

1031动静态库

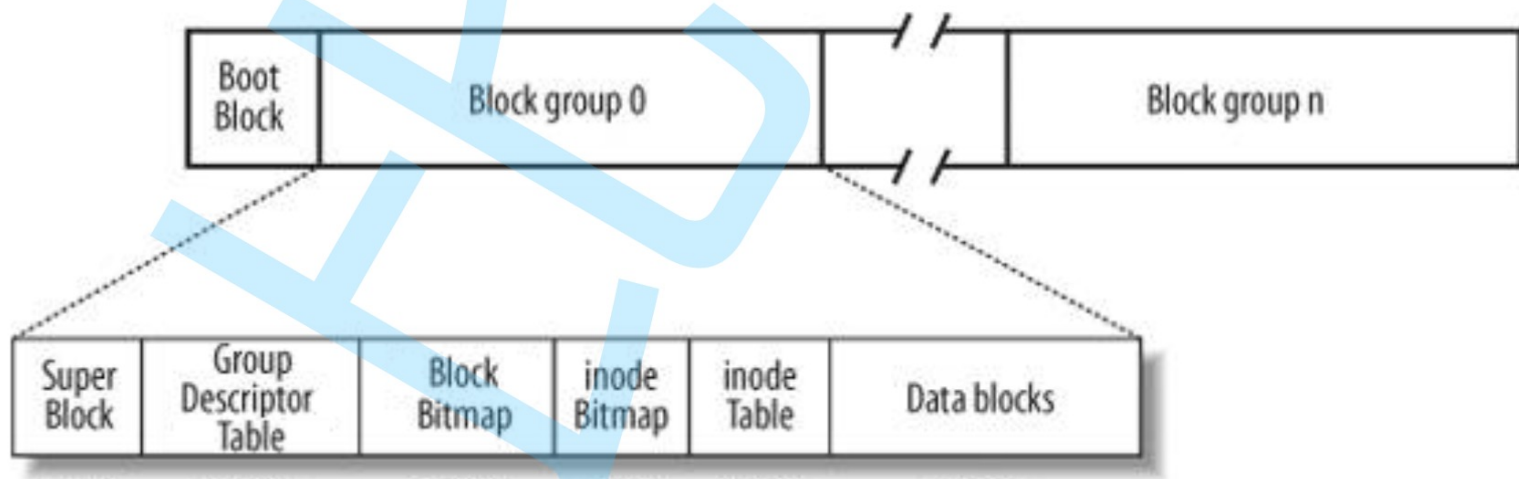
如果文件特别大的时候怎么办呢？

在data block中，不是所有的datablock只能存文件数据，也可以存其他块的块号！

我们通过索引找到一个块之后，可以通过这个块继续找到下面的块  
这样就解决了要存大文件的问题

### inode

为了能解释清楚inode我们先简单了解一下文件系统



找到一个文件的步骤：

inode 编号 -> 分区特定的bg -> inode -> 属性 -> 内容

那么现在的问题是

Inode编号是怎么得到的？

Linux中

inode属性里面，没有文件名这个说法！

为了理解这个概念，我们需要补充一些预备知识：

1. 一个目录下，可以保存很多文件，但是这些文件的名称是不能重复的！
2. 目录是文件吗？是 -> 有自己的inode，有自己的datablock！

**一个文件的文件名，是存在datablock里面存的！  
datablock里面存了：文件名和inode编号的映射关系！**

inode和文件名都是唯一的！（在一个目录下）  
它们是互为 key 的！

所以我们在一个目录下创建文件  
是需要写w权限的！

本质就是把，inode和文件名的映射写到这个目录的datablock等地方去！

所以，找到inode编号是要依托于目录结构的

下面我们要回答三个问题：

1. 创建文件，系统做了什么？
2. 删除文件，系统做了什么？
3. 查看文件，系统做了什么？

为什么删除总比拷贝快很多？

因为删除的时候，不用把内容这部分东西真的删掉  
只需要把位图标记改了就行了

所以，删了的东西能恢复吗？肯定是可以的，只是我们不会而已  
我们只要找到原来的inode，找到磁盘的位置（删除日志）  
只要它还没被覆盖  
就一定能找到

一道面试题：

为什么还有空间，但是一直不能创建文件呢？  
可能就是因为inode申请不下来  
文件无法创建

如果创建出来，也只有个文件名没有inode  
这个时候一写就会失败，一写就会失败的，无法写入

# 软硬链接

这个是我们的准备工作

```
• yufc@VM-12-12-centos:~/bit/1031$ ls -li
total 4
1063227 -rw-rw-r-- 1 yufc yufc 11 Feb  8 23:54 testLink.txt
• yufc@VM-12-12-centos:~/bit/1031$ mkdir dir
• yufc@VM-12-12-centos:~/bit/1031$ touch testLink1.txt
• yufc@VM-12-12-centos:~/bit/1031$ touch testLink2.txt
• yufc@VM-12-12-centos:~/bit/1031$ ls -li
total 8
1063228 drwxrwxr-x 2 yufc yufc 4096 Feb  8 23:54 dir
1063229 -rw-rw-r-- 1 yufc yufc  0 Feb  8 23:54 testLink1.txt
1063230 -rw-rw-r-- 1 yufc yufc  0 Feb  8 23:54 testLink2.txt
1063227 -rw-rw-r-- 1 yufc yufc 11 Feb  8 23:54 testLink.txt
• yufc@VM-12-12-centos:~/bit/1031$
```

通过这个命令，我们建立一个软链接

```
• yufc@VM-12-12-centos:~/bit/1031$ ln -s testLink.txt soft.link
• yufc@VM-12-12-centos:~/bit/1031$ ll
total 8
drwxrwxr-x 2 yufc yufc 4096 Feb  8 23:54 dir
lrwxrwxrwx 1 yufc yufc 12 Feb  8 23:56 soft.link -> testLink.txt
-rw-rw-r-- 1 yufc yufc  0 Feb  8 23:54 testLink1.txt
