Makefile进阶

开始准备

`template.c`

代码块

```
#include <stdio.h>
void fun()
{
    printf("hello fun!\n");
}
```

构建单目录多文件

`mkcode.sh`

代码块

```
#!/ bin / bash
num = 100 suffix =.c template = template.txt
                                    function
                                    mkcode()
{
    cnt = 0 while[$cnt - le $num] do sed "s/fun/fun${cnt}/" ${template} >
code${cnt} ${suffix};
    let cnt++;
    done
}
function mkmain()
{
    cnt = 0 > main.c printf "#include <stdio.h>\n\nint main()\n{\n" >> main.c
while[$cnt - le $num] do printf " fun${cnt}();\n" let cnt++ done >> main.c echo
 "}" >> main.c
}
mkcode mkmain
```

构建源文件

代码块

```
$ cat code99.c
#include <stdio.h>
    void fun99()
{
    printf("hello world!\n");
}
$ cat main.c
#include <stdio.h>
    int main()
{
    fun0();
    fun1();
    fun2();
    fun3();
    fun4();
    ...
}
```

场景1： 构建单模块，多文件项目

Step 1：显示当前目录下的源文件，并把他们的.c全部替换成为.o

`Makefile`

代码块

```
$ cat Makefile
#获取所有.c
    SRCS = $(wildcard *.c)
#转换.c变成.o
        OBJS = $(SRC :.c =.o)
#查看一下
            test : @echo "Src:"
                   @echo $(SRCS)
                       @echo "OBj:"
                   @echo $(OBJS)
```

要点：

· SRCS 使用了 wildcard 函数来自动获取当前目录下所有的 .c 文件

· 当你在 Makefile 中使用 wildcard 函数时，它会返回所有匹配给定模式的文件名列表，这些文件名是当前目录（或指定目录，如果函数参数中包含了路径）下的实际存在的文件。这

对于动态地确定需要编译的源文件列表特别有用，尤其是当源文件可能会随着项目的发展而增加或减少时。

· $(wildcard PATTERN...)

查看

代码块

```
$ make
Src:
code63.c code88.c code75.c code72.c code30.c code34.c code90.c code13.c
code4.c code53.c code76.c code6.c code99.c code80.c code11.c code100.c
code84.c code3.c code23.c code57.c code31.c code28.c code62.c code51.c
code26.c code95.c code49.c code16.c code55.c code18.c code65.c code67.c
code8.c code22.c code19.c code21.c code47.c code32.c code73.c code74.c
code69.c code15.c code87.c code36.c code92.c code44.c code56.c code71.c
code14.c code89.c code43.c code50.c code25.c code58.c code45.c code9.c
code33.c code1.c code64.c code93.c code98.c code20.c code66.c code83.c
code10.c code39.c code41.c code70.c code24.c code17.c code60.c code59.c
code61.c code27.c code86.c code37.c code85.c code54.c code5.c code46.c
code48.c code29.c code94.c code12.c code96.c code77.c code35.c code7.c
code40.c code42.c code97.c code68.c code78.c code52.c code91.c code79.c
code81.c code2.c main.c code82.c code38.c code0.c
OBjS:
code63.o code88.o code75.o code72.o code30.o code34.o code90.o code13.o
code4.o code53.o code76.o code6.o code99.o code80.o code11.o code100.o
code84.o code3.o code23.o code57.o code31.o code28.o code62.o code51.o
code26.o code95.o code49.o code16.o code55.o code18.o code65.o code67.o
code8.o code22.o code19.o code21.o code47.o code32.o code73.o code74.o
code69.o code15.o code87.o code36.o code92.o code44.o code56.o code71.o
code14.o code89.o code43.o code50.o code25.o code58.o code45.o code9.o
code33.o code1.o code64.o code93.o code98.o code20.o code66.o code83.o
code10.o code39.o code41.o code70.o code24.o code17.o code60.o code59.o
code61.o code27.o code86.o code37.o code85.o code54.o code5.o code46.o
code48.o code29.o code94.o code12.o code96.o code77.o code35.o code7.o
code40.o code42.o code97.o code68.o code78.o code52.o code91.o code79.o
code81.o code2.o main.o code82.o code38.o code0.o
```

Step 2：所有源文件，编译成为目标文件

更改 `Makefile`

代码块

```
#引入编译器
```

```
CC = gcc
#定义编译选项
    FLAGS = -c
#获取所有.c
                SRCS = $(wildcard *.c)
#转换.c变成.o
        OBJS = $(SRCS :.c =.o)
#定义目标，让Makefile进行推导
                        .PHONY : all
                                    all : $(OBJS) %
                        .o : %.c @echo "compling ... $<" @$(CC) $(FLAGS) $ <
                -o $ @
#清理项目
                        .PHONY : clean
                                    clean : @echo "clean ... Project"
                                            @echo "$(OBJS)"
                                            @rm -
                    f $(OBJS)
```

查看结果

代码块

```
$ make
compling ... code51.c
compling ... code33.c
compling ... code35.c
compling ... code18.c
compling ... code19.c
compling ... code38.c
compling ... code40.c
compling ... code77.c
compling ... code9.c
compling ... code65.c
compling ... code42.c
...

$ make clean
clean ... Project
code98.o code86.o code97.o code55.o code39.o code81.o code53.o code38.o
code76.o code5.o code6.o code90.o code80.o code8.o code3.o code48.o code47.o
code54.o code57.o code31.o code28.o code62.o code51.o code22.o code26.o
code67.o code95.o code16.o code100.o code18.o code65.o code99.o code42.o
code72.o code19.o code32.o code73.o code82.o code78.o code63.o code69.o
code71.o code15.o code87.o code36.o code92.o code46.o code56.o code21.o
code14.o code89.o code25.o code58.o code45.o code9.o code33.o code23.o
```

```
code1.o code64.o code93.o code20.o code66.o code27.o code83.o code10.o
code11.o code41.o code70.o code24.o code60.o code94.o code61.o code37.o
code59.o code29.o code44.o code7.o code12.o code96.o code4.o code35.o
code40.o code77.o code49.o code84.o code17.o code13.o code30.o code68.o
code52.o code91.o code79.o code74.o code50.o code2.o main.o code85.o code0.o
code43.o code88.o code75.o code34.o
```

