GNU 编译工具

autoconf

1 介绍

一个物理学家，一个工程师，和一个计算机科学家正在讨论上帝的天性。“他当然是一个物理学家”，物理学家说，“因为创造万物之初，上帝就制造了光；还有你们知道的，麦克斯韦的方程式，电磁波的二重性；还有相对论…”“他是工程师！”工程师说，“因为创造光之前，他得从混沌中分离出陆地和水；这得让一个工程师艰难的处理大量的泥巴，有序的分离出固体和液体…”计算机科学家叫道“混沌，你们认为它从什么地方来呢，嗯？”

------匿名者

Autconf 是一个工具，用来帮助生成能够自动配置软件源码包的命令行脚本，这个脚本适应众多类 Unix 的系统。由 autoconf 生成的脚本在运行时并不依赖于 autoconf，所以脚本的最终使用者不需要安装 autoconf。

autoconf,automake,libtool

为了生成脚本，autoconf 需要 GNU M4。它使用了 M4 的某些特性，可能某些 UNIX 版本的 M4，包括 GNU M4 1.3 都没有提供这些特性。你必须使用 GNU M4 1.4 以上的版本。

可以到 autoconf 的 web 页面，查看最新的信息，邮件列表明细，已知的 bug 列表等。
可以把您的建议邮件给 the Autoconf mailing 列表。

2 GNU 编译系统

Autoconf 解决了一个重要的问题，准确的发现系统详细的编译和运行时信息，但是这个只是开发一个可移植软件所要解决的疑惑中的一点。在这之后，GNU 项目已经开发了一套系统完整的工具来完成 autoconf 开始的工作；GNU 编译系统最重要的几个组件分别是：autoconf,automake 和 libtool。本章我们将为您介绍这些工具，引导您了解更多的信息，让您能够在您的软件中使用这一整套 GNU 的编译系统。

2.1 Automake

普遍存在的 make 让 makefile 看来是为软件发布自动编译规则的唯一可行的方式，但是很快我们会发现 make 存在很多的限制。Make 缺少自动跟踪依赖关系的支持，缺少子目录递归编译以及可靠时间戳（特别是，网络文件系统）等等，这些让软件开发者必须很艰难的（通常不能正确的）为每个项目重复发明“轮子”。Make 在很多系统上的诡异让“可移植”变得不简单。除此之外，为了实现用户期望的一些标准目标（make install,make distclean,make uninstall 等等），我们还有一些必须的体力活要做。使用 autoconf 之后，你还必须添加一些重复性的代码到 Makefile.in 中，以便于识别像@CC@,@CFLAGS@,以及其他 confiugre 提供的代名词。于是 Automake 出现，用来解决这些问题。

Automake 允许你在 Makefile.am 中制定你的编译需求，相对于制定 Makefile 来说，它更简单而且语法更强大。Automake 从 Makefile.am 生成一个可移植的 Makfile.in 来与 Autoconf 协作。例如，用户 Makefile.am 来编译并安装一个简单的”Hello world”程序，其内容可能就是这样

```
bin_PROGRAMS = hello
hello_SOURCES = hello.c
```

生成的 Makefile.in（大约 400 行）将自动支持所有的标准目标，autoconf 提供的代名词，并自动跟踪依赖关系和支持 VPATH 编译等等。你可以使用 make 来生成 hello 程序，让后使用 make install 来安装它（通常安装到/usr/local/bin,除非你使用—prefix 指定了其他的安装路径）。

越大的工程从 automake 得到的好处越多，特别是一个有子目录的工程。即使是一个小的程序也可以实实在在的获得移植和便利的好处。相信随着你的深入使用，还会有带给你更多…

2.2 Libtool

通常，你不仅仅生成应用程序，也需要生成库，这样其他的程序可以从你的劳动成果获得便利。理想的，你希望生成一个共享库（动态加载），这样使用这个库的程序不需要重复在磁盘和内存建立单独的拷贝，并且你还可以独立的更新这个库。但是产生一个可移植的共享库，是一个噩梦，因为每个系统都有自己互不兼容的工具，编译参数以及各样魔法般的咒语。幸运的是 GNU 提供了解决方案，这就是 Libtool。

Libtool 处理了建立一个共享库的所有需求，从可移植角度看，也是目前唯一的解决办法。Libtool 也解决了其他一些头疼的问题：比如在 Makefile 中针对不同后缀的动态库的交互命令，在

