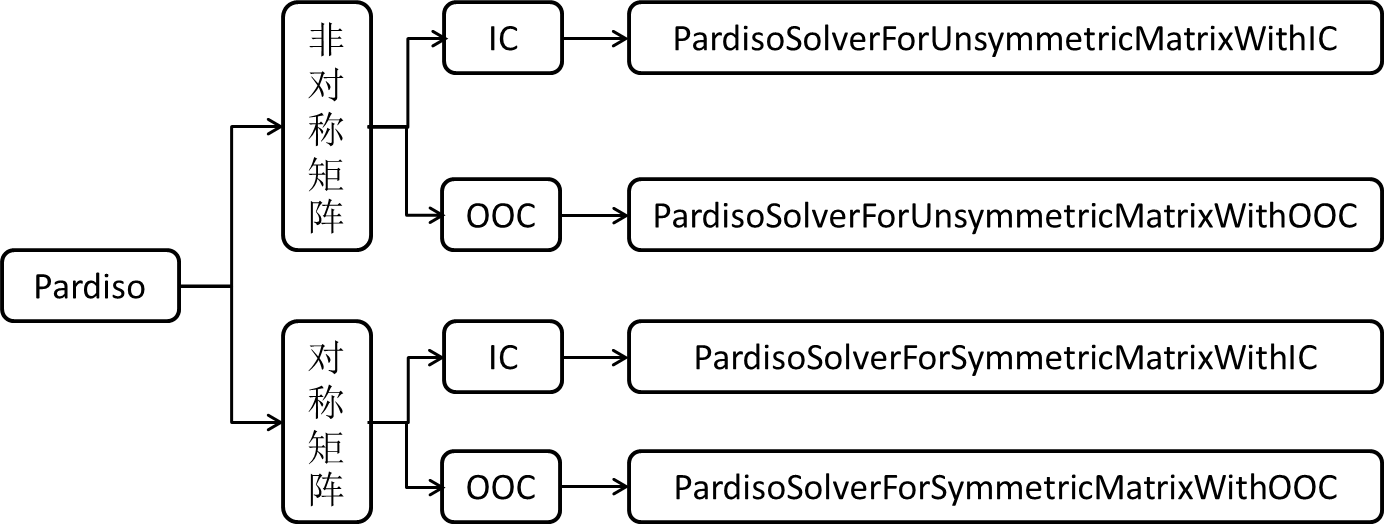


图 3‑8 不同功能不同模式下的Pardiso函数命名

对于每个函数，需要设置不同的参数，接下来，我们分别编写四个函数对应的C#代码（含注释）。

1. 对于函数PardisoSolverForUnsymmetricMatrixWithIC，写入以下代码：

|  |
| --- |
| /// <summary>  /// PardisoSolverForUnsymmetricMatrixWithIC——自定义Pardiso求解器函数，适用于非对称矩阵  /// 可以用来求解具有单个或多个右端项的稀疏线性方程组（in-Core模式）  /// </summary>  /// <param name="n">矩阵维数</param>  /// <param name="a">CSR存储格式——存储矩阵非零元素的一维数组</param>  /// <param name="ia">CSR存储格式——存储矩阵行索引值的一维数组</param>  /// <param name="ja">CSR存储格式——存储矩阵列索引值的一维数组</param>  /// <param name="b">存储载荷的列向量</param>  /// <param name="x">存储Pardiso求解器的计算结果</param>  /// <param name="error">返回分析、数值分解、求解阶段出现的错误</param>  public static void PardisoSolverForUnsymmetricMatrixWithIC(int n, double[] a, int[] ia, int[] ja, double[] b, double[] x, ref int error)  {  int mtype = 11;//实数非对称矩阵类型  int nrhs = 1;//右端项数量  //内部求解器内存指针  //32位系统上定义为：int pt[64];  //64位系统上定义为：long int pt[64];  //或者定义为：void \*pt[64];这对于两种系统结构应该都是可以的  double[] pt = new double[64];  int[] iparm = new int[64];//Pardiso控制参数  int maxfct = 1, mnum = 1, msglvl = 0;  int phase;  /\* -------------------------------------------------------------------- \*/  /\* 设置Pardiso控制参数\*/  /\* -------------------------------------------------------------------- \*/  for (int i = 0; i < 64; i++)  {  iparm[i] = 0;  }  iparm[0] = 1; // 不使用求解器默认iparm参数  iparm[1] = 2; // METIS软件包中的嵌套剖分算法  iparm[3] = 0; // 无迭代直接算法  iparm[4] = 0; // 不使用用户填充减少排列  iparm[5] = 0; // 将解写入x数组  iparm[6] = 0; // （输出参数）执行的迭代优化步骤的数量  iparm[7] = 0; // 迭代优化步骤数  iparm[8] = 