NMAKE 参考

来自 MSDN：http://msdn.microsoft.com/zh-cn/library/dd9y37ha.aspx

整理：Daway

2009-09-25

目 录

NMAKE 参考

更新：2007 年 11 月

Microsoft 程序维护实用工具 (NMAKE.EXE) 是一个基于说明文件中包含的命令生成项目的工具。

1 运行 NMAKE

更新：2007 年 11 月

NMAKE [option...] [macros...] [targets...] [@commandfile...]

备注

NMAKE 只生成指定的 *targets*，或者，如果未指定 *targets*，则生成生成文件中的第一个目标。第一个生成文件目标可以是生成其他目标的伪目标。NMAKE 使用指定 /F 选项的生成文件；如果未指定 /F 选项，则使用当前目录下的生成文件。如果未指定生成文件，则 NMAKE 使用推理规则生成命令行 *targets*。

commandfile 文本文件（或响应文件）包含命令行输入。其他输入可以位于 @*commandfile* 之前或之后。允许使用路径。在 *commandfile* 中，换行符被视为空格。如果宏定义包含空格，则将宏定义用引号引起来。

NMAKE 选项

Tools.ini 和 NMAKE

从 NMAKE 退出代码

1.1 NMAKE 选项

更新：2007 年 11 月

下表描述了 NMAKE 选项。选项前有斜杠 (/) 或短划线 (-)，并且不区分大小写。使用 !CMDSWITCHES 更改生成文件或 Tools.ini 中的选项设置。

选项	用途
/A	强制生成所有已评估的目标，即使这些目标相对于依赖项未过期。不强制不相关目标的生成。
/B	即使时间戳相等，也强制生成。建议只用于非常快的系统（解析为两秒或小于两秒）。
/C	取消默认输出，包括非致命的 NMAKE 错误或警告、时间戳以及 NMAKE 版权信息。取消 /K 选项发出的警告。
/D	当目标不存在时，显示每个已评估的目标、依赖项和消息的时间戳。与 /P 选项一起用于调试生成文件。使用 **!CMDSWITCHES** 设置或清除部分生成文件的 /D 选项。
/E	使环境变量重写生成文件宏定义。
/ERRORREPORT [NONE | PROMPT | QUEUE | SEND]	如果 nmake.exe 在运行时失败，则可以使用 /ERRORREPORT 将有关这些内部错误的信息发送给 Microsoft。 有关 /ERRORREPORT 的更多信息，请参见 /errorReport（报告内部编译器错误）。
/F *filename*	将 *filename* 指定为生成文件。空格或制表符可以位于 *filename* 的前面。为每个生成文件指定一次 /F 选项。若要从标准输入提供生成文件，请为 *filename* 指定短划线 (-)，并按 F6 或 Ctrl+Z 结束键盘输入。
/G	显示 !INCLUDE 指令中包含的生成文件。有关更多信息，请参见生成文件预处理指令。
/HELP, /?	显示 NMAKE 命令行语法的简短摘要。
/I	忽略所有命令的退出代码。若要设置或清除部分生成文件的 /I 选项，请使用 **!CMDSWITCHES**。若要忽略部分生成文件的退出代码，请使用短划线 (-) 命令修饰符或 .IGNORE。如果两者都指定了，则重写 /K 选项。
/K	如果命令返回错误，则继续生成不相关的依赖项。同时发出警告并返回退出代码 1。默认情况下，如果有任一命令返回非零退出代码，NMAKE 将暂停。来自 /K 选项的警告被 /C 选项取消；如果两者都指定了，则 /I 选项重写 /K 选项。

/N	显示但不执行命令；执行预处理命令。不在递归 NMAKE 调用中显示命令。对于调试生成文件和检查时间戳很有用。若要设置或清除部分生成文件的 /N 选项，请使用 **!CMDSWITCHES**。
/NOLOGO	取消 NMAKE 版权消息。
/P	显示标准输出的信息（宏定义、推理规则、目标、.SUFFIXES 列表），然后运行生成。如果不存在任何生成文件和命令行目标，则只显示信息。与 /D 选项一起用于调试生成文件。
/Q	检查目标的时间戳；不运行生成。如果所有目标都是最新的，则返回零退出代码；如果有任何目标不是最新的，则返回非零退出代码。执行预处理命令。从批处理文件运行 NMAKE 时很有用。
/R	清除 **.SUFFIXES** 列表并忽略在 Tools.ini 文件中定义的，或预定义的推理规则和宏。
/S	取消已执行命令的显示。若要取消部分生成文件中的显示，请使用 **@** 命令修饰符或 .SILENT。若要设置或清除部分生成文件的 /S 选项，请使用 **!CMDSWITCHES**。
/T	更新命令行目标（或第一个生成文件目标）的时间戳并执行预处理命令，但不运行生成。
/U	必须与 /N 选项一起使用。转储内联 NMAKE 文件，以便 /N 输出可用作批处理文件。
/X *filename*	将 NMAKE 错误输出发送到 *filename* 而不是标准错误。空格或制表符可以位于 *filename* 的前面。若要将错误输出发送到标准输出，请为 *filename* 指定短划线 (-)。不影响从命令到标准错误的输出。
/Y	禁用批模式推理规则。选定该选项后，所有批模式推理规则被视为常规推理规则。

1.2 Tools.ini 和 NMAKE

更新：2007 年 11 月

NMAKE 在读取生成文件之前读取 Tools.ini，除非使用 /R 选项。它首先在当前目录中查找 Tools.ini，然后在 INIT 环境变量指定的目录中查找。初始化文件中的 NMAKE 设置部分以 [NMAKE] 开头，并且可包含任何生成文件信息。在单独一行上指定注释，以数字符号 (#) 开头。