-rw-rw-r-- 1 yufc yufc  0 Feb  8 23:54 testLink2.txt
-rw-rw-r-- 1 yufc yufc 11 Feb  8 23:54 testLink.txt
• yufc@VM-12-12-centos:~/bit/1031$
```

通过这个命令，我们建立一个硬链接

```
• yufc@VM-12-12-centos:~/bit/1031$ ln testLink.txt hard.link
• yufc@VM-12-12-centos:~/bit/1031$ ll
total 12
drwxrwxr-x 2 yufc yufc 4096 Feb  8 23:54 dir
-rw-rw-r-- 2 yufc yufc  11 Feb  8 23:54 hard.link
lrwxrwxrwx 1 yufc yufc  12 Feb  8 23:56 soft.link -> testLink.txt
-rw-rw-r-- 1 yufc yufc   0 Feb  8 23:54 testLink1.txt
-rw-rw-r-- 1 yufc yufc   0 Feb  8 23:54 testLink2.txt
-rw-rw-r-- 2 yufc yufc  11 Feb  8 23:54 testLink.txt
• yufc@VM-12-12-centos:~/bit/1031$
```

那么它们有什么区别呢？

```
• yufc@VM-12-12-centos:~/bit/1031$ ls -li
total 12
1063228 drwxrwxr-x 2 yufc yufc 4096 Feb  8 23:54 dir
1063227 -rw-rw-r-- 2 yufc yufc  11 Feb  8 23:54 hard.link
1063231 lrwxrwxrwx 1 yufc yufc  12 Feb  8 23:56 soft.link -> testLink.txt
1063229 -rw-rw-r-- 1 yufc yufc   0 Feb  8 23:54 testLink1.txt
1063230 -rw-rw-r-- 1 yufc yufc   0 Feb  8 23:54 testLink2.txt
1063227 -rw-rw-r-- 2 yufc yufc  11 Feb  8 23:54 testLink.txt
• yufc@VM-12-12-centos:~/bit/1031$
```

软硬链接的本质区别：  
有没有独立的inode  
软链接有独立的inode  
软链接是一个独立的文件  
硬链接没有独立的inode  
硬链接不是一个独立的文件

我们发现，软链接的inode是一个新的inode  
硬链接没有新的inode

这个解释，应该很清晰了！

我们每次在别的路径如果要执行，就很麻烦  
所以，我们可以利用软链接的方式！

```
• yufc@VM-12-12-centos:~/bit/1031$ ll
total 12
drwxrwxr-x 3 yufc yufc 4096 Feb  9 00:03 dir
-rw-rw-r-- 2 yufc yufc   11 Feb  8 23:54 hard.link
lrwxrwxrwx 1 yufc yufc   16 Feb  9 00:06 myexe.link -> ./dir/dir1/myexe
lrwxrwxrwx 1 yufc yufc   12 Feb  8 23:56 soft.link -> testLink.txt
-rw-rw-r-- 1 yufc yufc    0 Feb  8 23:54 testLink1.txt
-rw-rw-r-- 1 yufc yufc    0 Feb  8 23:54 testLink2.txt
-rw-rw-r-- 2 yufc yufc   11 Feb  8 23:54 testlink.txt
• yufc@VM-12-12-centos:~/bit/1031$ ./myexe.link
hello link
hello link
hello link
hello link
hello link
hello link
hello link
```