Step 3：编译成为可执行程序

更改 `Makefile`

代码块

```
# 引入编译器
CC=gcc

# 定义编译选项
FLAGS=-c

# 获取所有.c
SRCS=$(wildcard *.c)

# 转换.c变成.o
OBJS=$(SRCS:.c=.o)

# 定义目标可执行
BIN=project

$(BIN):$(OBJS)
    @echo "Link *.o to $(BIN)"
    @$(CC) $^ -o $@
%.o:%.c
    @echo "compling ... $<"
    @$(CC) $(FLAGS) $< -o $@

# 清理项目
.PHONY:clean
clean:
    @echo "clean ... Project"
    @echo "$(OBJS)"
    @rm -f $(OBJS)
```

查看结果

代码块

```
$ make
...
compling ... code82.c
compling ... code55.c
compling ... code29.c
compling ... code44.c
compling ... code33.c
compling ... code12.c
compling ... code96.c
compling ... code7.c
compling ... code49.c
Link *.o to project
$ ./project
hello fun0
hello fun1
hello fun2
...
```

场景2：临时文件、目标可执行与源代码分离

我们编译的时候，会形成很多的.o临时文件，还有可执行程序(目前是一个),这么多的临时文件，和源文件放在一起，就显得非常不优雅，我们想在make的时候，直接将目标.o和可执行程序分别在不同的目录下，这样后期在清理的之后，只要删除目录，就可以去掉所有的临时文件了

更改 `Makefile`

代码块

```
# 引入编译器
CC=gcc

# 添加其他命令
RM=rm
RM_FLAGS=-rf
MKDIR=mkdir

# 定义编译选项
FLAGS=-c

# 获取所有.c
SRCS=$(wildcard *.c)

# 转换.c变成.o
OBJS=$(SRCS:.c=.o)

# 定义目标可执行
```

```
BIN=project

# 添加新建的目录名称
OBJS_DIR=objs
BIN_DIR=bin

# 给最终形成的可执行程序添加路径
# 这里必须使用:=,要不然makefile会有递归引用的问题.
BIN:=$(addprefix $(BIN_DIR)/, $(BIN)) # bin/project

# 给形成的临时.o都添加目标路径
OBJS:=$(addprefix $(OBJS_DIR)/, $(OBJS)) #objs/XX.o

# 需要新建的文件夹
DIR_LIST=$(OBJS_DIR) $(BIN_DIR)
all:$(DIR_LIST) $(BIN)
$(DIR_LIST):
    @$(MKDIR) $@
$(BIN):$(OBJS)
    @echo "Link *.o to $(BIN)"
    @$(CC) $^ -o $@

# 这里要添加路径,保证形成的.o放入指定路径下
$(OBJS_DIR)/%.o:%.c
    @echo "compling ... $<"
    @$(CC) $(FLAGS) $< -o $@

.PHONY:clean
clean:
    @echo "clean ... Project"
    @$(RM) $(RM_FLAGS) $(DIR_LIST)
```

运行结果

代码块

```
$ make
compling ... code51.c
compling ... code35.c
compling ... code18.c
compling ... code19.c
compling ... code38.c
...
$ tree bin
bin
└── project
```

```
$ tree objs
objs
├── code0.o
├── code100.o
├── code10.o
├── code11.o
├── code12.o
├── code13.o
├── code14.o
├── code15.o
├── code16.o
├── code17.o
├── code18.o
├── code19.o
├── code1.o
├── code20.o
├── code21.o
├── code22.o
├── code23.o
...
```

= 和 :=

=进行赋值，变量的值是整个makefile中最后被指定的值, 在make时，会把整个makefile展开，拉通决

定变量的值

代码块

```
a = hello
b = $(a) bit
a = world

#a = hello
#b := $(a) bit
#a = world

all:
    @echo $(a)
    @echo $(b)
$ make
world
world bit
```

”:=”就表示直接赋值，赋予当前位置的值,”:=”才是真正意义上的直接赋值

代码块

```
a = hello
b := $(a) bit
a = world

all:
    @echo $(a)
    @echo $(b)
$ make
world
hello bit
```

✏️ 📌 `$(addprefix prefix,names…)` 是一个函数，用于将指定的前缀添加到一组空格分隔

的文件名中。这个函数通常用于将相同的前缀添加到一组文件名或路径中，非常适合在

Makefile 中进行路径拼接操作。

- prefix：要添加的前缀。
- namesR：一组空格分隔的文件名或路径。

场景3：构建多模块，单可执行文件项目

代码块

```
#!/ bin / bash
num = 100 suffix =.c template = template.txt
                                    function
                                    mkcode()
{
    cnt = 1 while[$cnt - le $num] do sed "s/fun/fun${cnt}/" ${template} >
code${cnt} ${suffix};
    let cnt++;
    done
}
function mkmain()
{
    cnt = 1 > main.c printf "#include <stdio.h>\n\nint main()\n{\n" >> main.c
while[$cnt - le $num] do printf " fun${cnt}();\n" let cnt++ done >> main.c echo
 "}" >> main.c
}
function mkmodule()
{
```

```
    i = 0 j = 1 for ((; i < 10; i++)) do((end = j + 10))
                printf "build Module%d\n" $i
                    mkdir -
            p Module$i for ((; j < end; j++)) do mv code${j}.c Module$i
#mv Module$i / code${j }.c.#恢复文件结构
                done
                    done
                        mkdir -
            p main
                mv main.c main
}
function clean(){
    rm - f *.c rm - rf Module * rm - rf main} function Usage(){
    printf "Usage: $0 [-mkcode/-mkmain/-mkmodule/-clr]\n"} function main()
{
    if
        [$ # - ne 1];
    then
        Usage return fi
            opt = $1 if[$opt == "-mkcode"];
    then
        mkcode
            elif[$opt == "-mkmain"];
    then
        mkmain
            elif[$opt == "-mkmodule"];
    then
        mkmodule
            elif[$opt == "-clr"];
    then
        clean else Usage fi
}
main $ @
```