1.3 从 NMAKE 退出代码

更新：2007 年 11 月

NMAKE 返回下列退出代码：

代码	含义
0	无错误（可能是警告）
1	不完整的生成（只在使用 /K 选项时发出）
2	程序错误，可能由于下列原因之一：
	• 生成文件中的语法错误
	• 来自命令的错误或退出代码
	• 被用户中断
4	系统错误 — 内存不足
255	目标不是最新的（只在使用 /Q 选项时发出）

2 生成文件的内容

更新：2007 年 11 月

生成文件包含：

- 描述块
- 命令
- 宏
- 推理规则
- 点指令
- 预处理指令

备注

可以在生成文件中使用的其他功能是通配符、长文件名、注释和特殊字符。

有关示例，请参见示例生成文件。

2.1 通配符和 NMAKE

更新：2007 年 11 月

NMAKE 扩展依赖项行中的文件名通配符（* 和 ?）。命令中指定的通配符传递给该命令；NMAKE 不扩展它。

2.2 生成文件中的长文件名

更新：2007 年 11 月

将长文件名用双引号引起来，如下所示：

```
all : "VeryLongFileName.exe"
```

2.3 生成文件中的注释

更新：2007 年 11 月

在注释前放置一个数字符号 (#)。NMAKE 忽略从数字符号到下一个换行符之间的文本。示例：

```
# Comment on line by itself
OPTIONS = /MAP   # Comment on macro definition line

all.exe : one.obj two.obj   # Comment on dependency line
    link one.obj two.obj
# Comment in commands block
#    copy *.obj \objects   # Command turned into comment
    copy one.exe \release

.obj.exe:   # Comment on inference rule line
    link $<

my.exe : my.obj ; link my.obj   # Err: cannot comment this
 # Error: # must be the first character
.obj.exe: ; link $<   # Error: cannot comment this
```

若要指定数字符号，请在它前面添一插入符号 (^)，如下所示：

```
DEF = ^#define   #Macro for a C preprocessing directive
```

2.4 生成文件中的特殊字符

更新：2007 年 11 月

若要将 NMAKE 特殊字符用作文字字符，请在它前面添一插入符号 (^)。NMAKE 忽略其他字符前面的插入符号。特殊字符为：

: ; # ( ) $ ^ \ { } ! @ —

引号引起来的字符串内的插入符号 (^) 被视为 ^ 字符。在字符串或宏中，位于行尾的插入符号插入换行符。

在宏中，后面跟有换行符的反斜杠 (\) 由空格替换。

在命令中，百分号 (%) 是文件说明符。若要在命令中按原义表示 %，请指定两个百分号 (%%) 取代一个百分号。在其他情况中，NMAKE 按原义解释一个 %，但它总将两个 %% 解释为一个 %。因此，若要表示 %%，请指定三个百分号 %%%，或四个百分号 %%%%。

若要将美元符号 ($) 用作命令中的文字字符，请指定两个美元符号 ($$)。此方法还可用于 ^$ 有效的其他情况。

2.5 示例生成文件

更新：2007 年 11 月

此主题包含一个示例生成文件。

示例 **代码**

```
# Sample makefile

!include <win32.mak>

all: simple.exe challeng.exe

.c.obj:
  $(cc) $(cdebug) $(cflags) $(cvars) $*.c

simple.exe: simple.obj
  $(link) $(ldebug) $(conflags) -out:simple.exe simple.obj $(conlibs) lsapi32.lib

challeng.exe: challeng.obj md4c.obj
  $(link) $(ldebug) $(conflags) -out:challeng.exe $** $(conlibs) lsapi32.lib
```

3 描述块

更新：2007 年 11 月

描述块是后面可跟有命令块的依赖项行：

```
targets... : dependents...
    commands...
```

依赖项行指定一或多个目标以及零或多个依赖项。目标必须位于行首。用冒号 (:) 将目标和依赖项分开；允许使用空格或制表符。若要拆分行，请在目标或依赖项后面使用反斜杠 (\ )。如果目标不存在、目标的时间戳比依赖项早或者目标是伪目标，则 NMAKE 执行命令。如果某依赖项是其他地方的目标，并且不存在或对于自己的依赖项已过期，则 NMAKE 在更新当前依赖项之前更新该依赖项。

目标
依赖项

3.1 目标

更新：2007 年 11 月

在依赖项行中，使用任何有效的文件名、目录名或伪目标指定一个或多个目标。用一或多个空格或制表符分隔多个目标。目标不区分大小写。允许在文件名中使用路径。目标不能超过 256 个字符。如果位于冒号之前的目标是单个字符，则使用分隔空格；否则，NMAKE 将字母与冒号的组合解释为驱动器说明符。

伪目标
多个目标
累计依赖项
多个描述块中的目标
依赖项副作用

3.1.1 伪目标

更新：2007 年 11 月

伪目标是用于替换依赖项行中的文件名的标签。它被解释为不存在的一个文件，因此是过期的。NMAKE 假定伪目标的时间戳在它的所有依赖项中是最新的。如果它没有依赖项，则采用当前的时间。如果将伪目标用作目标，则总是执行它的命令。用作依赖项的伪目标还必须显示为其他依赖项中的目标。但是，该依赖项不需要有命令块。

伪目标名遵循目标的文件名语法规则。但是，如果名称没有扩展名（即不包含句点），则可以超过 8 个字符的文件名长度限制，最长可以达到 256 个字符。

3.1.2 多个目标

更新：2007 年 11 月

NMAKE 评估单个依赖项中的多个目标，就像每个目标都是在单独的描述块中指定的一样。

此...	...依此进行评估

bounce.exe leap.exe : jump.obj
    echo Building...
	bounce.exe : jump.obj
	    echo Building...
	leap.exe : jump.obj
	    echo Building...

