1105动静态库

静态库，是会把代码拷贝进可执行程序的（在链接的时候）

而动态库，而是让我们自己的程序和动态库和程序产生关联

```
mymath.o:mymath.c
    gcc -fPIC -c mymath.c -o mymath.o
myprint.o:myprint.c
    gcc -fPIC -c myprint.c -o myprint.o
```

-fPIC 表示生成一个和位置无关的二进制库

与位置无关表示：可以在任何地址在被加载

```
libhello.so:mymath.o myprint.o
    gcc -shared myprint_d.o mymath_d.o -o libhello.so
mymath.o:mymath.c
    gcc -fPIC -c mymath.c -o mymath_d.o
myprint.o:myprint.c
    gcc -fPIC -c myprint.c -o myprint_d.o
```

形成动态库不用使用ar命令了

直接使用 gcc 命令
但是，我怎么知道你要生成的是可执行程序还是动态库？
所以我们要带上一个 –shared 选项

```
yufc@VM-12-12-centos:~/bit/1105/myLib2/mklib$ ll
total 36
-rwxrwxr-x 1 yufc yufc 8152 Feb 10 23:17 libhello.so
-rw-rw-r-- 1 yufc yufc  356 Feb 10 23:15 Makefile
-rw-rw-r-- 1 yufc yufc  167 Feb 10 23:05 mymath.c
-rw-rw-r-- 1 yufc yufc   74 Feb 10 23:05 mymath.h
-rw-rw-r-- 1 yufc yufc 1280 Feb 10 23:17 mymath.o
-rw-rw-r-- 1 yufc yufc  100 Feb 10 23:05 myprint.c
-rw-rw-r-- 1 yufc yufc   87 Feb 10 23:05 myprint.h
-rw-rw-r-- 1 yufc yufc 1624 Feb 10 23:17 myprint.o
yufc@VM-12-12-centos:~/bit/1105/myLib2/mklib$
```

Makefile ~/.../myLib2/... × | Makefile ~/.../myLib/mklib

bit > 1105 > myLib2 > mklib > Makefile

```
1  .PHONY:all
2  all:libhello.so libhello.a
3
4  libhello.so:mymath_d.o myprint_d.o
5      gcc -shared myprint_d.o mymath_d.o -o libhello.so
6  mymath_d.o:mymath.c
7      gcc -c -fPIC  mymath.c -o mymath_d.o
8  myprint_d.o:myprint.c
9      gcc -c -fPIC  myprint.c -o myprint_d.o
10
11 libhello.a:mymath.o myprint.o
12     ar -rc libhello.a mymath.o myprint.o
13 mymath.o:mymath.c
14     gcc -c mymath.c -o mymath.o
15 myprint.o:myprint.c
16     gcc -c myprint.c -o myprint.o
17
18
19 .PHONY:output
20 output:
21     mkdir -p output/lib
22     mkdir -p output/include
23     cp -rf *.h output/include
24     cp -rf *.a output/lib
25     cp -rf *.so output/lib
26
27 .PHONY:clean
28 clean:
29     rm -f *.o *.a *.so
30     rm -rf output/
```

