

UC Day02

预习课

预习 内容

什么是库文件？

什么是库文件

编程模型的发展

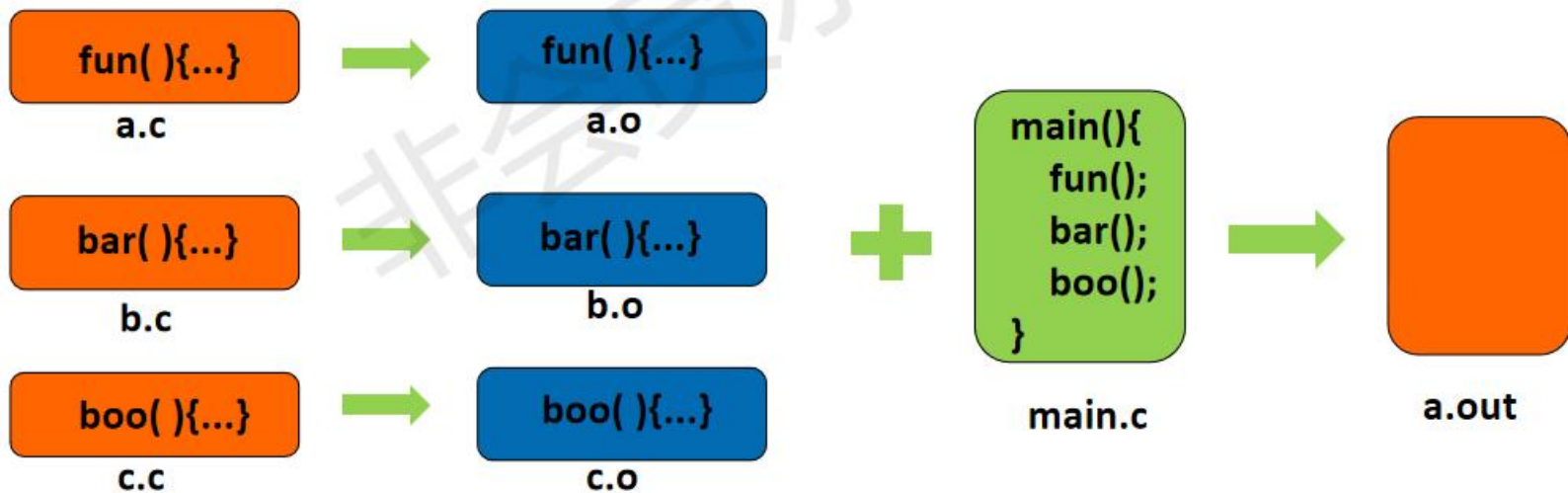
- 单一模型：
 - 将程序中所有功能全部实现于一个单一的源文件内部。编译时间长，不易于维护和升级，不易于协作开发。



编程模型的发展

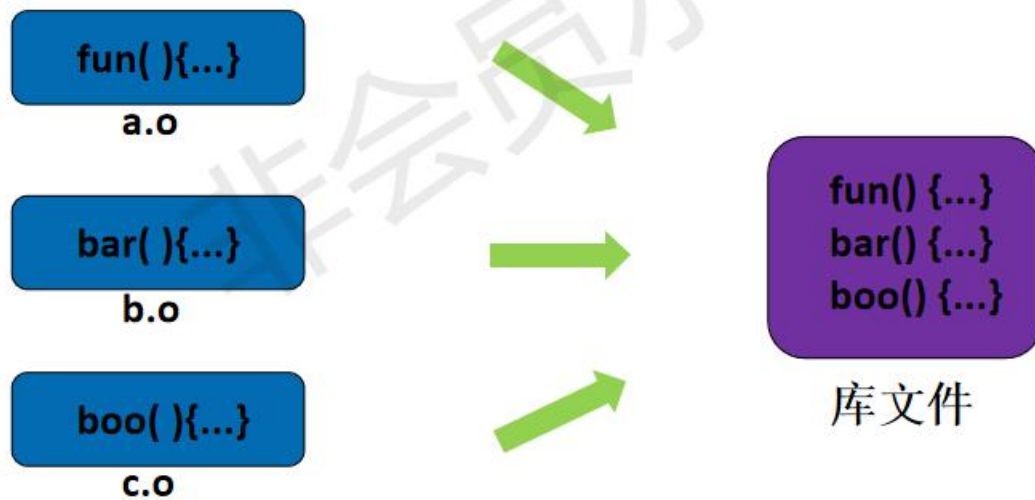
- 分离模型:

