火星时代教育

静态库与动态库

—虚幻引擎高级程序开发专业—

PART 1

概述

概述

程序设计中，我们应对的产品设计需求多种多样。但在长期的开发设计中，研发人员发现，在大多数产品设计中会用到相同的功能，例如数学算法，文件加密算法，通信算法。我们不必为每个产品都重新设计这些功能，只完成一份产品设计，然后不停的复用，库因此诞生了！**库帮助我们解决代码复用问题，并且提供跨工程项目的复用结构**。降低了产品设计成本。

注意：不同平台库的后缀名称不同，我们从库的介入方式，总体分为动态库和静态库。在Windows平台，静态库后缀是lib，动态库后缀是dll。

PART 2

静态库

情景分析

1. 某A公司需要IT从业人员王某二，编写在代号X软件内使用的数学函数工具类
2. A公司需要IT从业人员刘小某，编写在代号B软件中实现一套基本几何图形面积周长计算工具类
3. A公司事真多，IT从业人员赵凤某在编写代号XB的项目时，设计需求中有一项需求，要求软件能够计算几何图形面积并且包含常用数学函数

库

库是写好的现有的，成熟的，可以**复用的代码**。现实中每个程序都要依赖很多基础的底层库，不可能每个人的代码都从零开始，因此库的存在意义非同寻常。

静态库

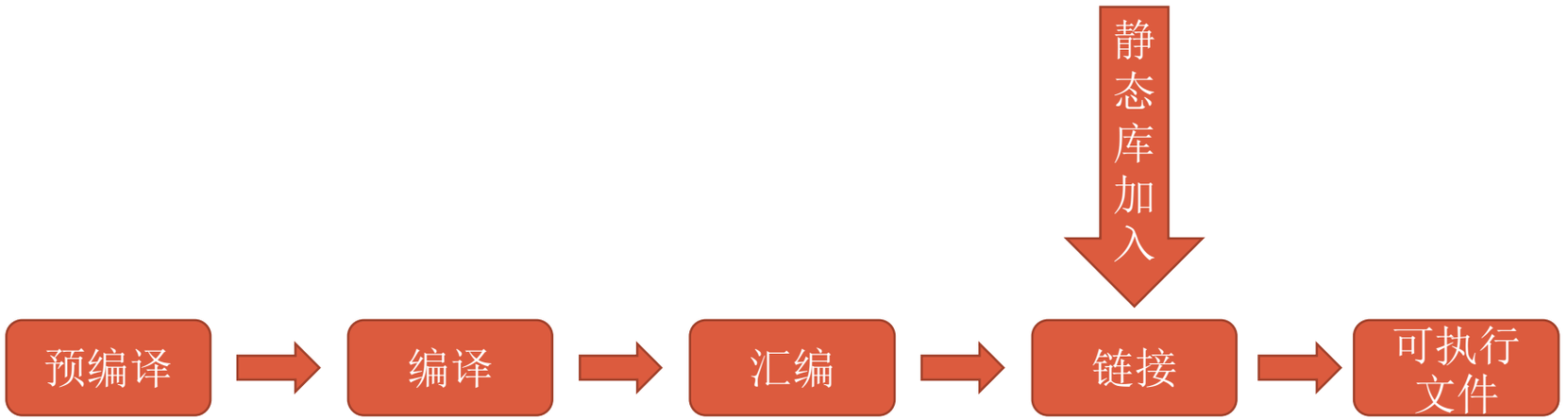
静态库是指在我们的应用中，有一些**公共代码**是需要反复使用，就把这些代码编译为“库”文件；在**链接步骤中**，连接器将从库文件取得**所需的代码，复制**到生成的可执行文件中的这种库。

静态库特点

1. 集合了一大部分常用公共操作函数
2. 常见的后缀名是.lib（windows）.a（linux）
3. 方便我们构建项目，减少重复编码工作
4. 使用方便，随用随引入，项目使用友好
5. 静态库对函数的使用是在链接阶段完成的
6. 运行阶段不依赖库文件，跨平台移植方便
7. 缺点，链接阶段会把所有相关文件和函数链接进软件，生成可执行文件，增加空间占用，浪费资源
8. 在编译时，需要注意，如果编译版本是32位库，导入使用的工程也只能是32位，如果是64位则只能提供给64位工程使用