动态库还没有由超级管理员安装时可靠的链接动态库，以及提供一致的版本系统（这样不同版本的库在安装和更新时保持二进制的兼容性）。尽管 libtool 和 autoconf 一样可以独立使用，但是它通常还是集成在 automake 中，只要共享库需要编译，libtool 就会自动被使用，你都不需要知道他的语法。

2.3 Pointers

对于一直使用朴素的 make 在单一系统生成小的工程的开发者来说，学习 automake 和 auroconf 的前景有些黯淡。当你的的软件发布给越来越多的用户，你可能会发现你投入了相当大的精力重复发明 GNU 编译工具已经提供的一些服务，而且你可能还经常犯一些曾经出现并解决过的错误。（如果你学习了 autoconf，automake 这将是小菜一碟！）

你可以从很多地方获得 GNU 编译工具的更多信息：

-WEB

Autoconf[1]，Automake[2]，Libtool的主站

-Automake 手册

参看 GNU automake 的章节 automake，可以获得更多关于 automake 的信息

-书籍

[1] Autoconf http://www.gnu.org/software/autoconf/
[2] Automake http://www.gnu.org/software/automake/

3 建立 configure 脚本

Autoconf 生成的配置脚本按照惯例命名为 configure。执行时，configure 创建一些文件，用合适的值替换他们中的配置参数。Configure 创建的文件有：

- 一个或者多个Makefile文件，一般软件包的每个子目录都有一个（参看4.7 节）。
- 可选的，一个C头文件，文件名可配置，包含#define指令（参看4.8 节）。
- 一个名为 config.status 的 shell 脚本。执行时，将重建上述文件（参看 14 章）
- 可选的，一个一般名为 config.cache（当使用 configure –config-cache 时创建）的 shell 脚本，保存了许多测试的结果（参看 7.3.2 节）。
- config.log，包含了编译器生成的任何信息，帮助发现 configure 出现的错误。

用 autoonf 创建 configure 脚本，你需要先编写一个 autoconf 的输入文件 configure.ac 或者是 confgiure.in，然后运行 autoconf。如果你编写了自己的特征检查，可能还需要编写文件 aclocal.m4 和 acsite.m4。如果你使用一个 C 头文件来包含#define 指令，可能还需要执行 autoheader，并将生成的 config.h.in 与软件包一起发布。

这里有个图描述了配置中使用的各样文件是如何生成的。Autoconf 和 autoheader 还会突区安装的 autconf 宏文件（读作 autoconf.m4）。

分发前，准备软件包时用到的文件，以及生成关系如下图：

分发后，用户配置一个软件包时用到的文件，以及相应关系如下图：

3.1 写 configure.ac

3.1.1 一个 shell 脚本编译器

3.1.2 Autoconf 语言

3.1.3 标准 configure.ac 布局

输少量例外，configure.ac 中调用 autoconf 宏的顺序并不重要。在所有检查之前 AC_INIT 是必须的，一般在末尾应有 AC_OUTPUT（参看 4.4 节）。另外有部分宏依赖于其他宏的处理结果，这些宏在其独立性描述中都有提及（参看第 5 章），如果他们没有按照特定顺序调用，在创建 configure 脚本时也会给出警告。

为了保证一致性，这里给出了一个建议性的调用顺序。一般来说，排在后面的项可能会依赖前面

的项，比如库函数一般会受到类型和库文件的影响。

Autoconf requirements

AC_INIT(package, version, bug-report-address)

information on the package
checks for programs
checks for libraries
checks for header files
checks for types
checks for structures
checks for compiler characteristics
checks for library functions
checks for system services

AC_CONFIG_FILES([file...])
AC_OUTPUT

3.2 使用 autoscan 创建 configure.ac

Autoscan 可以协助你对一个软件包的 conigure.ac 进行创建或者维护。Autoscan 检查通过命令行给定目录下（没有指定时默认为当前目录）各级目录的源文件。Autoscan 收集源文件中可能存在的移植性问题，并生成 confgiure.scan（可以作为软件包 configure.ac 的初始文件），并检查可能正存在的 configure.ac 文件的完整性。