## 硬链接

创建硬链接，是创建新文件吗？不是！


硬链接有inode吗，有inode

创建硬链接究竟做了什么呢？

就是在制定的目录下，建立了文件名和制定的inode的映射关系，仅此而已！

说白了，就是起别名

```
• yufc@VM-12-12-centos:~/bit/1031$ ll
total 12
drwxrwxr-x 3 yufc yufc 4096 Feb  9 00:03 dir
-rw-rw-r-- 2 yufc yufc  11 Feb  8 23:54 hard.link
lrwxrwxrwx 1 yufc yufc  16 Feb  9 00:06 myexe.link -> ./dir/dir1/myexe
lrwxrwxrwx 1 yufc yufc  12 Feb  8 23:56 soft.link -> testLink.txt
-rw-rw-r-- 1 yufc yufc   0 Feb  8 23:54 testLink1.txt
-rw-rw-r-- 1 yufc yufc   0 Feb  8 23:54 testLink2.txt
-rw-rw-r-- 2 yufc yufc  11 Feb  8 23:54 testLink.txt
• yufc@VM-12-12-centos:~/bit/1031$ rm hard.link
• yufc@VM-12-12-centos:~/bit/1031$ ll
total 8
drwxrwxr-x 3 yufc yufc 4096 Feb  9 00:03 dir
lrwxrwxrwx 1 yufc yufc  16 Feb  9 00:06 myexe.link -> ./dir/dir1/myexe
lrwxrwxrwx 1 yufc yufc  12 Feb  8 23:56 soft.link -> testLink.txt
-rw-rw-r-- 1 yufc yufc   0 Feb  8 23:54 testLink1.txt
-rw-rw-r-- 1 yufc yufc   0 Feb  8 23:54 testLink2.txt
-rw-rw-r-- 1 yufc yufc  11 Feb  8 23:54 testLink.txt
• yufc@VM-12-12-centos:~/bit/1031$
```



硬链接数！

硬链接数是什么？  
很好理解！

就是 shared\_ptr！  
这样说，就很清晰了！

所以刚才无论是rm hard.link  
还是删除testLink.txt  
其实都是引用计数 --  
当删到0点时候，才是真正的删除！



硬链接有什么用？

一个问题：为什么创建目录的硬链接数是2？

```
• yufc@VM-12-12-centos:~/bit/1031/part2/dir$ cd ..
• yufc@VM-12-12-centos:~/bit/1031/part2$ ls -li
total 4
1063238 drwxrwxr-x 2 yufc yufc 4096 Feb  9 00:17 dir
1063239 -rw-rw-r-- 1 yufc yufc    0 Feb  9 00:17 test.c
• yufc@VM-12-12-centos:~/bit/1031/part2$ cd dir
• yufc@VM-12-12-centos:~/bit/1031/part2/dir$ ls
• yufc@VM-12-12-centos:~/bit/1031/part2/dir$ ls -ali
total 8
1063238 drwxrwxr-x 2 yufc yufc 4096 Feb  9 00:17 .
1063237 drwxrwxr-x 3 yufc yufc 4096 Feb  9 00:17 ..
• yufc@VM-12-12-centos:~/bit/1031/part2/dir$
```

我们发现dir里面的  
代表当前路径的

的inode  
和dir的是一样的！

所以dir和里面的. 一共两个！

.. 这些的数字，其实也是同一个道理

所以，我们就能理解这里的这些数字了

所以怎不用进去么判断一个目录里面有几个目录？

看他的硬链接数 减去 2 就行了！

# 动静态库

1. 我如果想写一个库？
2. 如果我把库给别人，别人是怎么用的呢？

- 静态库 (.a)：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
- 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。

```
• yufc@VM-12-12-centos:~/bit/1031/part2/lib/mklib$ touch mymath.h
• yufc@VM-12-12-centos:~/bit/1031/part2/lib/mklib$ touch mymath.c
• yufc@VM-12-12-centos:~/bit/1031/part2/lib/mklib$ touch myprint.h
• yufc@VM-12-12-centos:~/bit/1031/part2/lib/mklib$ touch myprint.c
• yufc@VM-12-12-centos:~/bit/1031/part2/lib/mklib$ ll
total 0
-rw-rw-r-- 1 yufc yufc 0 Feb  9 16:43 mymath.c
-rw-rw-r-- 1 yufc yufc 0 Feb  9 16:43 mymath.h
-rw-rw-r-- 1 yufc yufc 0 Feb  9 16:43 myprint.c
-rw-rw-r-- 1 yufc yufc 0 Feb  9 16:43 myprint.h
○ yufc@VM-12-12-centos:~/bit/1031/part2/lib/mklib$
```

