**Lunix下编译静态库文件：**

.o后缀文件：编译生成的中间代码文件；

.a后缀文件：静态库文件，编译的时候会合到可执行程序中，文件比较大；

.so后缀文件：动态库文件，只是一个指向，不会合到可执行程序中，当要调用函数库的时候才使用；

用一个简单的例子阐述下在lunix环境下生成静态库文件的过程：

**一.   源码文件：**

so\_test.h：

void test\_a();

void test\_b();

void test\_c();

test\_a.c:

#include "so\_test.h"

void test\_a()

{

    printf("this is in test\_a...\n");

}

test\_b.c:

#include "so\_test.h"

void test\_a()

{

    printf("this is in test\_b...\n");

}

test\_c.c:

#include "so\_test.h"

void test\_a()

{

    printf("this is in test\_c...\n");

}

test.c:

#include "so\_test.h"

int main()

{

test\_a();

test\_b();

test\_c();

return 0;

}

**二.   makefile 文件：**

makefile：

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Copyright        :

#

# Author           :   kimi

# Date             :   2012-08-17

# Version          :   Demo2 静态链接库

# Description  :   Demo

#

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

SHELL = /bin/sh

LIB\_DIR = /cbs/lhbb/hcj/dy/lib/

BIN\_DIR = /cbs/lhbb/hcj/dy/bin/

OBJECT\_DIR = /cbs/lhbb/dy/test/obj/

APP\_DIR = /cbs/lhbb/hcj/dy/testapp/

$(shell mkdir -p ${LIB\_DIR})

$(shell mkdir -p ${BIN\_DIR})

$(shell mkdir -p ${OBJECT\_DIR})

RM = rm -fr

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CC = gcc

AR = ar rc

SRC\_OBJECT = $(APP\_DIR)test\_a.c $(APP\_DIR)test\_b.c $(APP\_DIR)test\_c.c

H\_OBJECT = $(APP\_DIR)so\_test.h

OBJECT = test\_a.o test\_b.o test\_c.o

DY\_SRC\_OBJECT = $(APP\_DIR)test.c

DY\_OBJECT=test.o

LIB\_OBJECT = libtest.a

BIN\_OBJECT = test

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.PHONY:all

all:$(LIB\_OBJECT) $(BIN\_OBJECT)

$(LIB\_OBJECT):$(OBJECT)

        $(AR) $(LIB\_OBJECT) $(OBJECT)

        mv $(LIB\_OBJECT) $(LIB\_DIR)

$(OBJECT):$(SRC\_OBJECT) $(H\_OBJECT)

        $(CC) -c $(SRC\_OBJECT)

$(BIN\_OBJECT):$(DY\_OBJECT)

        $(CC) -o $(BIN\_OBJECT) $(OBJECT\_DIR)$(DY\_OBJECT) $(LIB\_DIR)$(LIB\_OBJECT)

        mv $(BIN\_OBJECT) $(BIN\_DIR)

$(DY\_OBJECT):$(DY\_SRC\_OBJECT)

        $(CC) -c $(DY\_SRC\_OBJECT) -o $(DY\_OBJECT)

        mv $(OBJECT) $(DY\_OBJECT) $(OBJECT\_DIR)

clean:

        $(RM) $(LIB\_DIR) $(BIN\_DIR) $(OBJECT\_DIR)

三．**对makefile文件的简单阐述：**

1. 路径变量赋值：

LIB\_DIR = /cbs/lhbb/hcj/dy/lib/          -----存放生成的静态库文件，.a文件；

BIN\_DIR = /cbs/lhbb/hcj/dy/bin/         -----存放生成的可执行文件；

OBJECT\_DIR = /cbs/lhbb/dy/test/obj/     -----存放生成的中间代码文件，.o文件；

APP\_DIR = /cbs/lhbb/hcj/dy/testapp/     -----存放源文件；

2.调用shell脚本，创建目录：

$(shell mkdir -p ${LIB\_DIR})

$(shell mkdir -p ${BIN\_DIR})

$(shell mkdir -p ${OBJECT\_DIR})

3.编译变量赋值：

RM = rm –fr

CC = gcc

AR = ar rc       -----生成静态库文件命令

4.文件变量赋值：

SRC\_OBJECT = $(APP\_DIR)test\_a.c $(APP\_DIR)test\_b.c $(APP\_DIR)test\_c.c

----源文件变量；

H\_OBJECT = $(APP\_DIR)so\_test.h

----头文件变量；

OBJECT = test\_a.o test\_b.o test\_c.o

----中间代码文件变量；

DY\_SRC\_OBJECT = $(APP\_DIR)test.c

----生成test可执行文件所依赖的源文件变量；

DY\_OBJECT=test.o

----编译test.c生成的中间代码文件变量；

LIB\_OBJECT = libtest.a

----静态库文件变量；

BIN\_OBJECT = test

----生成的可执行文件变量；

这个makefile中基本都是使用的变量，这样编译不同的工程的时候，就只要修改变量的值，编译的整体框架就不需要修改，当然，这个框架只是本人刚开始研究makefile写的，不够好，望大神吐槽，谢谢！

5.编译主体阐述：

.PHONY:all

----固定用.PHONY定义一个伪目标，伪目标不是文件，只是一个标签，没有依赖文件；这边定义了一个伪目标all；

all:$(LIB\_OBJECT) $(BIN\_OBJECT)

----目标all包含了两个需要生成的目标文件，一个是需要生成的静态库文件libtest.a，一个是最终生成的可执行文件test；但是为什么这边需要用一个all来包含两个目标文件呢？原因是通常情况下，系统只会去编译生成makefile文件开头最早一个目标文件，下面的目标文件是不会去执行编译的，所以我在这边将两个目标文件写在了一个伪目标all中，这样就可以实现生成两个目标文件；

生成静态库文件的过程：

$(LIB\_OBJECT):$(OBJECT)      -----目标文件：依赖文件    libtest.a：test\_a.o test\_b.o test\_c.o

        $(AR) $(LIB\_OBJECT) $(OBJECT)    ----编译生成静态库文件命令，特点就是需要使用

　　　　　　　　　　　　　　　　　　　　　　　　ar rc命令去编译；

        mv $(LIB\_OBJECT) $(LIB\_DIR)      ----将生成的文件移动到对应的文件夹；

$(OBJECT):$(SRC\_OBJECT) $(H\_OBJECT)

        $(CC) -c $(SRC\_OBJECT)

生成可执行文件test的过程：

$(BIN\_OBJECT):$(DY\_OBJECT)

        $(CC) -o $(BIN\_OBJECT) $(OBJECT\_DIR)$(DY\_OBJECT) $(LIB\_DIR)$(LIB\_OBJECT)

        -----将静态库文件包含在命令中；

        mv $(BIN\_OBJECT) $(BIN\_DIR)

$(DY\_OBJECT):$(DY\_SRC\_OBJECT)

        $(CC) -c $(DY\_SRC\_OBJECT) -o $(DY\_OBJECT)

        mv $(OBJECT) $(DY\_OBJECT) $(OBJECT\_DIR)

执行make clean删除生成的文件：

clean:

        $(RM) $(LIB\_DIR) $(BIN\_DIR) $(OBJECT\_DIR)

四．**注意点：**

1.命令行必须以tab键打头，否则会报错；

2.生成静态库文件用命令：ar rc；

3.库文件以lib打头，.a为后缀；

五．**执行结果：**

**bin文件夹**中存放可执行文件test，运行./test结果如下：

this is in test\_a...

this is in test\_b...

this is in test\_c...

lib文件夹下存放生成的静态库文件libtest.a；