准备模块

代码块

```
$ ./mkcode.sh -mkcode
$ ./mkcode.sh -mkmain
$ ./mkcode.sh -mkmodule
build Module0
build Module1
build Module2
build Module3
build Module4
```

```
build Module5
build Module6
build Module7
build Module8
build Module9
$ ll
total 60
drwxrwxr-x 2 whb whb 4096 Sep 24 15:13 main
-rwxrwxr-x 1 whb whb 1258 Sep 24 13:56 mkcode.sh
drwxrwxr-x 2 whb whb 4096 Sep 24 15:13 Module0
drwxrwxr-x 2 whb whb 4096 Sep 24 15:13 Module1
drwxrwxr-x 2 whb whb 4096 Sep 24 15:13 Module2
drwxrwxr-x 2 whb whb 4096 Sep 24 15:13 Module3
drwxrwxr-x 2 whb whb 4096 Sep 24 15:13 Module4
drwxrwxr-x 2 whb whb 4096 Sep 24 15:13 Module5
drwxrwxr-x 2 whb whb 4096 Sep 24 15:13 Module6
drwxrwxr-x 2 whb whb 4096 Sep 24 15:13 Module7
drwxrwxr-x 2 whb whb 4096 Sep 24 15:13 Module8
drwxrwxr-x 2 whb whb 4096 Sep 24 15:13 Module9
-rw-rw-r-- 1 whb whb 60 Sep 24 13:06 template.txt
```

设计 `Makefile`

代码块

```
#main子模块名称
MODULES := main
#获取其他模块的名称
MODULES += $(shell ls -d Module*)

#形成目标文件的名称
BIN := project
RMF := rm -f

#循环获取指定模块下的所有.c源文件
SRCS := $(foreach module, $(MODULES), $(wildcard $(module)/*.c))

# 将所有的.c换成.o
OBJS := $(SRCS:.c=.o)

# 基本命令
CC=gcc

# 编译目标程序
$(BIN):$(OBJS)
    @$(CC) $^ -o $@
```

```
    @echo "链接 $^ 形成 $@"

# 形成目标文件
%.o:%.c
    @$(CC) -c $< -o $@
    @echo "编译$< 形成 $@"

# 清理项目
.PHONY:clean
clean:
    @$(RMF) $(OBJS) $(BIN)
    @echo "清理 $(OBJS) $(BIN)"

# 测试，查看测试结果
.PHONY:test
test:
    @echo $(MODULES)
    @echo "-----------------------"
    @echo $(SRCS)
    @echo "-----------------------"
    @echo $(OBJS)
    @echo "-----------------------"
```

编译

代码块

```
$ make
编译main/main.c 形成 main/main.o
编译Module0/code8.c 形成 Module0/code8.o
编译Module0/code3.c 形成 Module0/code3.o
编译Module0/code5.c 形成 Module0/code5.o
编译Module0/code2.c 形成 Module0/code2.o
...
编译Module9/code97.c 形成 Module9/code97.o
编译Module9/code93.c 形成 Module9/code93.o
编译Module9/code98.c 形成 Module9/code98.o
编译Module9/code96.c 形成 Module9/code96.o
链接 main/main.o Module0/code8.o Module0/code3.o Module0/code5.o
Module0/code2.o Module0/code7.o Module0/code10.o Module0/code4.o
Module0/code9.o Module0/code1.o Module0/code6.o Module1/code11.o
Module1/code17.o Module1/code12.o Module1/code16.o Module1/code18.o
Module1/code15.o Module1/code14.o Module1/code13.o Module1/code19.o
Module1/code20.o Module2/code24.o Module2/code27.o Module2/code23.o
Module2/code28.o Module2/code29.o Module2/code21.o Module2/code22.o
Module2/code30.o Module2/code25.o Module2/code26.o Module3/code39.o
```

```
Module3/code37.o Module3/code31.o Module3/code35.o Module3/code40.o
Module3/code32.o Module3/code36.o Module3/code34.o Module3/code38.o
Module3/code33.o Module4/code41.o Module4/code45.o Module4/code42.o
Module4/code48.o Module4/code44.o Module4/code49.o Module4/code43.o
Module4/code46.o Module4/code50.o Module4/code47.o Module5/code58.o
Module5/code59.o Module5/code57.o Module5/code51.o Module5/code60.o
Module5/code52.o Module5/code54.o Module5/code56.o Module5/code53.o
Module5/code55.o Module6/code70.o Module6/code61.o Module6/code64.o
Module6/code62.o Module6/code67.o Module6/code65.o Module6/code68.o
Module6/code63.o Module6/code69.o Module6/code66.o Module7/code73.o
Module7/code80.o Module7/code77.o Module7/code78.o Module7/code79.o
Module7/code74.o Module7/code71.o Module7/code75.o Module7/code72.o
Module7/code76.o Module8/code85.o Module8/code90.o Module8/code86.o
Module8/code89.o Module8/code81.o Module8/code87.o Module8/code88.o
Module8/code82.o Module8/code84.o Module8/code83.o Module9/code100.o
Module9/code95.o Module9/code94.o Module9/code99.o Module9/code91.o
Module9/code92.o Module9/code97.o Module9/code93.o Module9/code98.o
Module9/code96.o 形成 project
```

查看编译结果

代码块

```
tree .
.
├── main
│   ├── main.c
│   └── main.o
├── Makefile
├── mkcode.sh
├── Module0
│   ├── code10.c
│   ├── code10.o
│   ├── code1.c
│   ├── code1.o
│   ├── code2.c
│   ├── code2.o
│   ├── code3.c
│   ├── code3.o
│   ├── code4.c
│   ├── code4.o
...
```