准备工作，我们先创建好这四个文件

库里面，要不要main函数呢？  
不能！库是给别人用的！

现在我们需要给别人提供一组方法

## 准备工作：

```
h myprint.h  h mymath.h ×
bit > 1031 > part2 > lib > mklib > h mymath.h
1
2 #pragma once
3
4 #include<stdio.h>
5
6 extern int addToTarget(int from,int to);

C myprint.c  C mymath.c ●
bit > 1031 > part2 > lib > mklib > C mymath.c
1
2
3 #include "mymath.h"
4
5 int addToTarget(int from,int to)
6 {
7     int sum = 0;
8     for(int i = from; i<=to; i++)
9     {
10         sum += i;
11     }
12     return sum;
13 }
```

现在，我们的方法的声明和实现都写好了

但是我们不用去写main.c  
因为我们是准备当库来使用！

```
h myprint.h ×  h mymath.h
bit > 1031 > part2 > lib > mklib > h myprint.h
1
2 #pragma once
3
4 #include<stdio.h>
5 #include<time.h>
6
7 extern void Print(const char* str);

C myprint.c ×  C mymath.c ●
bit > 1031 > part2 > lib > mklib > C myprint.c
1
2
3 #include "myprint.h"
4
5
6 void Print(const char* str)
7 {
8     printf("%s[%d]\n",str,(int)time(NULL));
9 }
```

```
yufc@VM-12-12-centos:~/bit/1031/part2/lib$ mkdir usrlib
yufc@VM-12-12-centos:~/bit/1031/part2/lib$ ll
total 8
drwxrwxr-x 2 yufc yufc 4096 Feb  9 16:43 mklib
drwxrwxr-x 2 yufc yufc 4096 Feb  9 16:53 usrlib
yufc@VM-12-12-centos:~/bit/1031/part2/lib$
```

我们准备在usrlib这个文件夹里面去用mklib里面的方法

```

-rw-rw-r-- 1 yufc yufc 100 Feb  9 16:48 myprint.c
-rw-rw-r-- 1 yufc yufc  87 Feb  9 16:48 myprint.h
yufc@VM-12-12-centos:~/bit/1031/part2/lib/mklib$ gcc -c mymath.c -o my
mymath.c  mymath.h  myprint.c  myprint.h
• yufc@VM-12-12-centos:~/bit/1031/part2/lib/mklib$ gcc -c mymath.c -o mymath.o
• yufc@VM-12-12-centos:~/bit/1031/part2/lib/mklib$ ls
mymath.c  mymath.h  mymath.o  myprint.c  myprint.h
• yufc@VM-12-12-centos:~/bit/1031/part2/lib/mklib$ gcc -c myprint.c -o myprint.o
• yufc@VM-12-12-centos:~/bit/1031/part2/lib/mklib$ ll
total 24
-rw-rw-r-- 1 yufc yufc  140 Feb  9 16:50 mymath.c
-rw-rw-r-- 1 yufc yufc   74 Feb  9 16:49 mymath.h
-rw-rw-r-- 1 yufc yufc 1272 Feb  9 16:57 mymath.o
-rw-rw-r-- 1 yufc yufc  100 Feb  9 16:48 myprint.c
-rw-rw-r-- 1 yufc yufc   87 Feb  9 16:48 myprint.h
-rw-rw-r-- 1 yufc yufc 1576 Feb  9 16:57 myprint.o
• yufc@VM-12-12-centos:~/bit/1031/part2/lib/mklib$