- 将程序中的不同功能模块划分到不同的源文件中。缩短编译时间，易于维护和升级，易于协作开发



编程模型的发展

- 对多个目标文件的管理比较麻烦
 - 将多个目标文件统一整合合成为一个文件便于使用和管理，于是就有了库文件。



什么是库文件

- 为何要把一个程序分成多个源文件，并由每个源文件编译生成独立的目标文件？
 - 化整为零、易于维护、便于协作。
- 为何要把多个目标文件合并成一个库文件？
 - 集零为整、方便使用、易于复用。
- 可以简单的把库文件看成一种代码仓库，它提供给使用者一些可以直接拿来用的变量、函数或类
 - 库文件一般指计算机上的一类文件，分两种，一种是静态库，另一种是动态库



直播课见

UC

C/C++教学体系

目录

静态库的制作和使用

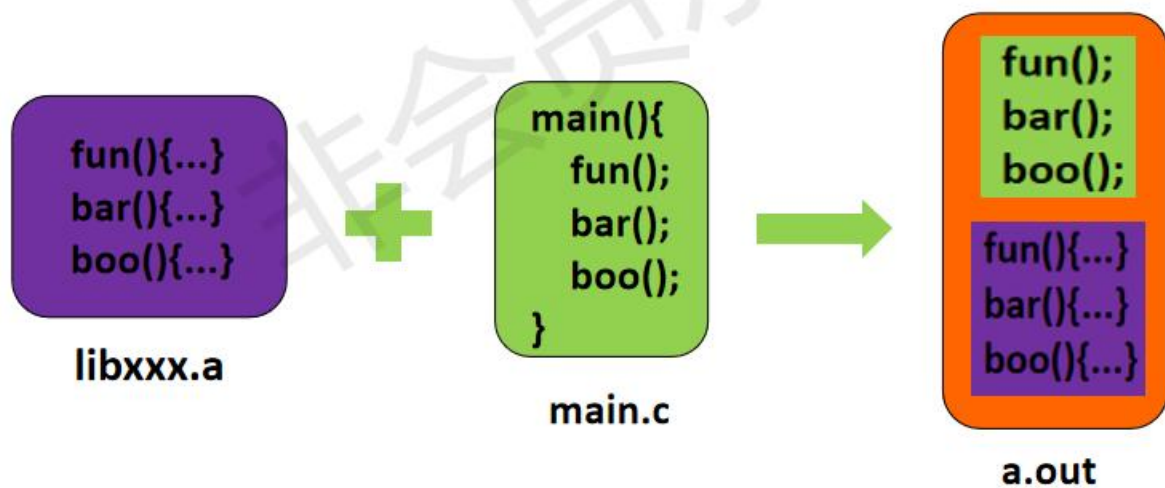
动态库的制作和使用

动态库的动态加载

静态库的制作和使用

静态库的制作和使用

- 静态库的本质就是将多个目标文件打包成一个文件
- 链接静态库的过程就是将库中被调用的代码复制到调用模块中
- 静态库的拓展名是 .a 例: libxxx.a



静态库的制作和使用

- 以构建数学库为例，静态库的构建顺序如下：

- 1、编辑库的实现代码和接口声明：

计算模块：calc.h、calc.c

显示模块：show.h、show.c

接口文件：math.h

- 2、编译成目标文件

```
gcc -c calc.c
```

```
gcc -c show.c
```

- 3、打包成静态库

```
ar -r libmath.a calc.o show.o
```



静态库的制作和使用

- ar 命令
 - ar [选项] <静态库文件> <目标文件列表>
 - r 将目标插入到静态库中，已存在则更新
 - q 将目标文件追加到静态库尾
 - d 从静态库中删除目标文件
 - t 列表显示静态库中的目标文件
 - x 将静态库展开为目标文件