注意:

可以看到，源文件和obj文件会在同一个目录下，这个就是 %.o:%.c 推导出来的

📌 foreach语法介绍

//TODO

场景4：构建多模块，单可执行文件项目，文件分离

改写 `makefile`

代码块

```
# main子模块名称
MODULES := main
# 获取其他模块的名称
MODULES += $(shell ls -d Module*)

# 形成目标文件的名称
BIN := project
RMF := rm -f
MKDIR := mkdir
# 循环获取指定模块下的所有.c源文件
SRCS := $(foreach module, $(MODULES), $(wildcard $(module)/*.c))

# 将所有的.c换成.o
# OBJS := $(SRCS:.c=.o)

# 给OBJS添加临时目录
OBJS_DIR=objs
OBJS := $(patsubst %.c, $(OBJS_DIR)/%.o, $(SRCS))

# 基本命令
CC=gcc

# 编译目标程序
$(BIN):$(OBJS)
    @$(CC) $^ -o $@
    @echo "连接 $^ 形成 $@"
# 形成目标文件
$(OBJS_DIR)/%.o:%.c
    @$(MKDIR) -p $(dir $@)
    @$(CC) -c $< -o $@
    @echo "编译$< 形成 $@"

# 清理项目
```

```
.PHONY:clean
clean:
    @$(RMF) $(OBJS) $(BIN)
    @echo "清理 $(OBJS) $(BIN)"

# 测试，查看测试结果
.PHONY:test
test:
    @echo $(MODULES)
    @echo "----------------------"
    @echo $(SRCS)
    @echo "----------------------"
    @echo $(OBJS)
    @echo "----------------------"
```

编译

代码块

```
$ make
编译main/main.c 形成 objs/main/main.o
编译Module0/code8.c 形成 objs/Module0/code8.o
编译Module0/code3.c 形成 objs/Module0/code3.o
编译Module0/code5.c 形成 objs/Module0/code5.o
编译Module0/code2.c 形成 objs/Module0/code2.o
编译Module0/code7.c 形成 objs/Module0/code7.o
编译Module8/code82.c 形成 objs/Module8/code82.o
编译Module8/code84.c 形成 objs/Module8/code84.o
编译Module8/code83.c 形成 objs/Module8/code83.o
编译Module9/code100.c 形成 objs/Module9/code100.o
编译Module9/code95.c 形成 objs/Module9/code95.o
编译Module9/code94.c 形成 objs/Module9/code94.o
编译Module9/code99.c 形成 objs/Module9/code99.o
编译Module9/code91.c 形成 objs/Module9/code91.o
编译Module9/code92.c 形成 objs/Module9/code92.o
编译Module9/code97.c 形成 objs/Module9/code97.o
编译Module9/code93.c 形成 objs/Module9/code93.o
编译Module9/code98.c 形成 objs/Module9/code98.o
编译Module9/code96.c 形成 objs/Module9/code96.o
连接 objs/main/main.o objs/Module0/code8.o objs/Module0/code3.o
objs/Module0/code5.o objs/Module0/code2.o objs/Module0/code7.o
objs/Module0/code10.o objs/Module0/code4.o objs/Module0/code9.o
objs/Module0/code1.o objs/Module0/code6.o objs/Module1/code11.o
objs/Module1/code17.o objs/Module1/code12.o objs/Module1/code16.o
objs/Module1/code18.o objs/Module1/code15.o objs/Module1/code14.o
objs/Module1/code13.o objs/Module1/code19.o objs/Module1/code20.o
```

```
objs/Module2/code24.o objs/Module2/code27.o objs/Module2/code23.o
objs/Module2/code28.o objs/Module2/code29.o objs/Module2/code21.o
objs/Module2/code22.o objs/Module2/code30.o objs/Module2/code25.o
objs/Module2/code26.o objs/Module3/code39.o objs/Module3/code37.o
objs/Module3/code31.o objs/Module3/code35.o objs/Module3/code40.o
objs/Module3/code32.o objs/Module3/code36.o objs/Module3/code34.o
objs/Module3/code38.o objs/Module3/code33.o objs/Module4/code41.o
objs/Module4/code45.o objs/Module4/code42.o objs/Module4/code48.o
objs/Module4/code44.o objs/Module4/code49.o objs/Module4/code43.o
objs/Module4/code46.o objs/Module4/code50.o objs/Module4/code47.o
objs/Module5/code58.o objs/Module5/code59.o objs/Module5/code57.o
objs/Module5/code51.o objs/Module5/code60.o objs/Module5/code52.o
objs/Module5/code54.o objs/Module5/code56.o objs/Module5/code53.o
objs/Module5/code55.o objs/Module6/code70.o objs/Module6/code61.o
objs/Module6/code64.o objs/Module6/code62.o objs/Module6/code67.o
objs/Module6/code65.o objs/Module6/code68.o objs/Module6/code63.o
objs/Module6/code69.o objs/Module6/code66.o objs/Module7/code73.o
objs/Module7/code80.o objs/Module7/code77.o objs/Module7/code78.o
objs/Module7/code79.o objs/Module7/code74.o objs/Module7/code71.o
objs/Module7/code75.o objs/Module7/code72.o objs/Module7/code76.o
objs/Module8/code85.o objs/Module8/code90.o objs/Module8/code86.o
objs/Module8/code89.o objs/Module8/code81.o objs/Module8/code87.o
objs/Module8/code88.o objs/Module8/code82.o objs/Module8/code84.o
objs/Module8/code83.o objs/Module9/code100.o objs/Module9/code95.o
objs/Module9/code94.o objs/Module9/code99.o objs/Module9/code91.o
objs/Module9/code92.o objs/Module9/code97.o objs/Module9/code93.o
objs/Module9/code98.o objs/Module9/code96.o 形成 project
```

