

鸡啄米

聚焦互联网、数码、软件开发和编程入门的IT休闲吧

首页 IT互联网 数码生活 软件开发 职场人生 娱乐休闲 编程课堂 安卓开发 留言簿

首页 » 软件开发 » DLL动态链接库编程入门之二：非MFC DLL

DLL动态链接库编程入门之二：非MFC DLL

分类标签: VC++ 编程入门 Windows

上一节中讲解的是[DLL概论及其调试和查看](#)，本节将为大家详解非MFC DLL的相关内容。

1、一个简单的DLL

上一节给出了以静态链接库方式提供add函数接口的方法，接下来我们来看看怎样用动态链接库实现一个同样功能的add函数。

如图1，在VC++中new一个Win32 Dynamic-Link Library工程dllTest。注意不要选择MFC AppWizard(dll)，因为用MFC AppWizard(dll)建立的将是后面要讲述的MFC动态链接库。

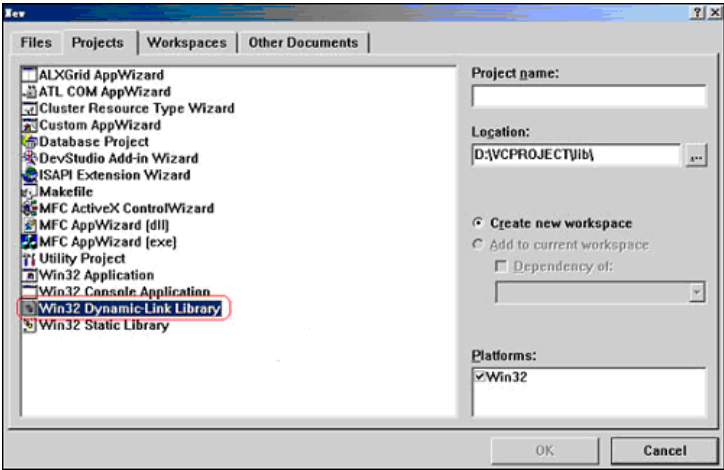


图1 建立一个非MFC DLL

在建立的工程中添加lib.h及lib.cpp文件，源代码如下：

```

c++代码
1.  /* 文件名: lib.h */
2.
3.  #ifndef LIB_H
4.  #define LIB_H
5.  extern "C" int __declspec(dllexport) add(int x, int y);
6.  #endif
7.
8.  /* 文件名: lib.cpp */
9.
10. #include "lib.h"
11.
12. int add(int x, int y)
13. {
14.     return x + y;
15. }
```

订阅鸡啄米

RSS + 订阅到QQ邮箱

分享

站内搜索

请输入搜索内容..

🔍

分类标签

- 编程入门 (135)
- C++ (96)
- VC++ (77)
- MFC (67)
- VS2010 (63)
- 程序员 (55)
- Android (51)
- Java (51)
- 苹果 (49)
- 智能手机 (47)
- 腾讯 (41)
- 百度 (37)
- 阿里巴巴 (33)
- 谷歌 (32)
- 平板电脑 (31)
- TCP/IP (29)
- iPhone (26)
- PHP (26)
- Javascript (25)
- 奇虎360 (24)
- Mysql (24)
- Windows (22)
- 软件架构 (20)
- 小米 (20)
- 设计模式 (19)
- iPad (18)
- Web (18)
- 职场攻略 (18)
- 三星 (16)
- 创业 (16)
- 微软 (13)
- iOS (13)
- 微信 (13)
- HTML (13)
- 应用程序 (12)

