昵称：~大器晚成~
园龄：8年8个月
粉丝：339
关注：3
+加关注

< 2012年2月 >

日	一	二	三	四	五	六
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	1	2	3
4	5	6	7	8	9	10

我的标签
面向对象(10)
数据结构与算法(8)
Web(8)
分布式(6)
机器学习(6)
生活(6)
C/C++(5)
Shell(4)
机器学习与算法(2)
Linux相关(2)
更多

随笔分类(72)
Hadoop分布式(7)
Linux相关(7)
Web技术(8)
程序相关(28)
机器学习与算法(15)
生活中来(7)
移动互联网

应该去的好地方
2011推荐系统高峰论坛
20本最好的Linux免费书籍
36氪
coolshell
Learn to code
Linux/Unix 新手和专家教程
Linux电子书
MIT计算机理论课程
TED Website
非常不错的编程技术教程
经典的C/C++教程
免费电子书列表
我爱自然语言处理
一些文章和各种资源

积分与排名
积分 - 115072
排名 - 1598

如何自己编写Makefile

相信很多朋友都有过这样的经历，看着开源项目中好几页的makefile文件，不知所云。在日常学习和工作中，也有意无意的去回避makefile，能改就不写，能用ide就用ide。其实makefile并没有想象的那么难写，只要你明白了其中的原理，自己实践几次。你也可以自己写makefile，让别人对你头来羡慕的目光。

下面本人介绍一下自己的学习成果，初学阶段，欢迎大家多多指正。

简单的说，makefile定义了一系列的规则来指定，哪些文件需要先编译，哪些文件需要后编译，哪些文件需要重新编译，甚至可以在makefile中执行shell脚本。makefile带来的好处就是—"自动化编译"，一旦写好，只需要一个make命令，整个工程完全自动编译，极大的提高了软件开发的效率。

关于程序的编译和链接

一般来说，无论是C还是C++，首先要把源文件编译成中间代码文件，在Windows下也就是 .obj 文件，UNIX下是 .o 文件，即 Object File，这个动作叫做编译（compile），一般来说，每个源文件都应该对应于一个中间目标文件（O文件或是OBJ文件）。然后再把大量的Object File合成执行文件，这个动作叫作链接（link）。

编译时，编译器需要的是语法的正确，函数与变量的声明的正确。对于后者，通常是你需要告诉编译器头文件的所在位置（头文件中应该只是声明，而定义应该放在C/C++文件中），只要所有的语法正确，编译器就可以编译出中间目标文件。

链接时，主要是链接函数和全局变量，所以，我们可以使用这些中间目标文件（O文件或是OBJ文件）来 链接我们的应用程序。链接器并不管函数所在的源文件，只管函数的中间目标文件（Object File），在大多数时候，由于源文件太多，编译生成的中间目标文件太多，而在链接时需要明显地指出中间目标文件名，这对于编译很不方便，所以，我们要给 中间目标文件打个包，在Windows下这种包叫"库文件"（Library File），也就是 .lib 文件，在UNIX下，是Archive File，也就是 .a 文件。

下面我们开始看看如何自己写出makefile。

Makefile的规则

目标 ： 需要的条件 （注意冒号两边有空格）

命令 （注意前面用tab键开头）

解释一下：

1 目标可以是一个或多个，可以是Object File，也可以是执行文件，甚至可以是一个标签。

2 需要的条件就是生成目标所需要的文件或目标

3 命令就是生成目标所需要执行的脚本

总结一下，就是说一条makefile规则规定了编译的依赖关系，也就是目标文件依赖于条件，生成规则用命令来描述。**在编译时，如果需要的条件的文件比目标更新的话，就会执行生成命令来更新目标。**