编译过程

静态库在链接阶段被组织到程序中，编译成可执行文件



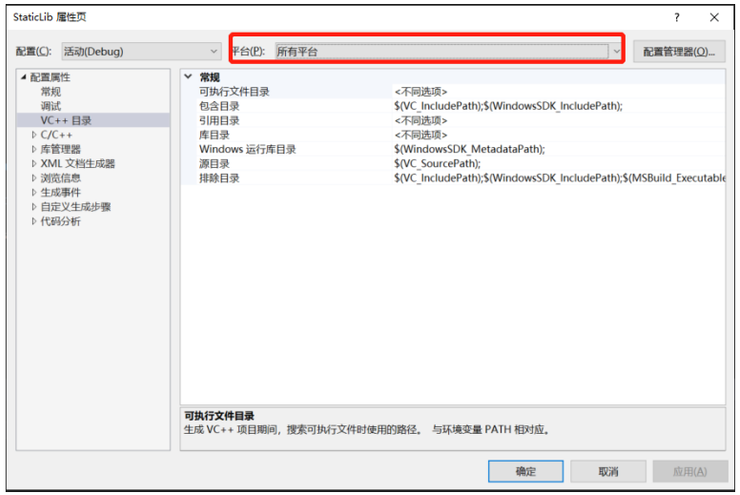
构建静态库

1. 创建静态库工程
2. 添加操作函数集
3. 编译工程
4. 创建新工程
5. 引入静态库到编辑器
6. 引入静态库头文件，使用静态库
7. 编译运行

编译静态库

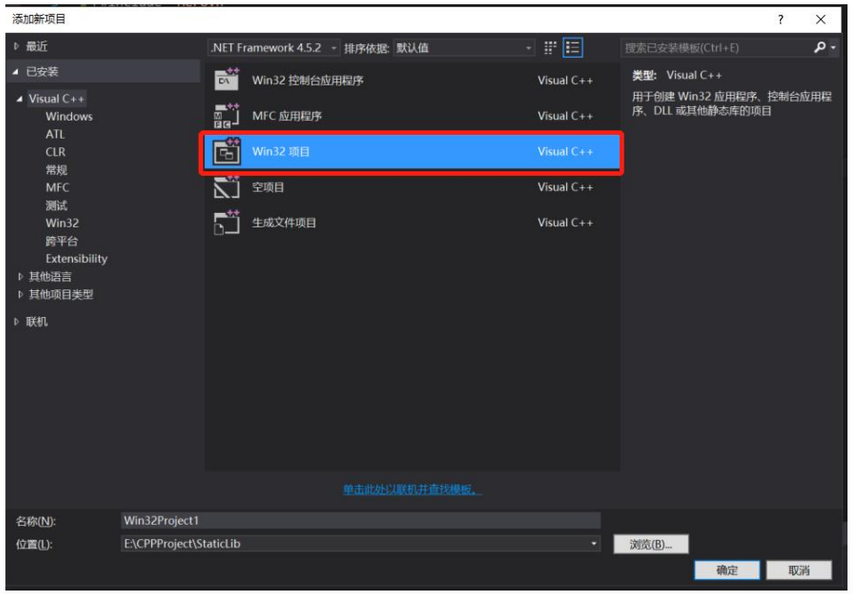
注意，引入头文件和库文件，是必须要做的。当编写完静态库后记得调整库的编译平台，可以在工程中右键属性中调整。配置管理器中可以调整编译目标平台。直接编译即可。