与上一节对静态链接库的调用相似，我们也建立一个与DLL工程处于同一工作区的应用工程dllCal I，它调用DLL中的函数add，其源代码如下：

c++代码

```
1.  #include <stdio.h>
2.  #include <windows.h>
3.
4.  typedef int (*lpAddFun) (int, int); //宏定义函数指针类型
5.
6.  int main(int argc, char *argv[])
7.  {
8.      HINSTANCE hDll; //DLL句柄
9.      lpAddFun addFun; //函数指针
10.
11.      hDll = LoadLibrary("../\\Debug\\dllTest.dll");
12.      if (hDll != NULL)
13.      {
14.          addFun = (lpAddFun)GetProcAddress(hDll, "add");
15.          if (addFun != NULL)
16.          {
17.              int result = addFun(2, 3);
18.              printf("%d", result);
19.          }
20.          FreeLibrary(hDll);
21.      }
22.      return 0;
23. }
```

分析上述代码，dllTest工程中的lib.cpp文件与上一节静态链接库版本完全相同，不同在于lib.h对函数add的声明前面添加了__declspec(dllexport)语句。这个语句的含义是声明函数add为DLL的导出函数。**DLL内的函数分为两种：**

- (1)DLL导出函数，可供应用程序调用；
- (2)DLL内部函数，只能在DLL程序使用，应用程序无法调用它们。

而应用程序对本DLL的调用和对上一节静态链接库的调用却有较大差异，下面我们来逐一分析。

首先，语句typedef int (* lpAddFun)(int,int)定义了一个与add函数接受参数类型和返回值均相同的函数指针类型。随后，在main函数中定义了lpAddFun的实例addFun；

其次，在函数main中定义了一个DLL HINSTANCE句柄实例hDll，通过Win32 API函数LoadLibrary动态加载了DLL模块并将DLL模块句柄赋给了hDll；

再次，在函数main中通过Win32 Api函数GetProcAddress得到了所加载DLL模块中函数add的地址并赋给了addFun。经由函数指针addFun进行了对DLL中add函数的调用；

最后，应用工程使用完DLL后，在函数main中通过Win32 Api函数FreeLibrary释放了已经加载的DLL模块。

通过这个简单的例子，我们获知DLL定义和调用的一般概念：

- (1)DLL中需以某种特定的方式声明导出函数（或变量、类）；
- (2)应用工程需以某种特定的方式调用DLL的导出函数（或变量、类）。

下面我们来对“特定的方式”阐述。

2、声明导出函数

- 新浪 (12)
- 微博 (11)
- 软件工程师 (10)
- 诺基亚 (10)
- 京东商城 (10)
- 比特币 (10)
- Facebook (9)
- 周鸿祎 (9)
- 操作系统 (8)
- Galaxy (8)
- 社交网络 (8)
- 搜索引擎 (8)
- 移动互联网 (8)
- C (8)
- 亚马逊 (7)
- 更多标签

分享

完全随机文章

- VS2010/MFC编程入门之五（MFC消息映...
从Facebook和Twitter获取新闻的用户...
- VS2010/MFC编程入门之二十五（常用控...
- VS2010/MFC编程入门之三十五（菜单： ...
- VS2010/MFC编程入门之四十一（文档、 ...
- VS2010/MFC编程入门教程之目录和总...
- 程序员的选择：技术vs管理
- 2015产品校招——阿里腾讯百度360小米...
- C、C++、python、Java、php、C#六种流行...
- App推广秘籍最全篇
- 程序员修炼指南——引导你成为真正的...
- 85后工作5年工资竟然涨了25倍——月薪...
- TCP/UDP网络编程入门教程之十五： TC...
- memcached使用场景和方法总结
- 从《奋斗》到《欢乐颂》看青年的价值观...
- StackOverflow 创始人推荐程序员看...
- 魅族的掉队已成事实，生态链不是那么...
- 详解HTML5 LocalStorage本地存储
- 如何避免成为下一个雅虎
- 说说Javascript闭包这点事

