lib与dll的区别

lib是静态库，dll一般是动态链接库（也有可能是别的）  
  
比如要编译个exe，lib在编译的时候就会被编译到exe里，作为程序的一部分  
  
而dll是不被编译进去，是运行的时候才调入的（可能是exe刚运行就调入，也可能运行了一半才调入）  
  
用法，lib需要个.lib文件和一个.h文件，程序正常使用.h的函数，在链接选项里加入.lib文件就ok  
  
**dll用法有2种**，一是 .h + .lib + .dll的，用法和前面一样，中间的lib是个中转，运行的时候会调用dll  
二是：直接用dll，需要知道dll的函数定义，用LoadLibrary和GetProcAddress把函数指针取出来，看msdn的例子吧

|  |
| --- |
|  |
|  | |  | | --- | | 什么是lib文件，lib和dll的关系如何 (2008-04-18 19:44:37)     (1)lib是编译时需要的，dll是运行时需要的。  如果要完成源代码的编译，有lib就够了。  如果也使动态连接的程序运行起来，有dll就够了。  在开发和调试阶段，当然最好都有。  (2)一般的动态库程序有lib文件和dll文件。lib文件是必须在编译期就连接到应用程序中的，而dll文件是运行期才会被调用的。如果有 dll文件，那么对应的lib文件一般是一些索引信息，具体的实现在dll文件中。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。  (3)在动态库的情况下，有两个文件，一个是引入库（.LIB）文件，一个是DLL文件，引入库文件包含被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。  一、开发和使用dll需注意三种文件        1、   dll头文件        它是指dll中说明输出的类或符号原型或数据结构的.h文件。当其它应用程序调用dll时，需要将该文件包含入应用程序的源文件中。        2、   dll的引入库文件        它是dll在编译、链接成功后生成的文件。主要作用是当其它应用程序调用dll时，需要将该文件引入应用程序。否则，dll无法引入。        3、   dll文件(.dll)        它是应用程序调用dll运行时，真正的可执行文件。dll应用在编译、链接成功后，.dll文件即存在。开发成功后的应用程序在发布时，只需要有.exe文件和.dll文件，不必有.lib文件和dll头文件。     动态链接库 (DLL) 是作为共享函数库的可执行文件。动态链接提供了一种方法，使进程可以调用不属于其可执行代码的函数。函数的可执行代码位 于一个 DLL 中，该 DLL 包含一个或多个已被编译、链接并与使用它们的进程分开存储的函数。DLL 还有助于共享数据和资源。多个应用程序可同时 访问内存中单个 DLL 副本的内容。   动态链接与静态链接的不同之处在于：动态链接允许可执行模块（.dll 文件或 .exe 文件）仅包含在运行时定位 DLL 函数的可执行代码所需的信息。在静态链接中，链接器从静态链接库获取所有被引用的函数，并将库同代码一起放到可执行文件中。   使用动态链接代替静态链接有若干优点。DLL 节省内存，减少交换操作，节省磁盘空间，更易于升级，提供售后支持，提供扩展 MFC 库类的机制，支持多语言程序，并使国际版本的创建轻松完成。   lib与dll文件最大区别在调用方面   dll可以静态陷入  lib与DLL  从这一章起，我讲述的内容将特定于windows平台。其实这篇文章也可看作是我在windows下的开发经验总结，因为以后我决定转unix了。  前面有一章说编译与链接的，说得很简略，其实应该放到这一章一块儿来说的。许多单讲C++的书其实都过于学院派，对于真实的工作环境，上百个源文件怎么结合起来，几乎没有提及。我引导读者一步步看看lib与DLL是怎么回事。  一个最简单的C++程序，只需要一个源文件，这个源文件包含了如下语句  int main(){return 0;}  自然，这个程序什么也不做。  当需程序需要做事情时，我们会把越来越多的语句添加到源文件中，例如，我们会开始在main函数中添加代码：  #include <stdio.h>  int main()  {  printf("Hello World!\n");  return 0;  }  由于人的智力水平的限制，当一个函数中包含了太多的语句时，便不太容易被理解，这时候开始需要子函数：  #include <stdio.h>  void ShowHello()  { | |
|  |  |
|  |  |