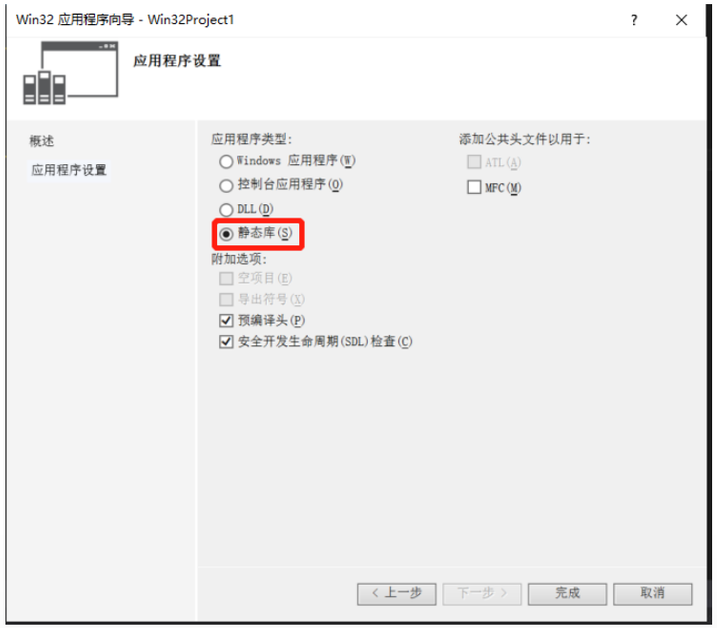
如果希望将库提供给他人使用，记得一定要将**头文件与库文件**一起提供。



创建静态库

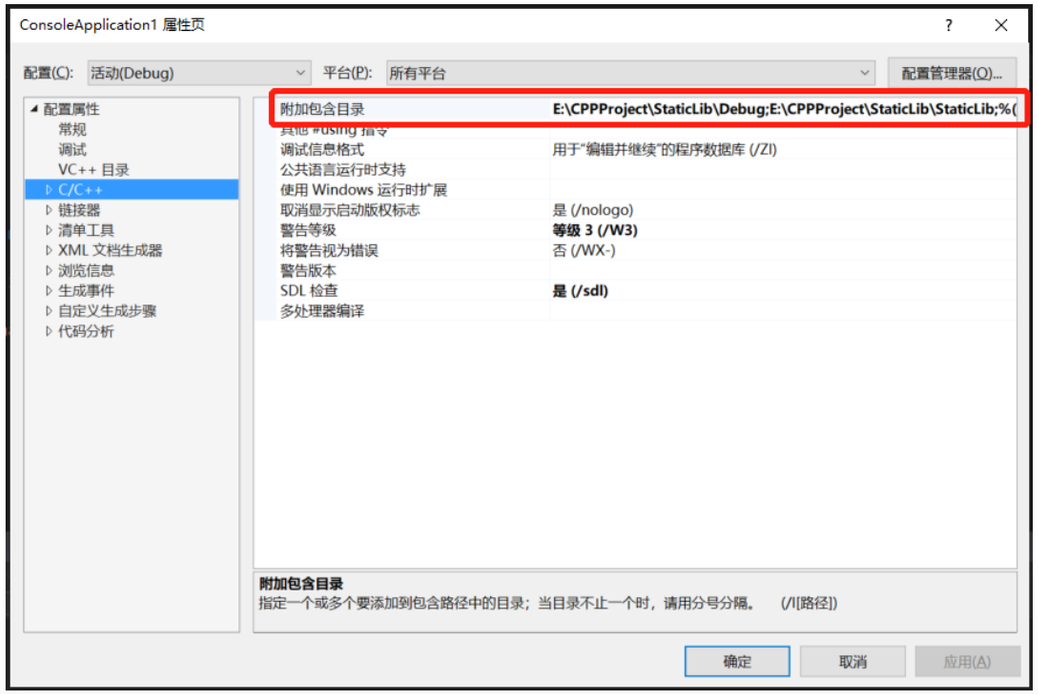
创建C++工程，类型选择Win32程序。然后选择构建为静态库程序





包含头文件

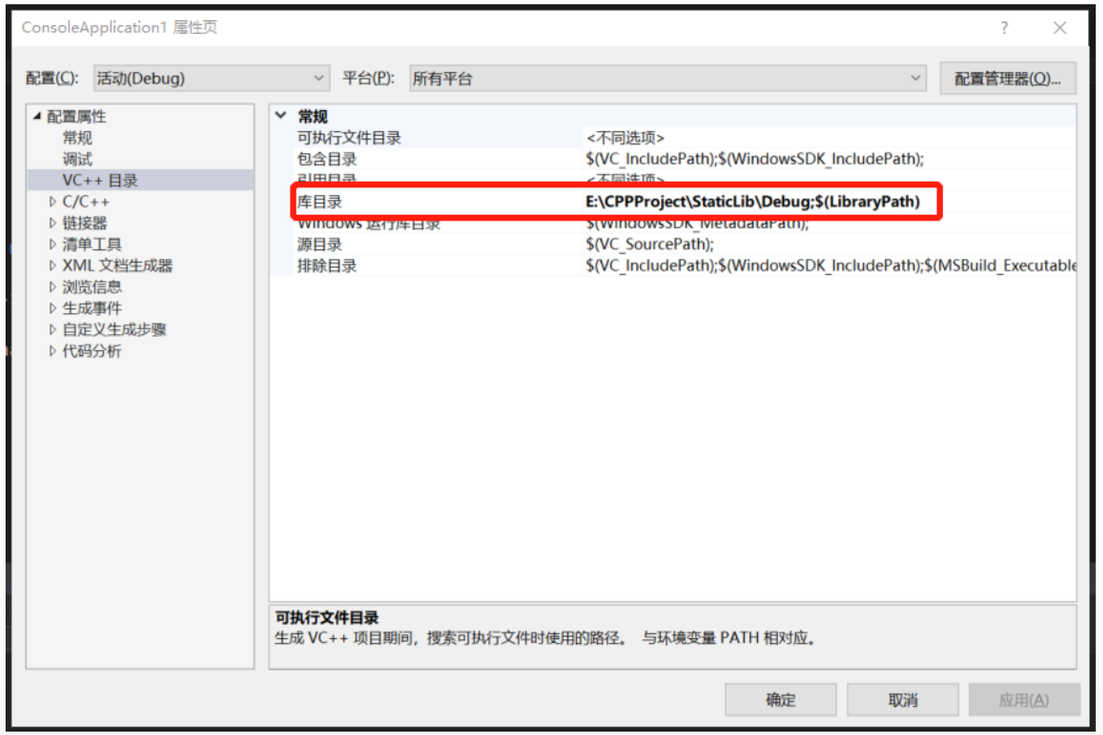
在使用静态库的工程中，右键工程属性内寻找C/C++下的**附加包含目录**，将**拷贝的静态库头文件所在路径**进行选取，即可在工程中引用头文件。