运行

代码块

```
$ ./project
hello fun1
hello fun2
hello fun3
hello fun4
hello fun5
...
```

查看形成结构

代码块

```
$ tree objs
objs
```

```
├── main
│   └── main.o
├── Module0
│   ├── code10.o
│   ├── code1.o
│   ├── code2.o
│   ├── code3.o
│   ├── code4.o
│   ├── code5.o
│   ├── code6.o
│   ├── code7.o
│   ├── code8.o
│   └── code9.o
├── Module1
│   ├── code11.o
│   ├── code12.o
│   ├── code13.o
│   ├── code14.o
│   ├── code15.o
│   ├── code16.o
│   ├── code17.o
│   ├── code18.o
│   ├── code19.o
│   └── code20.o
├── Module2
│   ├── code21.o
│   ├── code22.o
│   ├── code23.o
│   ├── code24.o
│   ├── code25.o
│   ├── code26.o
│   ├── code27.o
│   ├── code28.o
│   ├── code29.o
...
```

这样，所有的临时文件都在同一个临时目录下，而且目录结构也一致

清理项目

代码块

```
$ make clean
清理 objs/main/main.o objs/Module0/code8.o objs/Module0/code3.o
objs/Module0/code5.o objs/Module0/code2.o objs/Module0/code7.o
objs/Module0/code10.o objs/Module0/code4.o objs/Module0/code9.o
objs/Module0/code1.o objs/Module0/code6.o objs/Module1/code11.o
```

```
objs/Module1/code17.o objs/Module1/code12.o objs/Module1/code16.o
objs/Module1/code18.o objs/Module1/code15.o objs/Module1/code14.o
objs/Module1/code13.o objs/Module1/code19.o objs/Module1/code20.o
objs/Module2/code24.o objs/Module2/code27.o objs/Module2/code23.o
objs/Module2/code28.o objs/Module2/code29.o objs/Module2/code21.o
objs/Module2/code22.o objs/Module2/code30.o objs/Module2/code25.o
objs/Module2/code26.o objs/Module3/code39.o objs/Module3/code37.o
objs/Module3/code31.o objs/Module3/code35.o objs/Module3/code40.o
objs/Module3/code32.o objs/Module3/code36.o objs/Module3/code34.o
objs/Module3/code38.o objs/Module3/code33.o objs/Module4/code41.o
objs/Module4/code45.o objs/Module4/code42.o objs/Module4/code48.o
objs/Module4/code44.o objs/Module4/code49.o objs/Module4/code43.o
objs/Module4/code46.o objs/Module4/code50.o objs/Module4/code47.o
objs/Module5/code58.o objs/Module5/code59.o objs/Module5/code57.o
objs/Module5/code51.o objs/Module5/code60.o objs/Module5/code52.o
objs/Module5/code54.o objs/Module5/code56.o objs/Module5/code53.o
objs/Module5/code55.o objs/Module6/code70.o objs/Module6/code61.o
objs/Module6/code64.o objs/Module6/code62.o objs/Module6/code67.o
objs/Module6/code65.o objs/Module6/code68.o objs/Module6/code63.o
objs/Module6/code69.o objs/Module6/code66.o objs/Module7/code73.o
objs/Module7/code80.o objs/Module7/code77.o objs/Module7/code78.o
objs/Module7/code79.o objs/Module7/code74.o objs/Module7/code71.o
objs/Module7/code75.o objs/Module7/code72.o objs/Module7/code76.o
objs/Module8/code85.o objs/Module8/code90.o objs/Module8/code86.o
objs/Module8/code89.o objs/Module8/code81.o objs/Module8/code87.o
objs/Module8/code88.o objs/Module8/code82.o objs/Module8/code84.o
objs/Module8/code83.o objs/Module9/code100.o objs/Module9/code95.o
objs/Module9/code94.o objs/Module9/code99.o objs/Module9/code91.o
objs/Module9/code92.o objs/Module9/code97.o objs/Module9/code93.o
objs/Module9/code98.o objs/Module9/code96.o project
```

查看清理之后的结构

代码块

```
$ tree objs
objs
├── main
├── Module0
├── Module1
├── Module2
├── Module3
├── Module4
├── Module5
├── Module6
├── Module7
```

```
├── Module8
└── Module9
```

patsubst 语法

场景5: 引入头文件

自动生成头文件

代码块

```
#!/ bin / bash
num = 100 suffix =.c
                      include =.h template = template.txt
                                                 function
                                                 mkinclude()
{
    cnt = 1 while[$cnt - le $num] do echo "#pragma once" >> code${cnt}
${include} echo "#include<stdio.h>" > code${cnt} ${include} echo "void
fun${cnt}();" >> code${cnt} ${include} let cnt++ done
}
function mkcode()
{
    cnt = 1 while[$cnt - le $num] do sed "s/fun/fun${cnt}/" ${template} >
code${cnt} ${suffix};
    let cnt++;
    done
}
function mkmain()
{
    cnt = 1 > main.c printf "#include <stdio.h>\n" > main.c while[$cnt - le
$num] do printf "#include \"code$cnt.h\"\n" >> main.c

                                               let cnt++;
    done
        cnt = 1 printf "int main()\n{\n" >> main.c while[$cnt - le $num] do
printf " fun${cnt}();\n" let cnt++ done >> main.c echo "}" >> main.c
}
function mkmodule()
{
    i = 0 j = 1 for ((; i < 10; i++)) do((end = j + 10))
                printf "build Module%d\n" $i
                    mkdir -
              p Module$i for ((; j < end; j++)) do mv code${j}.c Module$i
```

```
                    mv code${j}
                        .h Module$i
#mv Module$i / code${j }.c.#恢复文件结构
                            done
                                done
                                    mkdir -
                p main
                    mv main.c main
}
function clean(){
    rm - f *.c rm - f *.h rm - rf Module * rm - rf main} function Usage(){
    printf "Usage: $0 [-mkcode/-mkinclude/-mkmain/-mkmodule/-clr]\n"} function
main()
{
    if
        [$ # - ne 1];
    then
        Usage return fi
            opt = $1 if[$opt == "-mkcode"];
    then
        mkcode
            elif[$opt == "-mkinclude"];
    then
        mkinclude
            elif[$opt == "-mkmain"];
    then
        mkmain
            elif[$opt == "-mkmodule"];
    then
        mkmodule
            elif[$opt == "-clr"];
    then
        clean else Usage fi
}
main $ @
```

