Makefile 教程

- 感谢徐海兵翻译
- 主讲人：秦风岭

起源

- 目前虽有众多依赖关系检查工具，但是 make 是应用最广泛的一个。这要归功于它被包含在 Unix 系统中。斯图亚特 · 费尔德曼在 1977 年在贝尔实验室里制作了这个软件。2003 年，斯图亚特 · 费尔德曼因发明了这样一个重要的工具而接受了美国计算机协会（ ACM ）颁发的软件系统奖。

  在 make 诞生之前， Unix 系统的编译系统主要由“ make”、“ install”shell 脚本程序和程序的源代码组成。它可以把不同目标的命令组成一个文件，而且可以抽象化依赖关系的检查和存档。这是向现代编译环境发展的重要一步。

不同版本

- GNU make

  GNU make 常和 GNU 编译系统一起被使用，是大多数 GNU Linux 安装的一部分。
- BSD make

  它编译目标的时候有并行计算的能力。它在 FreeBSD， NetBSD 和 OpenBSD 中不同程度的修改下存活了下来。
- Microsoft nmake

  广泛应用于微软的 Windows 。
- GNU make vs BSD make

  http://lists.freebsd.org/pipermail/freebsd-questions/2007-April/147533.html

其他软件构建工具介绍

ORIENT VIEW TECHNOLOGY

- Cmake

  “CMake”【“ cross platform make”】是一个跨平台的安装(编译)工具。为了解决美国国家医学图书馆出资的 Visible Human Project 专案下的 ITK 软件跨平台建构的需求而创造出来的， 2000 年中开始开发，它能够输出各种各样的 makefile 或 project 文件 ,Cmake 并不直接建构出最终的软件，而是产生标准的建构档（如 Unix 的 Makefile 或 Windows Visual C++ 的 projects/workspaces ），然后再依一般的建构方式使用。 http://www.cmake.org/

  使用案例： KDE,OPENCV,MySql

- Scons

  它是一个开放源代码计划，使用 Python 语言开发。第一个正式版本在 2010 年 3 月 23 日释出。 Scons 是下一代软件自动构建工具。跨平台，只需要编写一个 SConstruct 文件，根据此文件， scons 可以自动完成依赖关系的推导及编译链接等过程。 http://www.scons.org/

  使用案例： MongoDB,V8(Google's open source JavaScript engine),KDevelop

扫盲篇

- 编译：

  把高级语言书写的代码转换为机器可识别的机器指令。

  编译高级语言后生成的指令虽然可被机器识别，但是还不能被执行。编译时，编译器检查高级语言的语法、函数与变量的声明是否正确。只有所有的语法正确、相关变量定义正确编译器就可以编译出中间目标文件。通常，一个高级语言的源文件都可对应一个目标文件。目标文件在 Linux 中默认后缀为”.o”, 在 Windows 中默认后缀为”.obj”

- 例子：

  gcc -c hello.c -o hello.o

  file hello.o

- 链接：

  将多个 .o 文件和库文件链接成为可被操作系统执行的可执行程序 (Linux 环境下，可执行文件的格式为“ ELF” 格式， Window 环境下，可执行文件的格式为” EXE” 格式)。链接器不检查函数所在的源文件，只检查所有 .o 文件中的定义的符号。将 .o 文件中使用的函数和其它 .o 或者库文件中的相关符号进行合并，对所有文件中的符号进行重定位，并链接系统相关文件 (程序启动文件等) 最终生成可执行程序。链接过程使用 GNU 的“ ld” 工具。

- 例子：

  gcc -o hello hello.o

  file hello

- 静态库：

  称为文档文件或者归档文件 (Archive File)。它是多个 .o 文件的集合。 Linux 中静态库文件的后缀为“ .a”。 Windows 中静态库文件的后缀为”.lib”，静态库中的各个成员 (.o 文件) 没有特殊的存在格式，仅仅是一个 .o 文件的集合。使用“ ar” 工具维护和管理静态库。

- 例子：

  建立库 ----ar crv hello.lib hello.o

  解开库 ----ar x hello.lib

  file hello.lib

- 共享库：

  也是多个 .o 文件的集合，但是这些 .o 文件时有编译器按照一种特殊的方式生成 (Linux 中，共享库文件格式通常为“ ELF” 格式。共享库已经具备了可执行条件）。模块中各个成员的地址（变量引用和函数调用）都是相对地址。使用此共享库的程序在运行时，共享库被动态加载到内存并和主程序在内存中进行连接。多个可执行程序可共享库文件的代码段（多个程序可以共享的使用库中的某一个模块，共享代码，不共享数据）。另外共享库的成员对象可被执行（由 libdl.so 提供支持）。 Linux 中后缀名为”.so”,Windows 中后缀名为”.dll”。