最新评论及回复

- CAddSheet(LPCTSTR ...
- 有没有XTP的教学？求教
- 就画个界面，搞这么负责，难怪MFC要被淘汰
- 讲的太好了，很全，很清楚！楼主你的Q...
- 楼主，请问如何动态给重写的CList...
- 普通人只有被剥削的份
- 已点广，，，，，告支持楼主
- #include <afxco...
- 关掉王者荣耀。它就像鸦片，勾引小孩子...
- 关掉王者荣耀。它就像鸦片，勾引小孩子
- 楼主，请问CTabCtrl和CList...
- 一、初始化函数中在设置好子对话框位置...
- 创建两组Radio可以在Radio的...
- 一定要通过【类向导】添加类，【类向导...
- 写的太棒了。
- 谢谢楼主
- 加油
- 弱的问一声？符号常量的用法是否跟C语...
- 蛮实用的可是在8年后才看到[REV...
- 好的程序员一定是挣钱的

最近发表

- 鸡啄米开始承接项目啦

DLL中导出函数的声明有两种方式：一种为第1节例子中给出的在函数声明中加上`__declspec(dllexport)`，这里不再举例说明；另外一种方式是采用模块定义(.def)文件声明，.def文件为链接器提供了有关被链接程序的导出、属性及其他方面的信息。

下面的代码演示了怎样同.def文件将函数add声明为DLL导出函数（需在dllTest工程中添加lib.def文件）：

```
c++代码
1. ; lib.def : 导出DLL函数
2.
3. LIBRARY dllTest
4. EXPORTS
5. add @ 1
```

.def文件的规则为：

- (1)LIBRARY语句说明.def文件相应的DLL；
- (2)EXPORTS语句后列出要导出函数的名称。可以在.def文件中的导出函数名后加@n，表示要导出函数的序号为n（在进行函数调用时，这个序号将发挥其作用）；
- (3).def 文件中的注释由每个注释行开始处的分号(;)指定，且注释不能与语句共享一行。

由此可以看出，例子中lib.def文件的含义为生成名为“dllTest”的动态链接库，导出其中的add函数，并指定add函数的序号为1。

3、DLL的调用方式

在第1节的例子中我们看到了由“LoadLibrary-GetProcAddress-FreeLibrary”系统Api提供的三位一体“DLL加载-DLL函数地址获取-DLL释放”方式，这种调用方式称为DLL的动态调用。

动态调用方式的特点是完全由编程者用 API 函数加载和卸载 DLL，程序员可以决定 DLL 文件何时加载或不加载，显式链接在运行时决定加载哪个 DLL 文件。

与动态调用方式相对应的就是静态调用方式，“有动必有静”，这来源于物质世界的对立统一。“动与静”，其对立与统一竟无数次在技术领域里得到验证，譬如静态IP与DHCP、静态路由与动态路由等。从前文我们已经知道，库也分为静态库与动态库DLL，而想不到，深入到DLL内部，其调用方式也分为静态与动态。“动与静”，无处不在。《周易》已认识到有动必有静的动静平衡观，《易·系辞》曰：“动静有常，刚柔断矣”。哲学意味着一种普遍的真理，因此，我们经常可以在枯燥的技术领域看到哲学的影子。

静态调用方式的特点是由编译系统完成对DLL的加载和应用程序结束时 DLL 的卸载。当调用某DLL的应用程序结束时，若系统中还有其它程序使用该 DLL，则Windows对DLL的应用记录减1，直到所有使用该DLL的程序都结束时才释放它。静态调用方式简单实用，但不如动态调用方式灵活。

下面我们来看看静态调用的例子，将编译dllTest工程所生成的.lib和.dll文件拷入dllCall工程所在的路径，dllCall执行下列代码：

```
c++代码
1. #pragma comment(lib,"dllTest.lib")
2. // .lib文件中仅仅是关于其对应DLL文件中函数的重定位信息
3.
4. extern "C" __declspec(dllimport) add(int x,int y);
5.
6. int main(int argc, char* argv[])
7. {
8.     int result = add(2,3);
9.     printf("%d",result);
10.    return 0;
11. }
```

