vs2019 实现C#调用c++的dll两种方法_vs2019引用dll_吾梦汝梦的博客-CSDN博客

成就一亿技术人!

vs2019 实现C#调用c++的dll两种方法

注明：我的目的是利用C#为C++做界面设计
代码下载，免费的

1.托管与非托管的区别

链接地址（仅供参考）
除了链接中的，在实用角度出发：
非托管需要一个个声明引用，就很繁琐
但是托管（虽然麻烦）不用声明，只需要调好配置即可，还是比较方便的
为什么写这个博客，也是因为不同版本vs2017和vs2019有所区别，托管就容易踩坑，希望大家能看看，解决问题（我也很菜，大佬轻喷）

2.非托管类的实现

第一步：创建C++空项目（命名Caculate）添加一个类AddOperate

.h代码部分：

```
#pragma once

extern "C" _declspec(dllexport) int Sum(int a, int b);

class AddOperate
{
public:

};
```

1
2
3
4
5
6
7
8
9

.cpp代码部分：

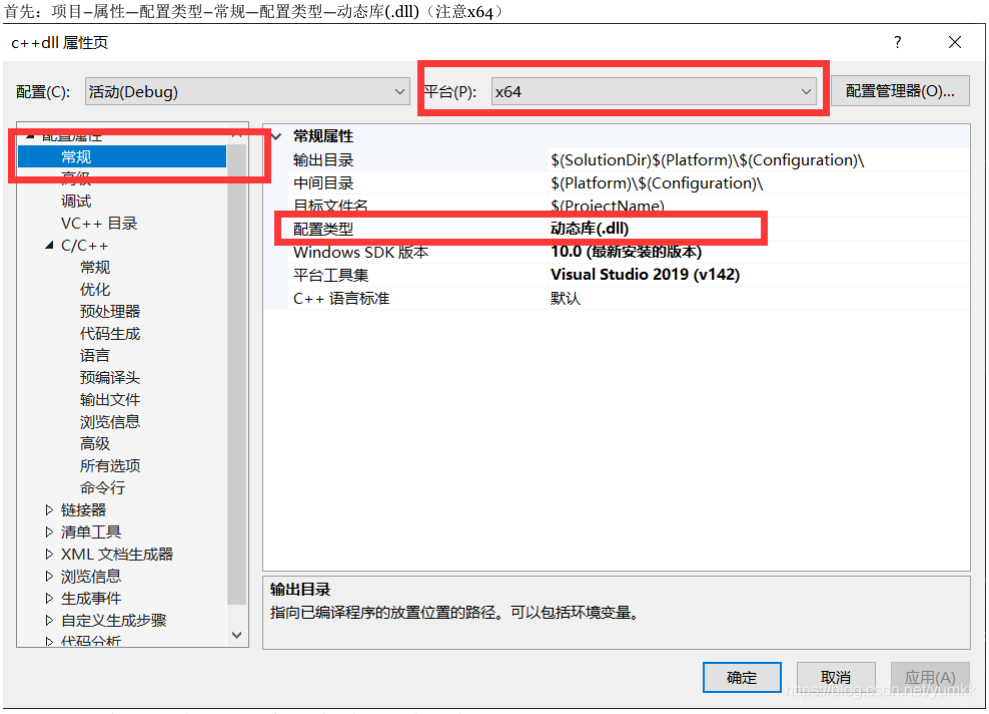
```
#include "AddOperate.h"
#include "iostream"
using namespace std;

int Sum(int a, int b)
{
    if (a - (int)a != 0 || b - (int)b != 0)
    {
        cout << "请输入整数" << endl;
        return -1;
    }
    return a + b;
}
```

1
2
3
4

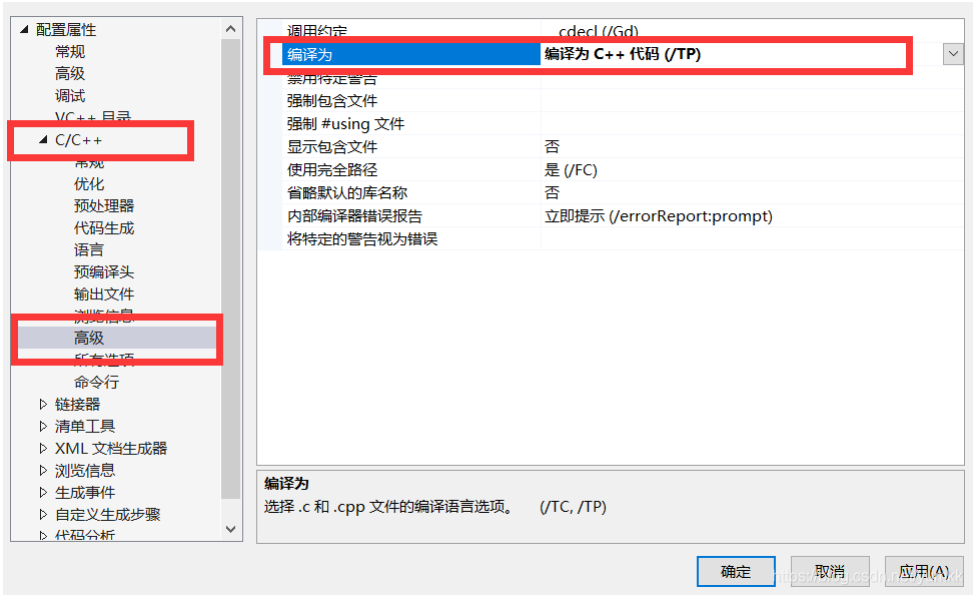
5
6
7
8
9
10
11
12
13

第二步：将C++代码编译成动态库dll

首先：项目–属性–配置类型–常规–配置类型–动态库(.dll)（注意x64）



然后：项目–属性–配置属性–C/C++–高级–编译为–便以为C++代码（/TP）



第三步：将dll拷贝到c#项目输入目录，一般在bin/debug下面

第四步：C#调用dll

C#代码如下：注意：C#也用x64

```
using System;
using System.Runtime.InteropServices;

namespace ConsoleApp_0001
{

    class Program
    {
```