0; // 默认设置为0  iparm[9] = 13; // 旋转摄动，非对称矩阵的默认值（mtype=11或mtype=13）  iparm[10] = 1; // 启用缩放向量，非对称矩阵的默认值  iparm[11] = 0; // 求解线性系统AX=B  iparm[12] = 1; // 启用最大加权匹配算法 （非对称矩阵的默认值）  iparm[13] = 0; // （输出参数）旋转摄动数  iparm[14] = 0; // （输出参数）符号分解阶段的峰值内存  iparm[15] = 0; // （输出参数）符号分解阶段的永久内存  iparm[16] = 0; // （输出参数）数值分解和求解阶段峰值内存的大小  iparm[17] = -1; // （输出参数）LU分解阶段非零元素的数量  iparm[18] = -1; // （输出参数）报告分解矩阵A所需的浮点运算数  iparm[19] = 0; // （输出参数）报告CG/CGS诊断  iparm[24] = 2; // 并行正向/反向求解控制  iparm[26] = 1; // 启用矩阵检查器  iparm[34] = 1; // 行索引和列索引从0开始  /\* -------------------------------------------------------------------- \*/  /\* 初始化内部求解器内存指针 \*/  /\* 这对于第一次调用Pardiso求解器是必须的 \*/  /\* -------------------------------------------------------------------- \*/  for (int i = 0; i < 64; i++)  {  pt[i] = 0;  }  int[] perm = new int[n];//用户自定义填充减少排列，仅在iparm[4]=1时生效  for (int i = 0; i < n; i++)  {  perm[i] = 0;  }  /\* -------------------------------------------------------------------- \*/  /\* 分析、数值分解、求解、迭代细化 \*/  /\* -------------------------------------------------------------------- \*/  phase = 13;  Pardiso(pt, ref maxfct, ref mnum, ref mtype, ref phase, ref n, a, ia, ja, perm, ref nrhs, iparm, ref msglvl, b, x, ref error);  /\* -------------------------------------------------------------------- \*/  /\* 内存的终止与释放 \*/  /\* -------------------------------------------------------------------- \*/  phase = -1;// 释放内部内存  Pardiso(pt, ref maxfct, ref mnum, ref mtype, ref phase, ref n, a, ia, ja, perm, ref nrhs, iparm, ref msglvl, b, x, ref error);  } |