静态库的制作和使用

- 编辑库的使用代码
 - main.c
- 编译并链接静态库
 - 直接链接静态库
gcc main.c libmath.a
 - 用-l指定库名, 用-L指定库路径
gcc main.c -lmath -L..
- 用-l指定库名, 用LIBRARY_PATH环境变量指定库路径
 - export LIBRARY_PATH=\$LIBRARY_PATH:..
 - gcc main.c -lmath



动态库的制作和使用

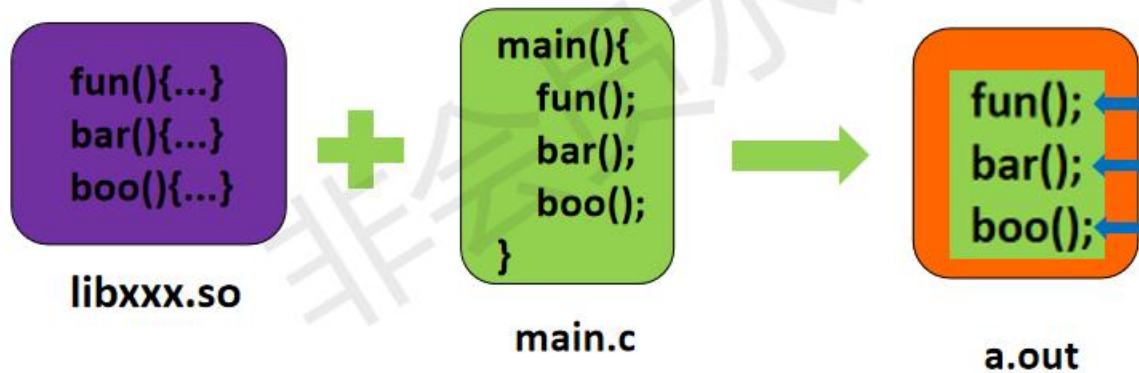
动态库的制作和使用

- 动态库和静态库不同，链接动态库不需要将被调用的函数代码复制到包含调用代码的可执行文件中，相反链接器会在调用语句处嵌入一段指令，在该程序执行，行到这段指令时，会加载该动态库并寻找被调用函数的入口地址并执行之。
- 如果动态库中的代码同时为多个进程所用，动态库在内存的实例仅需一份，为所有使用该库的进程所共享，因此动态库亦称共享库。
- 动态库的拓展名是 .so 例libxxx.so