小白照样读懂的VLAN原理讲解

SSH电商项目实战之十：商品类基本模块的搭建

SSH电商项目实战之九：添加和更新商品类别功能的实现

SSH电商项目实战之八：查询和删除商品类别功能的实现

SSH电商项目实战之七：Struts2和Json的整合

长文：内容产业的赢家与输家

SSH电商项目实战之六：基于DataGrid的数据显示

SSH电商项目实战之五：完成数据库的级联查询和分页

SSH电商项目实战之四：EasyUI菜单的实现

SSH电商项目实战之三：使用EasyUI搭建后台页面框架

SSH电商项目实战之二：基本增删查改、Service和Action的抽取以及使用注解替换xml

大妈：我们不懂ICO和X币，但知道比炒房厉害

SSH电商项目实战之一：整合Struts2、Hibernate和Spring

面临连续亏损，HTC出售手机还是VR业务？

分享

由上述代码可以看出，静态调用方式的顺利进行需要完成两个动作：

(1)告诉编译器与DLL相对应的.lib文件所在的路径及文件名，`#pragma comment(lib,"dllTest.lib")`就是起这个作用。

程序员在建立一个DLL文件时，连接器会自动为其生成一个对应的.lib文件，该文件包含了DLL 导出函数的符号名及序号（并不含有实际的代码）。在应用程序里，.lib文件将作为DLL的替代文件参与编译。

(2)声明导入函数，`extern "C" __declspec(dllimport) add(int x,int y)`语句中的`__declspec(dllimport)`发挥这个作用。

静态调用方式不再需要使用系统API来加载、卸载DLL以及获取DLL中导出函数的地址。这是因为，当程序员通过静态链接方式编译生成应用程序时，应用程序中调用的与.lib文件中导出符号相匹配的函数符号将进入到生成的EXE 文件中，.lib文件中所包含的与之对应的DLL文件的文件名也被编译器存储在EXE文件内部。当应用程序运行过程中需要加载DLL文件时，Windows将根据这些信息发现并加载DLL，然后通过符号名实现对DLL 函数的动态链接。这样，EXE将能直接通过函数名调用DLL的输出函数，就象调用程序内部的其他函数一样。

4、DllMain函数

Windows在加载DLL的时候，需要一个入口函数，就如同控制台或DOS程序需要main函数、WIN 32程序需要WinMain函数一样。在前面的例子中，DLL并没有提供DllMain函数，应用工程也能成功引用DLL，这是因为Windows在找不到DllMain的时候，系统会从其它运行库中引入一个不做任何操作的缺省DllMain函数版本，并不意味着DLL可以放弃DllMain函数。

根据编写规范，Windows必须查找并执行DLL里的DllMain函数作为加载DLL的依据，它使得DLL得以保留在内存里。这个函数并不属于导出函数，而是DLL的内部函数。这意味着不能直接在应用工程中引用DllMain函数，DllMain是自动被调用的。

我们来看一个DllMain函数的例子。

c++代码

```
1.  BOOL APIENTRY DllMain( HANDLE hModule,
2.  DWORD ul_reason_for_call,
3.  LPVOID lpReserved
4.  )
5.  {
6.      switch (ul_reason_for_call)
7.      {
8.          case DLL_PROCESS_ATTACH:
9.              printf("\nprocess attach of dll");
10.             break;
11.          case DLL_THREAD_ATTACH:
12.              printf("\nthread attach of dll");
13.              break;
14.          case DLL_THREAD_DETACH:
15.              printf("\nthread detach of dll");
16.              break;
17.          case DLL_PROCESS_DETACH:
18.              printf("\nprocess detach of dll");
19.              break;
20.      }
21.      return TRUE;
22.  }
```

DllMain函数在DLL被加载和卸载时被调用，在单个线程启动和终止时，DllMain函数也被调用，ul_reason_for_call指明了被调用的原因。原因共有4种，即PROCESS_ATTACH、PROCESS_DETACH、THREAD_ATTACH和THREAD_DETACH，以switch语句列出。

