C + +动态库封装及调用

1 一个程序从源文件编译生成可执行文件的步骤：

预编译 --> 编译 --> 汇编 --> 链接
(1)**预编译**，即预处理，主要处理在源代码文件中以“#”开始的预编译指令，如宏展开、处理条件编译指令、处理#include指令等。
(2)**编译**过程就是把预处理完的文件进行一系列词法分析、语法分析、语义分析以及优化后生成相应的汇编代码文件。
(3)**汇编**是将汇编代码转变成二进制文件。
(4)**链接**将二进制文件链接成一个可执行的命令，主要是把分散的数据和代码收集并合成一个单一的可加载并可执行的的文件。链接可发生在代码静态编译、程序被加载时以及程序执行时。链接过程的主要工作是符号解析和重定位。

2 库

库是一组目标文件的包，就是一些最常用的代码编译成目标文件后打包存放。而最常见的库就是运行时库（Runtime Library）,如C运行库CRT.

库分为两种:静态库（.a 、.lib）动态库（.so 、.dll ）所谓静态、动态是指链接过程。

3 静态库与动态库

（1）lib是编译时用到的，dll是运行时用到的。如果要完成源代码的编译，只需要lib；如果要使动态链接的程序运行起来，只需要dll。

（2）如果有dll文件，那么lib一般是一些索引信息，记录了dll中函数的入口和位置，dll中是函数的具体内容；如果只有lib文件，那么这个lib文件是静态编译出来的，索引和实现都在其中。使用静态编译的lib文件，在运行程序时不需要再挂动态库，缺点是导致应用程序比较大，而且失去了动态库的灵活性，发布新版本时要发布新的应用程序才行。

（3）动态链接的情况下，有两个文件：一个是LIB文件，一个是DLL文件。LIB包含被DLL导出的函数名称和位置，DLL包含实际的函数和数据，应用程序使用LIB文件链接到DLL文件。在应用程序的可执行文件中，存放的不是被调用的函数代码，而是DLL中相应函数代码的地址，从而节省了内存资源。DLL和LIB文件

必须随应用程序一起发行，否则应用程序会产生错误。如果不想用lib文件或者没有lib文件，可以用WIN32 API函数LoadLibrary、GetProcAddress装载。

DLL即动态链接库（Dynamic-Link Libaray）的缩写，相当于Linux下的共享对象。Windows系统中大量采用了DLL机制，甚至内核的结构很大程度依赖与DLL机制。Windows下的DLL文件和EXE文件实际上是一个概念，都是PE格式的二进制文件。一般的动态库程序有lib文件和dll文件，lib文件是编译时期连接到应用程序中的，而dll文件是运行时才会被调用的。

为了更好的理解DLL,首先介绍一下导出和导入的概念。

（1）导出与导入

在ELF(Linux下动态库的格式)，共享库中所有的全局函数和变量在默认情况下都可以被其他模块使用，即ELF默认导出所有的全局符号。DLL不同，需要显式地“告诉”编译器需要导出某个符号，否则编译器默认所有的符号都不导出。

程序使用DLL的过程其实是引用DLL中导出函数和符号的过程，即导入过程。对于从其他DLL导入的符号，需要使用“__declspec(dllimport)”显式声明某个符号为导入符号。在ELF中，使用外部符号时，不需要额外声明该符号是从其他共享对象导入的。