动态库的制作和使用

- 链接动态库过程



动态库的制作和使用

- 以构建数学库为例，动态库的构建顺序如下：

- 1、编辑库的实现代码和接口声明：

计算模块：calc.h、calc.c

显示模块：show.h、show.c

接口文件：math.h

- 2、编译成目标文件

```
gcc -c -fpic calc.c
```

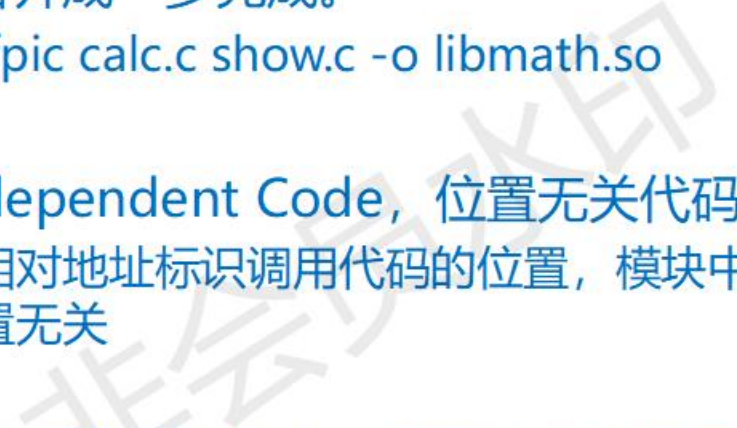
```
gcc -c -fpic show.c
```

- 3、打包成静态库

```
gcc -shared calc.o show.o -o libmath.so
```



动态库的制作和使用

- 编译链接也可以合并成一步完成。
 - `gcc -shared -fpic calc.c show.c -o libmath.so`
- PIC (Position Independent Code, 位置无关代码)。 
 - 调用代码通过相对地址标识调用代码的位置，模块中的指令与该模块被加载到内存中的位置无关
- -fPIC : 大模式，生成代码比较大，运行速度比较慢，所有平台都支持
- -fpic : 小模式，生成代码比较小，运行速度比较快，仅部分平台支持



动态库的制作和使用

- 编辑库的使用代码
 - main.c
- 编译并链接静态库
 - 直接链接动态库
 - gcc main.c libmath.a
 - 用-l指定库名, 用-L指定库路径
 - gcc main.c -lmath -L..
- 用-l指定库名, 用LIBRARY_PATH环境变量指定库路径
 - export LIBRARY_PATH=\$LIBRARY_PATH:..
 - gcc main.c -lmath



动态库的制作和使用

- 运行时需要保证LD_LIBRARY_PATH环境变量中包含共享库所在的路径用以告知链接器在运行时链接动态库。
 - export LD_LIBRARY_PATH=\$LD_LIBRARY_PATH:.
- 在可执行程序的链接阶段，并不将所调用函数的二进制代码复制到可执行程序中，而只是将该函数在共享库中的地址嵌入到调用模块中，因此运行时需要依赖共享库。



动态库的动态加载

动态库的动态加载

- 在程序执行的过程中，开发人员可以动态加载共享库（什么时候用什么时候加载），这样可以提高内存的利用效率。
- 在程序中动态加载共享库需要调用一组特殊的函数，它们被声明于一个专门的头文件中，并在一个独立的库中予以实现。
- 使用这组函数需要包含此头文件，并链接该库
 - `#include <dlfcn.h>`
 - `-ldl`



动态库的动态加载

- `void* dlopen(char const* filename, int flag);`
 - 功能：将共享库载入内存并获得其访问句柄
 - 参数：
 - filename 动态库路径，若只给文件名不带目录，则根据LD_LIBRARY_PATH环境变量的值搜索动态库
 - flag 加载方式，可取以下值：
 - RTLD_LAZY - 延迟加载，使用动态库中的符号时才真的加载进内存。
 - RTLD_NOW - 立即加载。
 - 返回值：成功返回动态库的访问句柄，失败返回NULL。
 - 句柄：句柄唯一地标识了系统内核所维护的共享库对象，将作为后续函数调用的参数



动态库的动态加载

- `void* dlsym(void* handle, char const* symbol);`
 - 功能：从已被加载的动态库中获取特定名称的符号地址
 - 参数：`handle` 动态库访问句柄
`symbol` 符号名
 - 返回值：成功返回给定符号的地址，失败返回NULL。
 - 该函数所返回的指针为void*类型，需要造型为与实际目标类型相一致的指针，才能使用。



动态库的动态加载

- `int dlclose(void* handle);`
 - 功能：从内存中卸载动态库
 - 参数：`handle` 动态库句柄
 - 返回值：返回值：成功返回0,失败返回非0。
 - 所卸载的共享库未必会真的从内存中立即消失，因为其他程序可能还需要使用该库
 - 只有所有使用该库的程序都显示或隐式地卸载了该库，该库所占用的内存空间才会真正得到释放
 - 无论所卸载的共享库是否真正被释放，传递给`dlclose`函数的句柄都会在该函数成功返回后立即失效



动态库的动态加载

- `char* dlerror(void);`
 - 功能：获取在加载、使用和卸载共享库过程中所发生的错误
 - 返回值：有错误则返回指向错误信息字符串的指针，否则返回NULL。