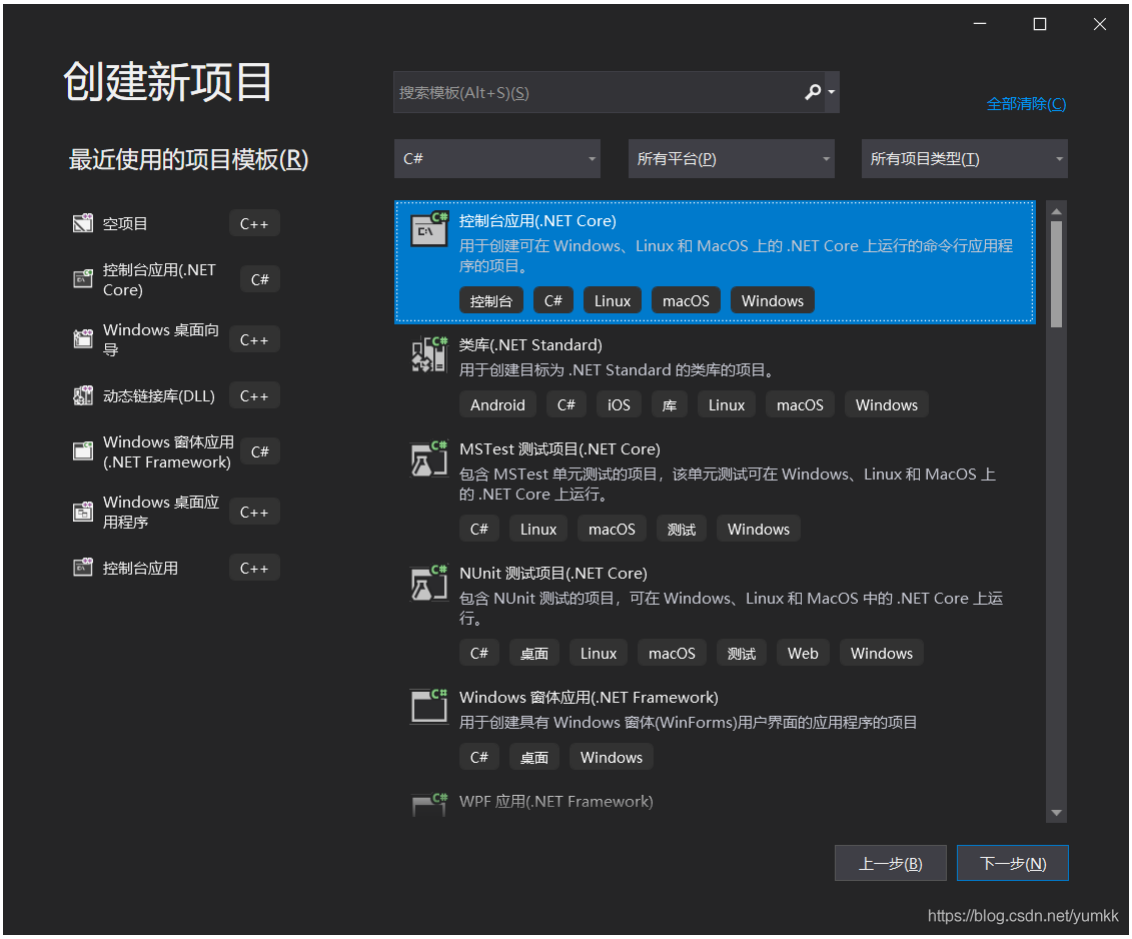
```
        [DllImport("Calculate.dll", CallingConvention = CallingConvention.Cdecl)]
        extern static int Sum(int a, int b);
        public static void Main(string[] args)
        {
            try
            {
                Console.WriteLine("请输入NumberA:");
                int numberA = Convert.ToInt32(Console.ReadLine());

                Console.WriteLine("请输入NumberB:");
                int numberB = Convert.ToInt32(Console.ReadLine());

                Console.WriteLine($"the numberA is:{numberA};numberB is:{numberB},The Sum is:{Sum(numberA, numberB)}");

            }
            catch (Exception ex)
            {
                Console.WriteLine($"ex:{ex}");
            }

            Console.ReadLine();
        }
    }
}
```


