
GNU make 在两个不同的阶段完成它的工作。 在第一阶段，它读取所有 makefile、包含的 makefile 等，并内化所有变量及其值以及隐式和显式规则，并构建所有目标及其先决条件的依赖关系图。 在第二阶段，make 使用这些内部化数据来确定需要更新哪些目标并运行更新它们所需的配方。

理解这种两阶段方法很重要，因为它直接影响变量和函数扩展的发生方式； 在编写 makefile 时，这通常是一些混乱的根源。 以下是可以在 makefile 中找到的不同构造的摘要，以及构造的每个部分发生扩展的阶段。

如果它发生在第一阶段，我们说扩展是立即的：make 将在解析 makefile 时扩展构造的那部分。 我们说，如果不是立即扩张，扩张就会被推迟。 延迟构造部分的扩展会延迟到使用扩展之前：无论是在直接上下文中引用它时，还是在第二阶段需要它时。

您可能还不熟悉其中的一些构造。 当您熟悉它们时，您可以在后面的章节中参考本节。

Variable Assignment (变量赋值)

变量定义解析如下：

```
immediate = deferred
immediate ?= deferred
immediate := immediate
immediate ::= immediate
immediate += deferred or immediate
immediate != immediate

define immediate
  deferred
endef

define immediate =
  deferred
endef

define immediate ?=
  deferred
endef

define immediate :=
  immediate
endef

define immediate ::=
  immediate
endef
```

```
define immediate +=
  deferred or immediate
endef

define immediate !=
  immediate
endef
```

对于附加运算符 '+=' ，如果变量先前设置为简单变量（ ':=' 或 '::=' ），则右侧被视为立即，否则为延迟。

对于 shell 赋值运算符 '!='，右侧立即计算并交给 shell。 结果存储在左侧命名的变量中，该变量成为一个简单变量（因此将在每个引用上重新评估）。

Conditional Directives（条件指令）

立即解析条件指令。 这意味着，例如，自动变量不能在条件指令中使用，因为在调用该规则的配方之前不会设置自动变量。 如果您需要在条件指令中使用自动变量，则必须将条件移动到配方中并改用 shell 条件语法。

Rule Definition（规则定义）

无论形式如何，规则总是以相同的方式展开：

```
immediate : immediate ; deferred
        deferred
```

也就是说，目标和先决条件部分立即展开，而用于构建目标的配方总是延迟。 这适用于显式规则、模式规则、后缀规则、静态模式规则和简单的先决条件定义。