✨ ✨所属专栏：Linux✨ ✨

✨ ✨作者主页：嵌某✨ ✨

什么是库

库是写好的现有的，成熟的可以复用的代码。现实中每个程序都需要依赖很多基础的底层库。世界上有很多大佬为了实现某一个功能，写了很多很NB的代码。他们把代码封装成一个库，这样我们不必写出像他们一样厉害的代码，只需要使用它们分享的库，也能使用对应的功能了。

本质上来说库是一种可执行代码的二进制形式，可以被操作系统载入内存执行。

- **静态库 .a[Linux/macOS]、.lib[windows]**
- **动态库 .so[Linux]、.dll[windows]、.dylib[macOS]**

静态库

- 程序在编译链接的时候把库的代码链接到可执行文件中，程序运行的时候将不再需要静态库。
- 一个一个可执行程序可能用到许多的库，这些库运行有的是静态库，有的是动态库，而我们的编译默认为动态链接库，只有在该库下找不到动态.so的时候才会采用同名静态库。我们也可以使用 gcc的 -static 强转设置链接静态库。

制作静态库

25/2_5 · 钦某/Code - 码云 - 开源中国 (gitee.com)

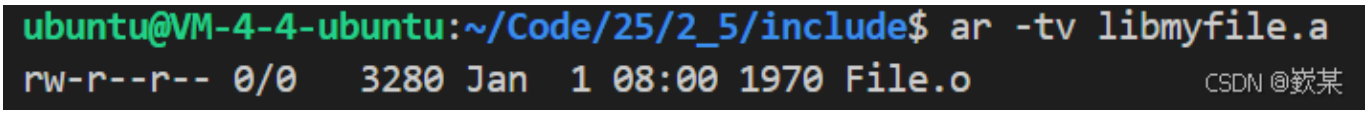
```
// Makefile

libmyfile.a:File.o
    @ar -rc $@ $^
    @echo "bulid $^ to $@ ... done"
%.o:%.c
    @gcc -c $<
    @echo "compling $< to $@ ... done"

.PHONY:output
output:
    mkdir -p ../lib/include
    mkdir -p ../lib/mylib
    cp -f *.h ../lib/include
    cp -f *.a ../lib/mylib
    tar czf lib.tgz lib

.PNOLY:clean
clean:
    @rm -rf *.a *.o *tgz
    @echo "clean ... done"
```

ar是gun归档工具，rc表示(replace and create)



- t：列出静态库中的文件
- v：verbose详细信息

使用静态库

```
// 任意目录下新建
// test.c,引入库头文件

#include "File.h"
#include

int main()
{
    MYFILE* file = my_openfile("log.txt","w");
    char* str = "abcdef";
    my_fwrite(file, str, strlen(str));
    my_closefile(file);

    return 0;
}

// 场景1：头文件和库文件安装到系统路径下
$ gcc main.c -lmyfile

// 场景2：头文件和库文件和我们自己的源文件在同一个路径下
$ gcc main.c -L. -lmyfile

// 场景3：头文件和库文件有自己的独立路径
$ gcc main.c -I头文件路径 -L库文件路径 -lmyfile
```

- -L(library头字母大写)：指定库路径
- -I(include头字母大写)：指定头文件搜索路径

- -l(library头字母小写)：指定库名
- 测试目标文件生成后，静态库删掉，程序照样可以运行
- 库名称 = 库文件名去掉前缀lib，去掉后缀.so，.a

```
ubuntu@VM-4-4-ubuntu:~/Code/25/2_5$ tree .
.
├── a.out
├── include
│   ├── File.c
│   ├── File.h
│   ├── File.o
│   ├── libmyfile.a
│   ├── lib.tgz
│   └── Makefile
├── lib
│   ├── include
│   │   └── File.h
│   └── mylib
│       └── libmyfile.a
├── Makefile
└── test.c

5 directories, 11 files
```

动态库

- 动态库（.so）：程序在运行的时候才去链接动态库的代码，多个程序共享使用库的代码。
- 一个与动态库链接的可执行文件仅仅包含它用到的函数入口地址的一个表，而不是外部函数所在目标文件的整个机器码
- 在可执行文件开始运行以前，外部函数的机器码由操作系统从磁盘上的该动态库中复制到内存中，这个过程称为动态链接（dynamic linking）
- 动态库可以在多个程序间共享，所以动态链接使得可执行文件更小，节省了磁盘空间。操作系统采用虚拟内存机制允许物理内存中的一份动态库被要用到该库的所有进程共用，节省了内存和磁盘空间。

生成动态库

```
// Makefile

libmyfile.so:File.o
    gcc -o $@ $^ -shared

%.o:%.c
    gcc -fPIC -c $<

.PHONY:output
output:
    mkdir -p ../lib/include
    mkdir -p ../lib/mylib
    cp -f *.h ../lib/include
    cp -f *.so ../lib/mylib
    tar czf lib.tgz lib

.PHONY:clean
clean:
    rm -rf *.o *.so lib.tgz
```

- shared：表示生成共享库格式
- **fPIC：产生位置无关码（position independent code）**
- 库名规则：libxxx.so

使用动态库

```
// 场景1：头文件和库文件安装到系统路径下
$ gcc main.c -lmyfile

// 场景2：头文件和库文件和我们自己的源文件在同一个路径下
$ gcc main.c -L. -lmyfile // 从左到右搜索-L指定的目录

// 场景3：头文件和库文件有自己的独立路径
$ gcc main.c -I头文件路径 -L库文件路径 -lmyfile

$ ldd libmystdio.so // 查看库或者可执行程序的依赖
linux-vdso.so.1 => (0x00007fffacbbf000)
```

```
libc.so.6 => /lib64/libc.so.6 (0x00007f8917335000)
/lib64/ld-linux-x86-64.so.2 (0x00007f8917905000)
```

```
ubuntu@VM-4-4-ubuntu:~/Code/25/2_6$ tree .
.
├── a.out
├── include
│   ├── File.c
│   ├── File.h
│   ├── File.o
│   ├── libmyfile.so
│   ├── lib.tgz
│   └── Makefile
├── lib
│   ├── include
│   │   └── File.h
│   └── mylib
│       └── libmyfile.so
├── Makefile
└── test.c

5 directories, 11 files
```

库运行搜索路径

问题

```
ubuntu@VM-4-4-ubuntu:~/Code/25/2_6$ ldd a.out
        linux-vdso.so.1 (0x00007ffcbe5ab000)
        libmyfile.so => not found
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x000077eea6a00000)
        /lib64/ld-linux-x86-64.so.2 (0x000077eea6cf5000)
```

- **解决方案**
- 拷贝 .so 文件到系统共享路径下，一般指/usr/lib、usr/local/lib、/lib64或者其他的默认库路径

- 向系统共享库路径下建立同名软连接
- 更改环境变量：LD_LIBRARY_PATH
- ldconfig方案：配置/etc/ld.so.conf.d/，ldconfig更新

在系统`/etc/ld.so.conf/`目录下新建文件`my_library.conf`，并在文件里写入库文件路径，关闭保存后使用`sudo ldconfig`命令，这将扫描 `/etc/ld.so.conf` 文件以及 `/etc/ld.so.conf.d/` 目录下的所有配置文件，并更新 `/etc/ld.so.cache` 缓存文件。

本期博客到这里就结束了，如果有什么错误，欢迎指出，如果对你有帮助，请点个赞，谢谢！