如何写makefile (http://casatwy.com/ru-he-xie-makefile.html)

Date Mon 10 November 2014 Tags makefile (http://casatwy.com/tag/makefile.html)

makefile简述

当你写完一个算法，只有一个源代码文件的时候，你要编译一下看看结果，这十分简单。

```
clang -o helloworld.run helloworld.c
./helloworld.run
```

但是当你的项目有几十个C文件，然后你要把你的项目编译出来跑跑看，那就坑爹了。

```
clang -o module_a.o -c module_a.c
clang -o module_b.o -c module_b.c
clang -o module_c.o -c module_c.c
...
clang -o project.run module_a.o module_b.o module_c.o ...
./project.run
```

你说你可以把上面这一堆命令写成bash，每次改完代码要编译的时候跑一下这个脚本就好了。但这样做又会有下面这些问题：

```
1. 如果模块之间相互依赖，用bash来描述和维护这样的依赖关系会很麻烦
2. 如果我只是修改了其中一个文件，跑一遍bash会把所有文件都编译一遍，浪费了时间
```

makefile就是做了类似bash的事情，同时解决了如下三个问题：

```
1. 简化编译整个项目的操作
2. 提供了相对简便地描述依赖的方案
3. 只编译改动过的文件
```

makefile本质上只是命令的组织，通过make能够理解的语法来描述编译的依赖。然而具体到命令的内容，则是根据编译器的不同而不同的，本文使用的是clang，如果你用gcc或者其他的编译器，那就需要对命令稍作修改。具体编译器使用命令的介绍不在本文范围内，不过这并不影响你学习如何写makefile。

Social

RSS (http://casatwy.com/feeds/all.atom.xml)

github (http://github.com/casatwy)

facebook (https://www.facebook.com/taloyum)

google+ (https://plus.google.com/u/0/108264119649922067163)

weibo (http://weibo.com/casatwy)

Tags

(http://casatwy.com/)

Links

casatwy (http://casatwy.com/)

刘坤的技术博客 (http://blog.cnbluebox.com)

齐道长的博客 (http://qitaos.github.io)

makefile是如何简化编译操作的

在你的项目目录里面创建一个 GNUmakefile 或 makefile 或 Makefile 文件，在这个文件里面写好内容（嗯，这篇文章就是教你怎么写makefile的内容的）。然后调用 make 程序就好了。

make程序会先找 GNUmakefile , 然后是 makefile , 然后是 Makefile 。一般来说 GNUmakefile 不太用，除非你的make是GNU make，否则别家的make程序识别不出来。然后比较推荐的是使用 Makefile ，因为这样在终端里面 ls 的时候比较醒目，它比较贴近目录列表，而且靠近 README 。

你也可以 make -f mymakefile 或者 make --file=mymakefile ，2B青年一般都喜欢这么做。

```
$  ls
makefile    src

$  make
clang -I./src -I../../utils -c src/demo.c -o build/demo.o
clang -I./src -I../../utils -c ../../utils/ArrayUtils.c -o buil
clang -I./src -I../../utils -o demo.run  ./build/demo.o ./build

$  ls
build       makefile    src         demo.run
```

1. demo.run 就是编译生成的可执行文件。大多数情况下，你 make 就够了。
2. build 是由makefile里面的命令指定生成的文件夹，用来存放编译过程的中间文件，如果你没有在makefile里面写相关创建文件夹的指令， build 不会自动出来。
3. src 是一个代码文件夹，里面存放源代码文件。

继续阅读前，你要知道的基础知识

1. 源代码先编译成中间文件，比如c文件编译成o文件
2. 链接中间文件形成可执行程序，比如一堆o文件连接成一个可执行文件
3. 源文件的相互调用是在链接的时候进行协调的，所以只管走第一步就好了

没了。

makefile是如何描述依赖的

编译时，模块之间会存在各种依赖，makefile也会指导编译器按照依赖顺序去编译文件。make程序会从makefile中的第一个标签开始解析，所以第一个标签会成为整个依赖树的根。

描述依赖规则的方法是这样的：

```
标签：依赖列表
    模块编译语句
```

注意，模块编译语句的句首一定要有一个tab。没有依赖的话，依赖列表就可以不写。

下面举个例子：

fruit程序由2个源代码文件组成，分别是apple.c pineapple.c， pineapple.c有调用apple.c中的函数， 那么makefile就可以像下面这么写：

```
# makefile里面的注释就是加个#就好了，跟bash一样

start: apple.o pineapple.o
    clang -o fruit apple.o pine.o

apple.o: apple.c
    clang -c apple.c -o apple.o

pineapple.o: pine.c
    clang -c pine.c -o pine.o
```