指定符号的导入导出一般有如下两种方法：

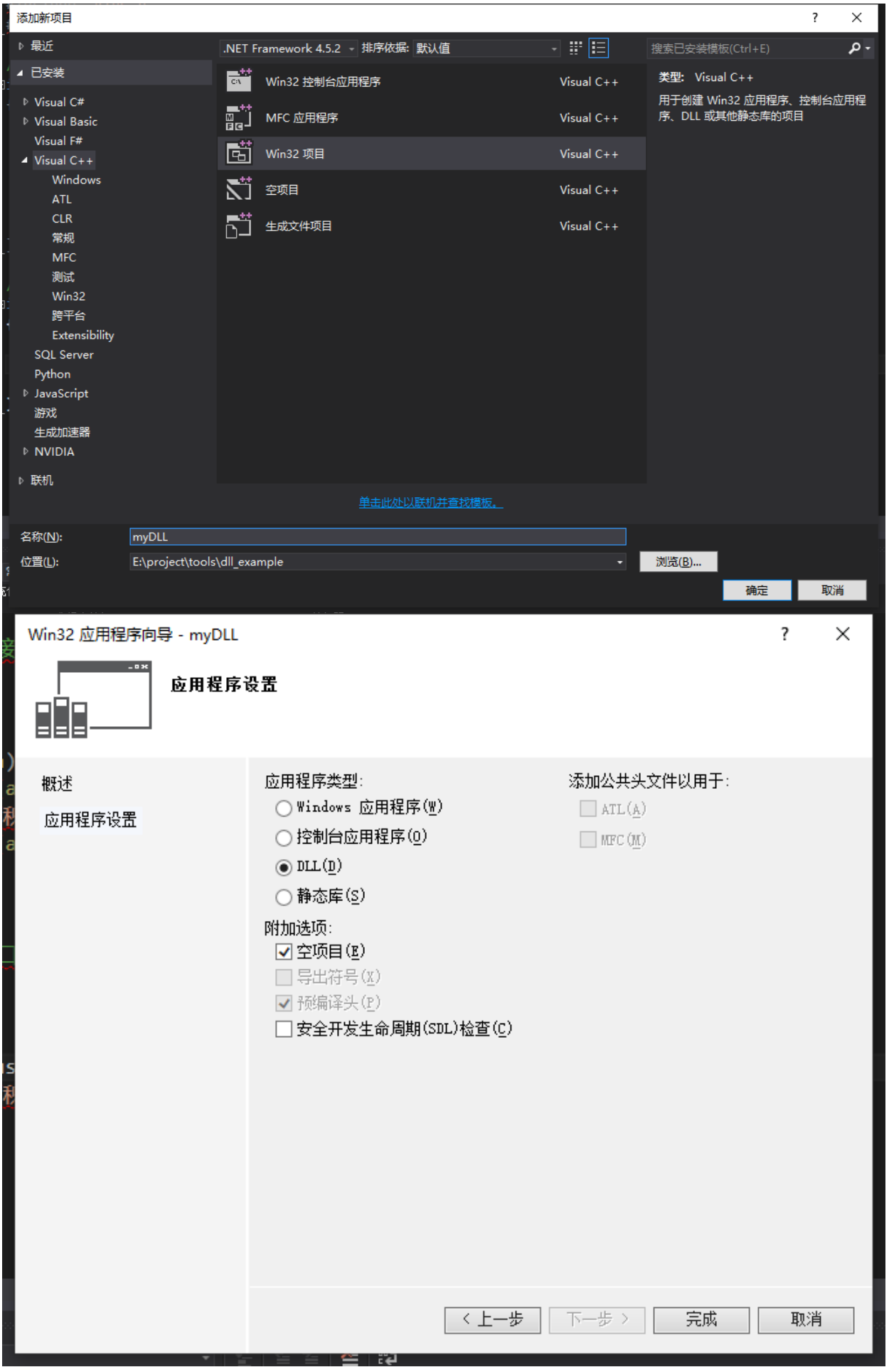
1）MSVC编译器提供了一系列C/C++的扩展来指定符号的导入导出，即**__declspec属性关键字**。

__declspec(dllexport) 表示该符号是从本DLL导出的符号

__declspec(dllimport) 表示该符号是从别的DLL中导入的

2）使用“.def”文件来声明导入到导出符号，参考《程序员的自我修养--链接、装载库》。

DLL创建


添加新项目
最近
已安装
Visual C#
Visual Basic
Visual F#
Visual C++
Windows
ATL
CLR
常规
MFC
测试
Win32
跨平台
Extensibility
SQL Server
Python
JavaScript
游戏
生成加速器
NVIDIA
联机
.NET Framework 4.5.2
排序依据: 默认值
Win32 控制台应用程序 Visual C++
MFC 应用程序 Visual C++
Win32 项目 Visual C++
空项目 Visual C++
生成文件项目 Visual C++
搜索已安装模板(Ctrl+E)
类型: Visual C++
用于创建 Win32 应用程序、控制台应用程序、DLL 或其他静态库的项目
单击此处以联机并查找模板。
名称(N): myDLL
位置(L): E:\project\tools\dll_example
浏览(B)...
确定
取消
Win32 应用程序向导 - myDLL
应用程序设置
概述
应用程序设置
应用程序类型:
Windows 应用程序(W)
控制台应用程序(O)
DLL(D)
静态库(S)
附加选项:
空项目(E)
导出符号(X)
预编译头(P)
安全开发生命周期(SDL)检查(C)
添加公共头文件以用于:
ATL(A)
MFC(M)
< 上一步
下一步 >
完成
取消

