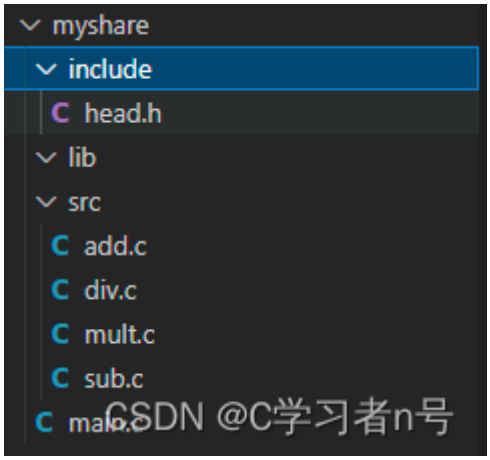
动态库的创建

1. 动态库的命名规则
   Linux : libxxx.so
   lib : 前缀（固定）
   xxx : 库的名字，自己起
   so : 后缀（固定）
   在Linux下是一个可执行文件
   2、制作
   目录如下：



第一步：gcc 得到 .o 文件，得到和位置无关的代码

> gcc -c –fpic/-fPIC a.c b.c

进入src目录中，使用一下命令生成.o文件，此时当前目录中就有了四个.o文件

```
gcc -c -fpic add.c sub.c mult.c div.c -I../include
```

第二步：gcc得到库文件libxxx.so

> gcc -shared a.o b.o -o libcalc.so

还是在src目录中，使用fcc命令生成库文件

```
gcc -shared add.o sub.o mult.o div.o -o libcalc.so
```

至此，libcalc.so动态库创建完毕。其中-shared、-fpic或者-fPIC都是gcc的参数。

动态库的使用

将创建出来的动态库放到lib目录中，并且回退到main.c所在的目录，也就是myshare中。此时main.c就需要使用动态库。

编译链接生成可执行文件app，执行完一下代码就会生成app可执行文件

```
gcc -o app main.c -I./include -lcalc -L./lib/
```

执行app文件，出现找不到动态库的问题，结果如下

```
./app
./app: error while loading shared libraries: libcalc.so: cannot open shared
object file: No such file or directory
```

动态库的工作原理

■ 静态库：gcc进行链接时，会把静态库中代码打包到可执行程序中
■ 动态库：gcc进行链接时，动态库的代码不会被打包到可执行程序中，只是会打包动态库的一些信息到其中。
■ 程序启动之后，动态库会被动态加载到内存中，通过 ldd （list dynamic dependencies）命令检查动态库依赖关系。可以看到libcalc.so动态库无法找到。

```
kiko@Hkiko:~/lesson/myshare$ ldd app
        linux-vdso.so.1 (0x00007ffefa2e6000)
        libcalc.so => not found
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f9de1c89000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f9de1e94000)
```

值得注意的是，如果程序是依赖的动态库的，且引用了动态库的头文件，但是在动态库中没有调用动态库中的任何API或者类等，此时并不会因为找不到动态库文件而执行失败。所以当应用程序在执行到动态库的API等的时候才会去寻找动态库载入其中的代码到内存中。
■ 如何定位共享库文件呢?
当系统加载可执行代码时候，能够知道其所依赖的库的名字，但是还需要知道**绝对路径**。此时就需要系统的动态载入器来获取该绝对路径。对于elf格式的可执行程序，是由ld-linux.so来完成的，它先后搜索elf文件的 DT_RPATH段 ——> 环境变量LD_LIBRARY_PATH ——> /etc/ld.so.cache文件列表 ——> /lib/，/usr/lib目录找到库文件后将其载入内存。

解决动态库加载失败问题

根据动态载入器的搜索路径，只要能够把动态库的路径存放到其中的一块地方即可。因为DT_RPATH段无法更改，所以我们只能从其他地方入手。

修改环境变量

- 修改用户级别的成员变量
  在用户的~目录下面有一个.bashrc的隐藏文件，通过vim命令进入编辑，在这个文本的最后加上一行如下代码:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/kiko/lesson/myshare/lib
```

:后面的路径就是存放动态库的绝对路径。
在终端还要通过如下命令才能使这个文件的修改生效:

```
source .bashrc    //或者 . .bashrc
```

此时，使用ldd命令就可以看到应用程序能够找到动态库所在的路径，自然也能够执行成功。

```
kiko@Hkiko:~/lesson/myshare$ ldd app
        linux-vdso.so.1 (0x00007ffc9df6d000)
        libcalc.so => /home/kiko/lesson/myshare/lib/libcalc.so
(0x00007fa69fffd000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fa69fdf9000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fa6a0009000)
```

- 修改系统级别环境变量
  通过一下命令直接进入系统级别的配置文件。同样的，在最后一行增加和上面一样的代码。

```
sudo vim /etc/profile
```

修改保存之后，使之生效:

```
source /etc/profile
```

进入应用程序所在目录，通过ldd命令查看app程序的依赖是否都能找到

```
kiko@Hkiko:~/lesson/myshare$ ldd app
        linux-vdso.so.1 (0x00007ffef19d0000)
        libcalc.so => /home/kiko/lesson/myshare/lib/libcalc.so
(0x00007fc0ef2cd000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fc0ef0c9000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fc0ef2d9000)
```

修改/etc/ld.so.cache文件列表

这个文件是一个二进制文件，无法直接修改，需要间接修改，即/ect/ld.so.conf文件。使用sudo打开这个文件，在最后一行增加需要加载的动态库的绝对路径的即可。

```
sudo vim /etc/ld.so.conf
```

编辑完之后需要使之生效:

```
sudo ldconfig
```

此时查看应用程序是否能够找到动态库的路径，显然是成功的。

```
kiko@Hkiko:~/lesson/myshare$ ldd app
        linux-vdso.so.1 (0x00007ffeac1fa000)
        libcalc.so => /home/kiko/lesson/myshare/lib/libcalc.so
(0x00007f3734e82000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f3734c90000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f3734ea0000)
```

/lib和/usr/lib

将生成的libcalc.so动态库文件放到这两个目录中的一个都可以使程序运行，但是不建议使用，因为这两个目录本身就存放了许多系统自带的一些动态库，有可能造成冲突，覆盖掉系统自带的库。