3.1.3 累计依赖项

更新：2007 年 11 月

如果重复目标，则在描述块中累计依赖项。

此...	...依此进行评估

bounce.exe : jump.obj
bounce.exe : up.obj
  echo Building
bounce.exe...
	bounce.exe : jump.obj
	up.obj
	  echo Building
	bounce.exe...

评估多个依赖项（在单个描述块中）中的多个目标，就像每个目标是在单独的描述块中指定的一样，但不在末尾依赖项行中的目标不使用命令块。

此...	...依此进行评估

bounce.exe leap.exe :
jump.obj
bounce.exe climb.exe :
up.obj
  echo Building...
	bounce.exe : jump.obj
	up.obj
	  echo Building
	bounce.exe...
	climb.exe : up.obj
	  echo Building
	climb.exe...
	leap.exe : jump.obj
	# invokes an inference rule

3.1.4 多个描述块中的目标

更新：2007 年 11 月

若要使用不同的命令更新多个描述块中的目标，请在目标与依赖项之间指定两个连续的冒号 (::)。

```
target.lib :: one.asm two.asm three.asm
    ml one.asm two.asm three.asm
    lib target one.obj two.obj three.obj
target.lib :: four.c five.c
    cl /c four.c five.c
    lib target four.obj five.obj
```

3.1.5 依赖项副作用

更新：2007 年 11 月

如果在位于不同位置的两个依赖项行中用冒号 (:) 指定一个目标，并且命令只出现在其中一行的后面，则 NMAKE 将依赖项解释为相邻的或组合的。它不为没有命令的依赖项调用推理规则，而是假定依赖项属于一个描述块并执行用其他依赖项指定的命令。

此...	...依此进行评估

bounce.exe : jump.obj
  echo Building
bounce.exe...
 	
bounce.exe : up.obj
	bounce.exe : jump.obj
	up.obj
	  echo Building
	bounce.exe...

如果使用两个冒号 (::)，则不会具有这种作用。

此...	...依此进行评估

bounce.exe :: jump.obj
  echo Building
bounce.exe...
 	
bounce.exe :: up.obj
	bounce.exe : jump.obj
	  echo Building
	bounce.exe...
	
	bounce.exe : up.obj
	# invokes an inference rule

3.2 依赖项

更新：2007 年 11 月

在依赖项行中，使用任何有效的文件名或伪目标在冒号 (:) 或两个冒号 (::) 后面指定零个或多个依赖项。用一或多个空格或制表符分隔多个依赖项。依赖项不区分大小写。允许在文件名中使用路径。

推导出的依赖项

依赖项的搜索路径

3.2.1 推导出的依赖项

更新：2007 年 11 月

推理依赖项从推理规则中派生，并先于显式依赖项进行评估。如果推理依赖项对于它的目标是过期的，则 NMAKE 调用该依赖项的命令块。如果推理依赖项不存在或对于它自己的依赖项已过期，则 NMAKE 首先更新推理依赖项。有关推理依赖项的更多信息，请参见推理规则。

3.2.2 依赖项的搜索路径

更新：2007 年 11 月

每个依赖项有一个可选的搜索路径，如以下所指定的：

语法

```
{directory[;directory...]}dependent
```

备注

NMAKE 首先在当前目录中查找依赖项，然后按指定的顺序从其他目录中查找。宏可以指定部分或全部搜索路径。将目录名括在大括号 ({ }) 内；用分号 (;) 分隔多个目录。不允许使用空格或制表符。

4 生成文件中的命令

更新：2007 年 11 月

如果依赖项已过期，则描述块或推理规则指定要运行的命令块。NMAKE 在运行命令之前显示每个命令，除非使用了 /S 选项、**.SILENT**、**!CMDSWITCHES** 或 @。如果描述块后面没有紧跟命令块，NMAKE 将查找匹配的推理规则。

命令块包含一个或多个命令，每个命令位于各自的命令行上。在依赖项（或规则）和命令块之间不能出现空行。但是可以出现只包含空格或制表符的行；该行被解释为空命令，并且不出现错误。命令行之间允许有空行。

命令行以一个或多个空格或制表符开始。后面紧有换行符的反斜杠 (\ ) 在命令中被解释为空格；在行尾使用反斜杠继续下一行命令。如果反斜杠后紧跟有其他任何字符（包括空格或制表符），则 NMAKE 按原义解释反斜杠。

无论后面是否紧跟有命令块，前面带分号 (;) 的命令可以出现在依赖项行上或推理规则中：

```
project.obj : project.c project.h ; cl /c project.c
```


4.1 命令修饰符

更新：2007 年 11 月

可以在命令前面指定一个或多个命令修饰符，可以选择用空格或制表符将其分隔。与命令一样，修饰符也必须缩进。

修饰符	用途
@*command*	防止显示命令。不取消命令显示。默认情况下，NMAKE 回显所有已执行的命令。使用 /S 选项取消显示整个生成文件；使用 **.SILENT** 取消显示部分生成文件。
-[*number*]*command*	关闭 *command* 的错误检查。默认情况下，当命令返回非零退出代码时，NMAKE 暂停。如果使用 -*number*，当退出代码超过 *number* 时，NMAKE 停止。空格或制表符不能出现在短划线和 number 之间。*number* 和 *command* 之间必须出现至少一个空格或制表符。使用 /I 选项关闭整个生成文件的错误检查；使用 **.IGNORE** 关闭部分生成文件的错误检查。
!*command*	如果 *command* 使用 **$****（依赖项中的所有依赖文件）或 **$?**，则为每个依赖文件执行 *command*。（依赖项中时间戳比目标的时间戳晚的所有依赖文件）。