下面是头文件内容：创建工程时有默认的导出函数，这是删除掉重新写的。

```
#ifdef MYDLL_EXPORTS
#define DLL_EXPORT_IMPORT __declspec(dllexport)
#else
#define DLL_EXPORT_IMPORT __declspec(dllimport)
#endif

class DLL_EXPORT_IMPORT AlgCPP{
```

这里将AlgCPP声明为导出类（或导出函数）；当定义了符号MYDLL_EXPORTS，DLL_EXPORT_IMPORT被设置为 **__declspec(dllexport)** 修饰符，
若未定义则DLL_EXPORT_IMPORT被设置为__declspec(dllimport)，
当DLL项目生成时，DLL_EXPORT_IMPORT默认是定义的，**所以默认设置的是__declspec(dllexport) 修饰符**， 可以在C/C++->预处理器查看。

然后编译就会生成对应的dll文件，同时也会生成对应的lib文件。

DLL的隐式调用

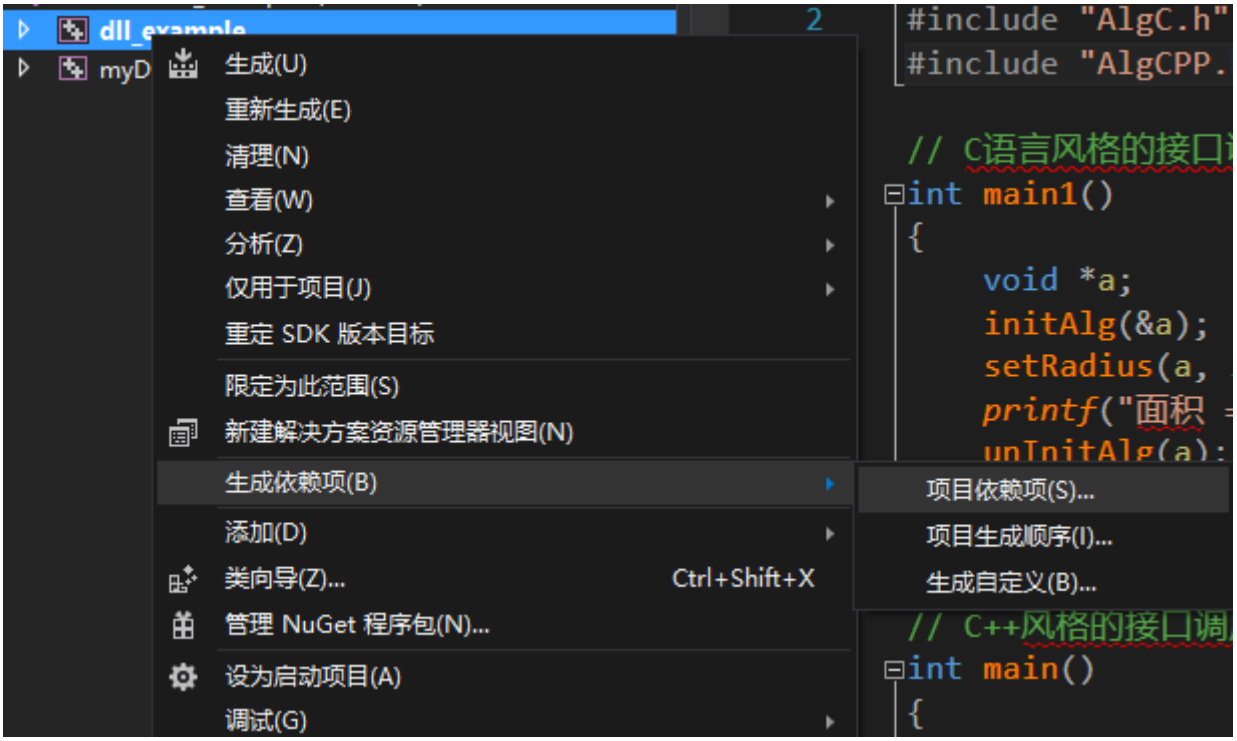
隐式链接采用静态加载的方式，比较简单，需要.h、.lib、.dll三件套。

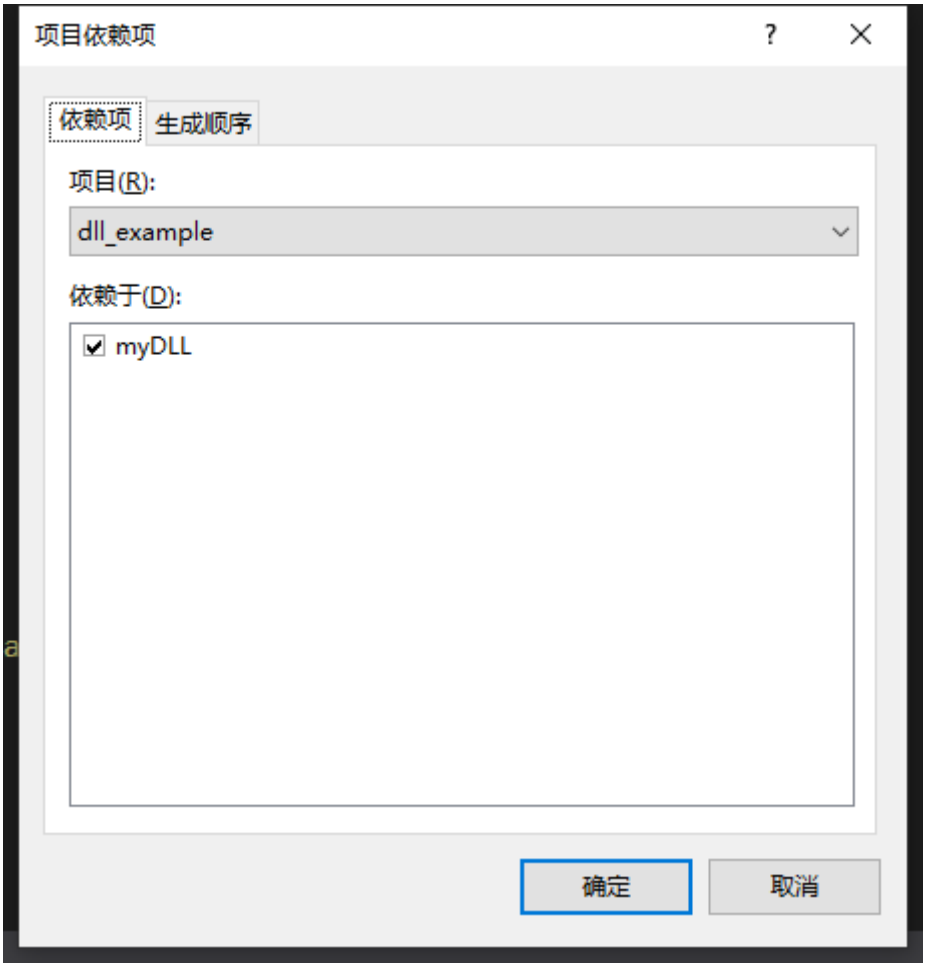
配置属性->C/C++->常规-> 在“包含目录”里添加头文件AlgCPP.h所在目录(..\myDLL)

配置属性->链接器->常规-> 在“附加库目录”里添加头文件myDLL.lib所在目录 ($(OutDir))

配置属性->链接器->输入-> 在“附加依赖项”里添加“myDLL.lib”

在主项目中设置DLL的依赖项


dll_example
myD
生成(U)
重新生成(E)
清理(N)
查看(W)
分析(Z)
仅用于项目(J)
重定 SDK 版本目标
限定为此范围(S)
新建解决方案资源管理器视图(N)
生成依赖项(B)
添加(D)
类向导(Z)... Ctrl+Shift+X
管理 NuGet 程序包(N)...
设为启动项目(A)
调试(G)
项目依赖项(S)...
项目生成顺序(I)...
生成自定义(B)...
#include "AlgC.h"
#include "AlgCPP.
// C语言风格的接口
int main1()
{
void *a;
initAlg(&a);
setRadius(a,
printf("面积
unInitAlg(a);
// C++风格的接口调
int main()
{


项目依赖项
依赖项
生成顺序
项目(R):
dll_example
依赖于(D):
myDLL
确定
取消