|  |  |  |
| --- | --- | --- |
|  | |  | | --- | | printf("Hello World!\n");  }  int main()  {  ShowHello();  return 0;  }  同样的道理，一个源文件中包含了太多的函数，同样不好理解，人们开始分多个源文件了  // main.cpp  void ShowHello();//[1]  int main()  {  ShowHello();  return 0;  }  // hello.cpp  #include <stdio.h>  void ShowHello()  {  printf("Hello World!\n");  }  将这两个文件加入到一个VC工程中，它们会被分别编译，最后链接在一起。在VC编译器的输出窗口，你可以看到如下信息  --------------------Configuration: hello - Win32 Debug--------------------  Compiling...  main.cpp  hello.cpp  Linking...   hello.exe - 0 error(s), 0 warning(s)  这展示了它们的编译链接过程。  接下来，大家就算不知道也该猜到，当一个工程中有太多的源文件时，它也不好理解，于是，人们想到了一种手段：将一部分源文件预先编译成库文件，也即lib文件，当要使用其中的函数时，只需要链接lib文件就可以了，而不用再理会最初的源文件。  在VC中新建一个static library类型的工程，加入hello.cpp文件，然后编译，就生成了lib文件，假设文件名为hello.lib。  别的工程要使用这个lib有两种方式：  1 在工程选项-〉link-〉Object/Library Module中加入hello.lib  2 可以在源代码中加入一行指令  #pragma comment(lib, "hello.lib")  注意这个不是C++语言的一部分，而是编译器的预处理指令，用于通知编译器需要链接hello.lib  根据个人爱好任意使用一种方式既可。  这种lib文件的格式可以简单的介绍一下，它实际上是任意个obj文件的集合。obj文件则是cpp文件编译生成的，在本例中，lib文件只包含 了一个obj文件，如果有多个cpp文件则会编译生成多个obj文件，从而生成的lib文件中也包含了多个obj，注意，这里仅仅是集合而已，不涉及到 link，所以，在编译这种静态库工程时，你根本不会遇到链接错误。即使有错，错误也只会在使用这个lib的EXE或者DLL工程中暴露出来。  关于静态lib，就只有这么多内容了，真的很简单，现在我们介绍另外一种类型的lib，它不是obj文件的集合，即里面不含有实际的实现，它只 是提供动态链接到DLL所需要的信息。这种lib可以在编译一个DLL工程时由编译器生成。涉及到DLL，问题开始复杂起来，我不指望在本文中能把DLL 的原理说清楚，这不是本文的目标，我介绍操作层面的东西。  简单的说，一个DLL工程和一个EXE工程的差别有两点：  1 EXE的入口函数是main或者WinMain，而DLL的入口函数是DllMain  2 EXE的入口函数标志着一段处理流程的开始，函数退出后，流程处理就结束了，而DLL的入口函数对系统来说，只是路过，加载DLL的时候路过 一次，卸载DLL的时候又路过一次[2]，你可以在DLL入口函数中做流程处理，但这通常不是DLL的目的，DLL的目的是要导出函数供其它DLL或 EXE使用。你可以把DLL和EXE的关系理解成前面的main.cpp和hello.cpp的关系，有类似，实现手段不同罢了。  先看如何写一个DLL以及如何导出函数，读者应该先尝试用VC创建一个新的动态链接库工程，创建时选项不选空工程就可以了，这样你能得到一个示例，以便开始在这个例子基础上工作。  看看你创建的例子中的头文件有类似这样的语句：  #ifdef DLL\_EXPORTS  #define DLL\_API \_\_declspec(dllexport)  #else  #define DLL\_API \_\_declspec(dllimport)  #endif  这就是函数的导出与使用导出函数的全部奥妙了。你的DLL工程已经在工程设置中定义了一个宏DLL\_EXPORTS，因此你的函数声明只要前面加 DLL\_API就表示把它导出，而DLL的使用者由于没有定义这个宏，所以它包含这个头文件时把你的函数看作导入的。通过模仿这个例子，你就可以写一系列 的标记为导出的函数了。  导出函数还有另一种方法，是使用DEF文件，DEF文件的作用，在现在来说只是起到限定导出函数名字的作用，这里，我们要引出第二种[4]使用 DLL的方法：称为显示加载，通过Windows API的LoadLibrary和GetProcAddress这两个函数来实现[5]，这里 GetProcAddress的参数需要一个字符串形式的函数名称，如果DLL工程中没有使用DEF文件，那么很可能你要使用非常奇怪的函数名称（形 如：?fnDll@@YAHXZ）才能正确调用，这是因为C++中的函数重载机制把函数名字重新编码了，如果使用DEF文件，你可以显式指定没编码前的函 数名。 | |
|  |  |
|  |  |