下面举个简单的例子说明。如果一个工程有3个头文件，和8个C文件，我们为了完成前面所述的那三个规则，我们的Makefile应该是下面的这个样子的。

```
edit : main.o kbd.o command.o display.o /
        insert.o search.o files.o utils.o
         cc -o edit main.o kbd.o command.o display.o /
                    insert.o search.o files.o utils.o

 main.o : main.c defs.h
          cc -c main.c
 kbd.o : kbd.c defs.h command.h
```

```
        cc -c kbd.c
command.o : command.c defs.h command.h
        cc -c command.c
display.o : display.c defs.h buffer.h
        cc -c display.c
insert.o : insert.c defs.h buffer.h
        cc -c insert.c
search.o : search.c defs.h buffer.h
        cc -c search.c
files.o : files.c defs.h buffer.h command.h
        cc -c files.c
utils.o : utils.c defs.h
        cc -c utils.c
clean :
        rm edit main.o kbd.o command.o display.o /
           insert.o search.o files.o utils.o
```

将上面的内容写入到Makefile文件中，然后执行make就可以进行编译，执行make clean就可以删除所有目标文件。解释一下，也就是说生成最终的目标文件edit，依赖于一系列的.o目标文件，而这些.o文件又是需要用源文件来编译生成的。

需要注意的是，clean后面没有条件，而clean本身也不是文件，它只不过是一个动作名字，其冒号后什么也没有，那么，make就不会自动去找文件的依赖性，也就不会自动执行其后所定义的命令。

make是如何工作的

在默认的方式下，也就是我们只输入make命令。那么，

1、make会在当前目录下找名字叫"Makefile"或"makefile"的文件。
2、如果找到，它会找文件中的第一个目标文件（target），在上面的例子中，他会找到"edit"这个文件，并把这个文件作为最终的目标文件。
3、如果edit文件不存在，或是edit所依赖的后面的 .o 文件的文件修改时间要比edit这个文件新，那么，他就会执行后面所定义的命令来生成edit这个文件。
4、如果edit所依赖的.o文件也不存在，那么make会在当前文件中找目标为.o文件的依赖性，如果找到则再根据那一个规则生成.o文件。（这有点像一个堆栈的过程）
5、当然，你的C文件和H文件是存在的啦，于是make会生成 .o 文件，然后再用 .o 文件生命make的终极任务，也就是执行文件edit了。

makefile中使用变量

前面的知识已经足以让你自己完成一个简单的makefile了，不过makefile的精妙之处远不止如此，下面来看看如何在makefile中使用变量吧。

在上面的例子中，先让我们看看edit的规则：

```
edit : main.o kbd.o command.o display.o /
           insert.o search.o files.o utils.o
       cc -o edit main.o kbd.o command.o display.o /
                 insert.o search.o files.o utils.o
```

我们可以看到[.o]文件的字符串被重复了两次，如果我们的工程需要加入一个新的[.o]文 件，那么我们需要在两个地方加（应该是三个地方，还有一个地方在clean中）。当然，我们的makefile并不复杂，所以在两个地方加也不累，但如果 makefile变得复杂，那么我们就有可能会忘掉一个需要加入的地方，而导致编译失败。所以，为了makefile的易维护，在makefile中我们 可以使用变量。makefile的变量也就是一个字符串，理解成C语言中的宏可能会更好。

于是，我们使用变量objects

```
objects = main.o kbd.o command.o display.o /
```

```
          insert.o search.o files.o utils.o
```

这样一来，原来的makefile变成如下的样子：

```
objects = main.o kbd.o command.o display.o /
          insert.o search.o files.o utils.o

edit : $(objects)
        cc -o edit $(objects)
main.o : main.c defs.h
        cc -c main.c
kbd.o : kbd.c defs.h command.h
        cc -c kbd.c
command.o : command.c defs.h command.h
        cc -c command.c
display.o : display.c defs.h buffer.h
        cc -c display.c
insert.o : insert.c defs.h buffer.h
        cc -c insert.c
search.o : search.c defs.h buffer.h
        cc -c search.c
files.o : files.c defs.h buffer.h command.h
        cc -c files.c
utils.o : utils.c defs.h
        cc -c utils.c
clean :
        rm edit $(objects)
```

