楼主

nginx_h

初级粉丝 1

陈皓

概述

——

什么是makefile？或许很多Winodws的程序员都不知道这个东西，因为那些Windows的IDE都为你做了这个工作，但我觉得要作一个好的和 professional的程序员，makefile还是要懂。这就好像现在有这么多的HTML的编辑器，但如果你想成为一个专业人士，你还是要了解 HTML的标识的含义。特别在Unix下的软件编译，你就不能不自己写makefile了，会不会写makefile，从一个侧面说明了一个人是否具备完 成大型工程的能力。

因为，makefile关系到了整个工程的编译规则。一个工程中的源文件不计数，其按类型、功能、模块分别放在若干个目录中，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，甚至于进行更复杂的功能操作，因为makefile就像一个Shell 脚本一样，其中也可以执行操作系统的命令。

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。make是 一个命令工具，是一个解释makefile中指令的命令工具，一般来说，大多数的IDE都有这个命令，比如：Delphi的make，Visual C++的nmake，Linux下GNU的make。可见，makefile都成为了一种在工程方面的编译方法。

现在讲述如何写makefile的文章比较少，这是我想写这篇文章的原因。当然，不同产商的make各不相同，也有不同的语法，但其本质都是在“文件依赖 性”上做文章，这里，我仅对GNU的make进行讲述，我的环境是RedHat Linux 8.0，make的版本是3.80。必竟，这个make是应用最为广泛的，也是用得最多的。而且其还是最遵循于IEEE 1003.2-1992 标准的（POSIX.2）。

在这篇文档中，将以C/C++的源码作为我们基础，所以必然涉及一些关于C/C++的编译的知识，相关于这方面的内容，还请各位查看相关的编译器的文档。这里所默认的编译器是UNIX下的GCC和CC。

关于程序的编译和链接

——————————

在此，我想多说关于程序编译的一些规范和方法，一般来说，无论是C、C++、还是pas，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）。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

Makefile详解（超级好）

只看楼主 收藏 回复

应于一个中间目标文件（O文件或是OBJ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。

总结一下，源文件首先会生成中间目标文件，再由中间目标文件生成执行文件。在编译时，编译器只检测程序语法，和函数、变量是否被声明。如果函数未被声明，编译器会给出一个警告，但可以生成Object File。而在链接程序时，链接器会在所有的Object File中找寻函数的实现，如果找不到，那到就会报链接错误码（Linker Error），在VC下，这种错误一般是：Link 2001错误，意思说是说，链接器未能找到函数的实现。你需要指定函数的Object File.

送TA礼物 分享

举报 1楼 2009-06-10 22:51 回复

楼主

nginx_h

初级粉丝 1

好，言归正传，GNU的make有许多的内容，闲言少叙，还是让我们开始吧。

Makefile 介绍

——————

make命令执行时，需要一个 Makefile 文件，以告诉make命令需要怎么样的去编译和链接程序。

首先，我们用一个示例来说明Makefile的书写规则。以便给大家一个感兴认识。这个示例来源于GNU的make使用手册，在这个示例中，我们的工程有8个C文件，和3个头文件，我们要写一个Makefile来告诉make命令如何编译和链接这几个文件。我们的规则是：

1）如果这个工程没有编译过，那么我们的所有C文件都要编译并被链接。

2）如果这个工程的某几个C文件被修改，那么我们只编译被修改的C文件，并链接目标程序。

3）如果这个工程的头文件被改变了，那么我们需要编译引用了这几个头文件的C文件，并链接目标程序。

只要我们的Makefile写得够好，所有的这一切，我们只用一个make命令就可以完成，make命令会自动智能地根据当前的文件修改的情况来确定哪些文件需要重编译，从而自己编译所需要的文件和链接目标程序。

贴吧热议榜

1 海贼王991话 1409830
2 五十万彩礼 1280753
3 伊布新冠检测阳性 1057010
4 拜仁夺得欧洲超级杯 832036
5 门迪加盟切尔西 463437
6 糖豆人内部构造 455222
7 断骨增高失败终身残疾 439028
8 德波尔出任荷兰主帅 335827
9 漫威多部新片档期推迟 256688
10 ios14 真香? 208143