1. 对于函数PardisoSolverForUnsymmetricMatrixWithOOC，写入以下代码：

|  |
| --- |
| /// <summary>  /// PardisoSolverForUnsymmetricMatrixWithOOC——自定义Pardiso求解器函数，适用于非对称矩阵  /// 可以用来求解具有单个或多个右端项的稀疏线性方程组（Out Of Core模式）  /// </summary>  /// <param name="n">矩阵维数</param>  /// <param name="a">CSR存储格式——存储矩阵非零元素的一维数组</param>  /// <param name="ia">CSR存储格式——存储矩阵行索引值的一维数组</param>  /// <param name="ja">CSR存储格式——存储矩阵列索引值的一维数组</param>  /// <param name="b">存储载荷的列向量</param>  /// <param name="x">存储Pardiso求解器的计算结果</param>  /// <param name="error">返回分析、数值分解、求解阶段出现的错误</param>  public static void PardisoSolverForUnsymmetricMatrixWithOOC(int n, double[] a, int[] ia, int[] ja, double[] b, double[] x, ref int error)  {  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*设置OOC模式所需环境变量\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  string ComputerMemory;  ComputerMemory = Convert.ToString(Convert.ToInt32(PublicClass.Computer.GetPhisicalMemory()) \* 1024 \* 4 / 5);//取电脑总RAM的五分之四用于PARDISO计算  Environment.SetEnvironmentVariable("MKL\_PARDISO\_OOC\_MAX\_CORE\_SIZE", ComputerMemory, EnvironmentVariableTarget.Process);//可用于 PARDISO 的 RAM 的最大大小（默认值为 2000 MB）  Environment.SetEnvironmentVariable("MKL\_PARDISO\_OOC\_MAX\_SWAP\_SIZE", "0", EnvironmentVariableTarget.Process);//可用于 PARDISO 的最大交换大小（默认值为 0 MB）  Environment.SetEnvironmentVariable("MKL\_PARDISO\_OOC\_KEEP\_FILE", "1", EnvironmentVariableTarget.Process);//如果将其设置为 1（默认值），则删除所有临时文件;如果将其设置为 0，则存储所有临时文件。  int mtype = 11;//实数非对称矩阵类型  int nrhs = 1;//右端项数量  //内部求解器内存指针  //32位系统上定义为：int pt[64];  //64位系统上定义为：long int pt[64];  //或者定义为：void \*pt[64];这对于两种系统结构应该都是可以的  double[] pt = new double[64];  int[] iparm = new int[64];//Pardiso控制参数  int maxfct = 1, mnum = 1, msglvl = 0;  int phase;  /\* -------------------------------------------------------------------- \*/  /\* 设置Pardiso控制参数\*/  /\* -------------------------------------------------------------------- \*/  for (int i = 0; i < 64; i++)  {  iparm[i] = 0;  }  iparm[0] = 1; // 不使用求解器默认iparm参数  iparm[1] = 3; // METIS软件包中的嵌套剖分算法(OpenMP并行)  iparm[3] = 0; // 无迭代直接算法  iparm[4] = 0; // 不使用用户填充减少排列  iparm[5] = 0; // 将解写入x数组  iparm[6] = 0; // （输出参数）执行的迭代优化步骤的数量  iparm[7] = 0; // 迭代优化步骤数  iparm[8] = 0; // 默认设置为0  iparm[9] = 13; // 旋转摄动，非对称矩阵的默认值（mtype=11或mtype=13）  iparm[10] = 1; // 启用缩放向量，非对称矩阵的默认值  iparm[11] = 0; // 求解线性系统AX=B  iparm[12] = 1; // 启用最大加权匹配算法 （非对称矩阵的默认值）  iparm[13] = 0; // （输出参数）旋转摄动数  iparm[14] = 0; // （输出参数）符号分解阶段的峰值内存  iparm[15] = 0; // （输出参数）符号分解阶段的永久内存  iparm[16] = 0; // （输出参数）数值分解和求解阶段峰值内存的大小  iparm[17] = -1; // （输出参数）LU分解阶段非零元素的数量  iparm[18] = -1; // （输出参数）报告分解矩阵A所需的浮点运算数  iparm[19] = 0; // （输出参数）报告CG/CGS诊断  iparm[24] = 1; // 并行正向/反向求解控制  iparm[26] = 1; // 启用矩阵检查器  iparm[34] = 1; // 行索引和列索引从0开始  iparm[59] = 2; // Pardiso使用Out Of Core模式  /\* -------------------------------------------------------------------- \*/  /\* 初始化内部求解器内存指针 \*/  /\* 这对于第一次调用Pardiso求解器是必须的 \*/  /\* -------------------------------------------------------------------- \*/  for (int i = 0; i < 64; i++)  {  pt[i] = 0;  }  int[] perm = new int[n];//用户自定义填充减少排列，仅在iparm[4]=1时生效  for (int i = 0; i < n; i++)  {  perm[i] = 0;  }  /\* -------------------------------------------------------------------- \*/  /\* 分析、数值分解、求解、迭代细化 \*/  /\* -------------------------------------------------------------------- \*/  phase = 13;  Pardiso(pt, ref maxfct, ref mnum, ref mtype, ref phase, ref n, a, ia, ja, perm, ref nrhs, iparm, ref msglvl, b, x, ref error);  /\* -------------------------------------------------------------------- \*/  /\* 内存的终止与释放 \*/  /\* -------------------------------------------------------------------- \*/  phase = -1;// 释放内部内存  Pardiso(pt, ref maxfct, ref mnum, ref mtype, ref phase, ref n, a, ia, ja, perm, ref nrhs, iparm, ref msglvl, b, x, ref error);  } |

