动态链接库中的函数导出成功后，其它EXE程序要使用导出函数，就会涉及到如何调用动态链接库。

调用DLL也分为两种方法：隐式调用和显示调用：

1、隐式调用：

隐式链接的特点是由编译器完成对DLL的加载和程序结束时对DLL的卸载工作，如果程序结束时如果还有其他应用程序使用该DLL，那么系统会使DLL的使用计数减1，当DLL的使用计数降为0时，会将DLL从内存中删除。

★优缺点：隐式链接DLL的方法简单实用，但缺少灵活性。

■使用方法：使用隐式链接DLL开发时，首先，需要将DLL的引入库文件(\*.lib--编译生成DLL时，会一起生成的)与应用程序进行静态链接，因为引入库文件包含DLL的各种输出资源，如导出函数，导出类等信息，这些信息指向DLL的函数指针等等，EXE执行时，DLL被 “自动” 加载，EXE退出时DLL被 “自动” 卸载。

对上节所讲的Add()函数进行隐式调用：

#pragma comment(lib,"D:\\DllTest\\Debug\\DllTest.lib")\_declspec(dllimport) int Add(int a,int b);

2、显示调用：

显示链接方式是完全由编程者用API 加载和卸载DLL，编程者可以决定何时加载DLL，加载哪个DLL，何时卸载DLL，卸载哪个DLL等。

★优缺点：显示链接方式充分体现了DLL的灵活性，是比较常用的调用DLL方式。但是与静态链接相比稍微复杂了些。

■使用方法：

LoadLibrary(...)：该 API 用于加载指定的DLL；

GetProcAddress(...)：该 API 用于获取DLL中导出函数的指针， 即导出函数的入口点；

FreeLibrary(...)：该 API 用于卸载指定的DLL。

〓注：如果程序中多次调用LoadLibrary(...)加载同一DLL时，在卸载的时候也要调用相应次数的FreeLibrary(...)进行卸载。

对上节所讲的Dec()函数进行显示调用：

typedef int (\*MYPROC)(int a,int b); HMODULE hMod = LoadLibrary("D:\\DllTest\\Debug\\DllTest.dll");//加载指定DLL模块 if (hMod) { MYPROC Func = (MYPROC)GetProcAddress(hMod,"Dec");//获取DLL中导出函数的指针，即导出函数的入口点 if (Func) { nRet=Func(10,6); printf("10-6=%d\n",nRet); } FreeLibrary(hMod);//卸载指定的DLL }

▲使用LoadLibrary显式链接，在这个函数的参数中可以指定DLL文件的完整路径。如果不指定路径，Windows将遵循如下的搜索顺序来定位DLL：

①EXE文件所在的目录：

②进程的当前工作目录：

③Windows系统目录：例如：C:\WINDOWS\system32

④Windows目录：例如：C:\WINDOWS

⑤环境变量的目录：我的电脑-->属性-->高级-->环境变量

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

函数指针：

对于一个函数：int Add(int a ,int b);

定义一个函数指针： int (\*pFunc)(int a ,int b); pFunc为函数指针

定义一个函数指针类型：typedef int (\*pFunc)(int a,int b); pFunc为函数指针类型

pFunc p; p为函数指针