- 例子：

  gcc --shared -o hello.so hello.o

  file hello.so

- GNU binutils 是一组二进制工具集。 [以后详细介绍]
  包括： ld,as,addr2line,ar,c++filt,dlltool,gold,gprof,nlmconv,nm,objcopy,objdump,ranlib,readelf,size,strings,strip,windmc,winres
- 常用简介
  ld - the GNU linker.（连接器）
  as - the GNU assembler.（汇编器）
  addr2line - Converts addresses into filenames and line numbers.
  ar - A utility for creating, modifying and extracting from archives.
  gprof - Displays profiling information.（静态性能分析工具）
  nm - Lists symbols from object files.
  objcopy - Copys and translates object files.
  objdump - Displays information from object files.
  ranlib - Generates an index to the contents of an archive.
  readelf - Displays information from any ELF format object file.
  strip - Discards symbols.
- 官方地址： http://www.gnu.org/software/binutils/

- GNU GCC

  GCC（ GNU Compiler Collection， GNU 编译器套装）
- 常用编译选项：（注意大小写）

  -E

  只激活预处理，这个不生成文件，你需要把他重定向到一个输出文件里面 .

  例子用法 :

  gcc -E hello.c > gcc_e.txt

  -S

  只激活预处理和编译，就是指把文件编 译成为汇编代码。

  例子用法

  gcc -S hello.c

  它将生成 .s 的汇编代码，你能用文本编辑器察看。学习汇编的好方法。

-c

只激活预处理，编译，和汇编，也就是生成 .o 文件

例子用法：

gcc -c hello.c -o hello.o

-o

输出目标名称，默认情况下 gcc 编译出来的文件是 a.out

例子用法

gcc hello.c

gcc -o hello.exe hello.c

gcc -o hello.asm -S hello.c

- 和依赖生成相关的选项

  -M

  生成文件关联的信息。生成目标文件所依赖的所有文件。

  例子用法

  gcc -M hello.c

  -MD

  和 -M 相同，不过输出将导入到 .d 的文件里面

  例子用法

  gcc -MD hello.c

-llibrary

指定编译的时候使用的库

-Ldir

指定编译的时候，搜索库的路径。比如你自己的库，能用他制定目录，不然编译器将只在标准库的目录找。这个 dir 就是目录的名称。

例子用法

```
cd hello_lib/lib
gcc -c welcome.c -o welcome.o
ar crv libwelcome.a welcome.o
cd ..
gcc hello.c -Llib -lwelcome
```

- 优化选项相关

  -O0 -O1 -O2 -O3

  编译器的优化选项的 4 个级别， -O0 表示没有优化 ,-O1 为缺省值， -O3 优化级别最高
- 比较个别的优化选项 -Os

  优化级别 2.5

- 调试信息相关的编译选项

  -g

  指示编译器，在编译的时候，产生调试信息。

  -ggdb

  此选项将尽可能的生成 gdb 的能使用的调试信息 .

- -static

  此选项将禁止使用动态库，所以，编译出来的东西，一般都非常大，不需要系统的动态库，就能运行 .

  -shared

  此选项将尽量使用动态库，所以生成文件比较小，不过需要系统有动态库 .

  -Wall

  显示所有警告信息 (warning all)

Lex 和 Yacc

- Lex:

  Lex 是 LEXical compiler 的缩写，主要功能是生成一个词法分析器 (scanner) 的 C 源码 , 描述规则采用正则表达式 (regular expression)。描述词法分析器的文件 *.l，经过 lex 编译后，生成一个 lex.yy.c 的文件，然后由 C 编译器编译生成一个词法分析器。词法分析器，简单来说，其任务就是将输入的各种符号，转化成相应的标识符，转化后的标识符很容易被后续阶段处理。对于 C 语言来说，它类似与关键字。