4.2 文件名部分语法

更新：2007 年 11 月

命令中的文件名部分语法表示第一个依赖项文件名（可能为暗含的依赖项）的组成部分。文件名组成部分是为文件指定的（而不是它在磁盘上存在的）驱动器、路径、基名称和扩展名。使用 **%s** 表示完整文件名。使用 **%|**[*parts*]**F**（百分号后面跟着一个竖线字符）表示部分文件名，其中 *parts* 可以是零个或多个下列字母（按任意顺序排列）。

字母	说明
无字母	完整名称（等同于 **%s**）
d	驱动器
p	路径
f	文件基名称
e	文件扩展名

例如，如果文件名为 c:\prog.exe，则：

- %s 将为 c:\prog.exe
- %|F 将为 c:\prog.exe
- %|dF 将为 c
- %|pF 将为 c:\
- %|fF 将为 prog
- %|eF 将为 exe

4.3 生成文件中的内联文件

更新：2007 年 11 月

内联文件包含您在生成文件中指定的文本。它的文件名在命令中可以用作输入（例如，LINK 命令文件），或者它可将命令传递到操作系统。当运行创建该文件的命令时在磁盘上创建该文件。

您想进一步了解什么？

指定内联文件
创建内联文件文本
重复使用内联文件
多个内联文件

4.3.1 指定内联文件

更新：2007 年 11 月

在显示 *filename* 的命令中指定两个尖括号 (<<)。尖括号不能是宏展开。

语法

```
<<[filename]
```

备注

运行命令时，尖括号由 *filename* 替换（如果已指定），或由 NMAKE 生成的唯一名称替换。如果指定了 *filename*，*filename* 必须跟在尖括号后面，之间没有空格或制表符。允许使用路径。不需要或采用扩展名。如果指定了 *filename*，则在当前目录或指定目录中创建该文件，改写现有的任何同名文件;否则在 TMP 目录（如果没有定义 TMP 环境变量，则在当前目录）中创建该文件。如果重复使用以前的 *filename*，NMAKE 将替换以前的文件。

4.3.2 创建内联文件文本

更新：2007 年 11 月

内联文件可以是临时的，也可以是永久的。

```
inlinetext
.
.
.
<<[KEEP | NOKEEP]
```

备注

在命令后的第一行上指定 *inlinetext*。用两个尖括号 (<<) 在单独一行的行首标记结束。文件包含分隔括号前面的所有 *inlinetext*。*inlinetext* 可以有宏展开和替换，但是没有指令或生成文件注释。空格、制表符和换行符按原义处理。

临时文件在会话期间存在，并可由其他命令重复使用。在右尖括号后指定 **KEEP** 以在 NMAKE 会话后保留文件；未命名的文件以生成的文件名保留在磁盘上。为临时文件指定 **NOKEEP** 或什么都不指定。**KEEP** 和 **NOKEEP** 不区分大小写。

4.3.3 重复使用内联文件

更新：2007 年 11 月

若要重新使用内联文件，请在定义并首次使用该文件的位置指定 <<*filename*，稍后再在同一命令或其他命令中重新使用不带 << 的 *filename*。创建内联文件的命令必须在使用该文件的所有命令运行之前运行。

4.3.4 多个内联文件

更新：2007 年 11 月

命令可以创建多个内联文件。

```
command << <<
inlinetext
<<[KEEP | NOKEEP]
inlinetext
<<[KEEP | NOKEEP]
```

备注

对于每个文件，指定一或多行内联文本，该文本后面要跟包含分隔符的结束行。在第一个文件分隔行后面的行上开始第二个文件的文本。

5 宏和 NMAKE

更新：2007 年 11 月

宏用另一个字符串替换生成文件中的特定字符串。使用宏可以：

- 创建可生成不同项目的生成文件。
- 指定命令选项。
- 设置环境变量。

可以定义您自己的宏或使用 NMAKE 的预定义宏。

定义 NMAKE 宏

使用 NMAKE 宏

特殊 NMAKE 宏

5.1 定义 NMAKE 宏

更新：2007 年 11 月

```
macroname=string
```

备注

macroname 是字母、数字和下划线 (_) 的组合，最多 1,024 个字符且区分大小写。*macroname* 可以包含调用的宏。如果 *macroname* 完全是由调用的宏组成的，则正调用的宏不能为空或未定义。

string 可以是零个或多个字符的任意序列。空字符串包含零个字符或只有空格或制表符。*string* 可以包含宏调用。

宏内的特殊字符

空宏和未定义的宏

定义宏的位置

宏定义中的优先级

5.1.1 宏内的特殊字符

更新：2007 年 11 月

位于定义之后的数字符号 (#) 指定注释。若要在宏内指定数字符号，请使用插入符号 (^)，如在 ^# 中。

美元符号 ($) 指定宏调用。若要指定文本 $，请使用 $$。

若要将定义扩展到新行，请以反斜杠 (\) 结束行。调用宏后，用空格替换反斜杠和换行符。若要在行尾指定文本反斜杠，请在它的前面添上插入符号 (^)，或后面跟注释说明符 (#)。

若要指定文本换行符，用插入符号 (^) 结束行，如下所示：

```
CMDS = cls^
dir
```

5.1.2 空宏和未定义的宏

更新：2007 年 11 月

空宏和未定义的宏都展开为空字符串，但定义为空字符串的宏被视为是在预处理表达式中定义的。若要将宏定义为空字符串，请不要在命令行或命令文件中的等号 (=) 后面指定除空格或制表符以外的任何字符，并将空字符串或定义引在双引号 (" ") 内。若要取消定义宏，请使用 **!UNDEF**。有关更多信息，请参见生成文件预处理指令。

