**DLL静态和动态调用**

**采用lib文件调用DLL（采用Lib文件的调用方式又被称为静态调用）**

**静态调用定义：**

**静态调用，也称为隐式调用，由编译系统完成对DLL的加载和应用程序结束时DLL卸载的编码（Windows系统负责对DLL调用次数的计数），调用方式简单，能够满足通常的要求。通常采用的调用方式是把产生动态连接库时产生的.LIB文件加入到应用程序的工程中，想使用DLL中的函数时，只须在源文件中声明一下。 LIB文件包含了每一个DLL导出函数的符号名和可选择的标识号以及DLL文件名，不含有实际的代码（这种lib文件被称为到入库，如果包含实际的可执行代码，则被称为静态库）。Lib文件包含的信息进入到生成的应用程序中，被调用的DLL文件会在应用程序加载时同时加载在到内存中。**

**Lib文件的分类以及本质（重要）**

**程序编译的时候，链接器需要使用此文件。里面封装了具体的函数实现，变量定义等。而函数接口，变量申明还是通过.h文件给出来。目前以lib后缀的库有两种：一种为静态链接库(StaticLibary，以下简称“静态库”)；另一种为动态连接库(DLL，以下简称“动态库”)的导入库(Import Libary，以下简称“导入库”）：注意另一种lib为DLL到入库。静态库和导入库（注意千万不要把lib文件的导入库，静态库的概念和DLL的动态调用，静态调用搞混淆了）的区别很大，他们实质是不一样的东西。静态库本身就包含了实际执行代码、符号表等等，而对于导入库而言，其实际的执行代码位于动态库中，导入库只包含了地址符号表，各个函数地址等等，确保程序找到对应函数的一些基本地址信息。**

**静态Lib库（可执行文件中间文件）：**

**静态库是一个或者多个obj文件的打包，所以有人干脆把从obj文件生成lib的过程称为Archive，即合并到一起。比如你链接一个静态库，如果其中有错，它会准确的找到是哪个obj有错，即静态lib只是壳子。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。静态编译的lib文件有好处：给用户安装时就不需要再挂动态库了。但也有缺点，就是导致应用程序比较大，而且失去了动态库的灵活性，在版本升级时，同时要发布新的应用程序才行。**

**导入lib库（符号集合）**

**有两个文件，而一个是导入库（.LIB）文件，一个是DLL文件，导入库文件包含被DLL导出的函数的名称和位置，DLL包含实际的函数和数据，应用程序使用LIB文件链接到所需要使用的DLL文件，库中的函数和数据并不复制到可执行文件中，因此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中所要调用的函数的内存地址，这样当一个或多个应用程序运行是再把程序代码和被调用的函数代码链接起来，从而节省了内存资源。从上面的说明可以看出，DLL和.LIB文件必须随应用程序一起发行，否则应用程序将会产生错误。**