1. 对于函数PardisoSolverForSymmetricMatrixWithIC，写入以下代码：

|  |
| --- |
| /// <summary>  /// PardisoSolverForSymmetricMatrixWithIC——自定义Pardiso求解器函数，适用于对称矩阵  /// 可以用来求解具有单个或多个右端项的稀疏线性方程组（in-Core模式）  /// </summary>  /// <param name="n">矩阵维数</param>  /// <param name="a">CSR存储格式——存储矩阵非零元素的一维数组</param>  /// <param name="ia">CSR存储格式——存储矩阵行索引值的一维数组</param>  /// <param name="ja">CSR存储格式——存储矩阵列索引值的一维数组</param>  /// <param name="b">存储载荷的列向量</param>  /// <param name="x">存储Pardiso求解器的计算结果</param>  /// <param name="error">返回分析、数值分解、求解阶段出现的错误</param>  public static void PardisoSolverForSymmetricMatrixWithIC(int n, double[] a, int[] ia, int[] ja, double[] b, double[] x, ref int error)  {  int mtype = -2;//实数对称矩阵类型  int nrhs = 1;//右端项数量  //内部求解器内存指针  //32位系统上定义为：int pt[64];  //64位系统上定义为：long int pt[64];  //或者定义为：void \*pt[64];这对于两种系统结构应该都是可以的  double[] pt = new double[64];  int[] iparm = new int[64];//Pardiso控制参数  int maxfct = 1, mnum = 1, msglvl = 0;  int phase;  /\* -------------------------------------------------------------------- \*/  /\* 设置Pardiso控制参数\*/  /\* -------------------------------------------------------------------- \*/  for (int i = 0; i < 64; i++)  {  iparm[i] = 0;  }  iparm[0] = 1; // 不使用求解器默认iparm参数  iparm[1] = 2; // METIS软件包中的嵌套剖分算法  iparm[3] = 0; // 无迭代直接算法  iparm[4] = 0; // 不使用用户填充减少排列  iparm[5] = 0; // 将解写入x数组  iparm[6] = 0; // （输出参数）执行的迭代优化步骤的数量  iparm[7] = 0; // 迭代优化步骤数  iparm[8] = 0; // 默认设置为0  iparm[9] = 8; // 旋转摄动，对称不定矩阵的默认值（mtype=-2或mtype=-4或mtype=6）  iparm[10] = 0; // 禁用缩放向量，对称不定矩阵的默认值  iparm[11] = 0; // 求解线性系统AX=B  iparm[12] = 0; // 禁用最大加权匹配算法 （对称不定矩阵的默认值）  iparm[13] = 0; // （输出参数）旋转摄动数  iparm[14] = 0; // （输出参数）符号分解阶段的峰值内存  iparm[15] = 0; // （输出参数）符号分解阶段的永久内存  iparm[16] = 0; // （输出参数）数值分解和求解阶段峰值内存的大小  iparm[17] = -1; // （输出参数）LU分解阶段非零元素的数量  iparm[18] = -1; // （输出参数）报告分解矩阵A所需的浮点运算数  iparm[19] = 0; // （输出参数）报告CG/CGS诊断  iparm[24] = 2; // 并行正向/反向求解控制  iparm[26] = 1; // 启用矩阵检查器  iparm[34] = 1; // 行索引和列索引从0开始  /\* -------------------------------------------------------------------- \*/  /\* 初始化内部求解器内存指针 \*/  /\* 这对于第一次调用Pardiso求解器是必须的 \*/  /\* -------------------------------------------------------------------- \*/  for (int i = 0; i < 64; i++)  {  pt[i] = 0;  }  int[] perm = new int[n];//用户自定义填充减少排列，仅在iparm[4]=1时生效  for (int i = 0; i < n; i++)  {  perm[i] = 0;  }  /\* -------------------------------------------------------------------- \*/  /\* 分析、数值分解、求解、迭代细化 \*/  /\* -------------------------------------------------------------------- \*/  phase = 13;  Pardiso(pt, ref maxfct, ref mnum, ref mtype, ref phase, ref n, a, ia, ja, perm, ref nrhs, iparm, ref msglvl, b, x, ref error);  /\* -------------------------------------------------------------------- \*/  /\* 内存的终止与释放 \*/  /\* -------------------------------------------------------------------- \*/  phase = -1;// 释放内部内存  Pardiso(pt, ref maxfct, ref mnum, ref mtype, ref phase, ref n, a, ia, ja, perm, ref nrhs, iparm, ref msglvl, b, x, ref error);  } |