|  |  |  |
| --- | --- | --- |
|  | |  | | --- | | 有了这些知识，你可以开始写一些简单的DLL的应用，但是我可以百分之百的肯定，你会遇到崩溃，而之前的非DLL的版本则没有问题。假如你通过显 式加载来使用DLL，有可能会是调用约定不一致而引起崩溃，所谓调用约定就是函数声明前面加上\_\_stdcall \_\_cdecl等等限定词，注意一些宏 如WINAPI会定义成这些限定词之一，不理解他们没关系，但是记住一定要保持一致，即声明和定义时一致，这在用隐式加载时不成问题，但是显示加载由于没 有利用头文件，就有可能产生不一致。  调用约定并不是我真正要说的，虽然它是一种可能。我要说的是内存分配与释放的问题。请看下面代码：  void foo(string& str)  {  str = "hello";  }  int main()  {  string str;  foo(str);  printf("%s\n", str.c\_str());  return 0;  }  当函数foo和main在同一个工程中，或者foo在静态库中时，不会有问题，但是如果foo是一个DLL的导出函数时，请不要这么写，它有可能 会导致崩溃[6]。崩溃的原因在于“一个模块中分配的内存在另一个模块中释放”，DLL与EXE分属两个模块，例子中foo里面赋值操作导致了内存分配， 而main中return语句之后，string对象析构引起内存释放。  我不想穷举全部的这类情况，只请大家在设计DLL接口时考虑清楚内存的分配释放问题，请遵循谁分配，谁释放的原则来进行。  如果不知道该怎么设计，请抄袭我们常见的DLL接口--微软的API的做法，如：  CreateDC  ReleaseDC  的成对调用，一个函数分配了内存，另外一个函数用来释放内存。  回到我们有可能崩溃的例子中来，怎么修改才能避免呢？  这可以做为一个练习让读者来做，这个练习用的时间也许会比较长，如果你做好了，那么你差不多就出师了。一时想不到也不用急，我至少见过两个有五年以上经验的程序员依然犯这样的错误。   注[1]：为了说明的需要，我这里使用直接声明的方式，实际工程中是应该使用头文件的。  注[2]: 还有线程创建与销毁也会路过DLL的入口，但是这对新手来说意义不大。  注[3]：DEF文件格式很简单，关于DEF文件的例子，可以通过新建一个ATL COM工程看到。  注[4]：第一种方法和使用静态库差不多，包含头文件，链接库文件，然后就像是使用普通函数一样，称为隐式加载。  注[5]：具体调用方法请参阅MSDN。  注[6]：之所以说有可能是因为，如果两个工程的设置都是采用动态连接到运行库，那么分配释放其实都在运行库的DLL中进行，那么这种情况便不会发生崩溃 | |