**如果既不想用静态Lib库，又不想用导入Lib库，就需要动态调用DLL了：需要自己LoadLibary调入DLL文件，然后再手工GetProcAddress获得对应函数了（动态调用DLL），前提是需要对DLL内部函数申明熟悉。**

**静态调用其步骤如下：**

**1.把你的youApp.DLL拷到你目标工程(需调用youApp.DLL的工程)的Debug目录下;**

**2.把你的youApp.lib拷到你目标工程(需调用youApp.DLL的工程)目录下;**

**3.把你的youApp.h(包含输出函数的定义)拷到你目标工程(需调用youApp.DLL的工程)目录下;**

**4.打开你的目标工程选中工程,选择Visual C++的Project主菜单的Settings菜单;**

**5.执行第4步后，VC将会弹出一个对话框，在对话框的多页显示控件中选择Link页。然后在Object/library**

**modules输入框中输入:youApp.lib**

**6.选择你的目标工程Head Files加入：youApp.h文件;**

**7.最后在你目标工程(\*.cpp,需要调用DLL中的函数)中包含你的:#include**

**"youApp.h"**

**在Delphi中，无法使用VCC的DLL对应的lib文件，所以，需要自己在pas文件中采用pascal语法申明该API函数，如下所示：**

**FunctionAddNum(Num1,Num2:integer):integer;stdcall;external 'AddNum.dll'**

**name 'AddNumber';**

**在.Net中，静态使用DLL中，跟Delphi比较类似，如下图代码所示：**

**[DllImport(@".\ihrAPI\EmulinAPI.dll", CharSet =**

**CharSet.Ansi, CallingConvention =**

**CallingConvention.Cdecl)]**

**public static**

**extern int emulin\_config();**

**使用LoadLibrary()使用DLL（采用LoadLibrary又被称为动态调用）**

**动态调用定义：**

**即显式调用方式，是由编程者用API函数加载和卸载DLL来达到调用DLL的目的，比较复杂，但能更加有效地使用内存，是编制大型应用程序时的重要方式，实际需要的时候才会载入DLL，用完过后，释放此DLL。在Windows系统中，与动态库调用有关的函数包括：**

**①LoadLibrary（或MFC 的AfxLoadLibrary），装载动态库。**

**②GetProcAddress，获取要引入的函数，将符号名或标识号转换为DLL内部地址。**

**③FreeLibrary（或MFC的AfxFreeLibrary），释放动态链接库。**

**方法如下:**

**#include**

**"stdafx.h"**

**#include**

**"afx.h "**

**#include**

**<iostream> //注意，不能加.h**

**#include <stdio.h>**

**typedef**

**int (\*SQLITE\_OPENTYPE)(**

**const**

**char \*filename, /\* Database filename(UTF-8) \*/**

**sqlite3 \*\*ppDb**

**/\* OUT: SQLite db handle \*/**

**); //可知，必须知道DLL中内部函数的类型，因此也就不需要额外的lib文件了**

**SQLITE\_OPENTYPE Func\_Open;**

**/\*Sample: use loadlibrary\*/**

**//HINSTANCE hInstLibrary =LoadLibrary("sqlite3.dll");**

**HINSTANCE hInstLibrary =**

**LoadLibrary(TEXT("./sqlite3.dll"));**

**if (hInstLibrary ==**

**NULL)**

**{**

**std::cout<<**

**"载入dll失败" << std::endl;**

**FreeLibrary(hInstLibrary);**

**system("pause");**

**return 1;**

**}**

**Func\_Open= (SQLITE\_OPENTYPE)GetProcAddress(hInstLibrary,"sqlite3\_open");**

**if (Func\_Open ==**

**NULL)//运行后程序进入这里，显示“转换失败”后结束**

**{**

**std::cout<<**

**"转换失败" << std::endl;**

**FreeLibrary(hInstLibrary);**

**system("pause");**

**return 1;**

**}**

**//以下程序没有运行**

**Func\_Open("test1.db", &db);**

**std::cout<< std::endl;**

**FreeLibrary(hInstLibrary);**

**通过上述两个例程，已经完成理解了C++中DLL的编译和加载。**

**---------------------**

**作者：ZY\_KFB**

**来源：CSDN**

**原文：https://blog.csdn.net/sya\_inn/article/details/53981440**

**版权声明：本文为博主原创文章，转载请附上博文链接！**

**1、 DLL的概念**  
DLL(Dynamic Linkable Library)，动态链接库，可以向程序提供一些函数、变量或类。这些可以直接拿来使用。  
静态链接库与动态链接库的区别：  
（1）静态链接库与动态链接库都是共享代码的方式。静态链接库把最后的指令都包含在最终生成的EXE文件中了；动态链接库不必被包含在最终EXE文件中，EXE文件执行时可以“动态”地引用和卸载这个与EXE独立的DLL文件。  
（2） 静态链接库中不能再包含其他的动态链接库或者静态库，而在动态链接库中还可以再包含其他的动态或静态链接库。  
动态链接库的分类：Visual C++支持三种DLL，它们分别是Non-MFC DLL（非MFC动态库）、MFC Regular DLL（MFC规则DLL）、MFC Extension DLL（MFC扩展DLL）。非MFC动态库不采用MFC类库结构，其导出函数为标准的C接口，能被非MFC或MFC编写的应用程序所调用；MFC规则DLL 包含一个继承自CWinApp的类，但其无消息循环；MFC扩展DLL采用MFC的动态链接版本创建，它只能被用MFC类库所编写的应用程序所调用。  
**2、 创建一个DLL**  
**2.1 非MFC的DLL**  
2.1.1声明导出函数：  
extern “C” \_\_declspec(dllexport) int add(int a, int b);  
其中 extern “C”为声明为C编译。由于C++编译器在编译的时候会造成其函数名的该变，在其他应用程序中导致函数不可调用，而C编译器则不会在编译后改变其函数名。这样如果用C编译的程序来调用该dll中的函数时，可能会造成找不到该函数。  
\_\_declspec（dllexport）表示该函数为DLL输出函数，即其他应用程序可以调用该函数  
从dll中声明输出函数有两种方式：  
（1） 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。  
（2） 用\_\_declspec(dllexport)来声明函数  
如果使用Visual C++来创建dll，对于同样用VC创建的exe来说，调用dll没有什么问题。而如果用其他工具来创建的exe来调用dll，就会出现问题。因为即使你不用C++编译器，Microsoft C编译器也会损害C函数。当用\_\_stdcall将函数输出时，C编译器会将函数改为[\_func@1](mailto:_func@1" \t "_blank)的形式。在这里需要在.def文件中加入EXPORTS节来输出函数：  
EXPORTS  
func  
这样，dll将用func函数名来输出函数。  
另一种方式是用#pragma (linker, “/exports:func=\_func@1”),告诉编译器输出函数func，这种方式没有前一种好。  
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](mailto:_functionname@number" \t "_blank)；而\_\_cdecl调用约定仅在输出函数名前面加下划线，形如\_functionname。  
; lib.def : 导出DLL函数

