gcc 生成静态库和动态库

我们通常把一些公用函数制作成函数库，供其它程序使用。函数库分为静态库和动态库两种。

静态库在程序编译时会被连接到目标代码中，程序运行时将不再需要该静态库。动态库在程序编译时并不会被连接到目标代码中，而是在程序运行是才被载入，因此在程序运行时还需要动态库存在。

在创建函数库前，我们先来准备举例用的源程序，并将函数库的源程序编译成.o 文件。

第 1 步：编辑得到举例的程序--hello.h、hello.c 和 main.c；

hello.c(见程序 2)是函数库的源程序，其中包含公用函数 hello，该函数将在屏幕上输出"
Hello XXX!"。hello.h(见程序 1)为该函数库的头文件。main.c(见程序 3)为测试库文件的主程序，在主程序中调用了公用函数 hello。

程序 1: hello.h

```
#ifndef HELLO_H
#define HELLO_H

void hello(const char *name);

#endif //HELLO_H
```

程序 2: hello.c

```
#include <stdio.h>

void hello(const char *name)
{
printf("Hello %s!\n", name);
}
```

程序 3: main.c

```
#include "hello.h"

int main()
{
hello("everyone");
return 0;
}
```

第 2 步：将 hello.c 编译成.o 文件；

无论静态库，还是动态库，都是由.o 文件创建的。因此，我们必须将源程序 hello.c 通过 gcc 先编译成.o 文件。

在系统提示符下键入以下命令得到 hello.o 文件。

```
# gcc -c hello.c
```

我们运行 ls 命令看看是否生存了 hello.o 文件。

```
# ls

hello.c hello.h hello.o main.c
```

在 ls 命令结果中，我们看到了 hello.o 文件，本步操作完成。

下面我们先来看看如何创建静态库，以及使用它。

第 3 步：由.o 文件创建静态库；

静态库文件名的命名规范是以 lib 为前缀，紧接着跟静态库名，扩展名为.a。例如：我们将创建的静态库名为 myhello，则静态库文件名就是 libmyhello.a。在创建和使用静态库时，需要注意这点。创建静态库用 ar 命令。

在系统提示符下键入以下命令将创建静态库文件 libmyhello.a。

```
# ar crv libmyhello.a hello.o
```

我们同样运行 ls 命令查看结果：

```
# ls
```

hello.c hello.h hello.o libmyhello.a main.c

ls 命令结果中有 libmyhello.a。

第 4 步：在程序中使用静态库；

静态库制作完了，如何使用它内部的函数呢？只需要在使用到这些公用函数的源程序中包含这些公用函数的原型声明，然后在用 gcc 命令生成目标文件时指明静态库名，gcc 将会从静态库中将公用函数连接到目标文件中。注意，gcc 会在静态库名前加上前缀 lib，然后追加扩展名.a 得到的静态库文件名来查找静态库文件。

在程序 3:main.c 中，我们包含了静态库的头文件 hello.h，然后在主程序 main 中直接调用公用函数 hello。下面先生成目标程序 hello，然后运行 hello 程序看看结果如何。

```
# gcc -o hello main.c -L.  -lmyhello

#gcc main.c libmyhello.a -o main

# ./hello

Hello everyone!
```

我们删除静态库文件试试公用函数 hello 是否真的连接到目标文件 hello 中了。

```
# rm libmyhello.a

rm: remove regular file `libmyhello.a'? y

# ./hello

Hello everyone!

#
```

程序照常运行，静态库中的公用函数已经连接到目标文件中了。

我们继续看看如何在 Linux 中创建动态库。我们还是从.o 文件开始。

第 5 步：由.o 文件创建动态库文件；

动态库文件名命名规范和静态库文件名命名规范类似，也是在动态库名增加前缀 lib，但其文件扩展名为.so。例如：我们将创建的动态库名为 myhello，则动态库文件名就是 libmyhello.so。