分享

来仔细解读一下DllMain的函数头BOOL APIENTRY DllMain(HANDLE hModule, WORD ul_reason_for_call, LPVOID lpReserved)。

APIENTRY被定义为__stdcall，它意味着这个函数以标准Pascal的方式进行调用，也就是WINAPI方式；

进程中的每个DLL模块被全局唯一的32字节的HINSTANCE句柄标识，只有在特定的进程内部有效，句柄代表了DLL模块在进程虚拟空间中的起始地址。在Win32中，HINSTANCE和HMODULE的值是相同的，这两种类型可以替换使用，这就是函数参数hModule的来历。

执行下列代码：

```
c++代码
1.  hDll = LoadLibrary("../Debug\\dllTest.dll");
2.  if (hDll != NULL)
3.  {
4.      addFun = (lpAddFun)GetProcAddress(hDll, MAKEINTRESOURCE(1));
5.      //MAKEINTRESOURCE直接使用导出文件中的序号
6.      if (addFun != NULL)
7.      {
8.          int result = addFun(2, 3);
9.          printf("ncall add in dll:%d", result);
10.     }
11.     FreeLibrary(hDll);
12. }
```

我们看到输出顺序为：

process attach of dll

call add in dll:5

process detach of dll

这一输出顺序验证了DllMain被调用的时机。

代码中的GetProcAddress (hDll, MAKEINTRESOURCE (1))值得留意，它直接通过.def文件中为add函数指定的顺序号访问add函数，具体体现在MAKEINTRESOURCE (1)，MAKEINTRESOURCE是一个通过序号获取函数名的宏，定义为（节选自winuser.h）：

```
c++代码
1.  #define MAKEINTRESOURCEA(i) (LPSTR) ((DWORD) ((WORD) (i)))
2.  #define MAKEINTRESOURCEW(i) (LPWSTR) ((DWORD) ((WORD) (i)))
3.  #ifdef UNICODE
4.      #define MAKEINTRESOURCE MAKEINTRESOURCEW
5.  #else
6.      #define MAKEINTRESOURCE MAKEINTRESOURCEA
```

5、__stdcall约定

如果通过VC++编写的DLL欲被其他语言编写的程序调用，应将函数的调用方式声明为__stdcall方式，WINAPI都采用这种方式，而C/C++缺省的调用方式却为__cdecl。__stdcall方式与__cdecl对函数名最终生成符号的方式不同。若采用C编译方式(在C++中需将函数声明为extern "C")，__stdcall调用约定在输出函数名前面加下划线，后面加"@"符号和参数的字节数，形如_functionname@number；而__cdecl调用约定仅在输出函数名前面加下划线，形如_functionname。

Windows编程中常见的几种函数类型声明宏都是与__stdcall和__cdecl有关的（节选自windef.h）：

```
c++代码
```

```
1. #define CALLBACK __stdcall //这就是传说中的回调函数
2. #define WINAPI __stdcall //这就是传说中的WINAPI
3. #define WINAPIV __cdecl
4. #define APIENTRY WINAPI //DllMain的入口就在这里
5. #define APIPRIVATE __stdcall
6. #define PASCAL __stdcall
```

在lib.h中，应这样声明add函数：

```
int __stdcall add(int x, int y);
```

在应用工程中函数指针类型应定义为：

```
typedef int(__stdcall *lpAddFun)(int, int);
```

若在lib.h中将函数声明为__stdcall调用，而应用工程中仍使用typedef int (* lpAddFun)(int,int)，运行时将发生错误（因为类型不匹配，在应用工程中仍然是缺省的__cdecl调用），弹出如图2所示的对话框。



