

## 一、 编写第一个入门级dll文件

1. 新建一个dlltest的dll工程，加入一源文件dlltest.cpp，包含add和subtract两个函数如下：

```
[cpp] 1. _declspec(dllexport) int add(int a,int b)
2. {
3.     return a+b;
4. }
5.
6. _declspec(dllexport) int subtract(int a,int b)
7. {
8.     return a-b;
9. }
```

注意：

在函数名前加上 `_declspec(dllexport)`，这样编译后在连接的时候才会生成dlltest.lib（引入库文件）和dlltest.exp（输出库文件），并且在命令提示符下用dumpbin命令可以看到导出了哪些函数。

方法：在命令提示符下，转到dll文件所在目录下，输入dumpbin -exports dlltest.dll,列出导出函数如下：

□

这里注意到函数名字已经被改成了 `?add@@@YAHHH@Z`，这种现象叫做 **名字粉碎**，是为了支持函数重载而做的。

2. 编写一个基于对话框的MFC 程序测试DLL,工程名为calldll，放置两个按钮add和subtract,响应按钮消息，调用这个DLL的add和subtract函数。

先添加响应按钮消息的函数 OnAdd和OnSubtract，然后在函数所在源文件中写完整函数体如下：

```
[cpp] 1. //extern int add(int,int);
2. //extern int subtract(int,int);
3. _declspec(dllimport) int add(int,int);
4. _declspec(dllimport) int subtract(int,int);
5.
6. void CCalldllDlg::OnAdd()
7. {
8.     // TODO: Add your control notification handler code here
9.     CString str;
10.    str.Format("2+3=%d",add(2,3));
11.    MessageBox(str);
12. }
13.
14. void CCalldllDlg::OnSubtract()
15. {
16.     // TODO: Add your control notification handler code here
17.     CString str;
18.    str.Format("3-2=%d",subtract(3,2));
19.    MessageBox(str);
20. }
```

这里采用 **隐式链接** 的方式加载动态链接库：

**注意：** 要用关键字extern先声明这两个函数,表明这两个函数是在外部定义的(不过程序中将它注释掉了)。最好是用 `_declspec(dllimport)` 表明函数是从动态链接库的lib文件中引入的，这样效率更高。

将文件dlltest.lib拷贝到此工程目录下，并在Project Settings的Link标签下添加此文件：（否则编译会成功，但连接时会出错提示找不到函数的定义）

编译连接生成calldll.exe，这时 用Dumpbin -imports calldll .exe 查看它的输入信息，可以看到它加载了 dlltest .dll。

运行 calldll.exe（要成功运行还需将dlltest .dll 拷贝到工程目录或此exe 所在目录下）

这样，最简单的第一个dll就完成了。

## 二、 编写涉及到类和头文件的dll文件

1. 新建一个dlltest 的dll工程，加入一头文件dlltest.h和之源文件dlltest.cpp。

[dlltest.h:](#)

```

1.  #ifdef DLL_API
2.  #else
3.  #define DLL_API __declspec(dllimport)
4.  #endif
5.
6.  DLL_API int add(int,int);
7.  DLL_API int subtract(int,int);
8.
9.  class DLL_API Point1          //将整个类导出
10. {
11. public:
12.     void output(int x,int y);
13.     void output1(int x,int y);
14. };
15.
16. class Point2
17. {
18. public:
19.     DLL_API void output(int x,int y);    //仅导出类中的一个函数
20.     void output1(int x,int y);
21. };