Makefile详解（超级好）

只看楼主 收藏 回复

在讲述这个Makefile之前，还是让我们先来粗略地看一看Makefile的规则。

target ... : prerequisites ...
command
...
...

target也就是一个目标文件，可以是Object File，也可以是执行文件。还可以是一个标签（Label），对于标签这种特性，在后续的“伪目标”章节中会有叙述。

prerequisites就是，要生成那个target所需要的文件或是目标。

command也就是make需要执行的命令。（任意的Shell命令）

这是一个文件的依赖关系，也就是说，target这一个或多个的目标文件依赖于prerequisites中的文件，其生成规则定义在command中。 说白一点就是说，prerequisites中如果有一个以上的文件比target文件要新的话，command所定义的命令就会被执行。这就是 Makefile的规则。也就是Makefile中最核心的内容。

说到底，Makefile的东西就是这样一点，好像我的这篇文档也该结束了。呵呵。还不尽然，这是Makefile的主线和核心，但要写好一个Makefile还不够，我会以后面一点一点地结合我的工作经验给你慢慢到来。内容还多着呢。：）

二、一个示例

正如前面所说的，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
```

⚠举报 2楼 2009-06-10 22:51 回复

Makefile详解（超级好）

只看楼主 收藏 回复

nginx_h

初级粉丝 1

insert.o : defs.h buffer.h
search.o : defs.h buffer.h
files.o : defs.h buffer.h command.h
utils.o : defs.h

.PHONY : clean
clean :
rm edit $(objects)

这种方法，也就是make的“隐晦规则”。上面文件内容中，“.PHONY”表示，clean是个伪目标文件。

关于更为详细的“隐晦规则”和“伪目标文件”，我会在后续给你一一道来。

六、另类风格的makefile

即然我们的make可以自动推导命令，那么我看到那堆[.o]和[.h]的依赖就有点不爽，那么多的重复的[.h]，能不能把其收拢起来，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

.PHONY : clean
clean :
rm edit $(objects)

这种风格，让我们的makefile变得很简单，但我们的文件依赖关系就显得有点凌乱了。鱼和熊掌不可兼得。还看你的喜好了。我是不喜欢这种风格的，一是文件的依赖关系看不清楚，二是如果文件一多，要加入几个新的.o文件，那就理不清楚了。

七、清空目标文件的规则

每个Makefile中都应该写一个清空目标文件（.o和执行文件）的规则，这不仅便于重编译，也很利于保持文件的清洁。这是一个“修养”（呵呵，还记得我的《编程修养》吗）。一般的风格都是：

clean:
rm edit $(objects)

更为稳健的做法是：

.PHONY : clean
clean :
-rm edit $(objects)

前面说过，.PHONY意思表示clean是一个“伪目标”，。而在rm命令前面加了一个小减号的意思就是，也许某些文件出现问题，但不要管，继续做后面 的事。当然，clean的规则不要放

Makefile详解（超级好）

只看楼主 收藏 回复

上面就是一个makefile的概貌，也是makefile的基础，下面还有很多makefile的相关细节，准备好了吗？准备好了就来。
一、Makefile里有什么？

Makefile里主要包含了五个东西：显式规则、隐晦规则、变量定义、文件指示和注释。

1、显式规则。显式规则说明了，如何生成一个或多的的目标文件。这是由Makefile的书写者明显指出，要生成的文件，文件的依赖文件，生成的命令。

2、隐晦规则。由于我们的make有自动推导的功能，所以隐晦的规则可以让我们比较粗糙地简略地书写Makefile，这是由make所支持的。

3、变量的定义。在Makefile中我们要定义一系列的变量，变量一般都是字符串，这个有点你C语言中的宏，当Makefile被执行时，其中的变量都会被扩展到相应的引用位置上。

4、文件指示。其包括了三个部分，一个是在一个Makefile中引用另一个Makefile，就像C语言中的include一样；另一个是指根据某些情 况指定Makefile中的有效部分，就像C语言中的预编译#if一样；还有就是定义一个多行的命令。有关这一部分的内容，我会在后续的部分中讲述。

5、注释。Makefile中只有行注释，和UNIX的Shell脚本一样，其注释是用“#”字符，这个就像C/C++中的“//”一样。如果你要在你的Makefile中使用“#”字符，可以用反斜框进行转义，如：“\#”。

举报 5楼 2009-06-10 22:51 回复

楼主

nginx_h

初级粉丝 1

1、读入所有的Makefile。
2、读入被include的其它Makefile。
3、初始化文件中的变量。
4、推导隐晦规则，并分析所有规则。
5、为所有的目标文件创建依赖关系链。
6、根据依赖关系，决定哪些目标要重新生成。
7、执行生成命令。

1-5步为第一个阶段，6-7为第二个阶段。第一个阶段中，如果定义的变量被使用了，那么，make会把其展开在使用的位置。但make并不会完全马上展 开，make使用的是拖延战术，如果变量出现在依赖关系的规则中，那么仅当这条依赖被决定要使用了，变量才会在其内部展开。

当然，这个工作方式你不一定要清楚，但是知道这个方式你也会对make更为熟悉。有了这个基础，后续部分也就容易看懂了。
书写规则
————

规则包含两个部分，一个是依赖关系，一个是生成目标的方法。

在Makefile中，规则的顺序是很重要的，因为，Makefile中只应该有一个最终目标，其它的目标都是被这个目标所连带出来的，所以一定要让 make知道你的最终目标是什么。一般来说，定义在Makefile中的目标可能会有很多，但是第一条规则中的目标将被确立为最终的目标。如果第一条规则 中的目标有很多个，那么，第一个目标会成为最终的目标。make所完成的也就是这个目标。