图2 调用约定不匹配时的运行错误

图2中的那段话实际上已经给出了错误的原因，即“This is usually a result of ...”。

6、DLL导出变量

DLL定义的全局变量可以被调用进程访问；DLL也可以访问调用进程的全局数据，我们来看看在应用工程中引用DLL中变量的例子。

c++代码

```
1. /* 文件名: lib.h */
2. #ifndef LIB_H
3. #define LIB_H
4. extern int dllGlobalVar;
5. #endif
6.
7. /* 文件名: lib.cpp */
8. #include "lib.h"
9. #include <windows.h>
10.
11. int dllGlobalVar;
12.
13. BOOL APIENTRY DllMain(HANDLE hModule, DWORD ul_reason_for_call, LPVOID lpReserved)
14. {
15.     switch (ul_reason_for_call)
16.     {
17.     case DLL_PROCESS_ATTACH:
18.         dllGlobalVar = 100; //在dll被加载时，赋全局变量为100
19.         break;
```

```
20.     case DLL_THREAD_ATTACH:
21.     case DLL_THREAD_DETACH:
22.     case DLL_PROCESS_DETACH:
23.         break;
24.     }
25.     return TRUE;
26. }
```

c++代码

```
1.  ;文件名: lib.def
2.  ;在DLL中导出变量
3.
4.  LIBRARY "dllTest"
5.  EXPORTS
6.  dllGlobalVar CONSTANT
7.  ;或dllGlobalVar DATA
8.  GetGlobalVar
```

从lib.h和lib.cpp中可以看出，全局变量在DLL中的定义和使用方法与一般的程序设计是一样的。
若要导出某全局变量，我们需要在.def文件的EXPORTS后添加：

变量名 CONSTANT ~~//过时的方法~~

~~——或~~

变量名 ~~DATA~~ ~~——~~ ~~//VC++提示的新方法~~

在主函数中引用DLL中定义的全局变量：

c++代码

```
1.  #include <stdio.h>
2.  #pragma comment(lib,"dllTest.lib")
3.  extern int dllGlobalVar;
4.
5.  int main(int argc, char *argv[])
6.  {
7.      printf("%d ", *(int*)dllGlobalVar);
8.      *(int*)dllGlobalVar = 1;
9.      printf("%d ", *(int*)dllGlobalVar);
10.     return 0;
11. }
```

特别要注意的是用extern int dllGlobalVar声明所导入的并不是DLL中全局变量本身，而是其地址，应用程序必须通过强制指针转换来使用DLL中的全局变量。这一点，从*(int*)dllGlobalVar可以看出。因此在采用这种方式引用DLL全局变量时，千万不要进行这样的赋值操作：

dllGlobalVar = 1;

其结果是dllGlobalVar指针的内容发生变化，程序中以后再也引用不到DLL中的全局变量了。

在应用工程中引用DLL中全局变量的一个更好方法是：

c++代码

```
1.  #include <stdio.h>
2.  #pragma comment(lib,"dllTest.lib")
3.
4.  extern int _declspec(dllimport) dllGlobalVar; //用_declspec(dllimport)
   导入
```



```
5.
6.  int main(int argc, char *argv[])
7.  {
8.      printf("%d ", dllGlobalVar);
9.      dllGlobalVar = 1; //这里就可以直接使用，无须进行强制指针转换
10.     printf("%d ", dllGlobalVar);
11.     return 0;
12. }
```