5.1.3 定义宏的位置

更新：2007 年 11 月

在命令行、命令文件、生成文件或 Tools.ini 文件中定义宏。

在生成文件或 Tools.ini 文件中，每个宏定义都必须显示在不同的行上，并且不能以空格或制表符开头。等号两旁的空格或制表符被忽略。所有字符串字符都是文本，包括两旁的引号和嵌入的空格。

在命令行或命令文件中，空格和制表符分隔参数并且不能出现在等号两旁。如果 *string* 有嵌入的空格或制表符，则将字符串本身或整个宏引在双引号 (" ") 内。

5.1.4 宏定义中的优先级

更新：2007 年 11 月

如果宏有多个定义，NMAKE 使用优先级最高的一个。以下列表由高至低显示了优先级的顺序：

1. 在命令行上定义的宏
2. 在生成文件或包含文件中定义的宏
3. 继承的环境变量宏
4. 在 Tools.ini 文件中定义的宏
5. 预定义的宏，如 CC 和 As

使用 /E 选项使从环境变量中继承的宏重写同名的生成文件宏。使用 **!UNDEF** 重写命令行。

5.2 使用 NMAKE 宏

更新：2007 年 11 月

若要使用宏，请将它的名称括在括号内并在前面加一个美元符号 ($)，如下所示。

语法

```
$(macroname)
```

备注

不允许有空格。如果 *macroname* 是单个字符，可以不使用括号。定义字符串替换 $(*macroname*)；未定义的宏由空字符串替换。

宏替换

5.2.1 宏替换

更新：2007 年 11 月

调用 *macroname* 时，其定义字符串中 *string1* 的每个匹配项由 *string2* 替换。

语法

```
$(macroname:string1=string2)
```

备注

宏替换区分大小写，并且是文本；*string1* 和 *string2* 不能调用宏。替换不修改原始定义。可以替换任何预定义宏（$$@除外）中的文本。

冒号前面没有空格或制表符；冒号后面的任何空格或制表符均被解释为文本。如果 *string2* 为空，将从宏的定义字符串中删除 *string1* 的所有匹配项。

5.3 特殊 NMAKE 宏

更新：2007 年 11 月

NMAKE 提供了几个特殊宏来表示不同的文件名和命令。其中某些宏的一个用途是用于预定义推理规则中。和所有的宏一样，NMAKE 提供的宏也区分大小写。

- 文件名宏
- 递归宏
- 命令宏和选项宏
- 环境变量宏

5.3.1 文件名宏

更新：2007 年 11 月

文件名宏被预定义为依赖项中指定的文件名（而不是磁盘上的完整文件名指定）。在调用时不需要将这些宏括在括号内；只需按如下方式指定 $。

宏	含义
$@	当前所指定的当前目标的全名（路径、基名称、扩展名）。
$$@	当前所指定的当前目标的全名（路径、基名称、扩展名）。仅在作为依赖项中的依赖项时有效。
$*	当前目标的路径和基名称，没有文件扩展名。
$**	当前目标的所有依赖项。
$?	时间戳比当前目标的时间戳晚的所有依赖项。
$<	时间戳比当前目标的时间戳晚的依赖文件。仅在推理规则的命令中有效。

若要指定部分预定义文件名宏，请追加宏修饰符并将修饰的宏括在括号内。

修饰符	结果文件名部分
D	驱动器和目录
B	基名称
F	基名称和扩展名
R	驱动器、目录和基名称

5.3.2 递归宏

更新：2007 年 11 月

使用递归宏递归调用 NMAKE。递归会话继承命令行和环境变量宏以及 Tools.ini 信息。它们不继承生成文件定义的推理规则或 .SUFFIXES 和 .PRECIOUS 规范。若要将宏传递到递归 NMAKE 会话，请要么在递归调用之前用 SET 设置环境变量，在命令中为递归调用定义宏，要么在 Tools.ini 中定义宏。

宏	定义
MAKE	最初用于调用 NMAKE 的命令。 $(MAKE) 宏提供 nmake.exe 的完整路径。
MAKEDIR	调用 NMAKE 时的当前目录。
MAKEFLAGS	当前有效的选项。用作 /$(MAKEFLAGS)。注意，未包括 /F。

5.3.3 命令宏和选项宏

更新：2007 年 11 月

为 Microsoft 产品预定义命令宏。选项宏表示这些产品的选项，并且在默认情况下不定义。两者都用于预定义的推理规则并可用于描述块或用户定义的推理规则。命令宏可以重新定义以表示部分或整个命令行，包括选项。如果不定义，选项宏生成空字符串。

Microsoft 产品	命令宏	定义为	选项宏
宏汇编	**AS**	ml	**AFLAGS**
Basic 编译器	**BC**	bc	**BFLAGS**
C 编译器	**CC**	cl	**CFLAGS**
C++ 编译器	**CPP**	cl	**CPPFLAGS**
C++ 编译器	**CXX**	cl	**CXXFLAGS**
资源编译器	**RC**	rc	**RFLAGS**

5.3.4 环境变量宏

更新：2007 年 11 月

NMAKE 继承会话开始前就存在的环境变量的宏定义。如果变量是在操作系统环境中设置的，则可以用作 NMAKE 宏。继承的名称被转换为大写。继承在预处理前发生。使用 /E 选项使从环境变量继承的宏重写生成文件中的任何同名宏。