同时生成静态和动态的

发布的时候把动态也放进去

```
yufc@VM-12-12-centos:~/bit/1105/myLib2/mklib$ ll
total 48
-rw-rw-r-- 1 yufc yufc 3074 Feb 10 23:30 libhello.a
-rwxrwxr-x 1 yufc yufc 8152 Feb 10 23:30 libhello.so
-rw-rw-r-- 1 yufc yufc  612 Feb 10 23:30 Makefile
-rw-rw-r-- 1 yufc yufc  167 Feb 10 23:05 mymath.c
-rw-rw-r-- 1 yufc yufc 1280 Feb 10 23:30 mymath_d.o
-rw-rw-r-- 1 yufc yufc   74 Feb 10 23:05 mymath.h
-rw-rw-r-- 1 yufc yufc 1280 Feb 10 23:30 mymath.o
-rw-rw-r-- 1 yufc yufc  100 Feb 10 23:05 myprint.c
-rw-rw-r-- 1 yufc yufc 1624 Feb 10 23:30 myprint_d.o
-rw-rw-r-- 1 yufc yufc   87 Feb 10 23:05 myprint.h
-rw-rw-r-- 1 yufc yufc 1576 Feb 10 23:30 myprint.o
yufc@VM-12-12-centos:~/bit/1105/myLib2/mklib$
```

```
yufc@VM-12-12-centos:~/bit/1105/myLib2/mklib$ make output
mkdir -p output/lib
mkdir -p output/include
cp -rf *.h output/include
cp -rf *.a output/lib
cp -rf *.so output/lib
yufc@VM-12-12-centos:~/bit/1105/myLib2/mklib$ tree output
output
├── include
│   ├── mymath.h
│   └── myprint.h
└── lib
    ├── libhello.a
    └── libhello.so

2 directories, 4 files
yufc@VM-12-12-centos:~/bit/1105/myLib2/mklib$
```

```
yufc@VM-12-12-centos:~/bit/1105/myLib2/usrlib$ gcc main.c -I output/include/ -L output/lib -lhello
yufc@VM-12-12-centos:~/bit/1105/myLib2/usrlib$ ll
total 20
-rwxrwxr-x 1 yufc yufc 8480 Feb 10 23:37 a.out
-rw-rw-r-- 1 yufc yufc  136 Feb 10 23:05 main.c
drwxrwxr-x 4 yufc yufc 4096 Feb 10 23:34 output
yufc@VM-12-12-centos:~/bit/1105/myLib2/usrlib$
```

这个是上节课的方法

问题是：默认用的是动态库还是静态库？

动态库！看下面这个情况

```
yufc@VM-12-12-centos:~/bit/1105/myLib2/usrlib$ ./a.out
./a.out: error while loading shared libraries: libhello.so: cannot open shared object file: No such file or di
rectory
yufc@VM-12-12-centos:~/bit/1105/myLib2/usrlib$ ldd a.out
        linux-vdso.so.1 =>  (0x00007ffc66f73000)
        libhello.so => not found
        libc.so.6 => /lib64/libc.so.6 (0x00007fab3c72f000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fab3cafd000)
yufc@VM-12-12-centos:~/bit/1105/myLib2/usrlib$
```

a.out 无法执行！

为什么上节课就能编过？

如果只有静态库，没办法，gcc只能针对该库进行静态链接
注意，在上节课这个命令中
只有libhello.a是静态链接的，别的都是动态链接的，gcc默认进行的是动态链接，如果想要全部都是静态链接，我们需要添加 -static 选项