LIBRARY dllTest

EXPORTS

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。

**2.1.2 Dll的调用方式：**   
DLL的调用分为两种方式：动态和静态  
（1） 动态调用：typedef int(\*lpAddFun)(int, int); //宏定义函数指针类型  
lpAddFun add;//函数指针  
HINSTANCE hDll=LoadLibrary(“path”);  
add=(lpAddFun)GetProcAddress(hDll, "add");//获得dll中的add函数指针  
FreeLibrary(hDll);  
在从dll调用中返回的函数、指针或者类都是以指针的方式会的，即返回的是函数、变量或类的地址。这里一定要注意，不能简单的用函数名来赋值。  
（2） 静态调用：将生成的.dll和.lib文件拷入到调用dll的工程中，用命令  
#pragma comment(lib,"dllTest.lib")，实现静态调用，即把该dll在编译的时候也编译到exe文件中去，而后在工程中调用时用下面的代码：  
#pragma comment(lib,"dllTest.lib")

//.lib文件中仅仅是关于其对应DLL文件中函数的重定位信息

extern "C" \_\_declspec(dllimport) add(int x,int y);

int main(int argc, char\* argv[])

{

int result = add(2,3);

printf("%d",result);

return 0;

}

　　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。  
另外一种显式调用的方式是设置vc中的目录和includefiles来实现  
　　(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。  
**2.1.3 DllMain函数**  
Windows在加载dll的时候，会首先需要一个入口函数DllMain。当在dll中不定义DllMain的时候，windows会从其他运行库中调用一个不做任何操作的DllMain函数，直接返回true。DllMain是dll内部的函数，这意味着在调用dll的程序中不能显式的调用。它是在 dll被调用时自动被调用的。  
BOOL APIENTRY DllMain( HANDLE hModule, DWORD ul\_reason\_for\_call,  
LPVOID lpReserved)  
{  
switch (ul\_reason\_for\_call)  
{  
case: DLL\_PROCESS\_ATTACH:  
break;  
case: DLL\_THREAD\_ATTACH:  
break;  
case: DLL\_THREAD\_DETACH:  
break;  
case: DLL\_PROCESS\_DETACH:  
break;  
return TRUE;