1. 对于函数PardisoSolverForSymmetricMatrixWithOOC，写入以下代码：

|  |
| --- |
| /// <summary>  /// PardisoSolverForSymmetricMatrixWithOOC——自定义Pardiso求解器函数，适用于对称矩阵  /// 可以用来求解具有单个或多个右端项的稀疏线性方程组（Out Of Core模式）  /// </summary>  /// <param name="n">矩阵维数</param>  /// <param name="a">CSR存储格式——存储矩阵非零元素的一维数组</param>  /// <param name="ia">CSR存储格式——存储矩阵行索引值的一维数组</param>  /// <param name="ja">CSR存储格式——存储矩阵列索引值的一维数组</param>  /// <param name="b">存储载荷的列向量</param>  /// <param name="x">存储Pardiso求解器的计算结果</param>  /// <param name="error">返回分析、数值分解、求解阶段出现的错误</param>  public static void PardisoSolverForSymmetricMatrixWithOOC(int n, double[] a, int[] ia, int[] ja, double[] b, double[] x, ref int error)  {  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*设置OOC模式所需环境变量\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  string ComputerMemory;  ComputerMemory = Convert.ToString(Convert.ToInt32(PublicClass.Computer.GetPhisicalMemory()) \* 1024 \* 4 / 5);//取电脑总RAM的五分之四用于PARDISO计算  Environment.SetEnvironmentVariable("MKL\_PARDISO\_OOC\_MAX\_CORE\_SIZE", ComputerMemory, EnvironmentVariableTarget.Process);//可用于 PARDISO 的 RAM 的最大大小（默认值为 2000 MB）  Environment.SetEnvironmentVariable("MKL\_PARDISO\_OOC\_MAX\_SWAP\_SIZE", "0", EnvironmentVariableTarget.Process);//可用于 PARDISO 的最大交换大小（默认值为 0 MB）  Environment.SetEnvironmentVariable("MKL\_PARDISO\_OOC\_KEEP\_FILE", "1", EnvironmentVariableTarget.Process);//如果将其设置为 1（默认值），则删除所有临时文件;如果将其设置为 0，则存储所有临时文件。  int mtype = -2;//实数对称矩阵类型  int nrhs = 1;//右端项数量  //内部求解器内存指针  //32位系统上定义为：int pt[64];  //64位系统上定义为：long int pt[64];  //或者定义为：void \*pt[64];这对于两种系统结构应该都是可以的  double[] pt = new double[64];  int[] iparm = new int[64];//Pardiso控制参数  int maxfct = 1, mnum = 1, msglvl = 1;  int phase;  /\* -------------------------------------------------------------------- \*/  /\* 设置Pardiso控制参数\*/  /\* -------------------------------------------------------------------- \*/  for (int i = 0; i < 64; i++)  {  iparm[i] = 0;  }  iparm[0] = 1; // 不使用求解器默认iparm参数  iparm[1] = 2; // METIS软件包中的嵌套剖分算法  iparm[3] = 0; // 无迭代直接算法  iparm[4] = 0; // 不使用用户填充减少排列  iparm[5] = 0; // 将解写入x数组  iparm[6] = 0; // （输出参数）执行的迭代优化步骤的数量  iparm[7] = 0; // 迭代优化步骤数  iparm[8] = 0; // 默认设置为0  iparm[9] = 8; // 旋转摄动，对称不定矩阵的默认值（mtype=-2或mtype=-4或mtype=6）  iparm[10] = 0; // 禁用缩放向量，对称不定矩阵的默认值  iparm[11] = 0; // 求解线性系统AX=B  iparm[12] = 0; // 禁用最大加权匹配算法 （对称不定矩阵的默认值）  iparm[13] = 0; // （输出参数）旋转摄动数  iparm[14] = 0; // （输出参数）符号分解阶段的峰值内存  iparm[15] = 0; // （输出参数）符号分解阶段的永久内存  iparm[16] = 0; // （输出参数）数值分解和求解阶段峰值内存的大小  iparm[17] = -1; // （输出参数）LU分解阶段非零元素的数量  iparm[18] = -1; // （输出参数）报告分解矩阵A所需的浮点运算数  iparm[19] = 0; // （输出参数）报告CG/CGS诊断  iparm[24] = 1; // 并行正向/反向求解控制  iparm[26] = 1; // 启用矩阵检查器  iparm[34] = 1; // 行索引和列索引从0开始  iparm[59] = 2; // Pardiso使用Out Of Core模式  /\* -------------------------------------------------------------------- \*/  /\* 初始化内部求解器内存指针 \*/  /\* 这对于第一次调用Pardiso求解器是必须的 \*/  /\* -------------------------------------------------------------------- \*/  for (int i = 0; i < 64; i++)  {  pt[i] = 0;  }  int[] perm = new int[n];//用户自定义填充减少排列，仅在iparm[4]=1时生效  for (int i = 0; i < n; i++)  {  perm[i] = 0;  }  /\* -------------------------------------------------------------------- \*/  /\* 分析、数值分解、求解、迭代细化 \*/  /\* -------------------------------------------------------------------- \*/  phase = 13;  Pardiso(pt, ref maxfct, ref mnum, ref mtype, ref phase, ref n, a, ia, ja, perm, ref nrhs, iparm, ref msglvl, b, x, ref error);  /\* -------------------------------------------------------------------- \*/  /\* 内存的终止与释放 \*/  /\* -------------------------------------------------------------------- \*/  phase = -1;// 释放内部内存  Pardiso(pt, ref maxfct, ref mnum, ref mtype, ref phase, ref n, a, ia, ja, perm, ref nrhs, iparm, ref msglvl, b, x, ref error);  } |