代码块

```
$ ./mkcode.sh -mkcode
$ ./mkcode.sh -mkinclude
$ ./mkcode.sh -mkmain
$ ./mkcode.sh -mkmodule
```

更改makefile

```
# main子模块名称
MODULES := main
# 获取其他模块的名称
MODULES += $(shell ls -d Module*)

# 形成目标文件的名称
BIN := project
RMF := rm -f
MKDIR := mkdir

# 循环获取指定模块下的所有.c源文件
SRCS := $(foreach module, $(MODULES), $(wildcard $(module)/*.c))
INCLUDES :=$(addprefix -I, $(MODULES))

# 将所有的.c换成.o
# OBJS := $(SRCS:.c=.o)

# 给OBJS添加临时目录
OBJS_DIR=objs
OBJS := $(patsubst %.c, $(OBJS_DIR)/%.o, $(SRCS))

# 基本命令
CC=gcc

# 编译目标程序
$(BIN):$(OBJS)
    @$(CC) $^ -o $@
    @echo "连接 $^ 形成 $@"

# 形成目标文件
$(OBJS_DIR)/%.o:%.c
    @$(MKDIR) -p $(dir $@)
    @$(CC) -c $< -o $@ $(INCLUDES)
    @echo "编译$< 形成 $@"

# 清理项目
.PHONY:clean
clean:
    @$(RMF) $(OBJS) $(BIN)
    @echo "清理 $(OBJS) $(BIN)"

# 测试，查看测试结果
.PHONY:test
test:
    @echo $(MODULES)
    @echo "----------------------"
    @echo $(SRCS)
```

```
48      @echo "----------------------"
49      @echo $(OBJS)
50      @echo "----------------------"
51      @echo $(INCLUDES)
52      @echo $(INCS)
53      @echo "----------------------"
```

编译

代码块

```
1   $ make
2   编译main/main.c 形成 objs/main/main.o
3   编译Module0/code3.c 形成 objs/Module0/code3.o
4   ...
5   编译Module9/code93.c 形成 objs/Module9/code93.o
6   编译Module9/code98.c 形成 objs/Module9/code98.o
7   编译Module9/code96.c 形成 objs/Module9/code96.o
8   连接 objs/main/main.o objs/Module0/code3.o objs/Module0/code5.o
9   objs/Module0/code2.o objs/Module0/code7.o objs/Module0/code10.o
10  objs/Module0/code8.o objs/Module0/code4.o objs/Module0/code9.o
11  objs/Module0/code1.o objs/Module0/code6.o objs/Module1/code11.o
12  objs/Module1/code17.o objs/Module1/code12.o objs/Module1/code16.o
13  objs/Module1/code18.o objs/Module1/code15.o objs/Module1/code14.o
14  objs/Module1/code13.o objs/Module1/code19.o objs/Module1/code20.o
15  objs/Module2/code24.o objs/Module2/code27.o objs/Module2/code23.o
16  objs/Module2/code28.o objs/Module2/code29.o objs/Module2/code21.o
17  objs/Module2/code22.o objs/Module2/code30.o objs/Module2/code25.o
18  objs/Module2/code26.o objs/Module3/code39.o objs/Module3/code37.o
19  objs/Module3/code31.o objs/Module3/code35.o objs/Module3/code40.o
20  objs/Module3/code32.o objs/Module3/code36.o objs/Module3/code34.o
21  objs/Module3/code38.o objs/Module3/code33.o objs/Module4/code41.o
22  objs/Module4/code45.o objs/Module4/code42.o objs/Module4/code48.o
23  objs/Module4/code44.o objs/Module4/code49.o objs/Module4/code43.o
24  objs/Module4/code46.o objs/Module4/code50.o objs/Module4/code47.o
25  objs/Module5/code58.o objs/Module5/code59.o objs/Module5/code57.o
26  objs/Module5/code51.o objs/Module5/code60.o objs/Module5/code52.o
27  objs/Module5/code54.o objs/Module5/code56.o objs/Module5/code53.o
28  objs/Module5/code55.o objs/Module6/code70.o objs/Module6/code61.o
29  objs/Module6/code64.o objs/Module6/code62.o objs/Module6/code67.o
30  objs/Module6/code65.o objs/Module6/code68.o objs/Module6/code63.o
31  objs/Module6/code69.o objs/Module6/code66.o objs/Module7/code73.o
32  objs/Module7/code74.o objs/Module7/code80.o objs/Module7/code77.o
33  objs/Module7/code78.o objs/Module7/code72.o objs/Module7/code71.o
34  objs/Module7/code79.o objs/Module7/code75.o objs/Module7/code76.o
35  objs/Module8/code85.o objs/Module8/code90.o objs/Module8/code86.o
```

```
objs/Module8/code89.o objs/Module8/code81.o objs/Module8/code87.o
objs/Module8/code88.o objs/Module8/code82.o objs/Module8/code84.o
objs/Module8/code83.o objs/Module9/code100.o objs/Module9/code95.o
objs/Module9/code94.o objs/Module9/code99.o objs/Module9/code91.o
objs/Module9/code92.o objs/Module9/code97.o objs/Module9/code93.o
objs/Module9/code98.o objs/Module9/code96.o 形成 project
```

代码块

```
$ tree Module0
Module0
├── code10.c
├── code10.h
├── code1.c
├── code1.h
├── code2.c
├── code2.h
├── code3.c
├── code3.h
├── code4.c
├── code4.h
├── code5.c
├── code5.h
├── code6.c
├── code6.h
├── code7.c
├── code7.h
├── code8.c
├── code8.h
├── code9.c
└── code9.h
```