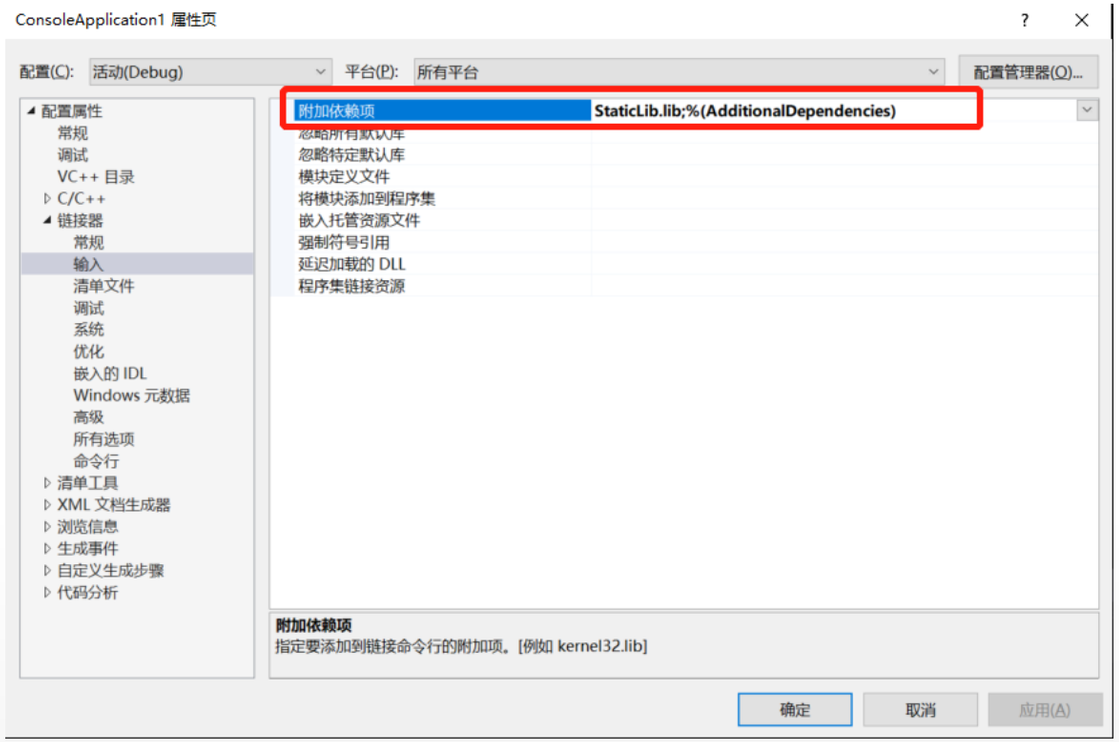


引入库文件

1、设置库目录路径

2、设置连接器中的库输入（填入库名称即可）





练习题

1. 构建静态工具库
2. 库内包含int类型的加减乘除（Add，Sub，Mul，Div）
3. 构建新的工程，引入静态库进行使用，并计算
    1. 50*23
    2. 22+66+41+11
    3. 171/3*225
    4. 50-65*31

PART 3

动态库

情景分析

1. A公司，生产的一款软件产品XB，市场反响非常不错，已经有1000万用户的使用量
2. A公司的员工，IT从业者赵凤某当初参与设计了软件，并且使用了刘小某和王某二的工具集构建了静态库，集成在了XB软件中
3. A公司PM提出了新的设计需求，要求更改静态库中的函数结构，并且快速提供给用户使用

动态库

1. 由于设计中静态库的全量更新十分不友好，动态库出生了
2. 动态库又称动态链接库英文为DLL，是Dynamic Link Library 的缩写形式，DLL是一个包含可由多个程序同时使用的代码和数据的库，DLL不是可执行文件，链接阶段不会被加入到目标软件，只在运行阶段调用使用。
3. 动态库的使用方式分为两种
    1. 静态加载(隐式调用)（load-time dynamic linking）

       在使用动态库时，启动工程即加载库，将库文件包含在项目中。
    2. 动态加载(显示调用)（run-time dynamic linking）

       在运行时使用到对应的库时，通过使用loadlibrary进行动态加载，使用完成可以主动释放

动态库特点（显示调用）

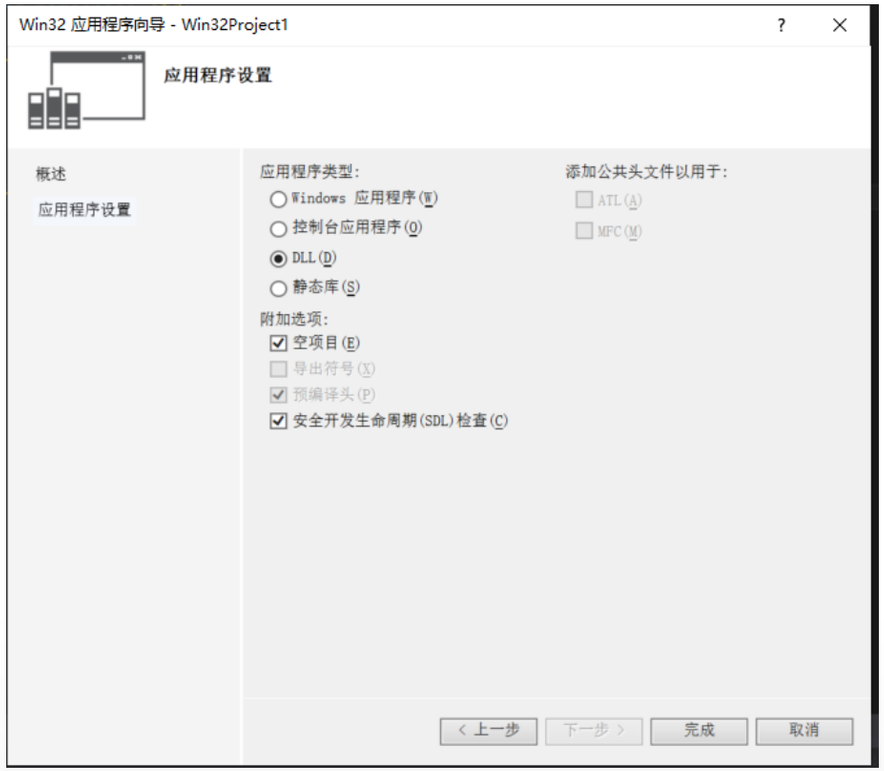
1. 程序编译阶段库内的文件及函数不会被打包到程序中规避空间浪费
2. 可以实现运行阶段动态链接载入库内函数
3. 更容易实现增量更新
4. 修改库的内容无需重新部署编译引用工程
5. 根据需求可在代码中显示调用和引入动态库
6. 可以实现进程间的资源共享
7. 动态库编译后会生成两个文件（Windows平台），dll动态链接库，lib函数引入库
8. 编译中，32位库只能应用到32位项目，64位库只能应用到64位工程