通过 `_declspec(dllexport)` 方式导入的就是DLL中全局变量本身而不再是其地址了，笔者建议在一切可能的情况下都使用这种方式。

7、DLL导出类

DLL中定义的类可以在应用工程中使用。

下面的例子里，我们在DLL中定义了 `point` 和 `circle` 两个类，并在应用工程中引用了它们。

c++代码

```
1.  //文件名: point.h, point类的声明
2.  #ifndef POINT_H
3.  #define POINT_H
4.  #ifdef DLL_FILE
5.      class _declspec(dllexport) point //导出类point
6.  #else
7.      class _declspec(dllimport) point //导入类point
8.  #endif
9.  {
10. public:
11.     float y;
12.     float x;
13.     point();
14.     point(float x_coordinate, float y_coordinate);
15. };
16. #endif
17.
18. //文件名: point.cpp, point类的实现
19. #ifndef DLL_FILE
20. #define DLL_FILE
21. #endif
22. #include "point.h"
23.
24. //类point的缺省构造函数
25. point::point()
26. {
27.     x = 0.0;
28.     y = 0.0;
29. }
30.
31. //类point的构造函数
32. point::point(float x_coordinate, float y_coordinate)
33. {
34.     x = x_coordinate;
35.     y = y_coordinate;
36. }
37.
```



```
38. //文件名: circle.h, circle类的声明
39. #ifndef CIRCLE_H
40. #define CIRCLE_H
41. #include "point.h"
42. #ifdef DLL_FILE
43. class _declspec(dllexport) circle //导出类circle
44. #else
45. class _declspec(dllimport) circle //导入类circle
46. #endif
47. {
48. public:
49.     void SetCentre(const point &rePoint);
50.     void SetRadius(float r);
51.     float GetGirth();
52.     float GetArea();
53.     circle();
54. private:
55.     float radius;
56.     point centre;
57. };
58. #endif
59.
60. //文件名: circle.cpp, circle类的实现
61. #ifndef DLL_FILE
62. #define DLL_FILE
63. #endif
64. #include "circle.h"
65. #define PI 3.1415926
66.
67. //circle类的构造函数
68. circle::circle()
69. {
70.     centre = point(0, 0);
71.     radius = 0;
72. }
73.
74. //得到圆的面积
75. float circle::GetArea()
76. {
77.     return PI *radius * radius;
78. }
79.
80. //得到圆的周长
81. float circle::GetGirth()
82. {
83.     return 2 *PI * radius;
84. }
85.
86. //设置圆心坐标
87. void circle::SetCentre(const point &rePoint)
88. {
89.     centre = centrePoint;
90. }
91.
```

```
92. //设置圆的半径
93. void circle::SetRadius(float r)
94. {
95.     radius = r;
96. }
```

类的引用：

c++代码

```
1. #include "..\circle.h" //包含类声明头文件
2. #pragma comment(lib,"dllTest.lib");
3.
4. int main(int argc, char *argv[])
5. {
6.     circle c;
7.     point p(2.0, 2.0);
8.     c.SetCentre(p);
9.     c.SetRadius(1.0);
10.    printf("area:%f girth:%f", c.GetArea(), c.GetGirth());
11.    return 0;
12. }
```

从上述源代码可以看出，由于在DLL的类实现代码中定义了宏DLL_FILE，故在DLL的实现中所包含的类声明实际上为：

c++代码

```
1. class _declspec(dllexport) point //导出类point
2. {
3.     ...
4. }
```

和

c++代码

```
1. class _declspec(dllexport) circle //导出类circle
2. {
3.     ...
4. }
```

而在应用工程中没有定义DLL_FILE，故其包含point.h和circle.h后引入的类声明为：

c++代码

```
1. class _declspec(dllimport) point //导入类point
2. {
3.     ...
4. }
```

和

c++代码

```
1. class _declspec(dllimport) circle //导入类circle
2. {
3.     ...
4. }
```

不错，正是通过DLL中的

C++代码

```
1.  class _declspec(dllexport) class_name //导出类circle
2.  {
3.      ...
4.  }
```

与应用程序中的

C++代码

```
1.  class _declspec(dllimport) class_name //导入类
2.  {
3.      ...
4.  }
```

配对来完成类的导出和导入的！

我们往往通过在类的声明头文件中用一个宏来决定使其编译为class _declspec(dllexport) class_name还是class _declspec(dllimport) class_name版本，这样就不再需要两个头文件。本程序中使用的是：

C++代码

```
1.  #ifdef DLL_FILE
2.  class _declspec(dllexport) class_name //导出类
3.  #else
4.  class _declspec(dllimport) class_name //导入类
5.  #endif
```

实际上，在MFC DLL的讲解中，您将看到比这更简便的方法，而此处仅仅是为了说明_declspec(dllexport)与_declspec(dllimport)配对的问题。