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32

3.托管类的实现

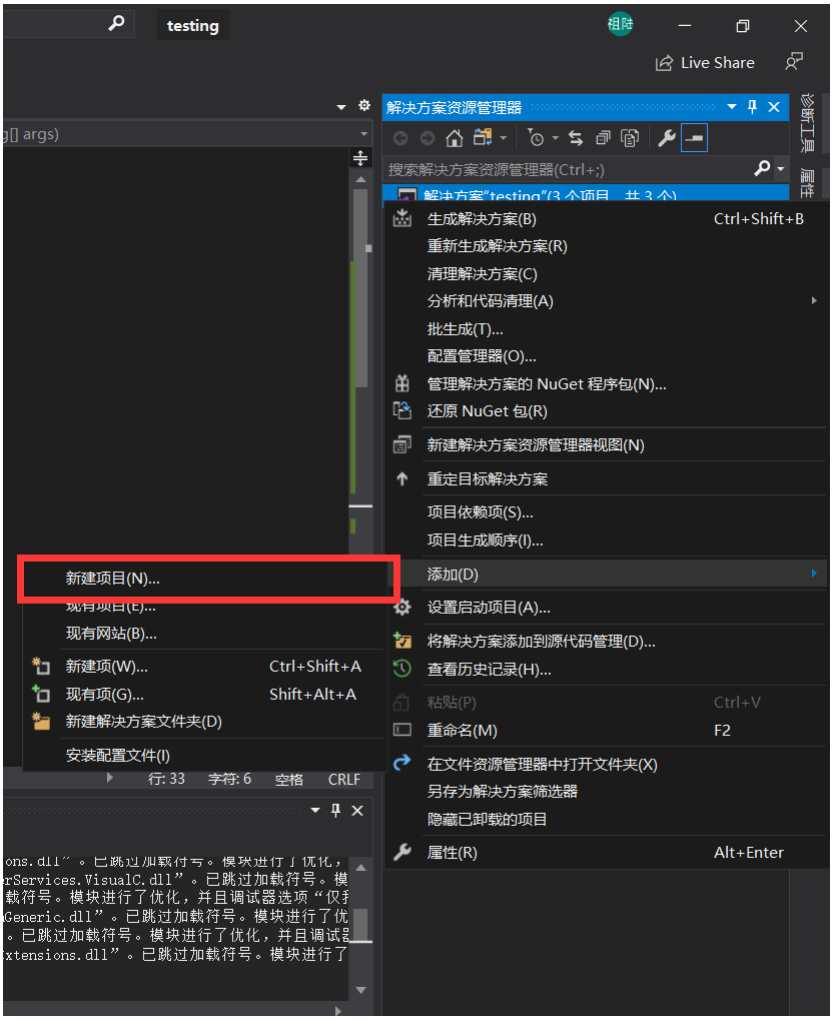
注明：windows窗体控制程序也可以这样

第一步：打开vs2019，新建新项目在C#里找到控制台应用（.NET Core）

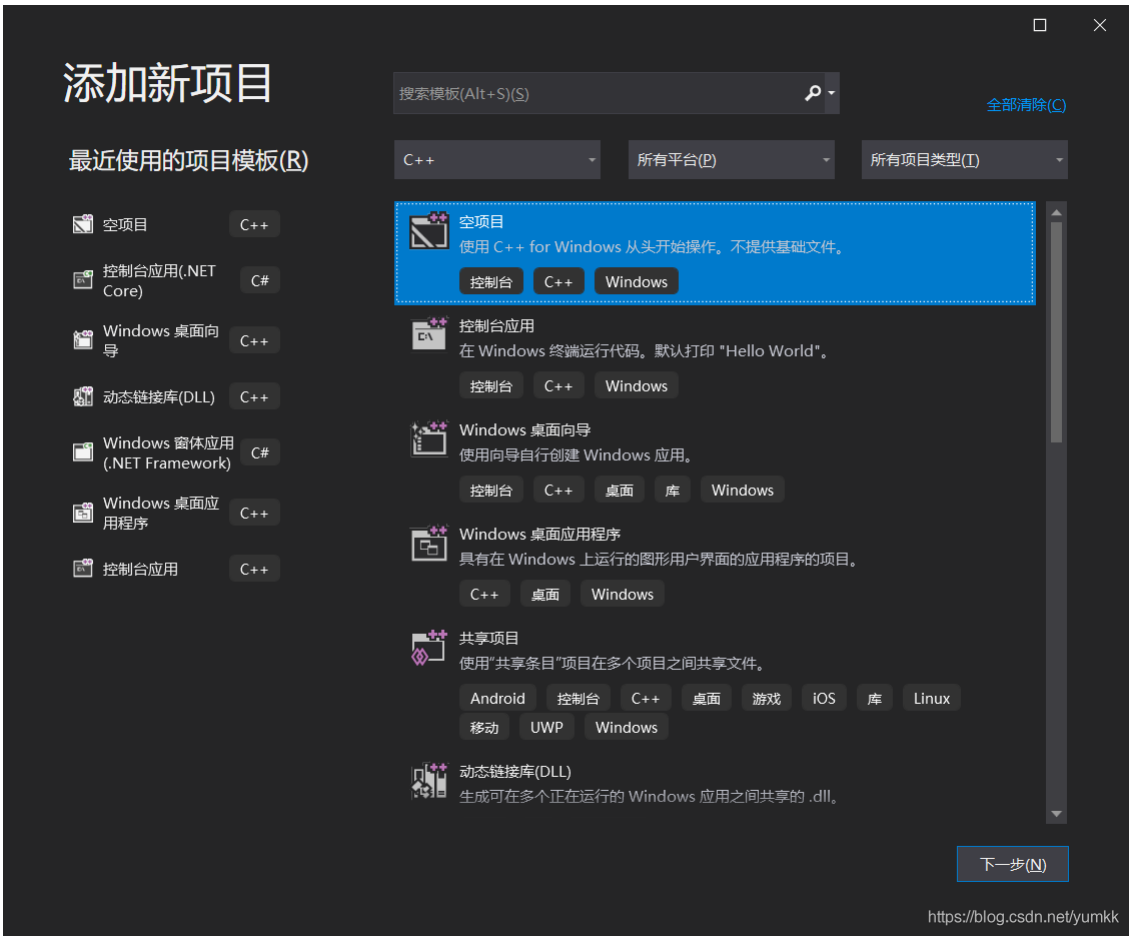


尽量把项目放到特定一个文件夹（经验）
第一步必须这样做

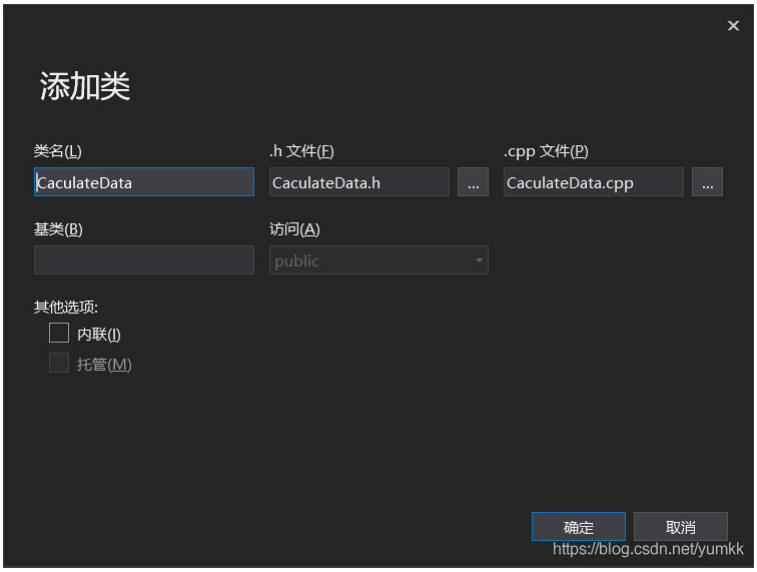
第二步：在你已经创建好的vs界面中，右击解决方案->添加->新建项目->C++空项目（取名Caculate）



新建的项目命名为Caculate



这个时候添加一个类CaculateData



CaculateData.h的代码如下

```
#pragma once

#include <stdio.h>
#include <stdlib.h>
#include <iostream>

using namespace std;

#ifdef CaculateDLL_EXPORTS
#define Calculate_EXPORTS __declspec(dllexport)
#else
#define Calculate_EXPORTS __declspec(dllimport)
#endif
```

```
extern "C" Calculate_EXPORTS int Add(int numberA, int numberB);