编写动态库

1. 创建动态库工程
2. 编写动态库内函数借助扩展修饰指令 __declspec(dllexport)
3. 编译动态库工程
4. 创建目标工程并导入动态库
5. 编写调用动态库代码
6. 编译运行

创建动态库

依照创建静态库的方法，在构建时选择动态库即可。（空项目选择随意）



编写动态库

动态库中主要是向外暴露函数接口，函数接口对外的类型为全局函数

```cpp
//向外导出全局函数 Cpp文件中
extern "C" _declspec(dllexport) int Max(int a, int b)
{
    return a > b ? a : b;
}
```

extern“C”

告知编译器，将此函数用C语言方式进行命名导出，由于C++函数具有重载特性，在通过显示链接时，名称访问会出现二义性。但C语言不具备重载，所以可以有效保证函数不会出现名称相同。（**用extern “C”标记的函数不能是重载函数**）。

此标记只能用于导出全局函数

_declspec(dllexport)

告知编译器导出一个函数，并且这个函数导出到dll库中，提供给其他人使用

调用动态库（动态加载）

调用动态库，需要先加载动态库，完成后构建函数指针获取函数。

注意需要引入头文件 Windows.h

```
//加载动态库    (引入头文件 windows.h)
HMODULE ndll = LoadLibrary(L"..\\Debug\\DynamicLib.dll");
if (!ndll)
{
    return 0;
}
//定义函数指针
typedef int(*pfun)(int, int);
//获取动态库导出的全局函数
pfun pf = pfun(GetProcAddress(ndll, "Max"));
if (pf)
{
    cout << pf(100, 60) << endl;
}
//释放动态库
FreeLibrary(ndll);
```

注意事项

1. 动态库搜寻目录（Windows平台）
    1. 当前工程运行目录 exe目录
    2. Windows系统的动态库目录
    3. Windows目录
    4. 环境变量中的目录
    5. 当前工程目录
2. 借助Windows头文件中加载动态库文件
    1. 先LoadLibrary获取的是动态库的句柄
    2. 声明函数指针，通过GetProcAddress获取函数指针
    3. 调用函数指针
    4. 如果动态库不需要使用，那么我们就FreeLibrary，释放加载的动态库

练习

1. 使用动态库构建加减乘除函数
2. 构建后并使用另一工程引入动态库并操作动态库

PART 4

动态库（面向对象）

描述

前面的章节中，我们构建了全局函数，通过使用C接口函数导出的方法，构建了全局函数操作，那么接下来我们讲解下如何导出成员函数以及如何导出库内的类。

目标：

- 通过库内构建全局对象，调用库内逻辑，对外暴露使用全局函数
- 导出库内类的静态成员函数
- 导出库内构建的类

PART 5

全局对象变量

通过全局变量调用库内逻辑

此操作，将逻辑全部封装在库内，通过导出C接口函数的方法进行导出。缺点，库内自定义数据结构外部无法使用，只能操作数据结果

因为导出C接口函数是全局的，构建全局对象，在导出函数中可以访问到，所以可以操作

源码

头文件

```
#pragma once
class LibStamp
{
public:
    LibStamp();
    ~LibStamp();

public:
    int Num;
};

```

源文件

```
#include "LibStamp.h"

LibStamp::LibStamp()
{
    Num = 10;
}

LibStamp::~LibStamp()
{
}
//构建全局变量
LibStamp lib;
//导出全局函数
extern "C" _declspec(dllexport) int Sum(int A, int B)
{
    return A + B + lib.Num;
}
```

调用

```
HMODULE dll = LoadLibrary("..\\Debug\\DynmaicLib.dll");
if (dll == nullptr)
{
    cout << "NULL" << endl;
}
else
{
    typedef int(*NSum)(int, int);
    NSum pf = NSum(GetProcAddress(dll, "Sum"));
    cout << pf(100, 100) << endl;
}
```

PART 5

静态成员函数

源码

头文件

```
#pragma once

#define LIBSTAMP_API _declspec(dllexport)

class LibStamp
{
public:
    LibStamp();
    ~LibStamp();
    static LIBSTAMP_API int LocSum(int A, int B);//只能导出类得静态成员函数

public:
    int Num;
};
```