至此，我们完成了四个函数的编写。

## 函数调用

对Pardiso函数的调用分为两步：

1. 调用函数的项目添加“引用”
2. 调用函数（命名空间→类→方法）

### 项目添加引用

首先介绍项目如何添加引用，这里以前处理项目“PreProcess”为例。

1. 启动SuperTire软件
2. 鼠标右键单击“解决方案资源管理器”中的“PreProcess”项目，如图 3‑9所示。

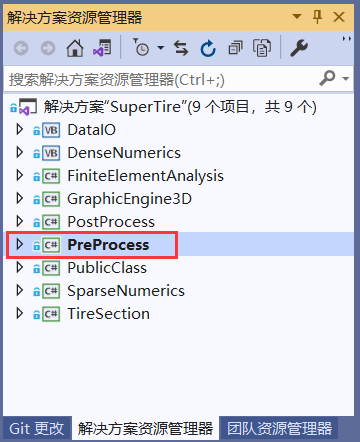


图 3‑9 解决方案资源管理器

1. 依次点击“添加(D)”、“引用(R)…”，如图 3‑10所示。进入“引用管理器-PreProcess”对话框，勾选“SparseNumerics”复选框，然后点击“确定”即可。

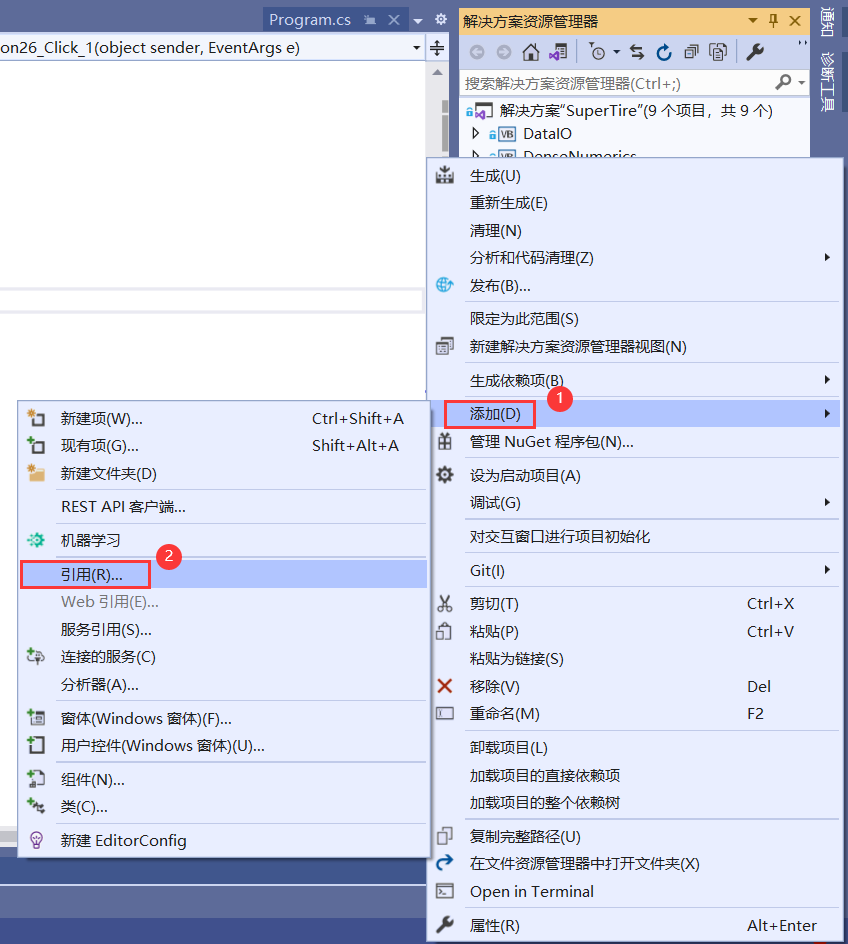


图 3‑10 添加引用

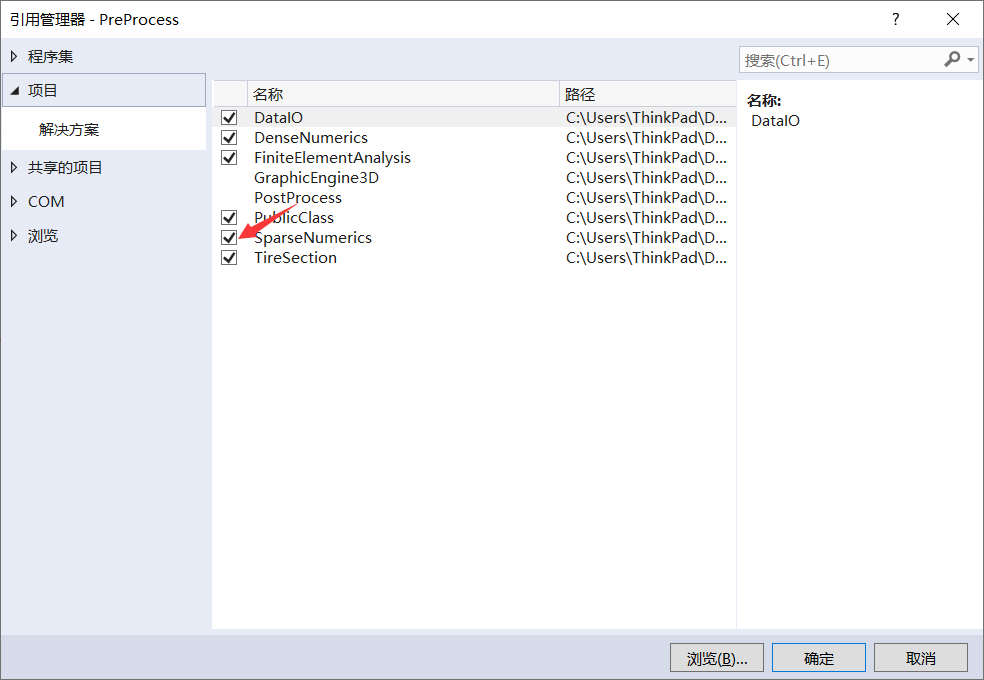


图 3‑11 添加SparseNumerics引用

### 函数调用示例

SuperTire软件具有测试功能，首先我们找到测试窗口“frmTest.cs”，分为以下步骤。

1. 点击“解决方案资源管理器”下的前处理项目“PreProcess”项目前的三角符号，如图 3‑12所示。

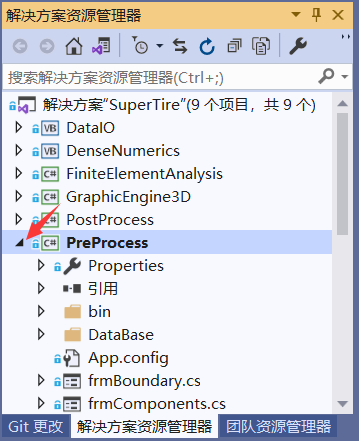


图 3‑12 解决方案资源管理器