可以在会话中重新定义环境变量宏，这将更改相应的环境变量。还可以用 SET 命令更改环境变量。但是，使用 SET 命令更改会话中的环境变量不会更改相应的宏。

例如：

```
PATH=$(PATH);\nonesuch

all:
    echo %PATH%
```

在该示例中，更改 PATH 从而更改了相应的环境变量 PATH；它将 \nonesuch 追加到路径中。

如果为环境变量定义的字符串的语法在生成文件中不正确，则没有宏创建，也没有警告生成。如果变量的值包含美元符号 ($)，则 NMAKE 将它解释为宏调用的开始。使用宏会导致意外行为。

6 推理规则

更新：2007 年 11 月

推理规则提供命令来更新目标并推理目标的依赖项。推理规则中的扩展名与具有相同基名称的单个目标和依赖项匹配。推理规则是用户定义的，或预定义的；预定义的规则可以重新定义。

如果过期的依赖项没有命令，并且如果 .SUFFIXES 包含依赖项的扩展名，则 NMAKE 使用其扩展名与当前或指定目录中的目标和现有文件匹配的规则。如果有多个规则与现有文件匹配，**.SUFFIXES** 列表将确定使用哪一个规则；列表优先级从左向右按降序排列。如果依赖文件不存在，并且未在另一个描述块中作为目标列出，则推理规则可以从具有相同基名称的另一个文件创建缺少的依赖项。如果描述块的目标没有依赖项或命令，推理规则可以更新目标。即使不存在描述块，推理规则也可以生成命令行目标。即使指定了显式依赖项，NMAKE 也可以调用推理依赖项的规则。

6.1 定义规则

更新：2007 年 11 月

fromext 表示依赖文件的扩展名，*toext* 表示目标文件的扩展名。

```
.fromext.toext:
    commands
```

备注

扩展名不区分大小写。可以调用宏来表示 *fromext* 和 *toext*；宏在预处理期间展开。*fromext* 前的句点 (.) 必须出现在行首。冒号 (:) 前面是零个或零个以上的空格或制表符。后面只能跟空格或制表符、分号 (;)（用于指定命令）、数字符号 (#)（用于指定注释）或换行符。不允许使用其他的空格。和在描述块中一样指定命令。

6.1.1 规则中的搜索路径

更新：2007 年 11 月

```
{frompath}.fromext{topath}.toext:
    commands
```

备注

只有当依赖项中指定的路径与推理规则路径完全匹配时，推理规则才应用于依赖项。在 *frompath* 中指定依赖项的路径，在 *topath* 中指定目标的目录；不允许有空格。为每个扩展名只指定一个路径。一个扩展名的路径需要另一个扩展名的路径。若要指定当前目录，请使用句点 (.) 或空的大括号 ({ })。宏可以表示 *frompath* 和 *topath*；在预处理期间调用。

示例 **代码**

```
{dbi\}.cpp{$(ODIR)}.obj::
        $(CC) $(CFLAGS) $(YUDBI) $<

{ilstore\}.cpp{$(ODIR)}.obj::
        $(CC) $(CFLAGS) $<

{misc\}.cpp{$(ODIR)}.obj::
        $(CC) $(CFLAGS) $(YUPDB) $<

{misc\}.c{$(ODIR)}.obj::
```

```
        $(CC) $(CFLAGS) $<

{msf\}.cpp{$(ODIR)}.obj::
        $(CC) $(CFLAGS) $<

{bsc\}.cpp{$(ODIR)}.obj::
        $(CC) $(CFLAGS) $(YUPDB) $<

{mre\}.cpp{$(ODIR)}.obj::
        $(CC) $(CFLAGS) $(YUPDB) $<

{namesrvr\}.cpp{$(ODIR)}.obj::
        $(CC) $(CFLAGS) $(YUPDB) $<

{src\cvr\}.cpp{$(ODIR)}.obj::
        $(CC) $(CFLAGS) $<
```

6.2 批模式规则

更新：2007 年 11 月

```
{frompath}.fromext{topath}.toext::
   commands
```

当 N 条命令通过推理规则时，批模式推理规则只调用一次该推理规则。如果没有批模式推理规则，将需要调用 N 条命令。N 是触发推理规则的依赖项的数目。

包含批模式推理规则的生成文件必须使用 NMAKE 1.62 版或更高版。若要检查 NMAKE 版本，请运行 NMAKE 1.62 版或更高版可用的 _NMAKE_VER 宏。此宏返回表示 Visual C++ 产品版本的字符串。

与标准推理规则相比，唯一的语法差别是：批模式推理规则以两个冒号 (::) 终止。

说明：

正在调用的工具必须能够处理多个文件。批模式推理规则必须将 $< 用作宏来访问依赖文件。

批模式推理规则可以加快生成过程。因为编译器驱动程序只调用一次，所以按批给编译器提供文件更快。例如，C 和 C++ 编译器在处理一组文件时性能更佳，因为它可以在处理期间使内存保持驻留。

下面的示例显示了如何使用批模式推理规则：

```
#
# sample makefile to illustrate batch-mode inference rules
#
O = .
S = .
Objs = $O/foo1.obj $O/foo2.obj $O/foo2.obj $O/foo3.obj $O/foo4.obj
CFLAGS = -nologo

all : $(Objs)

!ifdef NOBatch
{$S}.cpp{$O}.obj:
!else
{$S}.cpp{$O}.obj::
!endif
   $(CC) $(CFLAGS) -Fd$O\ -c $<

$(Objs) :

#end of makefile
```