extern "C" Calculate_EXPORTS int Subtract(int numberA, int numberB);

extern "C" Calculate_EXPORTS int Multiplication(int numberA, int numberB);

extern "C" Calculate_EXPORTS  int Divided(int numberA, int numberB);

class CaculateData
{

public:
    CaculateData();
    ~CaculateData();

};
```

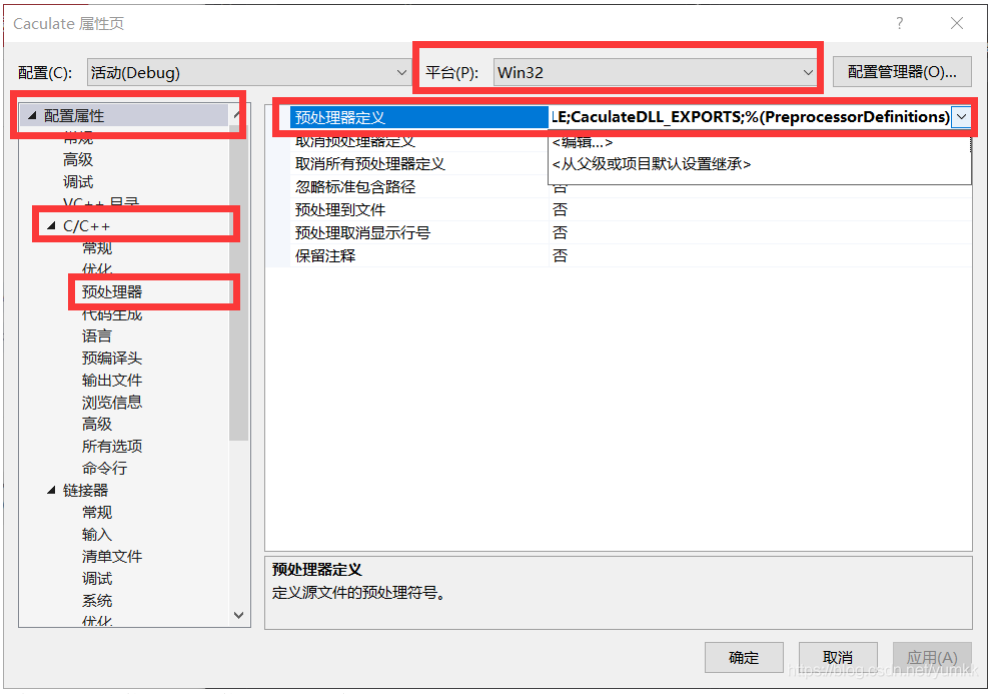
CaculateData.cpp代码如下

```
#include "CaculateData.h"

Calculate_EXPORTS int Add(int numberA, int numberB)
{
    return numberA + numberB;
}

Calculate_EXPORTS int Subtract(int numberA, int numberB)
{
    return numberA - numberB;
}

Calculate_EXPORTS int Multiplication(int numberA, int numberB)
{
    return numberA * numberB;
}

Calculate_EXPORTS int Divided(int numberA, int numberB)
```