如果动静态库都有，默认使用的是动态库！如果我非要用静态库呢
添加 – static 选项

-static 的意义：摒弃默认优先使用动态库的原则，而是直接使用静态库的方案！

此时我们来谈一下动态库的加载！

2. 站在是一个使用者(程序员)角度 -- 使用以动态库

a. 如果我们只有静态库，没办法，gcc只能针对该库进行静态链接

b. 如果动静态库同时存在，默认用的就是动态库

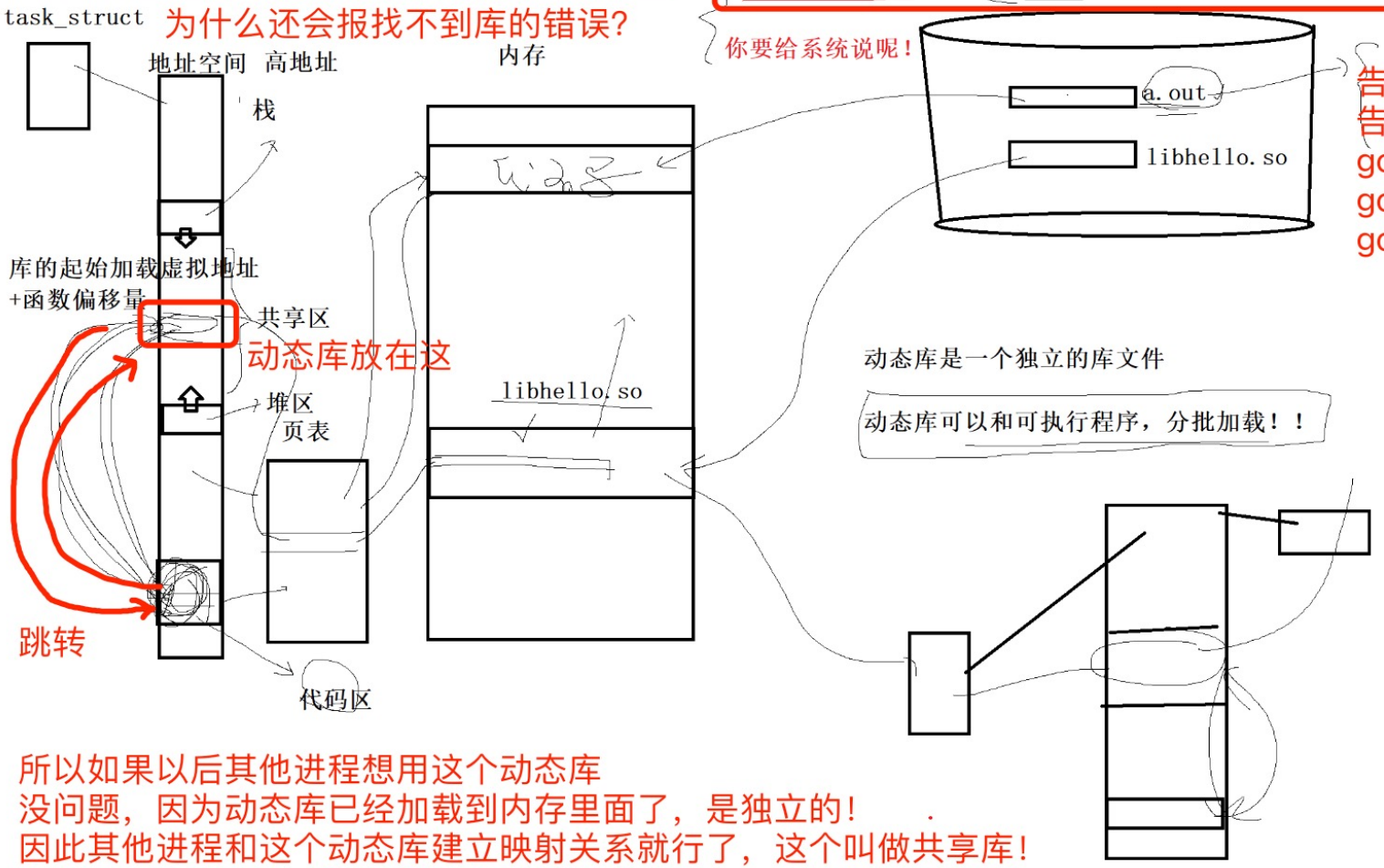
c. 如果动静态库同时存在，我非要使用静态库呢？ -static的意义： 摒弃默认优先使用动态库的原则，而是直接使用静态库的方案！

d. 我不是已经告诉了你我的动态库的路径了吗？？

给gcc说的！！！！，运行加载的时候，和gcc还有关系吗？？

操作系统加载器

你要给系统说呢！



告诉谁了？
告诉gcc了
gcc和运行加载有关系吗？
gcc是什么？
gcc是编译器！

动态库是一个独立的库文件

动态库可以和可执行程序，分批加载！！

所以如果以后其他进程想用这个动态库
没问题，因为动态库已经加载到内存里面了，是独立的！
因此其他进程和这个动态库建立映射关系就行了，这个叫做共享库！

所以，我们要告诉系统，动态库在哪

第一种方式：
拷贝进/lib64这些系统路径里面去
不过这种方式太粗糙了

另外的方法：

第一种方案：

```
yufc@VM-12-12-centos:~/bit/1105/myLib2$ echo $LD_LIBRARY_PATH
:/home/yufc/.VimForCpp/vim/bundle/YCM.so/el7.x86_64:/home/yufc/.VimForCpp/vim/bundle/YCM.so/el7.x86_64:/home/yufc/.VimForCpp/vim/bundle/YCM.so/el7.x86_64
yufc@VM-12-12-centos:~/bit/1105/myLib2$
```