```

我们生成两个 .o 文件之后

只需要和main.o 一链接

就能生成可执行了！

这个我们是知道的

如果此时  
我们只把 .h 和 .o 给别人去用，别人能用吗？  
可以！可以的！

但是，如果 .o 很多呢？用起来很麻烦  
如果 .o 丢了一个，怎么办？

所以，我们可以把这些 .o 打包！  
这个过程叫做形成静态库  
(在这种gcc情况下生成的.o完成打包，叫做静态库)

```

-rw-rw-r-- 1 yufc yufc 1576 Feb  9 16:57 myprint.o
• yufc@VM-12-12-centos:~/bit/1031/part2/lib/mklib$ ar -rc libhello.a mymath.o myprint.o
• yufc@VM-12-12-centos:~/bit/1031/part2/lib/mklib$ ll
total 28
-rw-rw-r-- 1 yufc yufc 3066 Feb  9 17:03 libhello.a
-rw-rw-r-- 1 yufc yufc  140 Feb  9 16:50 mymath.c
-rw-rw-r-- 1 yufc yufc   74 Feb  9 16:49 mymath.h
-rw-rw-r-- 1 yufc yufc 1272 Feb  9 16:57 mymath.o
-rw-rw-r-- 1 yufc yufc  100 Feb  9 16:48 myprint.c
-rw-rw-r-- 1 yufc yufc   87 Feb  9 16:48 myprint.h
-rw-rw-r-- 1 yufc yufc 1576 Feb  9 16:57 myprint.o
• yufc@VM-12-12-centos:~/bit/1031/part2/lib/mklib$

```

首先，命令叫做ar

然后 -rc

r表示替换，c表示创建

库的名字前缀必须是lib，没有原因

后面跟上要打包的.o

此时这个 .a 就是**静态库**！

现在我们把这个过程自动化一下，用Makefile实现一下

```

h myprint.h × C main.c Makefile × h mymath.h
bit > 1031 > part2 > lib > mklib > Makefile
1  libhello.a:mymath.o myprint.o
2      ar -rc libhello.a mymath.o myprint.o
3  mymath.o:mymath.c
4      gcc -c mymath.c -o mymath.o
5  myprint.o:myprint.c
6      gcc -c myprint.c -o myprint.o
7  .PHONY:clean
8  clean:
9      rm -f *.o libhello.a

```

在应用的时候，库一般是怎么给别人的呢？  
这些规范和习惯一定要注意

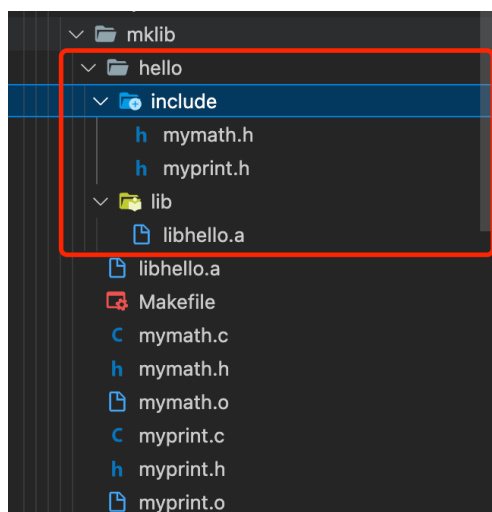
现在要将一个名字为hello的库给别人  
**hello**里面要有两个文件夹

**第一个叫做include 里面包含了所有的头文件**  
**第二个叫做lib里面包含了所有的库文件**

我们可以通过makefile去搞定这些事情

```
Makefile  x  C mymath.c  C myprint.c
bit > 1031 > part2 > myLib > mklib > Makefile
1  libhello.a:mymath.o myprint.o
2      ar -rc libhello.a mymath.o myprint.o
3  mymath.o:mymath.c
4      gcc -c mymath.c -o mymath.o
5  myprint.o:myprint.c
6      gcc -c myprint.c -o myprint.o
7
8  .PHONY:hello
9  hello:
10      mkdir -p hello/lib
11      mkdir -p hello/include
12      cp -rf *.h hello/include
13      cp -rf *.a hello/lib
14
15  .PHONY:clean
16  clean:
17      rm -f *.o libhello.a
18      rm -rf hello
```

```
total 20
-rw-rw-r-- 1 yufc yufc 335 Feb  9 17:15 Makefile
-rw-rw-r-- 1 yufc yufc 167 Feb  9 17:09 mymath.c
-rw-rw-r-- 1 yufc yufc  74 Feb  9 16:49 mymath.h
-rw-rw-r-- 1 yufc yufc 100 Feb  9 16:48 myprint.c
-rw-rw-r-- 1 yufc yufc  87 Feb  9 16:48 myprint.h
• yufc@VM-12-12-centos:~/bit/1031/part2/myLib/mklib$ make
gcc -c mymath.c -o mymath.o
gcc -c myprint.c -o myprint.o
ar -rc libhello.a mymath.o myprint.o
• yufc@VM-12-12-centos:~/bit/1031/part2/myLib/mklib$ make hello
mkdir -p hello/lib
mkdir -p hello/include
cp -rf *.h hello/include
cp -rf *.a hello/lib
• yufc@VM-12-12-centos:~/bit/1031/part2/myLib/mklib$
```



