## **一、变量的基础**     变量在声明时需要给予初值，而在使用时，需要给在变量名前**加上“$”符号，但最好用小括号“（）”或是大括号“{}”把变量给包括起来**。如果你要使用真实的“$”字符，那么你需要用“$$”来表示。 先看一个例子： objects = program.o foo.o utils.o program : $(objects) cc -o program $(objects) $(objects) : defs.h 变量会在使用它的地方精确地展开，就像C/C++中的宏一样，例如： foo = c prog.o : prog.$(foo) $(foo)$(foo) -$(foo) prog.$(foo) 展开后得到： prog.o : prog.c cc -c prog.c

## ****二、变量中的变量****

在Makefile中有两种方式来用变量赋值给变量。  
  
先看第一种方式，也就是简单的使用“=”号，我们可以把变量的真实值推到后面来定义，如：  
CFLAGS = $(include\_dirs) -O  
include\_dirs = -Ifoo -Ibar  
  
当“CFLAGS”在命令中被展开时，会是“-Ifoo -Ibar -O”。

我们可以使用make中的另一种用变量来定义变量的方法。这种方法使用的是“:=”操作符，如：  
x := foo  
y := $(x) bar  
x := later  
这种方法，前面的变量不能使用后面的变量

请先看一个例子，如果我们要定义一个变量，其值是一个空格，那么我们可以这样来：  
nullstring :=  
space := $(nullstring) # end of the line  
nullstring 是一个Empty变量，其中什么也没有，而我们的space的值是一个空格。请注意这里关于“#”的使用，注释符“#”的这种特性值得我们注意，如果我们这样定义一个变量：  
dir := /foo/bar # directory to put the frobs in  
dir这个变量的值是“/foo/bar”，后面还跟了4个空格，如果我们这样使用这样变量来指定别的目录——“$(dir)/file”那么就完蛋了。  
  
还有一个比较有用的操作符是“?=”，先看示例：  
FOO ?= bar  
其含义是，如果FOO没有被定义过，那么变量FOO的值就是“bar”

## ****三、变量高级用法****

这里介绍两种变量的高级使用方法

第一种是变量值的替换。  
其格式是“$(var:a=b)”或是“${var:a=b}”，其意思是，把变量“var”中所有以“a”字串“结尾”的“a”替换成“b”字串。这里的“结尾”意思是“空格”或是“结束符”。  
  
还是看一个示例吧：  
foo := a.o b.o c.o  
bar := $(foo:.o=.c)  
这个示例中，我们的“$(bar)”的值就是“a.c b.c c.c”。  
  
另外一种变量替换的技术是以“静态模式”（参见前面章节）定义的，如：  
foo := a.o b.o c.o  
bar := $(foo:%.o=%.c)  
  
这依赖于被替换字串中的有相同的模式，模式中必须包含一个“%”字符，这个例子同样让$(bar)变量的值为“a.c b.c c.c”。  
  
第二种高级用法是——“把变量的值再当成变量”。先看一个例子：  
x = y#注：x的值是”y”，而不是y变量的值  
y = z  
a := $($(x))  
在这个例子中，$(a)的值是“z”。（注意，是“x=y”，而不是“x=$(y)”）

再来看看结合第一种技术的例子：  
a\_objects := a.o b.o c.o  
1\_objects := 1.o 2.o 3.o  
sources := $($(a1)\_objects:.o=.c)  
这个例子中，如果$(a1)的值是“a”的话，那么，$(sources)的值就是“a.c b.c c.c”；如果$(a1)的值是“1”，那么$(sources)的值是“1.c 2.c 3.c”。

## ****四、追加变量值****

我们可以使用“+=”操作符给变量追加值，如：  
objects = main.o foo.o bar.o utils.o  
objects += another.o  
如果变量之前没有定义过，那么，“+=”会自动变成“=”，如果前面有变量定义，那么“+=”会继承于前次操作的赋值符

## 

## ****五、override 指示符****

如果有变量是通常make的命令行参数设置的，那么Makefile中对这个变量的赋值会被忽略。如果你想在Makefile中设置这类参数的值，那么，你可以使用“override”指示符。其语法是：  
override <variable> = <value>  
override <variable> := <value>

## ****六、多行变量****

还有一种设置变量值的方法是使用define关键字。

使用define关键字设置变量的值可以有换行，这有利于定义一系列的命令（前面我们讲过“命令包”的技术就是利用这个关键字）。  
下面的这个示例展示了define的用法：  
define two-lines  
echo foo  
echo $(bar)  
endef

## ****七、环境变量****

make 运行时的系统环境变量可以在make开始运行时被载入到Makefile文件中，但是如果Makefile中已定义了这个变量，或是这个变量由make命令行带入，那么系统的环境变量的值将被覆盖。

## ****八、目标变量****

我样可以为某个目标设置局部变量，这种变量被称为“Target-specific Variable”，它可以和“全局变量”同名，因为它的作用范围只在这条规则以及连带规则中  
其语法是：  
<target ...> : <variable-assignment>  
<target ...> : overide <variable-assignment>  
<variable-assignment>可以是前面讲过的各种赋值表达式，如“=”、“:=”、“+=”或是“？=”。第二个语法是针对于make命令行带入的变量，或是系统环境变量。  
  
这个特性非常的有用，当我们设置了这样一个变量，这个变量会作用到由这个目标所引发的所有的规则中去。如：  
prog : CFLAGS = -g  
prog : prog.o foo.o bar.o  
$(CC) $(CFLAGS) prog.o foo.o bar.o  
  
prog.o : prog.c  
$(CC) $(CFLAGS) prog.c

....  
在这个示例中，不管全局的$(CFLAGS)的值是什么，在prog目标，以及其所引发的所有规则中（prog.o foo.o bar.o的规则），$(CFLAGS)的值都是“-g”

## ****九、模式变量****

模式变量的好处就是，我们可以给定一种“模式”，可以把变量定义在符合这种模式的所有目标上。  
  
我们知道，make的“模式”一般是至少含有一个“%”的，所以，我们可以以如下方式给所有以[.o]结尾的目标定义目标变量：  
%.o : CFLAGS = -O  
同样，模式变量的语法和“目标变量”一样：  
<pattern ...> : <variable-assignment>  
<pattern ...> : override <variable-assignment>