如果不使用批模式推理规则，NMAKE 产生以下输出：

```
E:\tmp> nmake -f test.mak -a NOBatch=1

Microsoft (R) Program Maintenance Utility    Version 7.00.0000
Copyright (C) Microsoft Corp 1988-2001. All rights reserved.
        cl -nologo -Fd.\ -c .\foo1.cpp
foo1.cpp
        cl -nologo -Fd.\ -c .\foo2.cpp
foo2.cpp
        cl -nologo -Fd.\ -c .\foo3.cpp
foo3.cpp
        cl -nologo -Fd.\ -c .\foo4.cpp
foo4.cpp
```

使用批模式推理规则，NMAKE 产生以下结果：

```
E:\tmp> nmake -f test.mak -a

Microsoft (R) Program Maintenance Utility    Version 7.00.0000
Copyright (C) Microsoft Corp 1988-2001. All rights reserved.

        cl -nologo -Fd.\ -c .\foo1.cpp .\foo2.cpp .\foo3.cpp .\foo4.cpp
foo1.cpp
foo2.cpp
foo3.cpp
foo4.cpp
Generating Code...
```

6.3 预定义的规则

更新：2007 年 11 月

预定义的推理规则使用 NMAKE 提供的命令和选项宏。

规则	命令	默认操作	批处理规则	nmake 运行的平台
.asm.exe	$(AS) $(AFLAGS) $<	ml $<	否	x86
.asm.obj	$(AS) $(AFLAGS) /c $<	ml /c $<	是	x86
.asm.exe	$(AS) $(AFLAGS) $<	ml64 $<	否	x64
.asm.obj	$(AS) $(AFLAGS) /c $<	ml64 /c $<	是	x64
.s.obj	$(AS) $(AFLAGS) $<	ias /c $<	否	Itanium 处理器系列 (IPF)
.c.exe	$(CC) $(CFLAGS) $<	cl $<	否	所有
.c.obj	$(CC) $(CFLAGS) /c $<	cl /c $<	是	所有
.cc.exe	$(CC) $(CFLAGS) $<	cl $<	否	所有
.cc.obj	$(CC) $(CFLAGS) /c $<	cl /c $<	是	所有
.cpp.exe	$(CPP) $(CPPFLAGS) $<	cl $<	否	所有
.cpp.obj	$(CPP) $(CPPFLAGS) /c $<	cl /c $<	是	所有
.cxx.exe	$(CXX) $(CXXFLAGS) $<	cl $<	否	所有
.cxx.obj	$(CXX) $(CXXFLAGS) /c $<	cl /c $<	是	所有
.rc.res	$(RC) $(RFLAGS) /r $<	rc /r $<	否	所有

6.4 推导出的依赖项和规则

更新：2007 年 11 月

如果存在适用的推理规则，NMAKE 则为目标采用推理的依赖项。规则适用下列情况：

- *toext* 与目标的扩展名匹配。
- *fromext* 与具有目标的基名称并存在于当前或指定目录中的文件的扩展名匹配。
- *fromext* 位于 .SUFFIXES 中；匹配规则中没有其他的 *fromext* 有更高的 **.SUFFIXES** 优先级。
- 没有显式依赖项有更高的 **.SUFFIXES** 优先级。

推理的依赖项会导致意外的副作用。如果目标的描述块包含命令，NMAKE 将执行这些命令而不是规则中的命令。

6.5 推理规则中的优先级

更新：2007 年 11 月

如果推理规则有多个定义，NMAKE 使用优先级最高的定义。以下列表由高至低显示了优先级的顺序：

1. 在生成文件中定义的推理规则；越晚定义，优先级越高。
2. 在 Tools.ini 中定义的推理规则；越晚定义，优先级越高。
3. 预定义的推理规则。

7 点指令

更新：2007 年 11 月

在描述块之外的行首指定点指令。点指令以一个句点（.）开头，后跟一个冒号（:）。允许使用空格或制表符。点指令名区分大小写并且应为大写。

指令	用途
.IGNORE :	忽略从指定该指令的位置到生成文件末尾之间，由命令返回的非零退出代码。默认情况下，如果命令返回非零退出代码，NMAKE 将暂停。若要还原错误检查，请使用 **!CMDSWITCHES**。若要忽略单个命令的退出代码，请使用短划线 (-) 修饰符。若要忽略整个文件的退出代码，请使用 /I 选项。
.PRECIOUS :*targets*	若更新 *targets* 的命令暂停，则将 *targets* 保留在磁盘上；若命令通过删除文件处理中断，则该指令无效。用一或多个空格或制表符分隔目标名称。默认情况下，如果通过使用 Ctrl+C 或 Ctrl+Break 组合键中断生成，NMAKE 将删除目标。**.PRECIOUS** 的每一次使用都应用于整个生成文件；多次指定是累计的。
.SILENT :	取消从指定该指令的位置到生成文件末尾之间的已执行命令的显示。默认情况下，NMAKE 显示它调用的命令。若要还原回显，请使用 **!CMDSWITCHES**。若要取消单个命令的回显，请使用 **@** 修饰符。若要取消整个文件的回显，请使用 /S 选项。
.SUFFIXES :*list*	列出推理规则匹配的扩展名；预定义包括以下扩展名： .exe .obj .asm .c .cpp .cxx .bas .cbl .for .pas .res .rc .f .f90

若要更改 **.SUFFIXES** 列表顺序或指定新列表，请清除此列表并指定新的设置。若要清除此列表，请不要在冒号后指定扩展名：

```
.SUFFIXES :
```

若要将其他后缀添加到列表的末尾，请指定

```
.SUFFIXES : suffixlist
```

其中 *suffixlist* 是附加后缀的列表，由一或多个空格或制表符分隔。若要查看 **.SUFFIXES** 的当前设置，请运行选项为 /P 的 NMAKE。