1. make在解析这个makefile的时候，会先走到 `start` 标签，因为它是第一个(而不是因为它叫 `start`)。然后make看到apple.o是它的依赖，于是他会去找apple.o这个标签并执行下面的clang语句。
2. pineapple.c调用apple.c里面的函数这一茬，在start标签的命令里面，链接器会检查后面所有的.o文件的符号进行链接的，这个不用我们操心。
3. start里面的依赖列表的顺序是无所谓的，谁先谁后都可以

其实到这里，你完全可以自己写makefile来编译你自己的项目了，想知道更高级的技巧可以看后面。

makefile中的一些技巧

```
INCLUDE_PATH = -I./src -I../../utils
BUILD_DIR = ./build

ALL_OBJ_O = $(BUILD_DIR)/demo.o $(BUILD_DIR)/ArrayUtils.o
TARGET = demo.run

CC = clang $(INCLUDE_PATH)

start: prepare $(TARGET)

prepare:
    mkdir -p $(BUILD_DIR)

$(TARGET): $(ALL_OBJ_O)
    $(CC) -o $@  $(ALL_OBJ_O)

$(BUILD_DIR)/demo.o: ./src/demo.c
    $(CC) -c $< -o $@

$(BUILD_DIR)/ArrayUtils.o: ../../utils/ArrayUtils.c ../../utils
    $(CC) -c $< -o $@

clean:
    rm -rf $(BUILD_DIR)
    rm -rf ./$(TARGET)
```

1. 这里我定义了 `INCLUDE_PATH` 、 `BUILD_DIR` 、 `ALL_OBJ_O` 、 `TARGET` 、 `CC` 这些变量，方便将来的扩展和改变
2. 所有的依赖名其实都是基于字符串的，在start里面看到有prepare这个依赖，它并不是某个o文件。而且你也发现有些标签还带了路径，这些都没关系，make会找到对应字符串的标签，然后执行它
3. 编译语句里面写的其实还是bash命令，我把一些准备工作放到prepare标签下了
4. $<代表当前的依赖列表的第一项, 这里的标签正好就是要编译的c文件
5. $@代表标签名，这里的标签正好就是要输出的文件
6. clean标签在执行make的时候是不会走到的，但是执行make clean的时候就会走到了。聪明的你一定会发现，make后面跟什么标签，就会执行什么标签下的命令。你执行make prepare，那么就会执行prepare下的命令
7. `$(BUILD_DIR)/ArrayUtils.o: ../../utils/ArrayUtils.c ../../utils/ArrayUtils.h` 这里的h文件可以写也可以不写，写上的话，如果你修改了h文件，那么也会引起make重新链接target。如果不写，h文件即使有修改make也不管

makefile中的一些高级技巧

你可以省略将C文件编译成O文件的命令, make会自动根据标签名生成这样的命令： `cc -c -o 标签.o 标签.c` . 所以要是这么做，你就要指定一下编译器，export一下cc这个变量

```
export cc=clang

start: apple.o pineapple.o
    clang -o fruit apple.o pine.o

apple.o:
pineapple.o:

# 当然你也可以写成这样:
# apple.o: apple.h
# pineapple.o: pineapple.h
# 这样当h文件修改时，make就会重新链接了
```

如果你的某个标签可能跟某个文件重名，就可以使用 `.PHONY` 来标识这个标签, 详情参见Phony Targets (https://www.gnu.org/software/make/manual/make.html#Phony-Targets)

```
start: apple.o pineapple.o clean
    clang -o fruit apple.o pine.o clean

apple.o:
pineapple.o:
clean:
    clang -o clean -c clean.c

.PHONY: clean
clean:
    rm -rf ./build/.*

# 如果不用PHONY，那么在编译fruit的时候，make遇到clean就不知道要走哪个标
```

可以使用 `include` 指令来包含其他的makefile，描述文件名时可以用shell类似的拓展方法

```
bar = bish bash
include foo *.mk $(bar)

# 等价于
# include foo a.mk b.mk c.mk bish bash
```

总结

gnu上关于make的描述 (https://www.gnu.org/software/make/manual/make.html)很长很详细，不过其实只要理解了makefile是用来组织各种命令的，那剩下的问题其实就都是如何把makefile写得更好。

我们也见到很多工程会先跑一个configure脚本，然后生成makefile，大多数情况下，这是使用autoconf这个程序来生成configure脚本，然后进而生成makefile的。

我们也见过有通过automake来编译部署程序的。这些已经是后期发布时要操作的事情了，开发的时候写个简单的makefile其实够用。这两个坑我先挖在这儿，以后有空再填。

评论系统我用的是Disqus，不定期被墙。所以如果你看到文章下面没有加载出评论列表，翻个墙就有了。

我的博客拒绝挂任何广告，如果您觉得文章有价值，可以通过支付宝扫描下面的二维码捐助我。

Comments