

(make) makefile 概览

手稿

封面

本节简要介绍 makefile 的构成要素，以及 make 的简单运行。

概述

make 需要一个名为 makefile 的文件来告诉它如何工作。大多数情况下，makefile 被用于向 make 说明怎样编译并链接程序。

一个指明如何编译程序的 makefile 有多种使用方式，最简单的方式就是重新编译每个发生变化（out of date）的文件。通常，makefile 被写成当不带参数运行 make 时，就按照上述方式执行。

makefile 的构成要素

一个简单的 `makefile` 由以下形式的“规则”组成：

一条规则

目标 ... : 先决条件 ...

一份划单

命令

...

...

冒号 (:) 前面可以有多个目标 (target)，后面是这个目标的先决条件 (prerequisites)。

再下面是划单 (recipe, 计划清单)，它以一个水平制表符开始。

构成要素的概念

目标 通常是程序生成的文件的名称，例如可执行文件和对象文件（*.o）。目标也可以是待执行动作的名称，比如 `clean`（伪目标）。

先决条件 是创建目标时用作输入的文件，一个目标常常依赖于几个文件。

划单 即 `make` 执行的动作。一份划单可能有多个命令，要么位于同一行，要么单独一行。缺省情况下，划单以一个水平制表符开始，它看起来不太明显。如果你更喜欢用其他字符来作为划单的起始符，可以将变量 `.RECIPEPREFIX` 设置为其他代替字符。

规则的职责

通常，划单位于带有先决条件的规则中，如果任何先决条件发生了变化，它负责创建目标文件。不过，为目标指定了划单的规则并不一定要有先决条件。举例来说，包含与目标 `clean` 有关删除命令的规则就不带先决条件。

那么，规则就解释了如何以及何时重新生成特定文件，即特定规则的目标。`make` 在先决条件基础上执行划单来创建或更新目标。规则还可以解释如何以及何时执行动作。

除了规则之外，`makefile` 还可以包含其他文本，但对于相对简单的 `makefile` 来说，只需包含规则就够了。一些大型 `makefile` 可能看起来很复杂，但抽丝剥茧后你会发现，它们大致上仍然遵循的是前面的规则样式。

示例 makefile

对于下面一个示例 `makefile` 来说：

```
1 .PHONY: clean
2
3 CC = gcc
4 CFLAGS = -c -Wall -g
5
6 all: clean app
7
8 app : function_parameter.o
9      $(CC) $(LDFLAGS) -o $@ $^
10
11 %.o: %.c
12      $(CC) $(CFLAGS) -o $@ $<
13
14 clean:
15      rm -f app *.o
16
```

如果要生成目标 `app`，我们只需在命令行上键入命令：`make`

`make` 将第一个目标 `all` 作为标的，由于它的先决条件中包含 `clean` 和 `app`，因此 `make` 首先删除 `app` 和对象文件（`*.o`），然后重新生成 `app`。

指定 make 选项

有时候，你可能：

- ✦ 只想更新部分文件
- ✦ 使用不同的编译器或编译器选项
- ✦ 只是找到发生了变化但不改变它们

这种情况下，通过在运行 `make` 时指定选项，你可以实现上面任何一种需求，并且还能做更多事情。

举例来说，如果你想删除示例程序先前生成的可执行程序 `app` 和对象文件 (`*.o`)，那么可以在运行 `make` 时显式指定目标 `clean`。

```
$ make clean  
rm -f app *.o
```

make 的退出状态

make 的退出状态只有 3 种，它们是：

0

表示 make 成功运行

2

make 遇到了错误，它将打印出描述特定错误的消息。

1

指定了 -q 选项，而 make 发现有些目标已经不是最新的。

举例来说：

```
$ make
rm -f app *.o
gcc -c -Wall -g -o function_parameter.o function_parameter.c
gcc -o app function_parameter.o
```

```
$ echo $?
0
```

```
$ make clen
make: *** No rule to make target 'clen'.  Stop.
```

```
$ echo $?
2
```