有的程序为何只有dll没有lib

首先□dll的调用不一定需要*.lib这个引入库。

dll调用分为两种方式，一是隐式链接，二是显式链接。隐式链接就是使用*.lib的，这就需要在编译的时候有这个lib文件。而显式链接，就是使用LoadLibrary这个API函数来实现动态加载，因此，不需要lib这样的引入库。

再者，就算是使用隐式链接□lib只是在编译的时候确定DLL接口，使得调用者可以很便捷地使用dll文件，但在发布的时候，是不需要连同lib一起发布。

如果你要发布一个dll形式的SDK□就很有必要将头文件□dll文件连同引入库一起发布，以方便使用者调用。

动态链接库dll,导入库lib,静态链接库lib

https://www.cnblogs.com/tswcypy/p/4554041.html

目前以lib后缀的库有两种，一种为静态链接库(Static Libary□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 ”）。 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。 动态库一般会有对应的导入库，方便程序静态载入动态链接库，否则你可能就需要自己LoadLibary调入DLL文件，然后再手工GetProcAddress获得对应函数了。有了导入库，你只需要链接导入库后按照头文件函数接口的声明调用函数就可以了。 导入库和静态库的区别很大，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等，确保程序找到对应函数的一些基本地址信息。

这也是实际上很多开源代码发布的惯用方式：

1.预编译的开发包：包含一些.dll文件和一些.lib文件。其中这里的.lib就是导入库，而不要错以为是静态库。但是引入方式和静态库一样，要在链接路 径上添加找到这些.lib的路径。而.dll则最好放到最后产生的应用程序exe执行文件相同的目录。这样运行时，就会自动调入动态链接库。

2. 用户自己编译： 下载的是源代码，按照readme自己编译。生成很可能也是.dll + .lib(导入库)的库文件

3. 如果你只有dll□并且你知道dll中函数的函数原型，那么你可以直接在自己程序中使用LoadLibary调入DLL文件□GetProcAddress

DLL□ 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它允许可执行模块□.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。

API 就是应用程序编程接口。它是能用来操作组件、应用程序或者操作系统的一组函数。典型的情况下，API 由一个或多个提供某种特殊功能的 DLL 组成。 DLL 是一个文件，其中包含了在 Microsoft? Windows? 下运行的任何应用程序都可调用的函数。运行时，DLL 中的函数动态地链接到调用它的应用程序中。无论有多少应用程序调用 DLL 中的某个函数，在磁盘上只有一个文件包含该函数，且只在它调入内存时才创建该 DLL。 您听到最多的 API 可能是 Windows API。它包括构成 Windows 操作系统的各种 DLL。每个 Windows 应用程序都直接或间接地与 Windows API 互动。Windows API 保证 Windows 下运行的所有应用程序的行为方式一致。 注意 随着 Windows 操作系统的发展，现已发布了几个版本的 Windows API。Windows 3.1 使用 Win16 API。Microsoft? Windows NT?、Windows 95 和 Windows 98 平台使用 Microsoft? Win32? API。 除 Windows API 外，其他一些 API 也已发布。例如，邮件应用程序编程接口 (MAPI) 是一组可用于编写电子邮件应用程序的 DLL。 API 传统上是为开发 Windows 应用程序的 C 和 C++ 程序员编写的，但其他的编程语言（包括VBA）也可以调用 DLL 中的函数。因为大部分 DLL 主要是为 C 和 C++ 程序员编写和整理说明的，所以调用 DLL 函数的方法与调用 VBA 函数会有所不同。在使用 API 时必须了解如何给 DLL 函数传递参数。 警告 调用 Windows API 和 其他 DLL 函数可能会给您的应用程序带来不良影响。从自己的代码中直接调用 DLL 函数时，您绕过了 VBA 通常提供的一些安全机制。如果在定义或调用 DLL 函数时出现错误（所有程序员都不可避免），可能会在应用程序中引起应用程序错误（也称为通用性保护错误，或 GPF）。最好的解决办法是在运行代码以前保存该项目，并确保了解 DLL 函数调用的原理。

LIB 创建标准库、导入库和导出文件，在生成 32 位程序时可将它们与 LINK 一起使用。LIB 从命令提示运行。可在下列几种模式下使用 LIB。 生成或修改 COFF 库 将成员对象提取到文件中 创建导出文件和导入库 这些模式是互斥的；每次只能以一种模式使用 LIB。

1 静态链接库的优点

(1) 代码装载速度快，执行速度略比动态链接库快；

(2) 只需保证在开发者的计算机中有正确的.LIB文件，在以二进制形式发布程序时不需考虑在用户的计算机上.LIB文件是否存在及版本问题，可避免DLL地狱等问题。

2 动态链接库的优点

(1) 更加节省内存并减少页面交换；

(2) DLL文件与EXE文件独立，只要输出接口不变（即名称、参数、返回值类型和调用约定不变），更换DLL文件不会对EXE文件造成任何影响，因而极大地提高了可维护性和可扩展性；

(3) 不同编程语言编写的程序只要按照函数调用约定就可以调用同一个DLL函数；

(4)适用于大规模的软件开发，使开发过程独立、耦合度小，便于不同开发者和开发组织之间进行开发和测试。

3 不足之处

(1) 使用静态链接生成的可执行文件体积较大，包含相同的公共代码，造成浪费；

(2) 使用动态链接库的应用程序不是自完备的，它依赖的DLL模块也要存在，如果使用载入时动态链接，程序启动时发现DLL不存在，系统将终止程序并给出错误信息。而使用运行时动态链接，系统不会终止，但由于DLL中的导出函数不可用，程序会加载失败；速度比静态链接慢。当某个模块更新后，如果新模块与旧的模块不兼容，那么那些需要该模块才能运行的软件，统统撕掉。这在早期Windows中很常见。