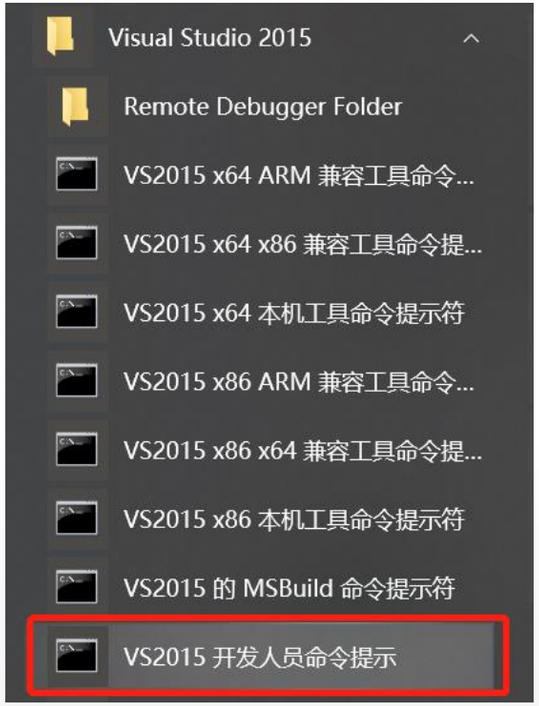
源文件

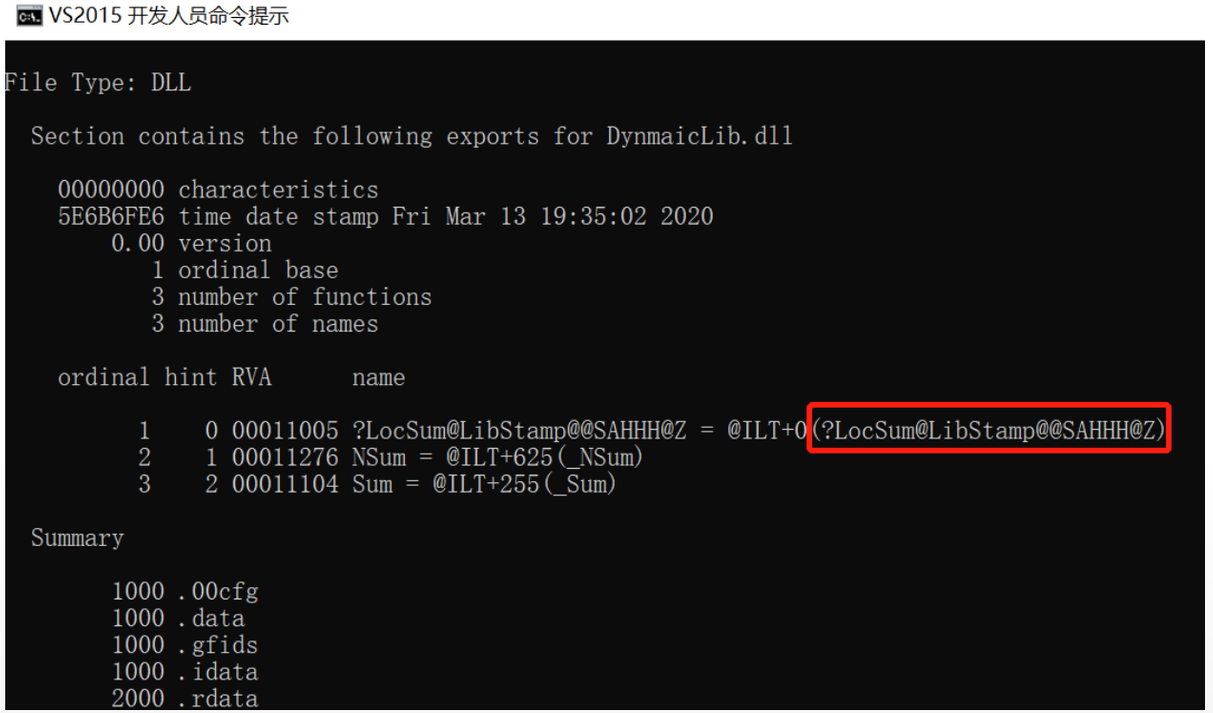
```
LIBSTAMP_API int LibStamp::LocSum(int A, int B)
{
    return A + B;
}
```

显示链接查询函数名称

由于导出的是静态成员函数，调用时无法只通过函数名称完成调用，所以必须通过VS得工具查询动态库函数名称，才可以显示链接进行调用。工具可以在开始，所有程序中找到。

找到项目文件夹，运行指令dumpbin –exports XXX.dll来查看名称





显示链接调用

代码

```
HMODULE dll = LoadLibrary("..\\Debug\\DynmaicLib.dll");
if (dll == nullptr)
{
    cout << "NULL" << endl;
}
else
{
    typedef int(*NSum)(int, int);
    NSum pf = NSum(GetProcAddress(dll, "?LocSum@LibStamp@@SAHHH@Z"));
    cout << pf(100, 100) << endl;
}
```

PART 6

静态调用

动态调用和静态调用

对于动态库的引入分为静态调用（隐式调用）和动态调用（显示调用），在之前我们的操作均是使用**显示调用**方式完成，但是显示调用有明显的问题。即当我们编写的内容量非常庞大时，我们的调用动作就变得异常的繁琐，其实在更多的时候我们可以通过隐式调用完成动态库的使用，这使得我们的操作更加的简单。