动态库的动态加载

- 辅助工具

- 查看符号表: nm
- 列出目标文件、可执行程序、静态库、或共享库中的符号
- 例: nm libmath.a
- 查看依赖: ldd
- 查看可执行文件或者共享库所依赖的共享库
- 例: ldd a.out



谢谢

UC

C/C++教学体系

目录

静态库的制作和使用

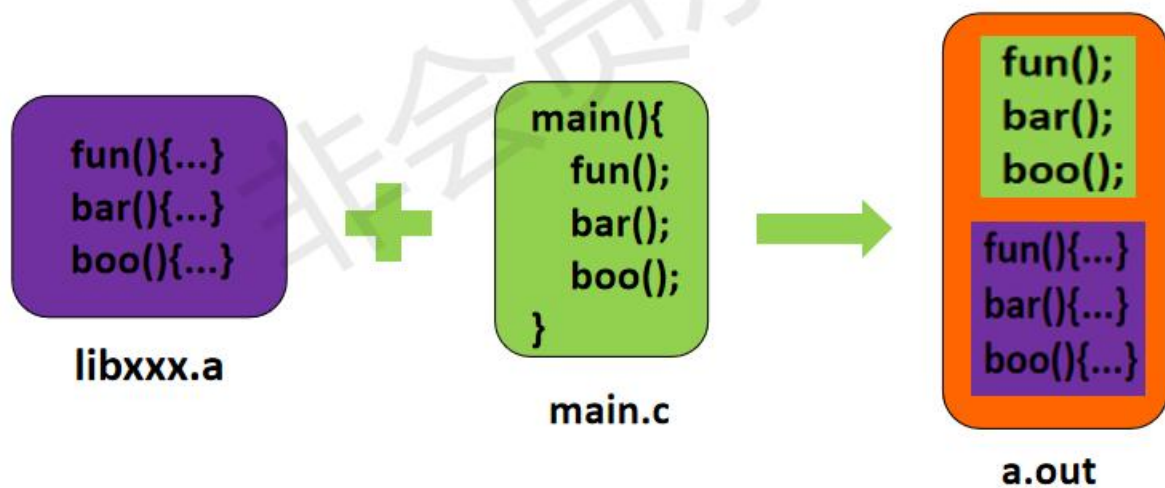
动态库的制作和使用

动态库的动态加载

静态库的制作和使用

静态库的制作和使用

- 静态库的本质就是将多个目标文件打包成一个文件
- 链接静态库的过程就是将库中被调用的代码复制到调用模块中
- 静态库的拓展名是 .a 例: libxxx.a



静态库的制作和使用

- 以构建数学库为例，静态库的构建顺序如下：

- 1、编辑库的实现代码和接口声明：

计算模块：calc.h、calc.c

显示模块：show.h、show.c

接口文件：math.h

- 2、编译成目标文件

```
gcc -c calc.c
```

```
gcc -c show.c
```

- 3、打包成静态库

```
ar -r libmath.a calc.o show.o
```



静态库的制作和使用

- ar 命令
 - ar [选项] <静态库文件> <目标文件列表>
 - r 将目标插入到静态库中，已存在则更新
 - q 将目标文件追加到静态库尾
 - d 从静态库中删除目标文件
 - t 列表显示静态库中的目标文件
 - x 将静态库展开为目标文件



静态库的制作和使用

- 编辑库的使用代码
 - main.c
- 编译并链接静态库
 - 直接链接静态库
gcc main.c libmath.a
 - 用-l指定库名, 用-L指定库路径
gcc main.c -lmath -L..
- 用-l指定库名, 用LIBRARY_PATH环境变量指定库路径
 - export LIBRARY_PATH=\$LIBRARY_PATH:..
 - gcc main.c -lmath