Linux下的动态链接库和静态链接库

动态库和静态库是做什么用的呢？链接的时候，使用的到底是动态库，还是静态库呢？

Linux中有两类函数库，分别是静态库和动态库。

静态函数库：

这类库的名字一般是libxxx.a□利用静态函数库编译成的文件比较大，因为整个函数库的所有数据都会被整合进目标代码中，他的优点就显而易见了，即编译后的执行程序不需要外部的函数库支持，因为所有使用的函数都已经被编译进去了。当然这也会成为他的缺点，因为如果静态函数库改变了，那么你的程序必须重新编译。

动态函数库：

这类库的名字一般是libxxx.so;相对于静态函数库，动态函数库在编译的时候并没有被编译进目标代码中，你的程序执行到相关函数时才调用该函数库里的相应函数，因此动态函数库所产生的可执行文件比较小。由于函数库没有被整合进你的程序，而是程序运行时动态的申请并调用，所以程序的运行环境中必须提供相应的库。动态函数库的改变并不影响你的程序，所以动态函数库的升级比较方便。

在Linux下编译程序的时候，我们通常使用-l选项来指明要链接的库，编译完了以后，使用 ldd命令检查发现程序链接的是动态库，原来，使用-l选项的时候，连接器在支持动态链接的系统上会优先搜索可用的动态库，如果有，则链接动态库，如果没有动态库，才会链接静态库。大家可以把/usr/lib下某个库的动态版本都移掉，之后，再编译程序，再使用ldd就会发现，不再那个动态库了。（如果在移掉动态库以后，编译的时候，报找不到dlopen之类的链接错误，需要把-ldl的编译选项加上）。

一、引言

通常情况下，对函数库的链接是放在编译时期□compile time□完成的。所有相关的对象文件□object file□与牵涉到的函数库□library□被链接合成一个可执行文件□executable file□□程序在运行时，与函数库再无瓜葛，因为所有需要的函数已拷贝到自己门下。所以这些函数库被成为静态库□static libaray□□通常文件名为“libxxx.a”的形式。

其实，我们也可以把对一些库函数的链接载入推迟到程序运行的时期□runtime□□这就是如雷贯耳的动态链接库□dynamic link library□技术。

二、动态链接库的特点与优势

首先让我们来看一下，把库函数推迟到程序运行时期载入的好处：

1. 可以实现进程之间的资源共享。

什么概念呢？就是说，某个程序的在运行中要调用某个动态链接库函数的时候，操作系统首先会查看所有正在运行的程序，看在内存里是否已有此库函数的拷贝了。如果有，则让其共享那一个拷贝；只有没有才链接载入。这样的模式虽然会带来一些“ 动态链接 ”额外的开销，却大大的节省了系统的内存资源□C的标准库就是动态链接库，也就是说系统中所有运行的程序共享着同一个C标准库的代码段。

2. 将一些程序升级变得简单。用户只需要升级动态链接库，而无需重新编译链接其他原有的代码就可以完成整个程序的升级□Windows 就是一个很好的例子。

3. 甚至可以真正坐到链接载入完全由程序员在程序代码中控制。

程序员在编写程序的时候，可以明确的指明什么时候或者什么情况下，链接载入哪个动态链接库函数。你可以有一个相当大的软件，但每次运行的时候，由于不同的操作需求，只有一小部分程序被载入内存。所有的函数本着“ 有需求才调入 ”的原则，于是大大节省了系统资源。比如现在的软件通常都能打开若干种不同类型的文件，这些读写操作通常都用动态链接库来实现。在一次运行当中，一般只有一种类型的文件将会被打开。所以直到程序知道文件的类型以后再载入相应的读写函数，而不是一开始就将所有的读写函数都载入，然后才发觉在整个程序中根本没有用到它们。

三、动态链接库的创建

由于动态链接库函数的共享特性，它们不会被拷贝到可执行文件中。在编译的时候，编译器只会做一些函数名之类的检查。在程序运行的时候，被调用的动态链接库函数被安置在内存的某个地方，所有调用它的程序将指向这个代码段。因此，这些代码必须实用相对地址，而不是绝对地址。在编译的时候，我们需要告诉编译器，这些对象文件是用来做动态链接库的，所以要用地址不无关代码□Position Independent Code □PIC□□□

对gcc编译器，只需添加上 -fPIC 标签，如：

gcc -fPIC -c file1.c gcc -fPIC -c file2.c gcc -shared libxxx.so file1.o file2.o

注意到最后一行□-shared 标签告诉编译器这是要建立动态链接库。这与静态链接库的建立很不一样，后者用的是 ar 命令。也注意到，动态链接库的名字形式为 “libxxx.so” 后缀名为 “.so”

四、动态链接库的使用

使用动态链接库，首先需要在编译期间让编译器检查一些语法与定义。

这与静态库的实用基本一样，用的是 -Lpath 和 -lxxx 标签。如：

gcc file1.o file2.o -Lpath -lxxx -o program.exe

编译器会先在path文件夹下搜索libxxx.so文件，如果没有找到，继续搜索libxxx.a□静态库）。

在程序运行期间，也需要告诉系统去哪里找你的动态链接库文件。在UNIX下是通过定义名为 LD_LIBRARY_PATH 的环境变量来实现的。只需将path赋值给此变量即可□csh 命令为：

setenv LD_LIBRARY_PATH your/full/path/to/dll

一切安排妥当后，你可以用 ldd 命令检查是否连接正常。

ldd program.exe

如果一切顺利，它将会输出你所用到的函数库的清单。否则，将会抱怨无法找到某个库。