}   
}

**2.2 在dll中导出变量**  
1、在dll中定义变量 extern int global;  
2、在.def中定义输出 EXPORTS:  
global DATA  
3、 在应用程序中调用：#pragma comment(lib,"dllTest.lib")  
extern int global;  
注意在此引入的变量global，是一个地址，在使用时需要强制转化为指针后再用，才能得到其值。  
（int \*）global=10;  
在应用工程中引用DLL中全局变量的一个更好方法是:  
extern int \_declspec(dllimport) global; //用\_declspec(dllimport)导入  
通过\_declspec(dllimport)方式导入的就是DLL中全局变量本身而不再是其地址了,建议在一切可能的情况下使用这种方式。  
**2.3 dll导出类**  
在定义的时候用 class \_declspec(dllexport) classname{  
}  
在类中引用的时候用  
加入类定义头文件：#include “classname.h”  
Class \_declspec(dllimport) classname 来导入类  
**3、 MFC规则Dll**  
MFC规则DLL的概念体现在两方面：  
　　（1） 它是MFC的  
　　“是MFC的”意味着可以在这种DLL的内部使用MFC；  
　　（2） 它是规则的  
　　“是规则的”意味着它不同于MFC扩展DLL，在MFC规则DLL的内部虽然可以使用MFC，但是其与应用程序的接口不能是MFC。而MFC扩展DLL与应用程序的接口可以是MFC，可以从MFC扩展DLL中导出一个MFC类的派生类。  
　　Regular DLL能够被所有支持DLL技术的语言所编写的应用程序调用，当然也包括使用MFC的应用程序。在这种动态连接库中，包含一个从CWinApp继承下来的类，DllMain函数则由MFC自动提供。  
（1）静态链接到MFC 的规则DLL  
　　静态链接到MFC的规则DLL与MFC库（包括MFC扩展 DLL）静态链接，将MFC库的代码直接生成在.dll文件中。在调用这种DLL的接口时，MFC使用DLL的资源。因此，在静态链接到MFC 的规则DLL中不需要进行模块状态的切换。  
　　使用这种方法生成的规则DLL其程序较大，也可能包含重复的代码。  
（2）动态链接到MFC 的规则DLL  
　　动态链接到MFC 的规则DLL 可以和使用它的可执行文件同时动态链接到 MFC DLL 和任何MFC扩展 DLL。在使用了MFC共享库的时候，默认情况下，MFC使用主应用程序的资源句柄来加载资源模板。这样，当DLL和应用程序中存在相同ID的资源时（即所谓的资源重复问题），系统可能不能获得正确的资源。因此，对于共享MFC DLL的规则DLL，我们必须进行模块切换以使得MFC能够找到正确的资源模板。  
我们可以在Visual C++中设置MFC规则DLL是静态链接到MFC DLL还是动态链接到MFC DLL。如图8，依次选择Visual C++的project -> Settings -> General菜单或选项，在Microsoft Foundation Classes中进行设置。

**3.1规则DLL的创建;**  
与非MFCdll不同的是，在其定义里面可以引入MFC类，其他与非MFC一样  
**3.2规则DLL的调用**  
（1）显示方式LoadLibrary , GetProcAdress , FreeLibrary  
(2) 我们照样可以在EXE程序中隐式调用MFC规则DLL，只需要将DLL工程生成的.lib文件和.dll文件拷入当前工程所在的目录，并在RegularDllCallDlg.cpp文件（图12所示对话框类的实现文件）的顶部添加：  
#pragma comment(lib,"RegularDll.lib")  
**3.3共享MFC DLL的规则DLL的模块切换**  
应用程序进程本身及其调用的每个DLL模块都具有一个全局唯一的HINSTANCE句柄，它们代表了DLL或EXE模块在进程虚拟空间中的起始地址。进程本身的模块句柄一般为0x400000，而DLL模块的缺省句柄为0x10000000。如果程序同时加载了多个DLL，则每个DLL模块都会有不同的 HINSTANCE。应用程序在加载DLL时对其进行了重定位。  
　　共享MFC DLL（或MFC扩展DLL）的规则DLL涉及到HINSTANCE句柄问题，HINSTANCE句柄对于加载资源特别重要。EXE和DLL都有其自己的资源，而且这些资源的ID可能重复，应用程序需要通过资源模块的切换来找到正确的资源。如果应用程序需要来自于DLL的资源，就应将资源模块句柄指定为 DLL的模块句柄；如果需要EXE文件中包含的资源，就应将资源模块句柄指定为EXE的模块句柄。  
模块的切换有三种方式：  
（1）在DLL函数中调用：AFX\_MANAGE\_STATE(AfxGetStaticModuleState());（推荐使用，最简单）

void ShowDlg(void)

{

//方法1:在函数开始处变更，在函数结束时恢复

//将AFX\_MANAGE\_STATE(AfxGetStaticModuleState());作为接口函数的第一//条语句进行模块状态切换

AFX\_MANAGE\_STATE(AfxGetStaticModuleState());