1. 在“PreProcess” 二级目录中找到“frmTest.cs”窗体应用程序并双击，如图 3‑13所示，得到测试窗体如图 3‑14所示。

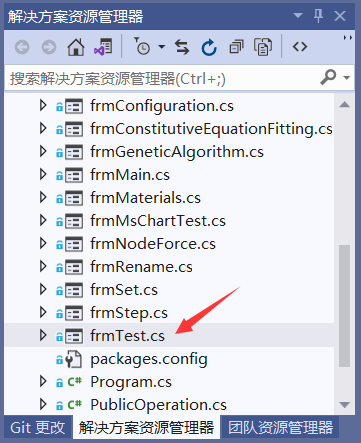


图 3‑13 解决方案资源管理器

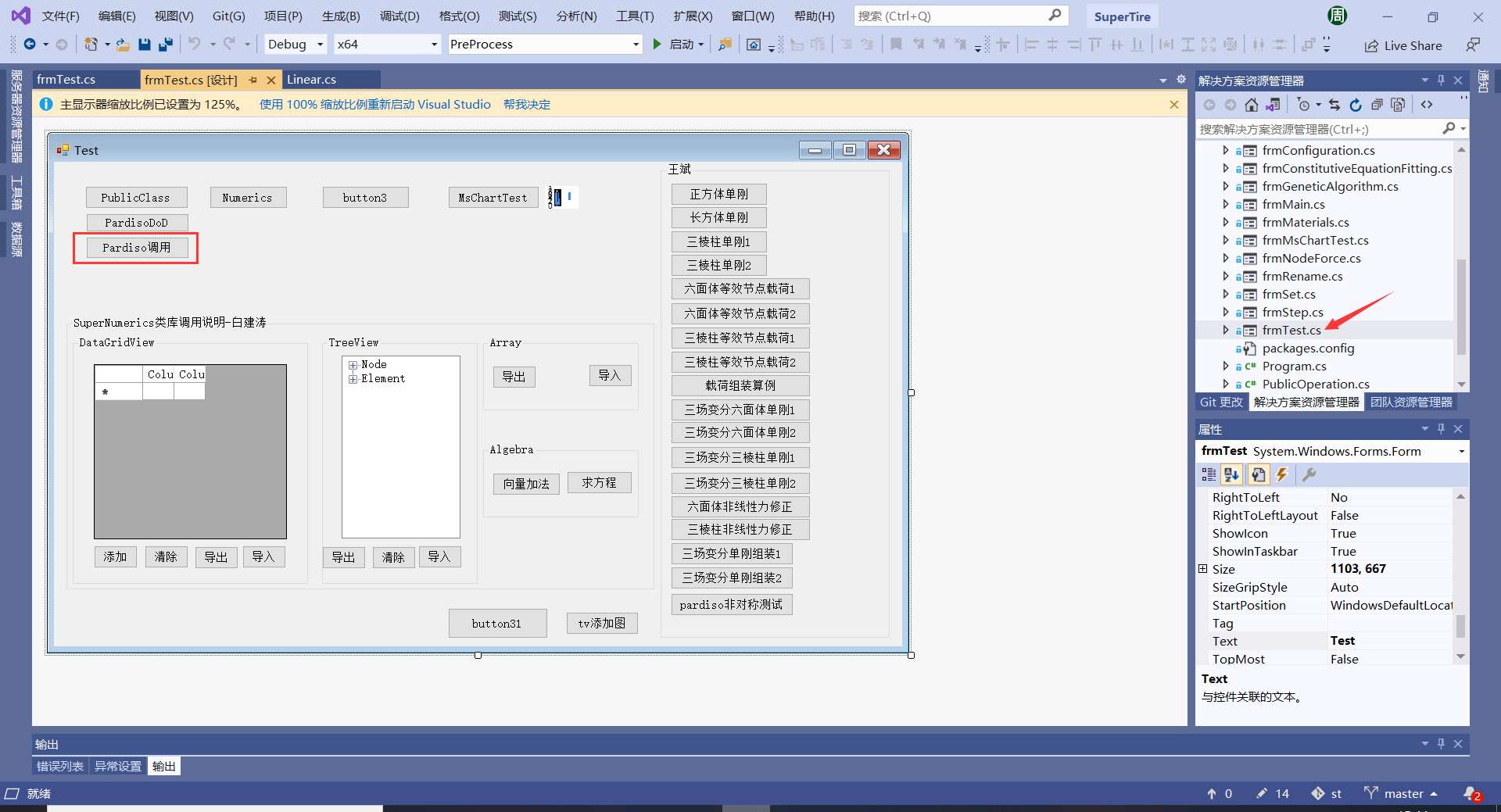


图 3‑14 SuperTire测试窗口

1. 在测试窗体中找到“Pardiso调用”按钮（图 3‑14红框位置）并双击，进入代码界面，“Pardiso调用”按钮下的代码为：

|  |
| --- |
| int error = 0;  int n = 8;  int[] ia = { 0, 4, 7, 9, 11, 14, 16, 17, 18 };  int[] ja = {0,2,5,6,  1,2,4,  2,7,  3,6,  4,5,6,  5,7,  6,  7  };  double[] a =  {7.0,1.0,2.0,7.0,  -4.0,8.0,2.0,  1.0,5.0,  7.0,9.0,  5.0,1.0,5.0,  -1.0,5.0,  11.0,  5.0  };  double[] b = new double[n];  for (int i = 0; i < n; ++i)  {  b[i] = 1;  }  double[] x = new double[n];  