Autoscan 生成的 configure.scan 在重命名为 confgiure.ac 之前，可能需要进行手工的调整。因为偶尔 autoscan 会把有关联的宏的顺序弄错，autoconf 运行时会给出警告，这时需要你进行手工调整。另外如果你希望使用配置信息头文件，你必须调用 AC_CONFIG_HEADERS（参看 4.8 节），这时你可能不得不在你的程序增加一系列的#if 指令（3.3 节将介绍一个工具来协助你的这个工作）。

当使用 autoscan 维护 configure.ac 是，致使简单的考虑增加他的一些建议。Autoscan 生成的 autoscan.log 包含了关于为什么一个宏是需要的详细的信息。

Autoscan 检查到源文件中的特殊符号时，根据几个数据文件（于 autoconf 一起安装的）来判断哪些宏应该输出。这些数据文件都是同样的格式：每一行由符号，空格，和遇到这个符号要输出的宏。以#起始的行是注释行。

Autoscan 接受如下选项：

参数	含义
--help -h	打印命令行选项的摘要信息并退出
--version -V	打印 autoconf 的版本号并退出

--verbose -v	打印检查的文件名以及发现的符号。输出可能是庞大的
--include=dir -I dir	追加 dir 到包含路径。可以多个
--prepend-include=dir -B dir	预加 dir 到包含路径。可以多个

3.3 使用 ifnames 列出条件

Ifnames 可以帮助你编写 configure.ac。他打印 C 预处理条件已经用到的标志符，如果软件已经安排了某些移植方面的考虑，ifnames 可以帮助你找出 configue 需要检查的点。这可以帮助你修复 autosan 生成的 confgiure.ac。

Ifnames 扫描命令行给出的 C 源文件，并排序输出所有出现在#if,#elif,#ifdef,或者#ifndef 指令上的标志符。输出格式为：每行一个标志符，空格，所在的文件名。

Ifnames 接受如下选项：

参数	含义
--help -h	打印命令行选项的摘要信息并退出
--version -V	打印 autoconf 的版本号并退出

3.4 使用 autoconf 创建 configure

从 configure.ac 创建 configure，只需要执行 autoconf（不需要任何参数）。Autoconf 让 M4 宏处理器使用 autoconf 宏处理 configure.ac。如果你使用一个参数运行 autoconf，它将把这个参数对应的文件作为输入文件（而不是 confgigure.ac），并将结果输出到标准 IO 设备（而不是 confgiure）。如果使用参数“-”，它将从标准输入中读取，并输出到标出输出设备。

Autoconf 的宏在几个文件中定义。其中一些与 autoconf 一起发布，autoconf 将首先读取这些文件，然后再 autoconf 安装目录搜寻 acsite.m4，最后在当前目录搜寻 aclocal.m4。这些文件可能包含了你机器或者包厢伽 un 的 autoconf 宏定义（参看第 9 章）。如果一个宏在多个地方定义，则最后加载的将覆盖之前加载的。

Autoconf 接受如下选项：

参数	含义
--help -h	打印命令行选项的摘要信息并退出
--version -V	打印 autoconf 的版本号并退出
--verbose	报告处理步骤.

-v	
--debug -d	不善处临时文件
--force -f	重建 configure，即使是比输入文件新
--include=dir -I dir	追加 dir 到包含路径。可多项
--prepend-include=dir -B dir	预加 dir 到包含路径。可多项
--output=file -o file	将输出保存到文件。“-”代表标准输出
--warnings=category -W category	报告 category（可以是，分隔的列表）相关的警告。参看 9.3 节，AC_DIAGNOSE，了解更全面的类别信息（categories），特别值如： “all” 报告所有警告 “none” 不报告 “error” 将警告当作错误 “no-category”
--trace=macro[:format] -t macro[:format]	不创建 confgiure 脚本，列出依照 formate 的宏调用。可以使用多个此选项指定多个宏。不能累计到同一个宏。Format 是一个正则表达式，可以包含新行，或者其它几个特殊字符。缺省时”$f:$1:n%”，参看 8.2.1 节，了解 format 的详细信息。
--initialization -i	缺省时，--trace 不输出 autoconf 宏的初始（例如 AC_DEFUN 定义）。这个结果明显加速，但是可以被这个选项禁用。