用 gcc 来创建动态库。

在系统提示符下键入以下命令得到动态库文件 libmyhello.so。

```
# gcc -shared -fPCI -o libmyhello.so hello.o
```

我们照样使用 ls 命令看看动态库文件是否生成。

```
# ls

hello.c hello.h hello.o libmyhello.so main.c
```

第 6 步：在程序中使用动态库；

在程序中使用动态库和使用静态库完全一样，也是在使用到这些公用函数的源程序中包含这些公用函数的原型声明，然后在用 gcc 命令生成目标文件时指明动态库名进行编译。我们先运行 gcc 命令生成目标文件，再运行它看看结果。

```
# gcc -o hello main.c -L. -lmyhello

# ./hello

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

出错。错误提示找不到动态库文件 libmyhello.so。程序在运行时，会在/usr/lib 和/lib 等目录中查找需要的动态库文件。若找到，则载入动态库，否则将提示类似上述错误而终止程序运行。我们将文件 libmyhello.so 复制到目录/usr/lib 中，再试试。

```
# mv libmyhello.so /usr/lib

# ./hello

Hello everyone!
```

成功。这也进一步说明了动态库在程序运行时是需要的。

我们回过头看看，发现使用静态库和使用动态库编译成目标程序使用的 gcc 命令完全一样，那当静态库和动态库同名时，gcc 命令会使用哪个库文件呢？

先删除除.c 和.h 外的所有文件，恢复成我们刚刚编辑完举例程序状态。

```
# rm -f hello hello.o /usr/lib/libmyhello.so

# ls

hello.c hello.h main.c
```

在来创建静态库文件 libmyhello.a 和动态库文件 libmyhello.so。

```
# gcc -c hello.c

# ar cr libmyhello.a hello.o

# gcc -shared -fPCI -o libmyhello.so hello.o

# ls

hello.c hello.h hello.o libmyhello.a libmyhello.so main.c
```

通过上述最后一条 ls 命令，可以发现静态库文件 libmyhello.a 和动态库文件 libmyhello.so 都已经生成，并都在当前目录中。然后，我们运行 gcc 命令来使用函数库 myhello 生成目标文件 hello，并运行程序 hello。

```
# gcc -o hello main.c -L. -lmyhello

# ./hello

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

从程序 hello 运行的结果中很容易知道，当静态库和动态库同名时， gcc 命令将优先使用动态库。

Note:
编译参数解析
最主要的是 GCC 命令行的一个选项:

-shared 该选项指定生成动态连接库（让连接器生成 T 类型的导出符号表，有时候也生成弱连接 W 类型的导出符号），不用该标志外部程序无法连接。相当于一个可执行文件

-fPIC：表示编译为位置独立的代码，不用此选项的话编译后的代码是位置相关的所以动态载入时是通过代码拷贝的方式来满足不同进程的需要，而不能达到真正代码段共享的目的。

-L.：表示要连接的库在当前目录中

-ltest：编译器查找动态连接库时有隐含的命名规则，即在给出的名字前面加上 lib，后面加上.so 来确定库的名称

LD_LIBRARY_PATH：这个环境变量指示动态连接器可以装载动态库的路径。当然如果有 root 权限的话，可以修改/etc/ld.so.conf 文件，然后调/sbin/ldconfig 来达到同样的目的，不过如果没有 root 权限，那么只能采用输 LD_LIBRARY_PATH 的方法了。

调用动态库的时候有几个问题会经常碰到，有时，明明已经将库的头文件所在目录 通过 “-I” include 进来了，库所在文件通过 “-L”参数引导，并指定了“-l”的库名，但通过 ldd 命令察看时，就是死活找不到你指定链接的 so 文件，这时你要作的就是通过修改 LD_LIBRARY_PATH 或者/etc/ld.so.conf 文件来指定动态库的目录。通常这样做就可以解决库无法链接的问题了。