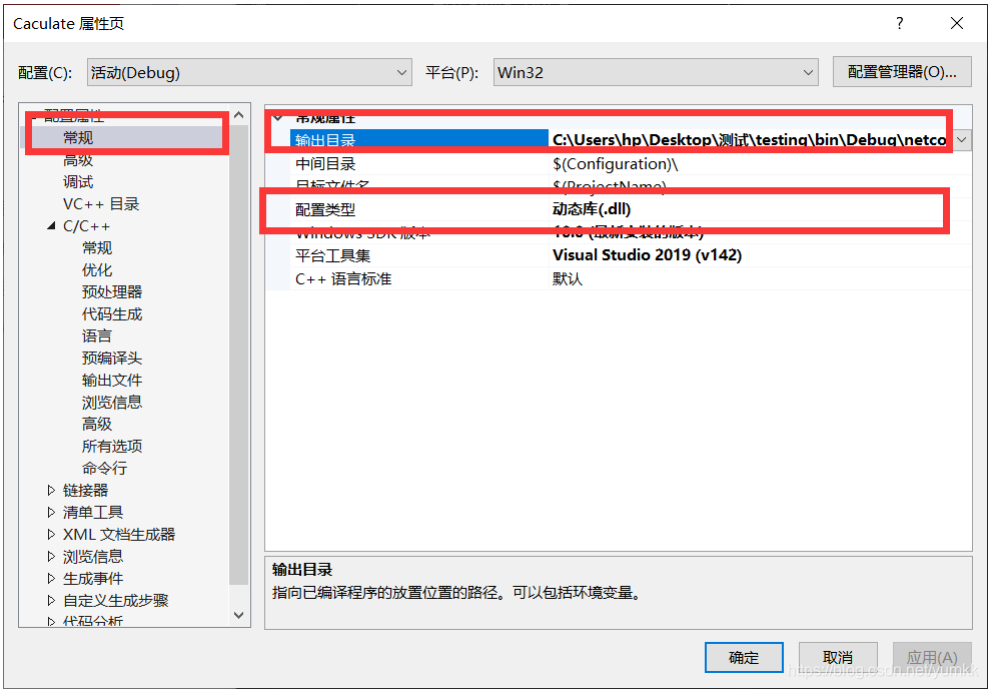
```
{
    if (numberB == 0) {
        std::cout << "除数不能为空" << std::endl;
    }
    return numberA / numberB;
}

CaculateData::CaculateData()
{
}

CaculateData::~CaculateData()
{
}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32

之后你需要配置你的项目
1.添加宏 CaculateDLL_EXPORTS
方法：在Caculate的属性页->配置属性->C/C++>预处理器->预处理器定义
（注意，我们需要平台为Win32）



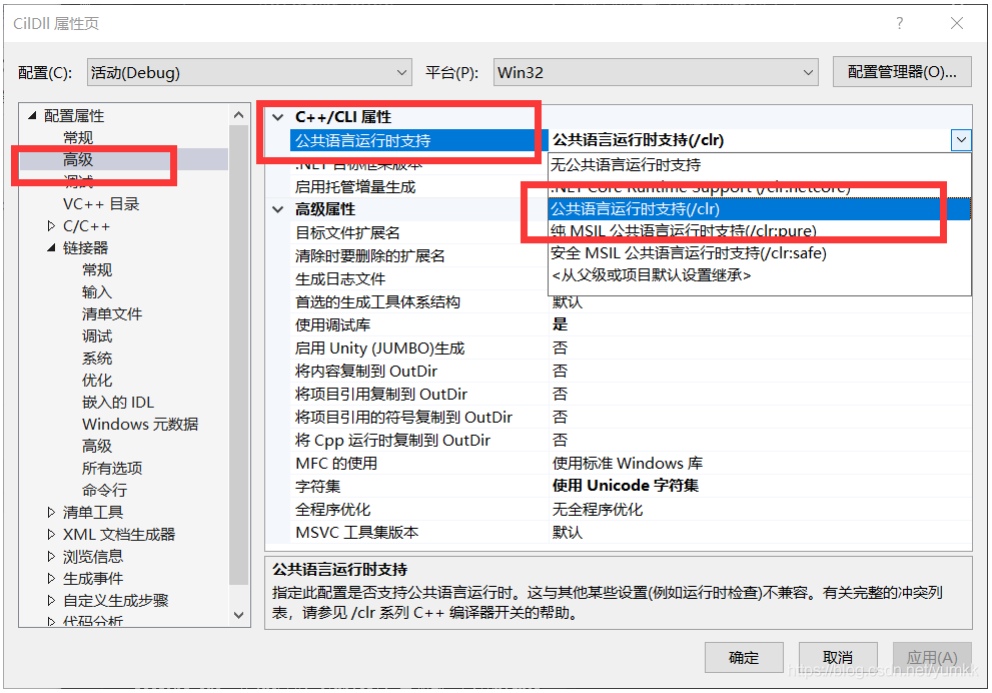
2.在Caculate的属性页->配置属性->常规->设置输出目录和配置类型
输出目录：C#文件夹的bin\Debug\netcoreapp3.1（版本更新会有区别）