8 生成文件预处理

更新：2007 年 11 月

可以通过使用预处理指令和表达式控制 NMAKE 会话。预处理指令可以放置在生成文件或 Tools.ini 文件中。使用指令可以有条件地处理生成文件，显示错误信息，包括其他生成文件，取消定义宏以及打开或关闭某些选项。

8.1 生成文件预处理指令

更新：2007 年 11 月

预处理指令不区分大小写。初始感叹号 (!) 必须出现在行首。感叹号后面可以有零个或多个空格或制表符，用于缩进。

!CMDSWITCHES

{**+**| **-**}*option*...打开或关闭列出的每个 *option*。空格或制表符必须出现在 + 或 - 运算符前面；运算符和选项字母之间不能出现任何内容。字母不区分大小写，并且不用反斜杠 (/) 指定。若要打开某些选项而关闭另外一些选项，请使用 **!CMDSWITCHES** 的分别指定。

生成文件中只能使用 /D、/I、/N 和 /S 选项。在 Tools.ini 文件中可以使用除 /F、/HELP、/NOLOGO、/X 和 /? 选项外的其他所有选项。一个描述块中指定的更改直到下一个描述块时才生效。该指令更新 **MAKEFLAGS**；如果指定了 **MAKEFLAGS**，则在递归期间继承更改。

!ERROR *text*

显示错误 U1050 中的 *text*，然后暂停 NMAKE，即便使用了 /K、/I、**.IGNORE**、**!CMDSWITCHES** 或短划线 (-) 命令修饰符。位于 *text* 之前的空格或制表符被忽略。

!MESSAGE *text*

显示标准输出的 *text*。位于 *text* 之前的空格或制表符被忽略。

!INCLUDE [<]*filename*[>]

将 *filename* 作为生成文件读取，然后继续当前的生成文件。NMAKE 首先在指定或当前目录中搜索 *filename*，然后在任何父生成文件的目录中递归搜索，最后，如果 *filename* 括在尖括号 (< >) 内，则在由 **INCLUDE** 宏（最初设置为 INCLUDE 环境变量）指定的目录中搜索。对于将 **.SUFFIXES** 设置、**.PRECIOUS** 和推理规则传递给递归生成文件很有用。

!IF *constantexpression*

如果 *constantexpression* 计算结果为非零值，则处理 **!IF** 和下一个 **!ELSE** 或 !ENDIF 之间的语句。

!IFDEF *macroname*

如果定义了 *macroname*，则处理 !IFDEF 和下一个 **!ELSE** 或 !ENDIF 之间的语句。空宏将被视为尚待定义。

!IFNDEF *macroname*

如果没有定义 *macroname*，则处理 **!IFNDEF** 和下一个 **!ELSE** 或 !ENDIF 之间的语句。

!ELSE[**IF** *constantexpression* | **IFDEF** *macroname*| **IFNDEF** *macroname*]

如果前面的 **!IF**、!IFDEF 或 **!IFNDEF** 语句计算结果为零值，则处理 **!ELSE** 和下一个 !ENDIF 之间的语句。可选关键字提供了进一步的预处理控制。

!ELSEIF

!ELSE IF 的同义词。

!ELSEIFDEF

!ELSE IFDEF 的同义词。

!ELSEIFNDEF

!ELSE IFNDEF 的同义词。

!ENDIF

标记 **!IF**、!IFDEF 或 **!IFNDEF** 块的结尾。同一行上 !ENDIF 后面的所有文本被忽略。

!UNDEF *macroname*

取消定义 *macroname*。

8.2 生成文件预处理中的表达式

更新：2007 年 11 月

!IF 或 **!ELSE IF***constantexpression* 由整型常数（使用十进制或 C 语言表示法）、字符串常数或命令组成。使用括号将表达式分组。表达式使用 C 样式的有符号长整型算法；数字为 32 位 2 的补数形式，范围在 - 2147483648 到 2147483647 之间。

表达式可以使用在常数值、命令的退出代码、字符串、宏和文件系统路径上使用的运算符。

生成文件预处理运算符

在预处理中执行程序

8.2.1 生成文件预处理运算符

更新：2007 年 11 月

DEFINED 运算符是宏名称上使用的逻辑运算符。如果定义了 *macroname*，则表达式 **DEFINED** (*macroname*) 为真。**DEFINED** 与 **!IF** 或 **!ELSE IF** 的组合等效于 !IFDEF 或 **!ELSE IFDEF**。但是，**DEFINED** 与这些指令不同，它可以用于使用二进制逻辑运算符的复杂表达式。

EXIST 运算符是文件系统路径上使用的逻辑运算符。如果 *path* 存在，则 **EXIST** (*path*) 为真。**EXIST** 的结果可用于二进制表达式。如果 *path* 包含空格，则用双引号将它引起来。

整型常数可以将一元运算符用于数字求反 (-)、1 的补数 (~) 和逻辑求反 (**!**)。

常数表达式可以使用下列二进制运算符。

运算符	说明	运算符	说明
+	加法	||	逻辑"或"
–	减法	**<<**	左移
*****	乘法	**>>**	右移
/	除法	==	相等
%	模数	!=	不相等
&	按位"与"	**<**	小于
|	按位"或"	**>**	大于
^	按位"异或"	**<=**	小于或等于
&&	逻辑"与"	**>=**	大于或等于

若要比较两个字符串，请使用相等 (==) 运算符和不相等 (!=) 运算符。用双引号将字符串引起来。

8.2.2 在预处理中执行程序

更新：2007 年 11 月

若要在预处理期间使用命令的退出代码，请在中括号 ([ ]) 内用任意参数指定命令。任何宏都在命令执行前展开。NMAKE 用命令的退出代码替换命令指定，此代码可用在控制预处理的表达式中。