好了，还是让我们来看一看如何书写规则。

Makefile详解（超级好）

只看楼主 收藏 回复

一、规则举例

foo.o : foo.c defs.h # foo模块
cc -c -g foo.c

看到这个例子，各位应该不是很陌生了，前面也已说过，foo.o是我们的目标，foo.c和defs.h是目标所依赖的源文件，而只有一个命令“cc -c -g foo.c”（以Tab键开头）。这个规则告诉我们两件事：

1、文件的依赖关系，foo.o依赖于foo.c和defs.h的文件，如果foo.c和defs.h的文件日期要比foo.o文件日期要新，或是foo.o不存在，那么依赖关系发生。
2、如果生成（或更新）foo.o文件。也就是那个cc命令，其说明了，如何生成foo.o这个文件。（当然foo.c文件include了defs.h文件）

二、规则的语法

targets : prerequisites
command
...

或是这样：

targets : prerequisites ; command
command
...

targets是文件名，以空格分开，可以使用通配符。一般来说，我们的目标基本上是一个文件，但也有可能是多个文件。

command是命令行，如果其不与“target:prerequisites”在一行，那么，必须以[Tab键]开头，如果和prerequisites在一行，那么可以用分号做为分隔。（见上）

prerequisites也就是目标所依赖的文件（或依赖目标）。如果其中的某个文件要比目标文件要新，那么，目标就被认为是“过时的”，被认为是需要重生成的。这个在前面已经讲过了。

如果命令太长，你可以使用反斜框（‘\’）作为换行符。make对一行上有多少个字符没有限制。规则告诉make两件事，文件的依赖关系和如何成成目标文件。

一般来说，make会以UNIX的标准Shell，也就是/bin/sh来执行命令。

三、在规则中使用通配符

如果我们想定义一系列比较类似的文件，我们很自然地就想起使用通配符。make支持三各通配符：“*”，“?”和“[...]”。这是和Unix的B-Shell是相同的。

波浪号（“~”）字符在文件名中也有比较特殊的用途。如果是“~/test”，这就表示当前用户的$HOME目录下的test目录。而 “~hchen/test”则表示用户hchen的宿主目录下的test目录。（这些都是Unix下的小知识了，make也支持）而在Windows或是 MS-DOS下，用户没有宿主目录，那么波浪号所指的目录则根据环境变量“HOME”而定。

通配符代替了你一系列的文件，如“*.c”表示所以后缀为c的文件。一个需要我们注意的是，如果我们的文件名中有通配符，如：“*”，那么可以用转义字符“\”，如“*”来表示真实的“*”字符，而不是任意长度的字符串。

Makefile详解（超级好）

只看楼主 收藏 回复

nginx_h

初级粉丝 1

因为，我们并不生成“clean”这个文件。“伪目标”并不是一个文件，只是一个标签，由于“伪目标”不是文件，所以make无法生成它的依赖关系和决定 它是否要执行。我们只有通过显示地指明这个“目标”才能让其生效。当然，“伪目标”的取名不能和文件名重名，不然其就失去了“伪目标”的意义了。

当然，为了避免和文件重名的这种情况，我们可以使用一个特殊的标记“.PHONY”来显示地指明一个目标是“伪目标”，向make说明，不管是否有这个文件，这个目标就是“伪目标”。

.PHONY : clean

只要有这个声明，不管是否有“clean”文件，要运行“clean”这个目标，只有“make clean”这样。于是整个过程可以这样写：

.PHONY: clean
clean:
rm *.o temp

伪目标一般没有依赖的文件。但是，我们也可以为伪目标指定所依赖的文件。伪目标同样可以作为“默认目标”，只要将其放在第一个。一个示例就是，如果你的 Makefile需要一口气生成若干个可执行文件，但你只想简单地敲一个make完事，并且，所有的目标文件都写在一个Makefile中，那么你可以使 用“伪目标”这个特性：

all : prog1 prog2 prog3
.PHONY : all

prog1 : prog1.o utils.o
cc -o prog1 prog1.o utils.o

prog2 : prog2.o
cc -o prog2 prog2.o

prog3 : prog3.o sort.o utils.o
cc -o prog3 prog3.o sort.o utils.o

我们知道，Makefile中的第一个目标会被作为其默认目标。我们声明了一个“all”的伪目标，其依赖于其它三个目标。由于伪目标的特性是，总是被执 行的，所以其依赖的那三个目标就总是不如“all”这个目标新。所以，其它三个目标的规则总是会被决议。也就达到了我们一口气生成多个目标的目的。 “.PHONY : all”声明了“all”这个目标为“伪目标”。

随便提一句，从上面的例子我们可以看出，目标也可以成为依赖。所以，伪目标同样也可成为依赖。看下面的例子：

.PHONY: cleanall cleanobj cleandiff

cleanall : cleanobj cleandiff
rm program

cleanobj :
rm *.o

cleandiff :
rm *.diff

“make clean”将清除所有要被清除的文件。“cleanobj”和“cleandiff”这两个伪目标有点像“子程序”的意思。我们可以输入“make cleanall”和“make cleanobj”和“make cleandiff”命令来达到清除不同种类文件的目的。

Makefile详解（超级好）

只看楼主 收藏 回复

Makefile的规则中的目标可以不止一个，其支持多目标，有可能我们的多个目标同的依赖于一个文件，并且其生成的命令大体类似。于是我们就能把其合并 起来。当然，多个目标的生成规则的执行命令是同一个，这可能会可我们带来麻烦，不过好在我们的可以使用一个自动化变量“$@”（关于自动化变量，将在后面 讲述），这个变量表示着目前规则中所有的目标的集合，这样说可能很抽象，还是看一个例子吧。