配置类型：动态库dll



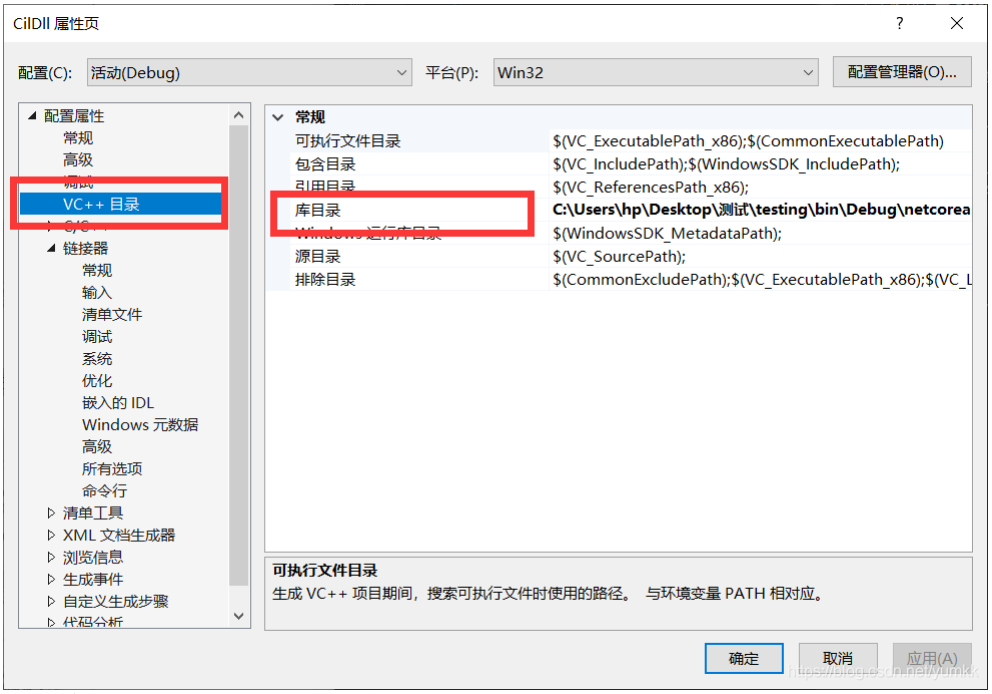
编译ctrl+B完成后在输出目录会出现Caculate.dll文件

第三步：再次右击解决方案->添加->新建项目->C++空项目（取名CilDll）

（这个项目是实现CLR项目）
首先，配置CLR（关键所在，否则会出大问题）
配置属性—高级—公共语言运行时支持

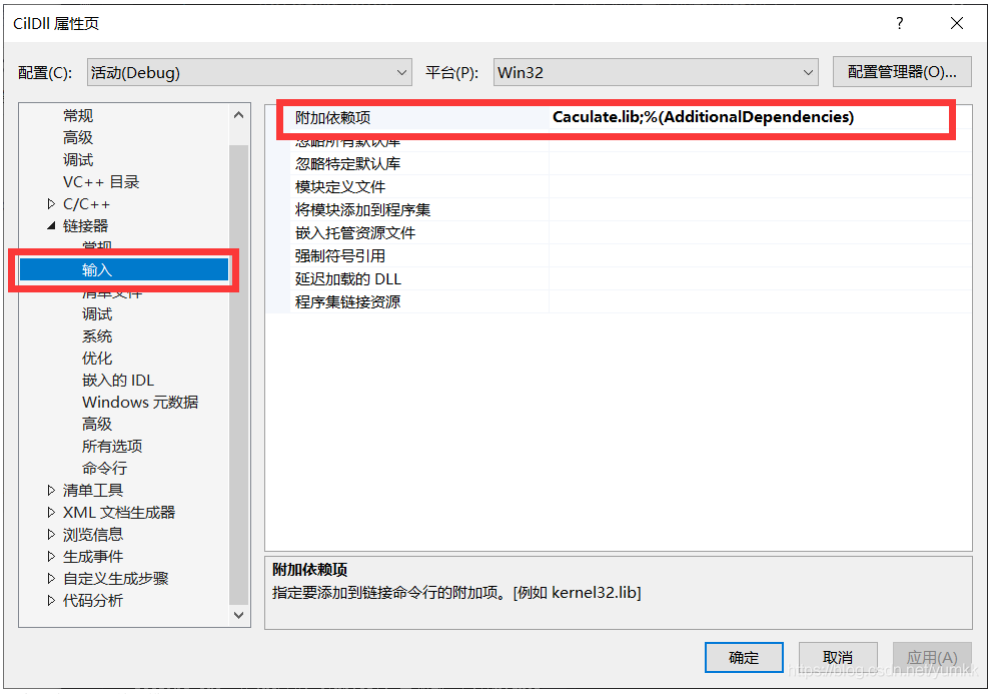


然后，配置属性—VC++目录—库目录（把刚刚生成Caculate.dll的路径添加进去）



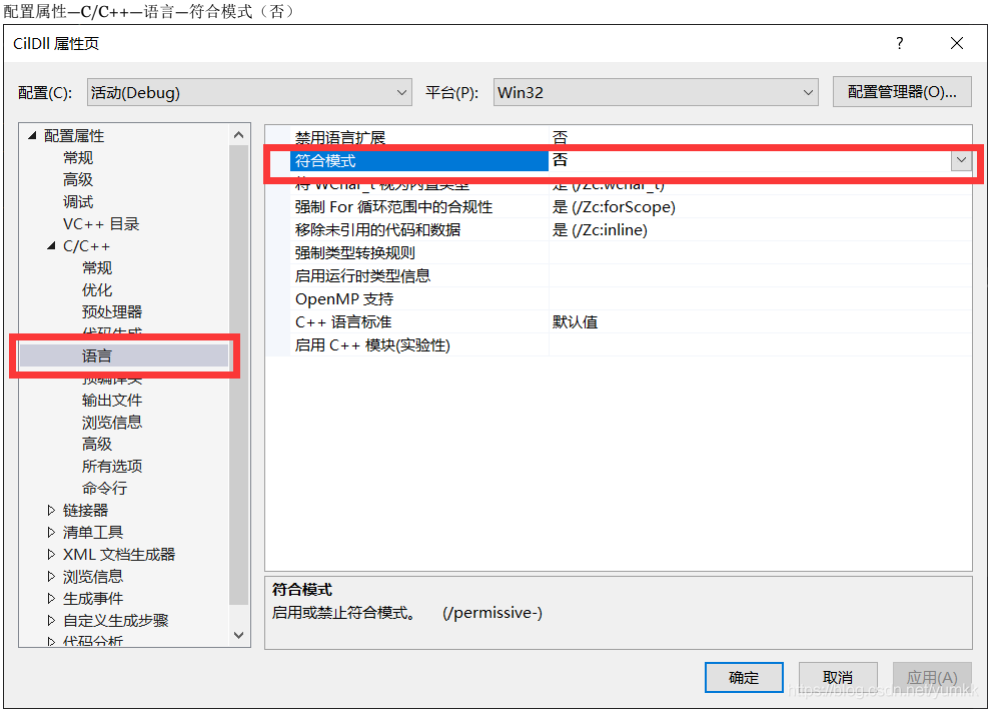
同时引用库：

配置属性—链接器—输入—附加依赖项（注意是.lib）

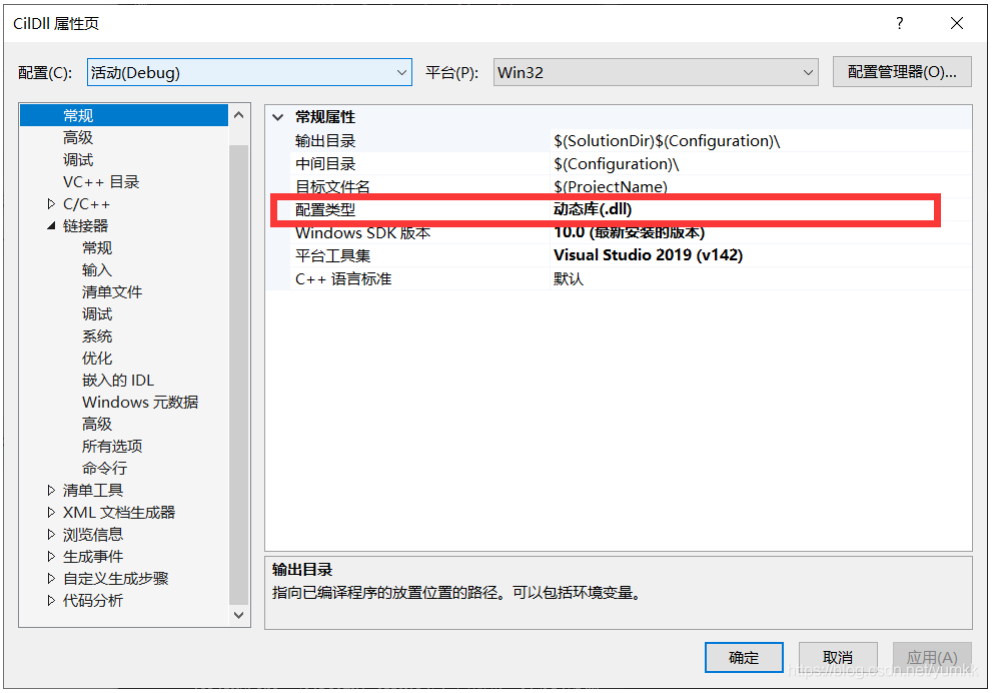


注意：查看语言符合模式（）

配置属性—C/C++—语言—符合模式（否）



最后导出为dll（注意Win32）



创建类InvokeCon

InvokeCon.h代码如下