对于静态调用，我们需要保证**dll库和lib库和头文件**，三者缺一不可。

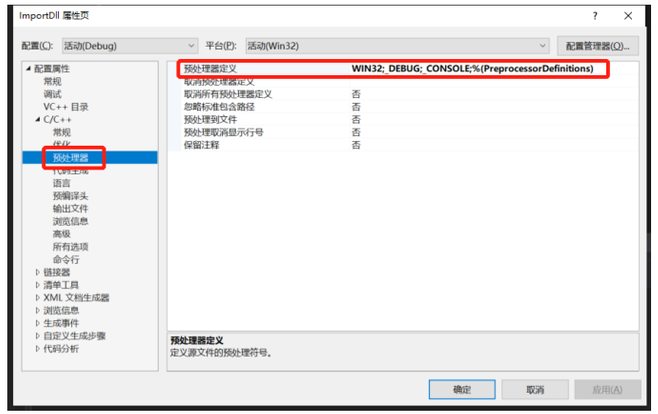
PART 7

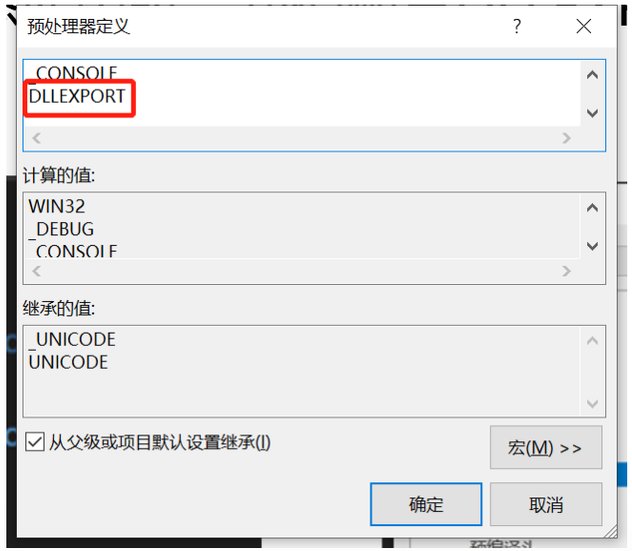
类（静态调用）

导出类

对于类的导出，一般我们也经常会使用，导出类的方法和导出函数基本一致，但是为了配合静态调用，我们需要构建导出宏，方便我们进行导出，参照以下代码。注意_declspec(dllimport)是用于导入操作，为了解决引入动态库重复声明问题，我们添加了导出宏，在动态库中时，我们配置DLLEXPORT宏，使得MYDLL_API为导出动作。DLLEXPORT宏可以直接配置在项目中，参照下图。

```
1   #pragma once
2
3   #ifdef DLLEXPORT
4   #define MYDLL_API _declspec(dllexport)
5   #else
6   #define MYDLL_API _declspec(dllimport)
7   #endif // DLLEXPORT
8
9   class MYDLL_API Human
10  {
11
12  public:
13
14      void Fun();
15
16
17  };
18
19
```