```
bigoutput littleoutput : text.g
generate text.g -$(subst output,,$@) >; $@
```

上述规则等价于：

```
bigoutput : text.g
generate text.g -big >; bigoutput
littleoutput : text.g
generate text.g -little >; littleoutput
```

其中，-$(subst output,,$@)中的“$”表示执行一个Makefile的函数，函数名为subst，后面的为参数。关于函数，将在后面讲述。这里的这个函数是截 取字符串的意思，“$@”表示目标的集合，就像一个数组，“$@”依次取出目标，并执于命令。

举报 9楼 2009-06-10 22:51 回复

219.234.81.*

好贴

举报 14楼 2009-09-02 23:14 回复

iamgong

初级粉丝

好 好

举报 15楼 2009-10-04 15:16 回复

超姐的弟弟

初级粉丝

赞！！

举报 16楼 2009-10-12 18:08 回复

sqddsunjian

你咋这么强的呢

贴吧热议榜

1 海贼王991话 1409830
2 五十万彩礼 1280753
3 伊布新冠检测阳性 1057010
4 拜仁夺得欧洲超级杯 832036
5 门迪加盟切尔西 463437
6 糖豆人内部构造 455222
7 断骨增高失败终身残疾 439028
8 德波尔出任荷兰主帅 335827
9 漫威多部新片档期推迟 256688
10 ios14 真香? 208143