```
#pragma once
#include <iostream>
#include "C:\Users\hp\Desktop\测试\Caculate\CaculateData.h"//引用库声明对应文件路径
public ref class InvokeCon
{
public:
    InvokeCon();

    int AddCli(int numberA, int numberB);
    int SubtractCli(int numberA, int numberB);
    int MultiplicationCli(int numberA, int numberB);
    int DividedCli(int numberA, int numberB);
};
```

1

2

3

4

5

6

7

8

9

10

11

12

13

InvokeCon.cpp代码如下

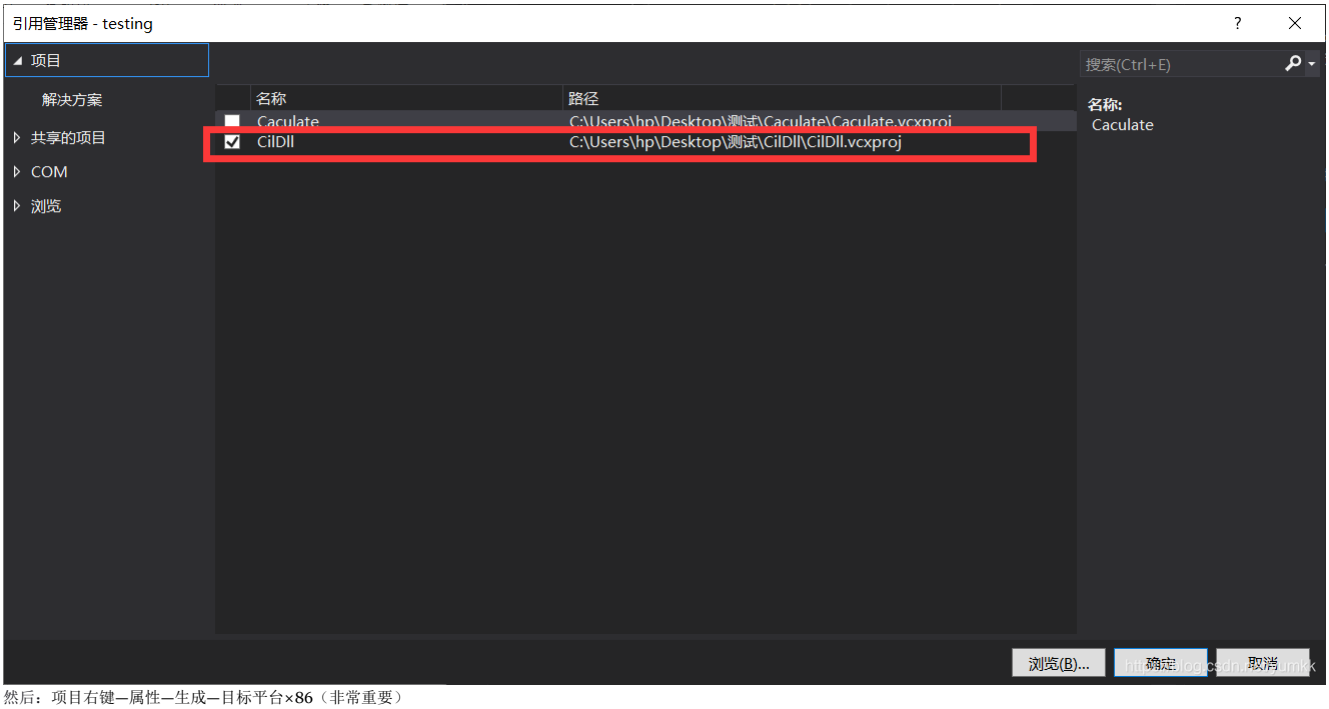
```
#include "InvokeCon.h"