动态库的制作和使用

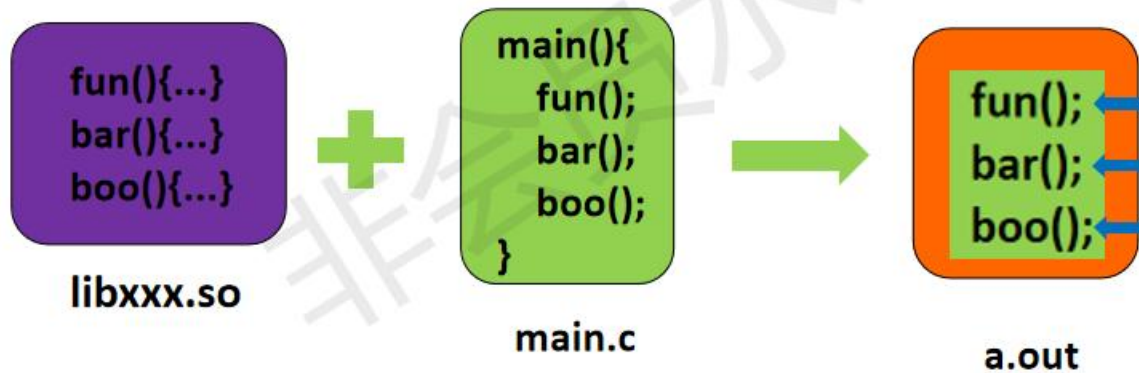
动态库的制作和使用

- 动态库和静态库不同，链接动态库不需要将被调用的函数代码复制到包含调用代码的可执行文件中，相反链接器会在调用语句处嵌入一段指令，在该程序执行，行到这段指令时，会加载该动态库并寻找被调用函数的入口地址并执行之。
- 如果动态库中的代码同时为多个进程所用，动态库在内存的实例仅需一份，为所有使用该库的进程所共享，因此动态库亦称共享库。
- 动态库的拓展名是 .so 例libxxx.so



动态库的制作和使用

- 链接动态库过程



动态库的制作和使用

- 以构建数学库为例，动态库的构建顺序如下：

- 1、编辑库的实现代码和接口声明：

计算模块：calc.h、calc.c

显示模块：show.h、show.c

接口文件：math.h

- 2、编译成目标文件

```
gcc -c -fpic calc.c
```

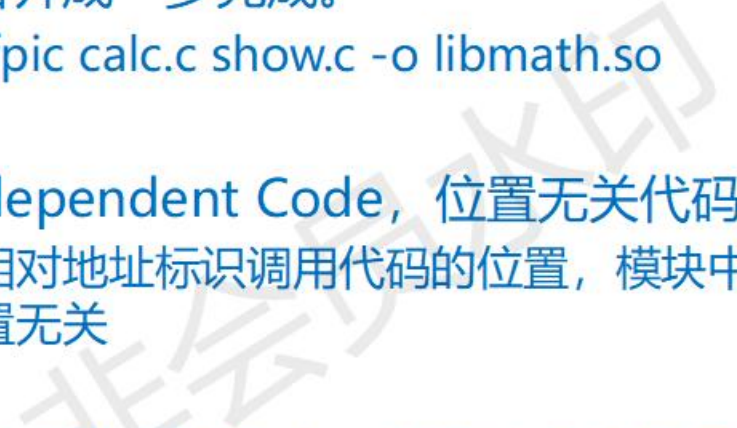
```
gcc -c -fpic show.c
```

- 3、打包成静态库

```
gcc -shared calc.o show.o -o libmath.so
```



动态库的制作和使用

- 编译链接也可以合并成一步完成。
 - `gcc -shared -fpic calc.c show.c -o libmath.so`
- PIC (Position Independent Code, 位置无关代码)。 
 - 调用代码通过相对地址标识调用代码的位置，模块中的指令与该模块被加载到内存中的位置无关
- -fPIC : 大模式，生成代码比较大，运行速度比较慢，所有平台都支持
- -fpic : 小模式，生成代码比较小，运行速度比较快，仅部分平台支持