通常有必要就检查 configure.ac 文件内容，但是这个工作给自己来做是极度难受和易错的。建议你依赖—trace 选项来检查 confgiure.ac。例如，用来查看被替换变量，如下使用：
#autoconf –t AC_SUBST

configure.ac:2:AC_SUBST:ECHO_C
configure.ac:2:AC_SUBST:ECHO_N
configure.ac:2:AC_SUBST:ECHO_T

等等

下面的例子显示 ‘$@’, ‘$*’, 和 $%.的差异：

$ cat configure.ac

AC_DEFINE(This, is, [an

[example]])

$ autoconf -t ’AC_DEFINE:@: $@

: $

$: $%’

@: [This],[is],[an

[example]]

*: This,is,an

[example]

$: This:is:an [example]

这个例子更自由，选择更多：

$ autoconf -t ’AC_SUBST:$$ac_subst{"$1"} = "$f:$l";’

$ac_subst{"ECHO_C"} = "configure.ac:2";
$ac_subst{"ECHO_N"} = "configure.ac:2";
$ac_subst{"ECHO_T"} = "configure.ac:2";

等等

一个长的分隔符可以让更复杂的结果可读性更高，并容易解析（例如没有一个单独的字符适合作为分隔符）
$ autoconf -t ’AM_MISSING_PROG:${|:::::|}*’

ACLOCAL|:::::|aclocal|:::::|$missing_dir
AUTOCONF|:::::|autoconf|:::::|$missing_dir
AUTOMAKE|:::::|automake|:::::|$missing_dir

等等

3.5 用 autoreconf 更新 configure 脚本

使 GNU 编译系统众多的组件运作是件乏味的事情：在每一个目录为 gettext 运行 autopoint，为 Makefile.in 运行 automake 等等。当你的系统上的 GNU 编译系统某些组件（比如 automake）被更新了，或者每个源码文件（如 configure.ac）被更新了，甚至当需要在一个新的路径安装 GNU 编译系统的时候，这些事情是需要的。

为了在指定目录或者是他们的子目录更新 GNU 编译系统，Autoreconf 会在合适的时候重复运

行 autoconf，autoheader，aclocal，automake，libtoolize 和 autopoint 这些命令。默认情况下，autoreconf 会只是重建那些比他们的源文件更老的文件。

如果你安装了某个工具的一个新版本，你可以使用带—force 选项的 autoreconf 重建所有的文件。

参看 4.7.4[Automatic Remaking]，25 页，当源文件改变之后，由 Makefile 规则自动重建 configure 脚本。这个方法合适的根据配置头模版的时间戳来处理，而且不需要传递”—autoconf-dir=dir”或者”—localdir=dir”。

Autoreconf 接受如下选项：

选项	含义
--help -h	打印命令行选项概览并退出
--version -V	打印 Autoconf 的版本并退出
--verbose	当 autoreconf 运行 autoconf（可能的时候还有 autoheader）时，打印每个目录的名字
--debug -d	不删除临时文件
--force -f	强制重建，尽管 configure 脚本或者配置头文件比输入文件（configure.ac，如果存在，还有 aclocal.m4）还新
--install -i	把辅助文件按照到软件包。缺省时复制，可以用选项—sumlink 改变。这个开关调用 automake –add-missing ，libtooloze，autopoint 等
--symlink -s	当与—install 一起使用时，安装指向辅助文件的符号链接，而不是复制
--make -m	当配置这些目录时，使用命令 ./config.status –recheck && ./configure.status ，让后运行 make
--include=dir -I dir	追加 dir 到包含路径。可多项
--prepend-include=dir -B dir	预加 dir 到包含路径。可多项
--warnings=category -W category	报告 category（可以是，分隔的列表）相关的警告。 “cross” 交叉编译执行相关 “obsolete”过时结构相关报告 “portability” 移植性执行 “all” 报告所有警告 “none” 不报告 “error” 将警告当作错误 “no-category” 禁止告警，？？

关于语法 syntax 的报告默认是打开的，一个包含逗号分隔的种类列表内容的环境变量 WARNINGS 是容许的。使用 -W category 选项实际上等价于—warnings=syntax,$WARNINGS,category。如果你希望禁用缺省的 WARNINGS，而打开过时结构的报警，可以这样使用”-W none,obsolete”。

4 初始化与文件输出

Autoconf 生成的 configure 脚本需要一些关于如何初始化（比如怎样发现包的源文件）和如何生成输出文件的信息。下面的章节描述了初始化和生成输出文件的内容。