由此可见，应用工程中几乎可以看到DLL中的一切，包括函数、变量以及类，这就是DLL所要提供的强大能力。只要DLL释放这些接口，应用程序使用它就如同使用本工程中的程序一样！

本节虽以VC++为平台讲解非MFC DLL，但是这些普遍的概念在其它语言及开发环境中也是相同的，其思维方式可以直接过渡。

非MFC DLL就讲到这里，下一节将介绍MFC规则DLL。

除非特别注明，**鸡啄米**文章均为原创

转载请标明本文地址：<http://www.jizhuomi.com/software/295.html>

2013年1月27日

作者:鸡啄米 分类:软件开发 浏览:144903 评论:7

相关文章:

[DLL动态链接库编程入门之一：DLL概论及其调试和查看](#) (2013-1-22 20:29:20)

[C++多线程编程入门之经典实例](#) (2013-1-9 21:50:38)

[给程序员五点建议--如何成为编程高手并以此创业](#) (2013-1-6 22:8:14)

[VS2010功能使用体验篇](#) (2013-1-3 20:47:9)

[C++编程开发学习的50条建议](#) (2012-12-29 22:55:43)

[Mysql C语言API编程入门讲解之详细篇](#) (2012-12-21 0:9:11)

[MFC六大核心机制之五、六：消息映射和命令传递](#) (2012-12-11 21:26:24)

[MFC六大核心机制之四：永久保存（串行化）](#) (2012-12-4 21:50:31)

[MFC六大核心机制之三：动态创建](#) (2012-11-30 21:43:21)

[MFC六大核心机制之二：运行时类型识别（RTTI）](#) (2012-11-26 21:5:33)

1楼. 阿军竞价博客

分享

分享

学习程序必须来的。

2013/1/28 15:14:03 [回复该留言](#)

[2楼. proe5.0免费下载](#)

代码对我来说就是不懂的。

2013/1/28 15:50:10 [回复该留言](#)

[3楼. 吴克难的博客](#)

很专业，继续来学习

2013/1/28 20:17:09 [回复该留言](#)

[4楼. 润初颜](#)

依然强大呀！

2013/1/29 9:49:38 [回复该留言](#)

[5楼. 足球比分](#)

读书的时候没学好，现在后悔莫及啊

2013/1/29 10:41:40 [回复该留言](#)

[6楼. 咎宏秋](#)

在学校的时间都荒废了啊！！

鸡啄米 于 2013-02-17 22:42:07 回复
现在努力也完全来得及

2013/2/15 20:31:53 [回复该留言](#)

[7楼. earlyboy](#)

请问：


用“__declspec(dllexport)”这种方法测试成功。
用模块方法：
lib.h中函数声明把__declspec(dllexport)去掉；
把lib.def放在跟dllTest.cpp同一目录下。
工程中也加入了lib.def
但调试时，addFun为0,会提示出错。
请问是怎么回事呀

2014/4/22 17:10:37 [回复该留言](#)

上一篇：[程序员的选择：技术vs管理](#)

下一篇：[RIM破釜沉舟之作：BlackBerry 10](#)

发表评论：

<input type="text"/>	名称(*)
<input type="text"/>	邮箱(选填)
<input type="text"/>	网站链接(选填)
<input type="text"/>	验证(*) 


正文(*) (留言最长字数:1000)

☐ 记住我,下次回复时不用重新输入个人信息

[\[URL\]](#) [\[URL2\]](#) [\[EMAIL\]](#) [\[EMAIL2\]](#) [\[B\]](#) [\[I\]](#) [\[U\]](#) [\[S\]](#) [\[QUOTE\]](#) [显示UBB表情>>](#)

• 欢迎参与讨论，请在这里发表您的看法、交流您的观点。

Copyright © 2011-2020 鸡啄米. 版权所有.

联系邮箱:jizhuomi@126.com Powered By Z-Blog 

[无觅相关文章插件](#)

分享