<https://www.cnblogs.com/woainilsr/archive/2013/07/10/3182891.html>

今天要用到静态库和动态库，于是写了几个例子来巩固一下基础。  
hello1.c ————————————————————

#include <stdio.h>  
void print1(int i) { int j; for(j=0;j<i;j++) { printf("%d \* %d = %d\n",j,j,j\*j); }    }  
hello2.c \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include <stdio.h>  
void print2(char \*arr) { char c; int i=0; while((c=arr[i++])!='\0') { printf("%d\*\*\*\*%c\n",i,c); } }  
hello.c \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

void print1(int); void print2(char \*);  
int main(int argc,char \*\*argv) { int i=100; char \*arr="THIS IS LAYMU'S HOME!"; print1(i); print2(arr);  
return 0; }

可以看到hello.c要用到hello1.c中的print1函数和hello2.c中的print2函数。所以可以把这两个函数组合为库，以供更多的程序作为组件来调用。

方法一：将hello1.c和hello2.c编译成静态链接库.a

[root@localhost main5]#gcc -c hello1.c hello2.c

  //将hello1.c和hello2.c分别编译为hello1.o和hello2.o，其中-c选项意为只编译不链接。

[root@localhost main5]#ar -r libhello.a hello1.o hello2.o

//将hello1.o和hello2.o组合为libhello.a这个静态链接库

[root@localhost main5]#cp libhello.a /usr/lib

//将libhello.a拷贝到/usr/lib目录下，作为一个系统共享的静态链接库

[root@localhost main5]#gcc -o hello hello.c -lhello

//将hello.c编译为可执行程序hello，这个过程用到了-lhello选项，这个选项告诉gcc编译器到/usr/lib目录下去找libhello.a的静态链接库

以上的过程类似于windows下的lib静态链接库的编译及调用过程。  
方法二：将hello1.o和hello2.o组合成动态链接库.so

[root@localhost main5]#gcc -c -fpic hello1.c hello2.c

 //将hello1.c和hello2.c编译成hello1.o和hello2.o，-c意为只编译不链接，-fpic意为位置独立代码，指示编译程序生成的代码要适合共享库的内容这样的代码能够根据载入内存的位置计算内部地址。

[root@localhost main5]#gcc -shared hello1.o hello2.o -o hello.so

  //将hello1.o和hello2.o组合为shared object，即动态链接库

[root@localhost main5]#cp hello.so /usr/lib

  //将hello.so拷贝到/usr/lib目录下

[root@localhost main5]#gcc -o hello hello.c hello.so

//将hello.c编译链接为hello的可执行程序，这个过程用到了动态链接库hello.so

在这里要废话几句，其实一切的二进制信息都有其运作的机制，只要弄清楚了它的机制，并能够实现之，则任何此时此刻无法想象之事都将成为现实。当然，这两者之间的巨大鸿沟需要顶级的设计思想和顶级的代码来跨越。