4.1 初始化 configure

在做其他事情之前，configure 脚本需要先调用 AC_INIT 宏，另外一个需要的宏是 AC_OUTPUT（参看 4.4 节）。

AC_INIT（package,version,[bug-report],[tarname]）

处理命令行参数，执行各种初始化和验证

设置软件包的名字和版本号，如执行 configure 的时候，典型的用法是”--version”。Bug-report 是可选参数，用来填写用户发送错误报告时使用的 email 地址。Tarname 与 packeage 存在区别：后者指名完整的软件包名称（如：’GNU Autoconf’），而前者指定发布的文件名称（如：’autoconf’）。缺省时，tarname 来自于 package，但是将被去掉 GNU，使用小写，将非数字和字母的字符（包括下划线）转换为’-‘号。

建议 AC_INIT 的参数最好是静态的，不可以使用脚本运算，但是可以来自于 M4 的宏运算。

下面的这些 M4 宏（如：AC_PACKAGE_NAME），输出变量（如：PACKAGE_NAME），以及预处理符号（如：PACKAGE_NAME）由 AC_INIT 定义。

宏	含义
AC_PACKAGE_NAME PACKAGE_NAME	软件包名称
AC_PACKAGE_TARNAME PACKAGE_TARNAME	软件包 tar 文件名
AC_PACKAGE_VERSION PACKAGE_VERSION	版本号
AC_PACKAGE_STRING PACKAGE_STRING	软件包名称和软件版本号
AC_PACKAGE_BUGREPORT PACKAGE_BUGREPORT	软件包 BUG 报告 email 地址

4.2 Configure 需要注意的几点

下面的这些宏伟 configure 脚本管理版本号，但并非必需的。

4.2.1 AC_PREREQ(version)

确保当前环境有一个足够高的 autoconf 版本可用。如果创建 configure 的 autoconf 版本比这个版本低，将打印错误信息并退出系统（错误状态为 63）。例如：

AC_PREREQ（2.59）

这个宏是唯一一个可以在 AC_INTI 前使用的，但是为了一致性，建议不要这样做。

4.2.2 AC_COPYRIGHT(copyright-notice)

表明，除了在 autoconf 宏中的自由软件基金会的版权外，你的 configure 脚本的部分内容也包含了 copyright-notice。Copyright-notice 会在 configure 脚本开始处以及使用’configure –version’时出现。

4.2.3 AC_REVISION（revision-info）

将修订时间戳（revision-info）去掉美元号、双引号之后，拷贝到 configure 脚本中。这个宏让你可以通过’configure.ac’放置一个修订时间戳到 configure 中，当你使用 RCS 或者 CVS 没有同步更新 confgirue 和 configure.ac 的时候，你可以判定哪一个修改的 configure.ac 与特定的 configure 对应。

例如，如果在 configure.ac 有一行：

AC_REVISION($Revision:1.30 $)

那么，configure 中将产生：

#! /bin/sh

From configure.ac Revision:130

4.3 寻找 configure 的输入

4.3.1 AC_CONFIG_SRCDIR(unique-file-in-source-dir)

Unique-file-in-source-dir 是一个位于软件包源码目录的文件，configure 将检查这个文件是否存在，以此判断被告知的目录确实存在源码。有时用户偶尔会使用’—srcdir’指定一个错误的目录，这是一个安全的检查。更多信息可参看 13.9 节。

手工配置或者使用安装程序的软件包可能需要告诉 configure 到哪里寻找其他的 shell 脚本，你可以使用 AC_CONFIG_AUX_DIR 来完成这个动作，尽管 configure 自己默认的地方通常是正确的。

4.3.2 AC_CONFIG_AUX_DIR(dir)

配置中使用的辅助文件目录 dir，指定用到的辅助编译工具（如：’install-sh’,’config.sub’,’config.guess’,Cygnus configure,automake and libtool script 等）dir 可以是绝对路径，也可以是相对于’srcdir’的相对路径。默认路径可以是’srcdir’或者’srcdir/..’或者’srcdir/../..’,实际使用的是第一次真正包含了 install-sh 的路径。其他的文件并不会检查，所以使用 AC_PROG_INSTALL 并不自动要求其他辅助文件的分发。它也检查’install.sh’，但是这个名字已经过时，因为一些 make 有一条规则：当 makefile 不存在时，从它创建 install。