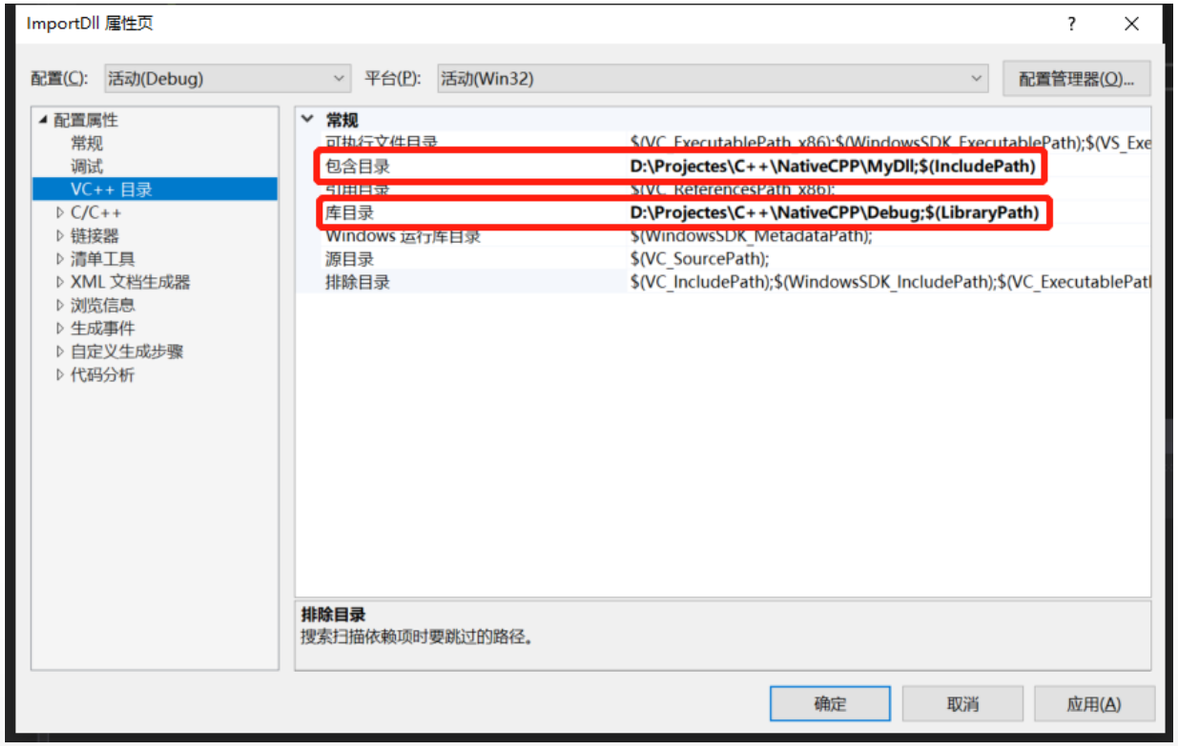




使用动态库

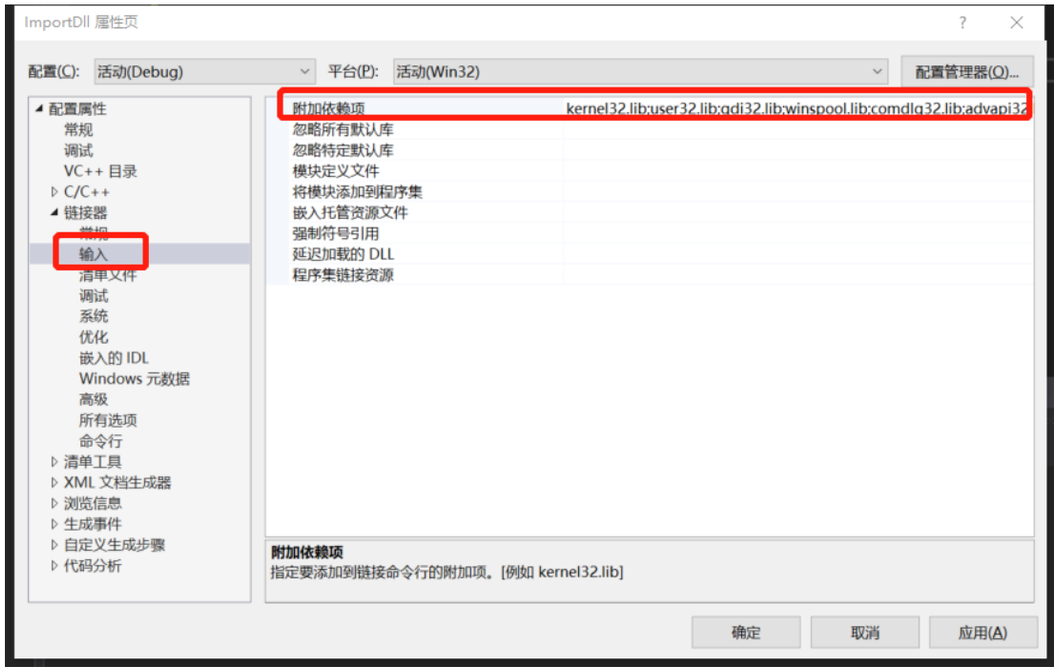
通过导出的静态库lib帮助我们关联动态库内容，在使用的项目中，我们需要完成以下配置才可使用。

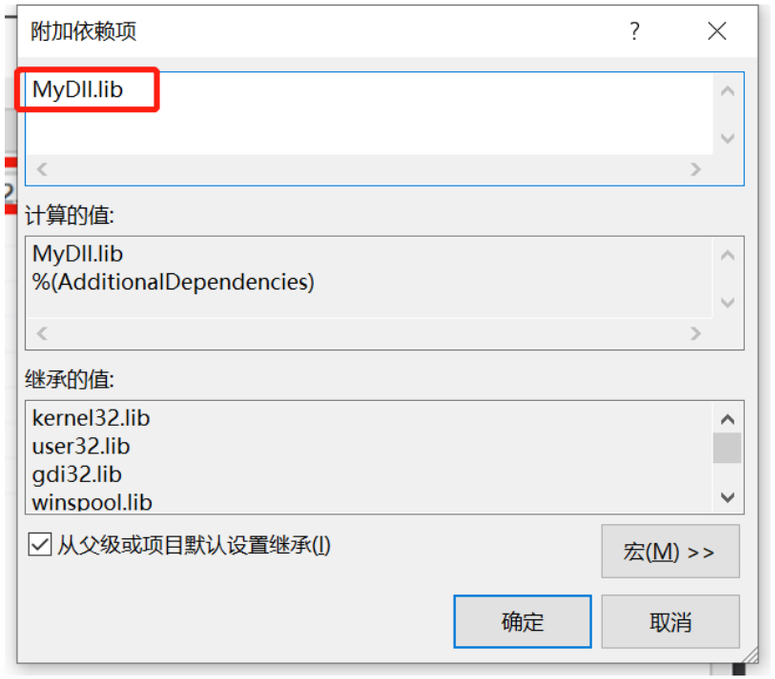
- 包含目录：即头文件所在目录
- 库目录：lib所在目录



配置链接库

在项目连接器中选择输入，在依赖项中填写lib库名称





使用库内容

将**动态库放置到项目执行目录中**，通过正常引入头文件，即可使用库内容。

动态库的优势，即修改动态库内容后，只要声明结构没有发生变化，则只要编译动态库即可，新的逻辑即被应用到原项目中。

```
#include <iostream>
#include "Human.h"


int main()
{
    Human* hman = new Human;
    hman->Fun();

    std::cout << "Hello World!\n";
}
```

THANK YOU

虚幻4高级程序开发专业