|  |
| --- |
| DLL生成与使用 操作环境平台 以下教程保证在Windows 8.1 x64下使用Visual Studio 2013操作的正确性。 目录组织 根目录为解决方案文件夹“DLLtest”，其下有一个二进制文件（dll、lib和exe文件）的输出目录“Release”文件夹以及分别对应同名的两个项目的“DLLexport”和“DLLimport”文件夹。 生成DLL基本步骤新建工程 选择控制台应用程序，在向导中“应用程序类型”处选择DLL。    此外，勾选“附加选项”中的“导出符号”可以自动生成一个示例，演示如何导出类、全局变量和函数。 编辑头文件 假设我们要导出到dll的只有一个函数 ***f*** ，可以新建一个头文件（如本例中的“DLLexport.h”），向文件中添加以下代码：  // File : DLLexport.h  #ifdef DLLEXPORT\_EXPORTS  #define DLLEXPORT\_API \_\_declspec(dllexport)  #else  #define DLLEXPORT\_API \_\_declspec(dllimport)  #endif      DLLEXPORT\_API int f(); 编辑源文件 在一个cpp文件中（如本例中的“DLLexport.cpp”）实现“DLLexport.h”中声明的函数 ***f*** ：  // File : DLLexport.cpp  #include "DLLexport.h"      DLLEXPORT\_API int f()  {      return 87;  } 生成解决方案 生成解决方案之后，可以在输出目录（如解决方案中的Release文件夹）中看到“DLLexport.lib”和“DLLexport.dll”，实际名称取决于项目名称，你也可以在项目属性中更改用自己的名称替换默认命名。 新建项目向导背后的工作 如果你觉得Visual Studio自动生成的文件很烦人，你也可以选择新建空项目。那么，项目模板为你做了哪些工作呢？ 配置类型 控制台应用程序的配置类型都是exe，而在这里我们要选中dll。   预定义宏 Visual Studio会预先定义一个名称为 ***$(ProjectName)\_EXPORTS*** 的宏，其中 ***$(ProjectName)*** 为工程名中所有字母的大写。利用这个宏，我们可以进行条件编译，使同一个头文件既可以作为生成dll的工程的源码，也可以用于使用隐式链接方式使用dll的工程的源码。例如我们创建了一个名为“DLLexport”的工程， ***DLLEXPORT\_EXPORTS*** 宏将被预先定义好。   其他 对比空项目的项目配置和使用dll项目模板创建的项目属性还可以发现其他的不同之处，但是似乎不影响dll的正常编译和使用。 为隐式链接做准备 隐式链接有多种方式，其中有些方法需要对项目属性进行额外的设置，为了省略这些步骤，我们可以通过在头文件中添加适当的 ***#pragma*** 指示来化简该步骤。在头文件中添加几行，变成如下代码：  // File : DLLexport.h  #ifdef DLLEXPORT\_EXPORTS  #define DLLEXPORT\_API \_\_declspec(dllexport)  #else  #define DLLEXPORT\_API \_\_declspec(dllimport)  #define DLL\_PATH "../Release/DLLexport.dll" // code added  #define LIB\_PATH "../Release/DLLexport.lib" // code added  #pragma comment(lib, LIB\_PATH)              // code added  #endif      DLLEXPORT\_API int f();  这几行代码定义了.dll和.lib文件的路径，并指示编译器寻找对应的库文件。由于之后要使用的 ***LoadLibrary*** 函数加载dll时可以自动添加默认后缀“.dll”，而此处的静态库加载指示会自动添加默认后缀“.lib”，故可以省略后缀。  注意， ***LIB\_PATH*** 宏要根据你编译程序时的实际情况设置对应的值，而 ***DLL\_PATH*** 要根据部署时的实际情况设置对应的值。 使用DLL隐式链接直接使用（推荐） 如果你添加了2.3节中提到的代码，那么恭喜你已经可以像使用一个普通的函数一样调用dll中的函数了！  新建一个空项目，新建一个cpp文件（如本例中的“main.cpp”），在该文件中添加如下代码：  // File : main.cpp  #include <iostream>    #include "../DLLexport/DLLexport.h"      using namespace std;      int main()  {      cout << f() << endl;  }  注意其中的头文件要设置正确的路径，在本例中按1.2节的目录进行组织。 添加引用 如果没有添加2.3节中提到的代码，但是生成dll的项目和使用dll的项目在同一个解决方案里，可以通过如下步骤进行隐式链接。  在项目上点鼠标右键，选择添加引用：      勾选生成dll的项目并确定：   配置附加库 将2.1.4节中提到的“DLLexport.lib”文件的目录添加到“附加库目录”中（建议使用相对路径）：    然后将库文件添加到“附加依赖项”中：   显示链接 使用显示链接可以不用添加2.3节中提到的代码。 查看被导出的函数 Visual Studio提供了dumpbin工具用来查看dll文件中被导出了的函数。  对于C++项目，对应工具的位置位于“C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\bin\dumpbin.exe”（我的Visual Studio使用了默认安装路径）。  使用该工具的 /exports 选项可以导出我们需要的内容：    其中 ordinal 列中的编号将在3.2.2节中使用， name 列中等号左侧的名称将在将在3.2.3节中使用。 使用编号链接 将3.1.1节中的代码改为（注意，因为没include头文件，所以需要重新定义 ***DLL\_PATH*** 宏）：  // File : main.cpp  #include <windows.h>  #include <iostream>      #define DLL\_PATH "../Release/DLLexport.dll"      using namespace std;      typedef int( \*Func )();      int main()  {      HINSTANCE hDLL;     // Handle to DLL        hDLL = LoadLibrary( DLL\_PATH );  if (hDLL != NULL) {          Func f = (Func)GetProcAddress( hDLL, MAKEINTRESOURCE( 1 ) );          cout << (f ? f() : 0) << endl;      }        FreeLibrary( hDLL );      return 0;  } 使用名称链接 使用名称链接只需将          Func f = (Func)GetProcAddress( hDLL, MAKEINTRESOURCE( 1 ) );  替换为：          Func f = (Func)GetProcAddress( hDLL, "?f@@YAHH@Z" );  即可。 使用.def文件 该方法需要自行编写“.def”文件，并在项目属性中指定该文件的位置。自动化程度较低，不推荐使用。在此仅为方法列举的完整性将其列出。 |