同样的，使用了 aclocal 的软件包可以用 AC_CONFIG_MACRO_DIR 指定本地宏定义的路径。

4.3.3 AC_CONFIG_MACRO_DIR (dir)

未来的 autopoint,libtoolize,aclocal 和 autoreconf 都会使用 dir 指定的路径作为本地额外的宏路径。请直接在 configure.ac 中直接使用这个宏，这样 aclocal 可以在使用”—trace”之前找到宏的声明，为安装了这些宏的工具就能够被安全的调用。

4.4 输出文件

每个 autoconf 脚本，例如 configure.ac，应该在结尾调用 AC_OUTPUT。这个宏生成并运行 config.status，config.status 将用来从配置信息创建 makefile 以及其他文件。这也是除了 AC_INIT 之外唯一必须的宏（参看 4.3 节）。

4.4.1 AC_OUTPUT

生成 config.status 并运行，在 configure.ac 尾部调用此宏一次。

Config.status将执行所有的配置动作：所有的输出文件 （4.6 节,AC_CONFIG_FILES）,头文件 (4.8 节 ,AC_CONFIG_HEADERS), 命令 (4.9 节 ,AC_CONFIG_COMMANDS), 链接 (4.10 节,AC_CONFIG_LINKS),子目录(4.11 节,AC_CONFIG_SUBDIRS)

AC_OUTPUT 的位置正是配置动作开始的位置:其后的代码在 configure 运行 config.status 之后将被执行.如果你要加入自己的动作到 configure.status(使与 configure 是否运行无关),参看 4.9 节.

历史上,AC_OUTPUT 的用法稍微存在一些不同.参看 15.4 节[过时的宏],可以看到以往用法中 AC_OUTPU 的参数描述.

在子目录中执行 make 时,应该使用变量 MAKE.大多数版本的 make 将变量 MAKE 设置为 make 程序名加上它提供的一些可选项(但是也有一些没在命令行宝航这些变量的值,以致无法自动带入这些参数).有一些老版本的 make 没有设置变量.下面的宏允许你在这些老版本中也可以这样用.

4.4.2 AC_PROG_MAKE_SET

如果 make 预定义了变量 MAKE,则将 SET_MAKE 定义为空.否则定义包含”MAKE=make”值

的 SET_MAKE，并为其调用 AC_SUBST。

如果你使用这个宏，在每个 makefile.in 加上如下这样一行，就可以在其他目录运行 MAKE。

@SET_MAKE@

4.5 执行配置动作

Configure 本设计的看起来好像是独立处理了所有的事情，但是实际上还有一个潜在的工作者 config.status。configure 负责检查你的系统，config.status 则是实际的根据 configure 的检查结果采用合适的动作。最典型的工作就是 config.status 用来实例化各式各样的文件。

本章节描述四种标准的实例化宏 AC_CONFIG_FILES,AC_CONFIG_HEADERS,AC_CONFIG_COMMANDS,AC_CONFIG_LINKS 的常用行为，他们的原型如下：

AC_CONFIG_FOOS(tag…,[command],[init-cmds])

详细参数列表如下：

参数	含义
Tag…	以空格分隔的标签列表。一般是一些需要实例化的文件名 我们鼓励完全字面化的定义标签，特别的你应该避开 … && my_foos=”$my_foos fooo” … && my_foos=”$my_foos foooo” 而应该这样做 … && AC_CONFIG_FOOS(fooo) … && AC_CONFIG_FOOS(foooo) 宏 AC_CONFIG_FILES 和 AC_CONFIG_HEADERS 使用专门的标签：他们可以是‘output’或者‘output:inputs’的格式。文件 output 从自己的模板 inputs 实例化出来，缺省的模板名是 output.in 例如‘AC_CONFIG_FILES(Makefile:boiler/top.mk:boiler/bot.mk)’要求串联的从 boiler/top.mk 和 boiler/bot.mk 展看，生成 makefile 标签中可能会用到‘-’，作为特殊的含义表示标准输出（当用在 output 中）和输入（当用在 input 中）。很可能你不需要在 configure.ac 中使用到它，但是如果在./config.status 中使用时则较方便，更多信息可以参考 14 章。 Inputs 可以使用绝对路径，也可以使用相对路径。当使用相对路径时，搜索路径先后次序是先编译路径，后源码路径。
Commands	Shell 命令，直接输入到 config.status。并与一个标签关联起来，便于用户告知 config.status 执行哪一个命令。每当 tag 出现一次，命令就会被 config.status 执行一次，典型情况下，tag 对应的文件也就被创建。 Configure 执行中设置的变量，在这些命令执行时并不存在，你可以先使用 init-cmds 来时设置他们。但是下面的变量是可用的： Srcdir 从顶级编译目录到顶级源码目录的路径。就是 configure 选项‘--srcdir’所指定的。 ac_top_srcdir 从当前编译目录到顶级源码目录的路径。