```

dlltest.cpp :

```

1.  #define DLL_API __declspec(dllexport)
2.  #include "dlltest.h"
3.  #include <windows.h>
4.  #include <stdio.h>
5.
6.  int add(int a,int b)
7.  {
8.      return a+b;
9.  }
10.
11. int subtract(int a,int b)
12. {
13.     return a-b;
14. }
15.
16. void Point1::output(int x,int y)
17. {
18.     HWND hwnd=GetForegroundWindow();
19.     HDC hdc=GetDC(hwnd);
20.     char buf[50];
21.     memset(buf,0,50);
22.     sprintf(buf,"x=%d,y=%d",x,y);
23.     TextOut(hdc,0,0,buf,strlen(buf));
24. }
25.
26. void Point1::output1(int x,int y)
27. {
28.     HWND hwnd=GetForegroundWindow();
29.     HDC hdc=GetDC(hwnd);
30.     char buf[50];
31.     memset(buf,0,50);
32.     sprintf(buf,"x=%d,y=%d",x,y);
33.     TextOut(hdc,0,0,buf,strlen(buf));
34. }
35.
36. void Point2::output(int x,int y)
37. {
38.     HWND hwnd=GetForegroundWindow();
39.     HDC hdc=GetDC(hwnd);
40.     char buf[50];
41.     memset(buf,0,50);
42.     sprintf(buf,"x=%d,y=%d",x,y);
43.     TextOut(hdc,0,20,buf,strlen(buf));
44. }
45.
46. void Point2::output1(int x,int y)
47. {
48.     HWND hwnd=GetForegroundWindow();
49.     HDC hdc=GetDC(hwnd);
50.     char buf[50];
51.     memset(buf,0,50);
52.     sprintf(buf,"x=%d,y=%d",x,y);
53.     TextOut(hdc,0,20,buf,strlen(buf));
54. }

```

2. 编写一个基于对话框的MFC 程序测试DLL,此时在工程中包含上面这个头文件，不用在声明导入的类和函数了。

### 三、使用模块定义文件和动态加载动态链接库

1. 新建dlltest.dll工程，加入一源文件dlltest.cpp，包含add和subtract两个函数如下：

```
[cpp]
1.  int add(int a,int b)
2.  {
3.      return a+b;
4.  }
5.
6.  int subtract(int a,int b)
7.  {
8.      return a-b;
9.  }
```

2. 使用模块定义文件防止文件名改变，在目录中新建dlltest.def 文件，增加到工程。

```
[plain]
1.  LIBRARY dlltest
2.  EXPORTS
3.  add
4.  subtract
```

注：LIBRARY 和 EXPORTS 的用法参照 MSDN.

3. 编译后用 dumpbin 查看到函数名没有改变。

□

4. 用动态加载的方法来调用 dll 文件。

```
[cpp]
1.  void CCallDllDlg::OnAdd()
2.  {
3.      // TODO: Add your control notification handler code here
4.      HINSTANCE hInst;
5.      hInst=LoadLibrary("dlltest.dll");
6.      typedef int (*_stdcall*) *ADDPROC(int a,int b);
7.      //ADDPROC Add=(ADDPROC)GetProcAddress(hInst,"?add@YAHHH@Z");
8.      //ADDPROC Add=(ADDPROC)GetProcAddress(hInst,MAKEINTRESOURCE(1));
9.      ADDPROC Add=(ADDPROC)GetProcAddress(hInst,"add");
10.     if(!Add)
11.     {
12.         MessageBox("获取函数地址失败!");
13.         return;
14.     }
15.     CString str;
16.     str.Format("2+3=%d",Add(2,3));
17.     MessageBox(str);
18.     FreeLibrary(hInst);
19. }
```

注意：

1. 这里调用的 GetProcAddress 函数中的第二个参数是动态链接库中导出的函数名,如果动态链接库中没有用模块定义文件防止函数名粉碎,则要用注释掉的 GetProcAddress(hInst,"?add@YAHHH@Z"),另外也可按序号访问 GetProcAddress(hInst,MAKEINTRESOURCE(1))。
2. 使用模块定义文件后,如果改变调用约定为 \_stdcall,函数名也不会被改变,但如果加上 \_stdcall 定义函数,调用时也需要加上 \_stdcall,否则会出错。
3. 动态加载不需要将文件dlltest.lib拷贝到此工程目录下,并在Project Settings的Link标签下添加此文件,只需将dll文件拷贝到工程目录下即可,并且通过dumpbin -imports call.dll.exe查看它的输入信息时,可以看到它并没有加载dlltest.dll。
4. 动态加载的好处是需要时再加载,可以提高执行的效率。当不使用 DLL 时,调用 FreeLibrary 减少 DLL 的使用计数,释放 DLL 资源,减少系统负担。
5. 不使用模块定义文件,也可用 extern "C" 使函数名保持不变,如 #define DLL1\_API extern "C" \_\_declspec(dllexport),但它只能导出全局函数,不能导出类的成员函数,并且如果调用约定被改成了别的方式,此时函数名也被改变,所以一般不用这种方式。

文章标签： C++ DLL 入门

个人分类： 纯编程