CDialog dlg(IDD\_DLL\_DIALOG);//打开ID为2000的对话框

dlg.DoModal();

}

（2）在DLL函数中调用AfxGetResourceHandle();

AfxSetResourceHandle(HINSTANCE xxx);  
（3）由应用程序自身切换（不推荐，最麻烦）  
**4、扩展MFCDLL**  
MFC扩展DLL的内涵为MFC的扩展，用户使用MFC扩展DLL就像使用MFC本身的DLL一样。除了可以在MFC扩展DLL的内部使用MFC以外， MFC扩展DLL与应用程序的接口部分也可以是MFC。我们一般使用MFC扩展DLL来包含一些MFC的增强功能，譬如扩展MFC的CStatic、 CButton等类使之具备更强大的能力。  
导出一个类，直接在类声明头文件中使用AFX\_EXT\_CLASS即可，最后别忘了在调用dll的程序中加入class的头文件

**5、总结：**

综上所述:以上几种dll主要由以下几种区别：

1、动态链接库是将exe程序在程序执行的时候动态加载的，而静态链接库是在编译的时 将其编译在代码之中的

2、动态链接库可以输出变量、函数和类。其中每种输出的方式与调用方式不尽相同：

（1）变量：在dll中定义 extern int global;

在.def文件中输出 EXPORTS

global DATA

或extern \_declspec(dllexport)int global(不用输出文件了)

在程序中调用：

静态调用： #pragma comment(lib,"dllTest.lib")

extern int \_declspec(dllimport) global;

动态调用:

（2）函数:在dll中定义

extern “C” \_\_declspec(dllexport) int add(int a, int b);

也可以在.def文件中输出该函数

EXPORTS

[add @ 1](mailto:add@1)

在程序中调用：

静态调用：

#pragma comment(lib,"dllTest.lib")

extern "C" \_\_declspec(dllimport) add(int x,int y);

动态调用：

typedef int(\*lpAddFun)(int, int); //宏定义函数指针类型  
lpAddFun add;//函数指针  
HINSTANCE hDll=LoadLibrary(“path”);  
add=(lpAddFun)GetProcAddress(hDll, "add");//获得dll中的add函数指针  
FreeLibrary(hDll);  
在从dll调用中返回的函数、指针或者类都是以指针的方式会的，即返回的是函数、变量或类的地址。这里一定要注意，不能简单的用函数名来赋值。

（3）类：在dll中定义：  
在定义的时候用 class \_declspec(dllexport) classname{  
}  
在类中引用的时候用  
加入类定义头文件：#include “classname.h”  
Class \_declspec(dllimport) classname 来导入类  
3、除了扩展MFC的dll外，其他的dll均可被其他语言编写的应用程序来调用。

转自：[http://blog.sina.com.cn/s/blog\_597fcab10100ck4a.html](http://blog.sina.com.cn/s/blog_597fcab10100ck4a.html" \t "_blank)

【补充】静态调用和动态调用区别：

    总的来说，采用动态库技术对于升级软件版本更加容易。与静态库（Static Link Library）不同，动态库里面的函数不是执行程序本身的一部分，而是根据执行需要按需载入，其执行代码可以同时在多个程序中共享。  
(1)静态调用，也称为隐式调用，由编译系统完成对DLL的加载和应用程序结束时DLL卸载的编码（Windows系统负责对DLL调用次数的计数），调用方式简单，能够满足通常的要求。通常采用的调用方式是把产生动态连接库时产生的.LIB文件加入到应用程序的工程中，想使用DLL中的函数时，只须在源文件中声明一下。   
LIB文件包含了每一个DLL导出函数的符号名和可选择的标识号以及DLL文件名，不含有实际的代码。Lib文件包含的信息进入到生成的应用程序中，**被调用的DLL文件会在应用程序加载时同时加载在到内存中**。   
(2)动态调用，即显式调用方式，是由编程者用API函数加载和卸载DLL来达到调用DLL的目的，比较复杂，但能更加有效地使用内存，是编制大型应用程序时的重要方式。在Windows系统中，与动态库调用有关的函数包括：   
①LoadLibrary（或MFC 的AfxLoadLibrary），装载动态库。   
②GetProcAddress，获取要引入的函数，将符号名或标识号转换为DLL内部地址。   
③FreeLibrary（或MFC的AfxFreeLibrary），释放动态链接库。