现在我们把这个库交给usrlib里面去用

我们讲三种方法：

第一种：

gcc 的默认搜索路径：  
头文件：/usr/include  
库：/lib64 or /usr/lib64

我们把我们hello的include拷贝到默认的头文件搜索路径下  
把hello的lib拷贝到默认的库路径下

```
yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib/hello/include$ ls
mymath.h  myprint.h
yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib/hello/include$ sudo cp -rf ./ * /usr/include/
[sudo] password for yufc:
yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib/hello/include$ cd ..
yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib/hello$ ls
include  lib
yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib/hello$ cd lib
yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib/hello/lib$ sudo cp -rf ./ * /lib64/
yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib/hello/lib$ ls /usr/include/mymath.h
/usr/include/mymath.h
yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib/hello/lib$ ls /usr/include/myprint.h
/usr/include/myprint.h
yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib/hello/lib$ ls /lib64/libhello.a
/lib64/libhello.a
yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib/hello/lib$
```

此时，我们就拷贝过去了！



```

• yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib/hello$ ls
include lib
• yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib/hello$ cd ..
• yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib$ ls
hello main.c
⊗ yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib$ gcc main.c
/tmp/cc0NB6MW.o: In function `main':
main.c:(.text+0xa): undefined reference to `Print'
main.c:(.text+0x19): undefined reference to `addToTarget'
collect2: error: ld returned 1 exit status
○ yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib$ █

```

此时，我们编译还是没有通过，为什么？

因为这些我们自己写的叫做第三方库  
而C语言的静态库，是自己会去链接的  
第三方库不会，所以我们gcc的时候要自己加选项链接

```

⊗ yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib$ gcc main.c
/tmp/ccGKCtpN.o: In function `main':
main.c:(.text+0xa): undefined reference to `Print'
main.c:(.text+0x19): undefined reference to `addToTarget'
collect2: error: ld returned 1 exit status
• yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib$ gcc main.c -lhello
• yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib$ ll
total 20
-rwxrwxr-x 1 yufc yufc 8584 Feb  9 17:29 a.out
drwxrwxr-x 4 yufc yufc 4096 Feb  9 17:18 hello
-rw-rw-r-- 1 yufc yufc 136 Feb  9 16:55 main.c
○ yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib$ █

```

-l表示链接静态库

libhello.a 把前缀和后缀去掉

-lhello

把库拷贝到系统更多默认路径下面，这个操作，叫做库的安装！

不建议用上面那种方法，  
因为我们自己写的库是没有进行过可靠性测试的  
也就是没经过测试的

直接安装可能会污染别人的头文件和库

所以我们赶紧把刚刚拷贝进去的rm掉，这个过程叫做**卸载**

```
⊗ yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrLib$ gcc main.c
main.c:3:21: fatal error: myprint.h: No such file or directory
#include "myprint.h"
                  ^
compilation terminated.
○ yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrLib$
```

```
collect2: error: ld returned 1 exit status
⊗ yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib$ gcc main.c -I ./hello/include -L ./hello/lib
/tmp/cckmMsfY.o: In function `main':
main.c:(.text+0xa): undefined reference to `Print'
main.c:(.text+0x19): undefined reference to `addToTarget'
collect2: error: ld returned 1 exit status
○ yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib$
```

通过 -I 和 -L 显性调用，为什么还是报错？

我们告诉系统库在 ./hello/lib 下，但是有没有告诉系统，是哪一个库呢？

```
● yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib$ gcc main.c -I ./hello/include -L ./hello/lib -lhello
● yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib$ ll
total 20
-rwxrwxr-x 1 yufc yufc 8584 Feb  9 17:39 a.out
drwxrwxr-x 4 yufc yufc 4096 Feb  9 17:18 hello
-rw-rw-r-- 1 yufc yufc  136 Feb  9 16:55 main.c
○ yufc@VM-12-12-centos:~/bit/1031/part2/myLib/usrlib$
```

剩下的两种调用库的方式，我们在动态库的时候用的比较多  
我们下节课再讲