动态库的制作和使用

- 编辑库的使用代码
 - main.c
- 编译并链接静态库
 - 直接链接动态库
 - gcc main.c libmath.a
 - 用-l指定库名, 用-L指定库路径
 - gcc main.c -lmath -L..
- 用-l指定库名, 用LIBRARY_PATH环境变量指定库路径
 - export LIBRARY_PATH=\$LIBRARY_PATH:..
 - gcc main.c -lmath



动态库的制作和使用

- 运行时需要保证LD_LIBRARY_PATH环境变量中包含共享库所在的路径用以告知链接器在运行时链接动态库。
 - export LD_LIBRARY_PATH=\$LD_LIBRARY_PATH:.
- 在可执行程序的链接阶段，并不将所调用函数的二进制代码复制到可执行程序中，而只是将该函数在共享库中的地址嵌入到调用模块中，因此运行时需要依赖共享库。



动态库的动态加载

动态库的动态加载

- 在程序执行的过程中，开发人员可以动态加载共享库（什么时候用什么时候加载），这样可以提高内存的利用效率。
- 在程序中动态加载共享库需要调用一组特殊的函数，它们被声明于一个专门的头文件中，并在一个独立的库中予以实现。
- 使用这组函数需要包含此头文件，并链接该库
 - `#include <dlfcn.h>`
 - `-ldl`



动态库的动态加载

- `void* dlopen(char const* filename, int flag);`
 - 功能：将共享库载入内存并获得其访问句柄
 - 参数：
 - filename 动态库路径，若只给文件名不带目录，则根据LD_LIBRARY_PATH环境变量的值搜索动态库
 - flag 加载方式，可取以下值：
 - RTLD_LAZY - 延迟加载，使用动态库中的符号时才真的加载进内存。
 - RTLD_NOW - 立即加载。
 - 返回值：成功返回动态库的访问句柄，失败返回NULL。
 - 句柄：句柄唯一地标识了系统内核所维护的共享库对象，将作为后续函数调用的参数



动态库的动态加载

- `void* dlsym(void* handle, char const* symbol);`
 - 功能：从已被加载的动态库中获取特定名称的符号地址
 - 参数：`handle` 动态库访问句柄
`symbol` 符号名
 - 返回值：成功返回给定符号的地址，失败返回NULL。
 - 该函数所返回的指针为void*类型，需要造型为与实际目标类型相一致的指针，才能使用。



动态库的动态加载

- `int dlclose(void* handle);`
 - 功能：从内存中卸载动态库
 - 参数：`handle` 动态库句柄
 - 返回值：返回值：成功返回0,失败返回非0。
 - 所卸载的共享库未必会真的从内存中立即消失，因为其他程序可能还需要使用该库
 - 只有所有使用该库的程序都显示或隐式地卸载了该库，该库所占用的内存空间才会真正得到释放
 - 无论所卸载的共享库是否真正被释放，传递给`dlclose`函数的句柄都会在该函数成功返回后立即失效



动态库的动态加载

- `char* dlerror(void);`
 - 功能：获取在加载、使用和卸载共享库过程中所发生的错误
 - 返回值：有错误则返回指向错误信息字符串的指针，否则返回NULL。



动态库的动态加载

- 辅助工具

- 查看符号表: nm
- 列出目标文件、可执行程序、静态库、或共享库中的符号
- 例: nm libmath.a
- 查看依赖: ldd
- 查看可执行文件或者共享库所依赖的共享库
- 例: ldd a.out



谢谢

UC Day02

复习课

静态库和动态库比较

使用静态库的优缺点

- 优点:

链接静态库生成的可执行程序，执行速度更快
可执行程序的执行不依赖库的存在

- 缺点:

链接静态库生成的可执行程序，文件体积相对较大
更新困难，维护成本高



使用动态库的优缺点

- 优点：
链接动态库生成的可执行程序体积更小节省空间
易于链接，便于更新维护
- 缺点：
链接动态库生成的可执行程序执行速度相对较慢
可执行程序的执行依赖库文件的存在



下节课见