场景6：多 Makefile ，多可执行程序

可以把之前的内容看做项目的一个部分，拷贝形成多个部分，也可以用如下脚本自动形成:

`mkcode.sh`

代码块

```
$ cat mkcode.sh
#!/ bin / bash
    num = 100 suffix =.c
                          include =.h template = template.txt
                                                    function mkinclude()
{
```

```
7       cnt = 1 while[$cnt - le $num] do echo "#pragma once" >> code${cnt}
    ${include} echo "#include<stdio.h>" > code${cnt} ${include} echo "void
    fun${cnt}();" >> code${cnt} ${include} let cnt++ done
8   }
9   function mkcode()
10  {
11      cnt = 1 while[$cnt - le $num] do sed "s/fun/fun${cnt}/" ${template} >
    code${cnt} ${suffix};
12      let cnt++;
13      done
14  }
15  function mkmain()
16  {
17      cnt = 1 > main.c printf "#include <stdio.h>\n" > main.c while[$cnt - le
    $num] do printf "#include \"code$cnt.h\"\n" >> main.c
18
                                                  let cnt++;
19      done
20          cnt = 1 printf "int main()\n{\n" >> main.c while[$cnt - le $num] do
    printf " fun${cnt}();\n" let cnt++ done >> main.c echo "}" >> main.c
21  }
22  function mkmodule()
23  {
24      i = 0 j = 1 for ((; i < 10; i++)) do((end = j + 10))
25                      printf "build Module%d\n" $i
26                          mkdir -
27                  p Module$i for ((; j < end; j++)) do mv code${j}.c Module$i
28                      mv code${j}
29                          .h Module$i
30  #mv Module$i / code${j }.c.#恢复文件结构
31                              done
32                                  done
33                                      mkdir -
34                  p main
35                      mv main.c main
36  }
37  #根据模版makefile，在各个目录下形成makefile
38  function mkfile()
39  {
40      i = 0 for ((; i < 10; i++)) do sed "s/project/project${i}/g" subMakefile >
    Module$i / Makefile
41  #构建每一个模块的测试main.c
42
        echo "#include <stdio.h>" >
43          Module$i / main.c echo "int main() {}" >> Module$i / main.c done
44  }
45  function clean(){
```

```
46      rm - f *.c rm - f *.h rm - rf Module * rm - rf main} function Usage(){
47      printf "Usage: $0 [-mkcode/-mkinclude/-mkmain/-mkmodule/-clr/-mkfile]\n"}
    function main()
48  {
49      if
50          [$ # - ne 1];
51      then
52          Usage return fi
53              opt = $1 if[$opt == "-mkcode"];
54      then
55          mkcode
56              elif[$opt == "-mkinclude"];
57      then
58          mkinclude
59              elif[$opt == "-mkmain"];
60      then
61          mkmain
62              elif[$opt == "-mkmodule"];
63      then
64          mkmodule
65              elif[$opt == "-mkfile"];
66      then
67          mkfile
68              elif[$opt == "-clr"];
69      then
70          clean else Usage fi
71  }
72  main $ @
```

样板 `subMakefile`

代码块

```
1   BIN := project
2   CC := gcc
3   SRCS := $(wildcard *.c)
4   OBJS := $(SRCS:.c=.o)
5
6   $(BIN):$(OBJS)
7       $(CC) -o $@ $^
8   *.o:%.c
9       $(CC) -c $<
10
11  .PHONY:clean
12  clean:
13      rm -f *.o project
```

```
14
15  .PHONY:output
16  output:
17      mkdir -p ../bin
18      mv project ../bin
19      #mv project ../
```

构建好之后，每一个模块都可以独立编译

代码块

```
1   $ tree Module0/
2   Module0/
3   ├── code10.c
4   ├── code10.h
5   ├── code1.c
6   ├── code1.h
7   ├── code2.c
8   ├── code2.h
9   ├── code3.c
10  ├── code3.h
11  ├── code4.c
12  ├── code4.h
13  ├── code5.c
14  ├── code5.h
15  ├── code6.c
16  ├── code6.h
17  ├── code7.c
18  ├── code7.h
19  ├── code8.c
20  ├── code8.h
21  ├── code9.c
22  ├── code9.h
23  ├── main.c
24  └── Makefile
```

整体文件目录结构

代码块

```
1   $ ll
2   total 56
3   -rw-rw-r-- 1 whb whb 426 Oct 8 16:10 Makefile
4   -rwxrwxr-x 1 whb whb 2145 Oct 8 15:57 mkcode.sh
5   drwxrwxr-x 2 whb whb 4096 Oct 8 16:12 Module0
6   drwxrwxr-x 2 whb whb 4096 Oct 8 16:12 Module1
```

```
drwxrwxr-x 2 whb whb 4096 Oct 8 16:12 Module2
drwxrwxr-x 2 whb whb 4096 Oct 8 16:12 Module3
drwxrwxr-x 2 whb whb 4096 Oct 8 16:12 Module4
drwxrwxr-x 2 whb whb 4096 Oct 8 16:12 Module5
drwxrwxr-x 2 whb whb 4096 Oct 8 16:12 Module6
drwxrwxr-x 2 whb whb 4096 Oct 8 16:12 Module7
drwxrwxr-x 2 whb whb 4096 Oct 8 16:12 Module8
drwxrwxr-x 2 whb whb 4096 Oct 8 16:12 Module9
-rw-rw-r-- 1 whb whb 240 Oct 8 16:11 subMakefile
-rw-rw-r-- 1 whb whb 60 Sep 24 17:06 template.txt
```