这个环境变量
告诉系统
每次加载默认搜索库的路径
所以我们可以导入环境变量

```
yufc@VM-12-12-centos:~/bit/1105/myLib2$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/yufc/bit/1105/myLib2/usrlib/output/lib/
yufc@VM-12-12-centos:~/bit/1105/myLib2$ echo $LD_LIBRARY_PATH
:/home/yufc/.VimForCpp/vim/bundle/YCM.so/el7.x86_64:/home/yufc/.VimForCpp/vim/bundle/YCM.so/el7.x86_64:/home/yufc/.VimForCpp/vim/bundle/YCM.so/el7.x86_64:/home/yufc/bit/1105/myLib2/usrlib/output/lib/
yufc@VM-12-12-centos:~/bit/1105/myLib2$
```

$ 原来的路径
： 新路径

```
yufc@VM-12-12-centos:~/bit/1105/myLib2/usrlib$ ./a.out
hello lib[1676087822]
3
yufc@VM-12-12-centos:~/bit/1105/myLib2/usrlib$
```

此时可以成功运行了！

这个方法是临时的，因为导入的环境变量是内存级别的环境变量

第二种方案：

修改配置文件
永久保存我们修改后默认搜索路径

> - 3、ldconfig 配置/etc/ld.so.conf.d/，ldconfig更新

```
[root@localhost linux]# cat /etc/ld.so.conf.d/bit.conf
/root/tools/linux
[root@localhost linux]# ldconfig
```

具体怎么弄
可以看看视频

第三种方案：

把这个动态库建立一个快捷方式放到 /lib64/下面就行了

```
yufc@VM-12-12-centos:~/bit/1105/myLib2/usrlib$ sudo ln -s /home/yufc/bit/1105/myLib2/usrlib/output/lib/libhell
o.so /lib64/libhello.so
[sudo] password for yufc:
yufc@VM-12-12-centos:~/bit/1105/myLib2/usrlib$ ls /lib64/libhello.so
/lib64/libhello.so
yufc@VM-12-12-centos:~/bit/1105/myLib2/usrlib$ ll /lib64
lrwxrwxrwx. 1 root root 9 Mar  7  2019 /lib64 -> usr/lib64
yufc@VM-12-12-centos:~/bit/1105/myLib2/usrlib$ ls -l /lib64/libhello.so
lrwxrwxrwx 1 root root 56 Feb 11 12:06 /lib64/libhello.so -> /home/yufc/bit/1105/myLib2/usrlib/output/lib/libh
ello.so
yufc@VM-12-12-centos:~/bit/1105/myLib2/usrlib$
```

```
yufc@VM-12-12-centos:~/bit/1105/myLib2/usrlib$ ldd a.out
        linux-vdso.so.1 =>  (0x00007ffd896c3000)
        libhello.so => /lib64/libhello.so (0x00007fa5bb968000)
        libc.so.6 => /lib64/libc.so.6 (0x00007fa5bb59a000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fa5bbb6a000)
yufc@VM-12-12-centos:~/bit/1105/myLib2/usrlib$ ./a.out
hello lib[1676089055]
3
yufc@VM-12-12-centos:~/bit/1105/myLib2/usrlib$
```

第四种方案：

改配置文件
~/.bashrc
~/.bash_profie
这两个文件会在我们登陆的时候执行
可以在里面导入环境变量

这样我们启动的时候就自动export了

不建议这样搞，出了问题不好调整

为什么要有库？

1. 站在使用库的角度，库的存在，可以大大减少我们开发的周期，提高软件本身的质量
2. 站写写库的人的角度 --- a.简单 b.代码安全

推荐两个好玩的库

1. **ncurses --- 字符的界面库 --- centos 7 yum 安装ncurses**
2. **Boost --- C++的准标准库**