Linux工程管理工具 make入门

make工具的功能

- 主要负责一个软件工程中多个源代码的**自动编译**工作
- 还能进行环境检测、后期处理等工作；
- make工具可以识别出工程中哪些文件已经被修改，并且在再次编译的时候只编译这些文件，从而提高编译的效率
- make的**主要任务**是根据makefile文件（一个脚本文件）中定义的规则和步骤，根据各个模块的更新情况，自动完成整个软件项目的维护和目标程序生成工作。让程序员把注意力放在“代码本身”，而不是“编译代码”上

makefile文件

makefile告诉make该做什么、怎么做。make工具在当前目录下寻找名为“Makefile”或“makefile”的文件并解释执行其指令。

makefile主要包含了

1. **一系列规则（显示规则、隐式规则、模式规则）**
2. **变量定义**
3. **文件指示（指示包含其他makefile文件，或根据情况指定makefile文件中的有效部分）**

Makefile的规则

一条规则包含3个方面的内容

1. 要创建的**目标**（文件）
2. 创建目标（文件）**所依赖的文件列表**
3. 通过依赖文件创建目标文件的命令组

```
规则一般形式
 target: dependency_files
<tab>command
<tab>...
例如
test:test.c
    gcc -O -o test test.c
Makefile 与 makefile
    makefile优先
```

Makefile 中的变量

变量可以指代一个长的字符串

使用变量可以

- 降低错误风险
- 简化makefile

变量类型

- 简单变量VAR := var

- 递归变量VAR = var
- 自动变量

常见自动变量有

命令格式	含义
$*	不包含扩展名的目标文件名称
$+	所有的依赖文件，以空格分开，并以出现的先后为序，可能包含重复的依赖文件
$<	第一个依赖文件的名称
$?	所有时间戳比目标文件晚的依赖文件，并以空格分开
$@	目标文件的完整名称
$^	所有不重复的依赖文件，以空格分开
$%	如果目标是归档成员，则该变量表示目标的归档成员名称

makefile中的规则

- 显式规则
- 隐式规则
- 模式规则

见makefile例子代码，abc.h和xyz.h

```
# make命令默认执行第一个规则 其它命令需要名称
EXE = test
CC = gcc
CFLAGS = -Wall
OBJ = main.c

# all所以目标文件
build:$(OBJ)
        $(CC) $(CFLAGS) -O0  $< -o m0
        $(CC) $(CFLAGS) -O1  $< -o m1
        $(CC) $(CFLAGS) -O2  $< -o m2
        $(CC) $(CFLAGS) -O3  $< -o m3

# 自己添加的run规则
run:$@
        ./m0
        ./m1
        ./m2
        ./m3
clean:
        @ -rm m0 m1 m2 m3 -f
```

隐式规则

常见的隐含规则

1. **C 程序的编译和链接**：
   - 编译 `.c` 文件为 `.o` 文件：

     ```
     makefile复制代码%.o: %.c
         $(CC) $(CFLAGS) -c -o $@ $<
     ```

     这表示所有的 `.c` 文件都会被编译为对应的 `.o` 文件。
   - 链接 `.o` 文件生成可执行文件：

     ```
     makefile复制代码%: %.o
         $(CC) $(LDFLAGS) -o $@ $^
     ```

     这表示所有的 `.o` 文件将会被链接成对应的可执行文件。
2. **C++ 程序的编译和链接**：
   - 编译 `.cpp` 文件为 `.o` 文件：

     ```
     makefile复制代码%.o: %.cpp
         $(CXX) $(CXXFLAGS) -c -o $@ $<
     ```

     这表示所有的 `.cpp` 文件会被编译为对应的 `.o` 文件。
   - 链接 `.o` 文件生成可执行文件：

     ```
     makefile复制代码%: %.o
         $(CXX) $(LDFLAGS) -o $@ $^
     ```

     这表示所有的 `.o` 文件将会被链接成对应的可执行文件