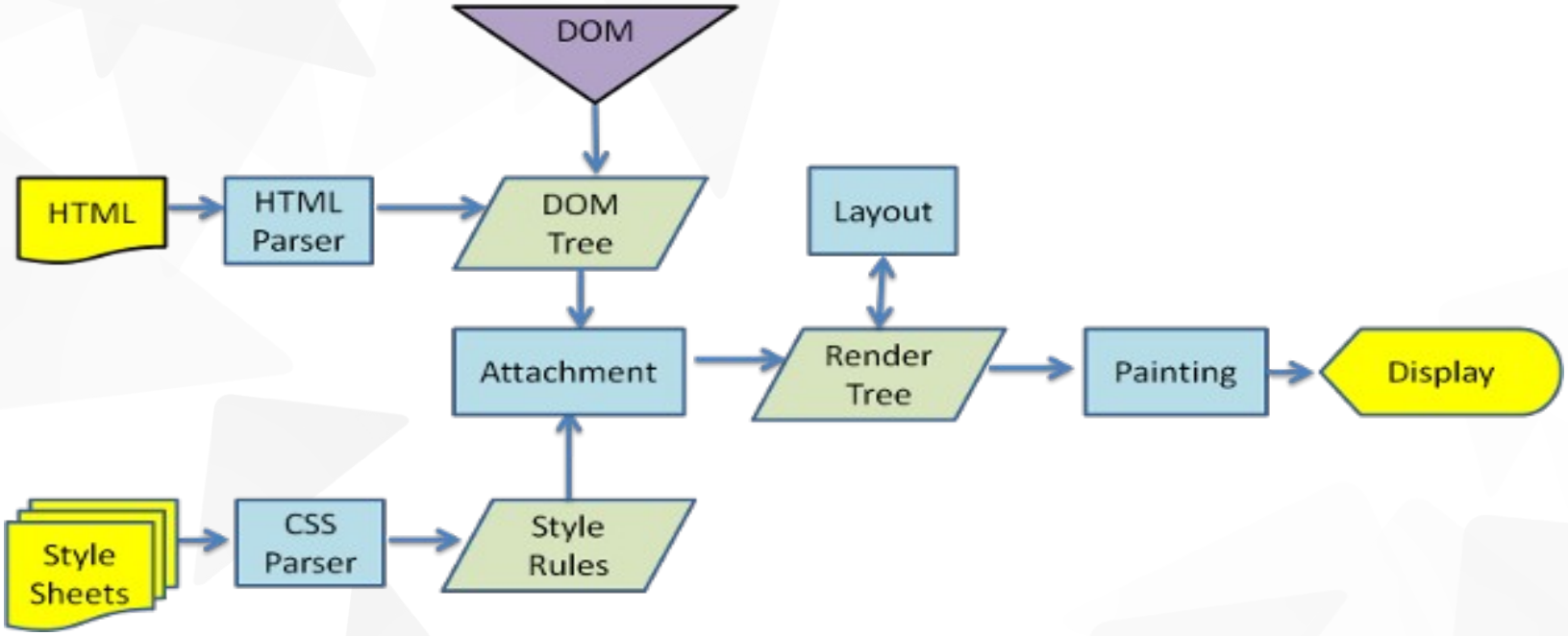
- Yacc:

  是 Unix/Linux 上一个用来生成编译器的编译器（编译器代码生成器）。 yacc 生成的编译器主要是用 C 语言写成的语法解析器（ Parser ），需要与词法解析器 Lex 一起使用，根据语言的语法规则分析文档结构，从而构建解析树。

- 参考文献：

  https://www.ibm.com/developerworks/cn/linux/sdk/lex/
  http://hi.baidu.com/hins_pan/blog/item/54dfe47696b65c04b151b97c.html

- Lex 和 Yacc 应用案例：
  Webkit 中解析（ HTML， CSS ）与 DOM 树构建。



参考文献：

前端必读：浏览器内部工作原理
http://kb.cnblogs.com/page/129756/#chapter3

- 学会自己查找帮助
- man

  man 工具可以显示系统手册页中的内容，这些内容大多数都是对命令的解释信息。通过查看系统文档中的 man 页可以得到程序的更多相关主题信息和 Linux 的更多特性。
- 例子：

  1.man man 【 cp,fork,fprintf,/dev/tty,passwd 】 -a passwd

  2.Ubuntu 安装 manpage

  命令： sudo apt-get install manpages manpages-posix manpages-posix-dev

  说明：

  manpages 包含 GNU/Linux 的基本操作。

  manpages-posix 包含 POSIX 所定义公用程序的方法。

  manpages-posix-dev 包含 POSIX 的 header files 和 library calls 的用法。

  注：中文 manpage

  sudo apt-get install manpage-zh

- Info

  Info 工具是一个基于菜单的超文本系统，由 GNU 项目开发并由 Linux 发布。 info 工具包括一些关于 Linux shell、工具、 GNU 项目开发程序的说明文档。如在命令行中键入： info fprintf