SparseNumerics.Equation.Linear.PardisoSolverForUnsymmetricMatrixWithIC(n, a, ia, ja, b, x, ref error);  for (int i = 0; i < n; ++i)  {  Console.WriteLine(string.Format("x\_{0}={1}", i, x[i]));  } |

代码分为数据准备部分和调用函数求解部分。

对于方程组

**数据准备部分代码解释**

n：系数矩阵的维数。

a：包含了与ja中各指标对应的系数矩阵A的非零元素。

ia：对于系数矩阵的CSR存储格式，ia保存了系数矩阵的行索引。

ja：对于系数矩阵的CSR存储格式，ja保存了系数矩阵的列索引。

b：以列向量的形式保存载荷列阵。

x：以列向量的形式保存求解结果。

error：（输出数据）当求解器发生错误时，通过查看error的返回值来分析求解出错的原因。

**调用函数求解部分代码解释**

调用函数的代码为：

|  |
| --- |
| SparseNumerics.Equation.Linear.PardisoSolverForUnsymmetricMatrixWithIC(n, a, ia, ja, b, x, ref error); |

其中，

命名空间：SparseNumerics.Equation

类：Linear

方法：PardisoSolverForUnsymmetricMatrixWithIC

输入函数的相应参数即可完成求解。

其他项目调用函数的方法与此基本相同，不再赘述。