这样看起来方便多了吧，也更加省事了。如果有新的.o文件怎么办？当然是在objects里面添加了，这样只需要一处改变，很方便吧。

让make自动推导

GNU的make很强大，它可以自动推导文件以及文件依赖关系后面的命令，于是我们就没必要去在每一个[.o]文件后都写上类似的命令，因为，我们的make会自动识别，并自己推导命令。

只要make看到一个[.o]文件，它就会自动的把[.c]文件加在依赖关系中，如果make找到一 个whatever.o，那么whatever.c，就会是whatever.o的依赖文件。并且 cc -c whatever.c 也会被推导出来，于是，我们的makefile再也不用写得这么复杂。我们的是新的makefile又出炉了。

```
objects = main.o kbd.o command.o display.o /
          insert.o search.o files.o utils.o

edit : $(objects)
        cc -o edit $(objects)


main.o : defs.h
kbd.o : defs.h command.h
command.o : defs.h command.h
display.o : defs.h buffer.h
insert.o : defs.h buffer.h
search.o : defs.h buffer.h
files.o : defs.h buffer.h command.h
utils.o : defs.h

clean :
        rm edit $(objects)
```

当然，如果你觉得那么多[.o]和[.h]的依赖有点不爽的话，好吧，没有问题，这个对于make来说很容易，谁叫它提供了自动推导命令和文件的功能呢？来看看最新风格的makefile吧。

```
objects = main.o kbd.o command.o display.o /
          insert.o search.o files.o utils.o

edit : $(objects)
```

```
        cc -o edit $(objects)

$(objects) : defs.h
kbd.o command.o files.o : command.h
display.o insert.o search.o files.o : buffer.h
clean :
        rm edit $(objects)
```

不过话说回来，本人并不推荐这种方法。**虽然简单，但是这种方法破坏了文件本身的依赖关系。**如果文件过多的话，可能你自己都不清楚了。

怎么样，makefile是不是既简单又强大？其实makefile远比这更强大，容我日后再慢慢介绍，今天就先到这里。

如果大家觉得看我的文章不解渴，可以去这里看看 http://blog.csdn.net/haoel/article/details/2887，大牛的文章，我个人是很喜欢的。

最后，欢迎大家拍砖啊。

分类：程序相关

标签：C/C++

~大器晚成~
关注 - 3
粉丝 - 339
+加关注

« 上一篇：推荐系统的常用算法概述
» 下一篇：如何自己编写Makefile(高级篇)

posted @ 2012-02-05 15:33 ~大器晚成~ 阅读(60555) 评论(8) 编辑 收藏

发表评论

#1楼 2012-02-05 17:44 | Charles Yan

呵呵，写的不错，顶一下。

支持(0) 反对(0)

#2楼[楼主] 2012-02-05 23:45 | ~大器晚成~

@墨守橙规
多谢支持～

支持(0) 反对(0)

#3楼 2012-02-06 08:47 | yhexie

不错啊!

支持(0) 反对(0)

#4楼[楼主] 2012-02-06 09:52 | ~大器晚成~

@yhexie
也是初学，互相学习，呵呵

支持(0) 反对(0)

#5楼 2012-02-10 16:09 | sina_esf_bj[未注册用户]

main.o：main.c defs.h

#6楼 2012-02-10 16:10 | sina_esf_bj[未注册用户]

main.o：main.c defs.h
只想问一句，是不是这样写了，main.c中就不用写 #include "defs.h"了?

#7楼 2014-07-23 12:36 | 近海之林

哇噻，小哥你写的太好了！一目了然！我终于搞懂了...

支持(0) 反对(0)

#8楼 2015-09-06 09:53 | 王者之路

学习了

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。