  当出现帮助信息内容后：

  按下？键，可以列出 info 窗口中的相关命令。

  按下 SPACE 键，可以在菜单项中进行滚动浏览。

- “--help” 选项

  “--help” 是一个工具选项，大部分的 GNU 工具都具备这个选项，“ --help” 选项可以用来显示一些工具的信息，如在命令行中键入： fdisk –help

- /usr/share/doc

  大多数软件包将自己的文档放在 /usr/share/doc 的某个子目录中，一般子目录名与特定的软件包相同。例如： /usr/share/doc/nfs-common/README.Debian.nfsv4

概述

- Linux 环境下如使用 GNU Make 来构建和管理工程，需要我们投入一些时间完成一个或者多个 Makefile 的编写。 Makefile 文件描述了整个工程编译，链接等规则。其中包括：工程中的哪些源文件需要编译以及如何编译、需要创建那些库文件以及如何创建这些库文件、如何产生我们最后想要的可执行文件。使用正确的 Makefile ， 编译工程仅需要在 shell 下输入 make 即可。

- make 是一个命令工具，它解释 Makefile 中的规则。在 Makefile 文件中描述了整个工程所有文件的编译顺序、编译规则。Makefile 有自己的书写格式、关键字、函数。而且在 Makefile 中可以使用系统 shell 所提供的任何命令来完成想要的工作。

Make 做了什么

1. 所有的源文件没有被编译过，则对各个 C 源文件进行编译并进行链接，生成最后的可执行程序；
2. 每一个在上次执行 make 之后修改过的 C 源代码文件在本次执行 make 时将会被重新编译；
3. 头文件在上一次执行 make 之后被修改。则所有包含此头文件的 C 源文件在本次执行 make 时将会被重新编译。

Makefile 规则介绍

- Makefile 规则：

  TARGET... : PREREQUISITES...

  COMMAND

  ...

  ...

  TARGET: 规则的目标。通常是最后需要生成的文件名或者为了实现这个目的而必需的中间过程文件名。可以是 .o 文件、也可以是最后的可执行程序的文件名等。另外，目标也可以是一个 make 执行的动作的名称，如目标“ clean”，我们称这样的目标是“伪目标”。

PREREQUISITES: 规则的依赖。生成规则目标所需要的文件名列表。通常一个目标依赖于一个或者多个文件。

COMMAND: 规则的命令行。是规则所要执行的动作 (任意的 shell 命令或者是可在

shell 下执行的程序) 。它限定了 make 执行这条规则时所需要的动作。

注意 :

每一个命令行必须以 [Tab] 字符开始 ,[Tab] 字符告诉 make 此行是一个命令行。

Makefile 的例子

- 需要演示 Makefile 的编译三种情况。

  参照 hello_make

  1. 编译 hello.c welcome.c
  2. 仅编译 hello.c 【改动 hello.c,hello.h 】
  3. 仅编译 welcome.c 【改动 welcome.c,welcome.h 】

make 如何工作

默认的情况下 ,make 执行的是 Makefile 中的第一个规则 , 此规则的第一个目标称之为“最终目的”或者“终极目标”。

Makefile 中的变量

```
OBJS = hello.o welcome.o
SRC = hello.c welcome.c
OUTPUT = hello
```

自动推导规则

在使用 make 编译 .c 源文件时，
编译 .c 源文件规则的命令可以不用明确给出。
这是因为 make 本身存在一个默认的规则，能够自动完成对 .c 文件的编译并生成对应的 .o 文件。
它执行命令“ cc -c” 来编译 .c 源文件。

Makefile 的书写规则

书写规则建议的方式是 : 单目标 , 多依赖。就是说尽量要做到一个规则中只存在一个目标文件 , 可有多个依赖文件。尽量避免使用多目标 , 单依赖的方式。

例子：

多目标的坏处。

记得清理垃圾

清除当前目录中编译过程中产生的临时文件

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

1. 通过“ .PHONY” 将“ clean” 目标声明为伪目标。避免当磁盘上存在一个名为“ clean” 文件时 , 目标“ clean” 所在规则的命令无法执行。

2. 在命令行之前使用“ -”, 意思是忽略命令“ rm” 的执行错误。

谢谢大家！

- 联系方式
- 手机： 15010404682
- 邮箱： qin_fengling@126.com
- QQ： 415508683