Makefile结构

代码块

```
$ find . -name Makefile
./Module0/Makefile
./Module3/Makefile
./Module7/Makefile
./Module8/Makefile
./Module5/Makefile
./Module1/Makefile
./Module2/Makefile
./Module6/Makefile
./Module9/Makefile
./Module4/Makefile
./Makefile
$ find . -name main.c
./Module0/main.c
./Module3/main.c
./Module7/main.c
./Module8/main.c
./Module5/main.c
./Module1/main.c
./Module2/main.c
./Module6/main.c
./Module9/main.c
./Module4/main.c
```

顶级makefile内容

代码块

```
# 获取其他模块的名称
MODULES += $(shell ls -d Module*)

```

```
.PHONY: build
build:
    @for module in $(MODULES); do \
        (cd $$module && make) || exit 1; \
        echo "build $$module ... done";\
    done

.PHONY: clean
clean:
    @for module in $(MODULES); do \
        (cd $$module && make clean); \
        echo "clean $$module ... done";\
    done

.PHONY: output
output:
    @for module in $(MODULES); do \
        (cd $$module && make output); \
    done
```

编译

代码块

```
$ make
make[1]: Entering directory `/home/whb/mkfile/Module0'
gcc -c -o code3.o code3.c
gcc -c -o code5.o code5.c
gcc -c -o code2.o code2.c
gcc -c -o code7.o code7.c
gcc -c -o code10.o code10.c
gcc -c -o main.o main.c
gcc -c -o code8.o code8.c
gcc -c -o code4.o code4.c
gcc -c -o code9.o code9.c
gcc -c -o code1.o code1.c
gcc -c -o code6.o code6.c
gcc -o project0 code3.o code5.o code2.o code7.o code10.o main.o code8.o
code4.o code9.o code1.o code6.o
make[1]: Leaving directory `/home/whb/mkfile/Module0'
build Module0 ... done
...
```

编译结果

```
Module0
├── code10.c
├── code10.h
├── code10.o
├── code1.c
├── code1.h
├── code1.o
├── code2.c
├── code2.h
├── code2.o
├── code3.c
├── code3.h
├── code3.o
├── code4.c
├── code4.h
├── code4.o
├── code5.c
├── code5.h
├── code5.o
├── code6.c
├── code6.h
├── code6.o
├── code7.c
├── code7.h
├── code7.o
├── code8.c
├── code8.h
├── code8.o
├── code9.c
├── code9.h
├── code9.o
├── main.c
├── main.o
├── Makefile
└── project0
...
```

清理

代码块

```
$ make clean
make[1]: Entering directory `/home/whb/mkfile/Module0'
rm -f *.o project0
make[1]: Leaving directory `/home/whb/mkfile/Module0'
clean Module0 ... done
```

```
make[1]: Entering directory `/home/whb/mkfile/Module1'
rm -f *.o project1
make[1]: Leaving directory `/home/whb/mkfile/Module1'
clean Module1 ... done
make[1]: Entering directory `/home/whb/mkfile/Module2'
rm -f *.o project2
make[1]: Leaving directory `/home/whb/mkfile/Module2'
clean Module2 ... done
make[1]: Entering directory `/home/whb/mkfile/Module3'
rm -f *.o project3
make[1]: Leaving directory `/home/whb/mkfile/Module3'
clean Module3 ... done
make[1]: Entering directory `/home/whb/mkfile/Module4'
...
```

查看清理结果

代码块

```
Module0
├── code10.c
├── code10.h
├── code1.c
├── code1.h
├── code2.c
├── code2.h
├── code3.c
├── code3.h
├── code4.c
├── code4.h
├── code5.c
├── code5.h
├── code6.c
├── code6.h
├── code7.c
├── code7.h
├── code8.c
├── code8.h
├── code9.c
├── code9.h
├── main.c
└── Makefile
```

编译并发布

```
$ make && make output
```

查看结果

代码块

```
$ ll
total 60
2drwxrwxr-x 2 whb whb 4096 Oct 8 16:23 bin
-rw-rw-r-- 1 whb whb 424 Oct 8 16:20 Makefile
-rwxrwxr-x 1 whb whb 2145 Oct 8 15:57 mkcode.sh
drwxrwxr-x 2 whb whb 4096 Oct 8 16:23 Module0
drwxrwxr-x 2 whb whb 4096 Oct 8 16:23 Module1
drwxrwxr-x 2 whb whb 4096 Oct 8 16:23 Module2
drwxrwxr-x 2 whb whb 4096 Oct 8 16:23 Module3
drwxrwxr-x 2 whb whb 4096 Oct 8 16:23 Module4
drwxrwxr-x 2 whb whb 4096 Oct 8 16:23 Module5
drwxrwxr-x 2 whb whb 4096 Oct 8 16:23 Module6
drwxrwxr-x 2 whb whb 4096 Oct 8 16:23 Module7
drwxrwxr-x 2 whb whb 4096 Oct 8 16:23 Module8
drwxrwxr-x 2 whb whb 4096 Oct 8 16:23 Module9
-rw-rw-r-- 1 whb whb 240 Oct 8 16:11 subMakefile
-rw-rw-r-- 1 whb whb 60 Sep 24 17:06 template.txt
$ ll bin
total 120
-rwxrwxr-x 1 whb whb 9096 Oct 8 16:23 project0
-rwxrwxr-x 1 whb whb 9112 Oct 8 16:23 project1
-rwxrwxr-x 1 whb whb 9112 Oct 8 16:23 project2
-rwxrwxr-x 1 whb whb 9112 Oct 8 16:23 project3
-rwxrwxr-x 1 whb whb 9112 Oct 8 16:23 project4
-rwxrwxr-x 1 whb whb 9112 Oct 8 16:23 project5
-rwxrwxr-x 1 whb whb 9112 Oct 8 16:23 project6
-rwxrwxr-x 1 whb whb 9112 Oct 8 16:23 project7
-rwxrwxr-x 1 whb whb 9112 Oct 8 16:23 project8
-rwxrwxr-x 1 whb whb 9112 Oct 8 16:23 project9
```

形成多可执行程序