InvokeCon::InvokeCon()
{
}

int InvokeCon::AddCli(int numberA, int numberB)
{
    return Add(numberA, numberB);
}

int InvokeCon::SubtractCli(int numberA, int numberB)
{
    return Subtract(numberA, numberB);
}

int InvokeCon::MultiplicationCli(int numberA, int numberB)
```

```
{
    return Multiplication(numberA, numberB);
}

int InvokeCon::DividedCli(int numberA, int numberB)
{
    return Divided(numberA, numberB);
}
```

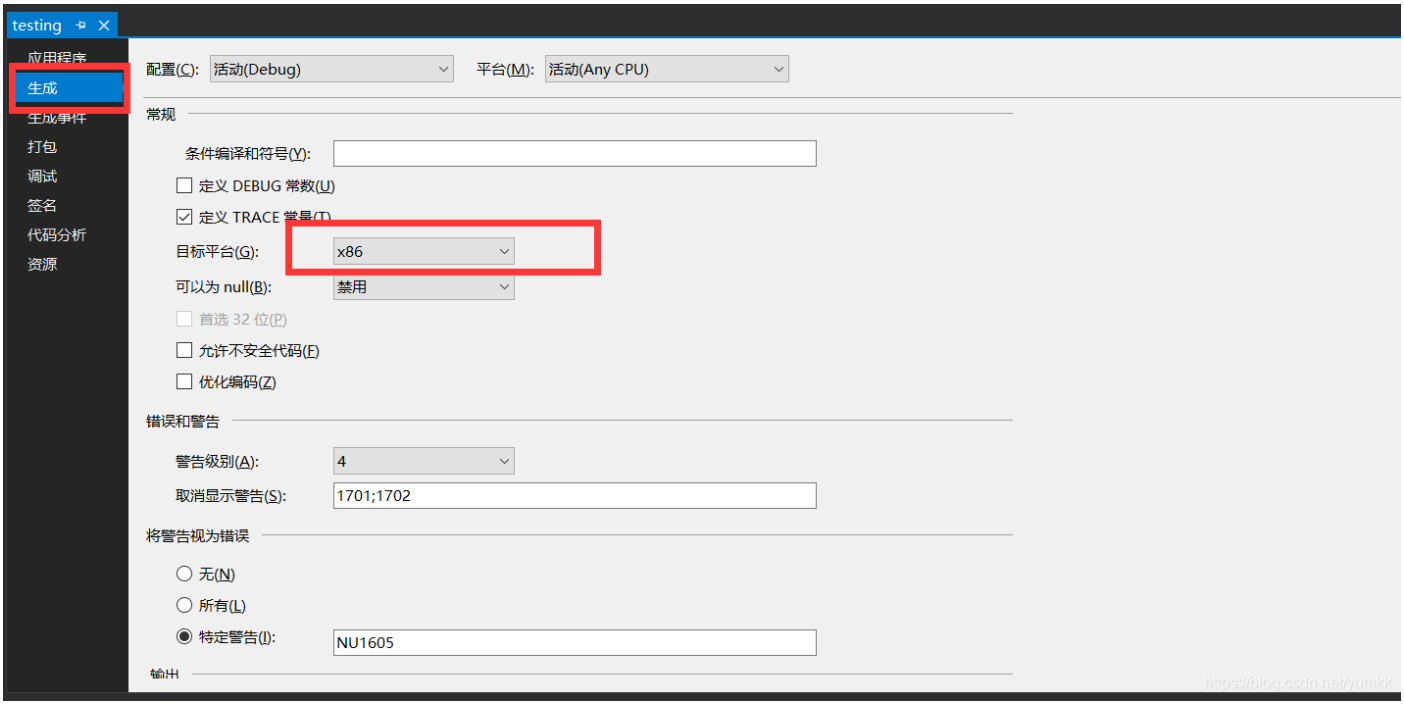
编译ctrl+B

第四步：回到C#进行配置

首先：引用CilDll.dll



然后：项目右键—属性—生成—目标平台×86（非常重要）



C#代码如下

```
using System;

namespace testing
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                Console.WriteLine("---------c#通过CLI调用C++类方法---------");
                Console.Write("请输入numberA:");
                int numberA = Convert.ToInt32(Console.ReadLine());
                Console.Write("请输入numberB:");
                int numberB = Convert.ToInt32(Console.ReadLine());

                InvokeCon invoke = new InvokeCon();
                int addResult = invoke.AddCli(numberA, numberB);
                int subResult = invoke.SubtractCli(numberA, numberB);
                int mutilResult = invoke.MultiplicationCli(numberA, numberB);
                int divResult = invoke.DividedCli(numberA, numberB);

                Console.WriteLine($"the {numberA} And {numberB} sum is:{addResult};sub is:{subResult};Mutil is:{mutilResult};div is:{divResult}");
            }
            catch (Exception ex)
            {
                Console.WriteLine($"ex:{ex}");
            }

            Console.WriteLine("执行成功");
            Console.ReadLine();
        }
    }
}
```