	ac_top_builddir 从当前编译目录到顶级编译目录的路径
	ac_srcdir 从当前编译目录到对应的源码目录的路径。
	当前目录（或者虚拟目录）指向的是包含 tags 中的输入部分的目录。
	例如，使用—srcdir=../package 运行如下宏
	AC_CONFIG_COMMANDS([deep/dir/out:in/in.in],[…],[…])
	将生成如下的宏：
	#参数--srcdir
	Srcdir='../package'
	#参考 deep/dir
	Ac_top_builddir='../../'
	#将$ac_top_builddir 与 srcdir 串联
	Ac_top_srcdir='../../../package'
	#将$ac_top_srcdir 与 deep/dir 串联
	Ac_srcdir='../../../package/deep/dir'
	与 in/in.in 无关

Init-cmds	指定的 Shell 命令，不加引用的增加到 config.status 的开始处。 Config.status 运行一次，这里的命令也执行一次（不管 tag 如何）。因为他们没有被引用，所以'$var'将输出成为 var 的值。Init-cmds 的典型应用是让 configure 执行之后，为 config.status 提供一些 commands 需要的变量。 你应该特别消息自定义变量的命名，应为 init-cmds 使用的是与其他变量同样的名字空间。你可能会不可预知的覆盖了其他的变量。Sorry…

所有这些宏可以被调用多次，当然应该使用不同的标签。

4.6 创建配置文件

请一定阅读4.5节。

4.6.1 AC_CONFIG_FILES(file……,[cmds],[init-cmds])

让 AC_OUTPUT 读入每一个输入文件（默认为 file.in），替换其中的变量为具体的变量值生成每一个 file。这是实例化宏中的一个，可以从如 4.5，4.7 节了解更多关于如何使用变量的信息，从 7.2 节了解更多如何生成这些变量的信息。如果 file 所在目录不存在，这个宏将创建这些目录。通常 makefile 的目录就是这样创建的。但是其他文件，比如‘.gdbinit‘做好还是指定好。

AC_CONFIG_FILES 的典型用法如下：

AC_CONFIG_FILES([Makefile src/Makefile man/Makefile X/Imakefile])

AC_CONFIG_FILES([autoconf],[chmod +x autoconf])

你可以使用冒号:，来指定一个输入文件，例如：

AC_CONFIG_FILES([Makefile:boiler/top.mk:boiler/bot.mk]

[lib/Makefile:boiler/lib.mk])

这样的做法，允许你指定一个与 MS-DOS 兼容的文件名，或者预先为输出文件 file 设置或增加模板文件。

4.7 Makefile 中的置换

发行包中的每一个包含了一些需要编译或者安装的内容的目录都应该包含一个makefile.in，这样configure将为这些目录创建makefile。Configure首先根据系统环境或者用户设置确定各个变量的值，让后对出现在makefile.in中的@variable@进行替换。我们把这些出现在makefile.in中将被替换内容的变量叫做输出变量（output variable），它们通常都是一些configure设置的shell变量。如果需要configure替换一个特定的变量，可以使用AC_SUBST宏（将变量名作为其参数），具体用法参考 7.8 节。出现的其他‘@variable@’则保持不变。

使用 configure 的发行包必须与 makefile.in 一起发布，而不是 makefile。这样用户在编译前根据具体的环境合理的配置 makefile。

关于 makefile 的更多信息，请参看《The GNU Coding Standards》的“Makefile Conventions”一节。

4.7.1 预设输出变量

有一些输出变量是由 autoconf 的宏预先设置的。Autoconf 的一些宏设置额外的输出变量，在这些宏的说明中会被提到。B.2 节给出了完整的输出变量列表。4.7.2 列出了安装目录相关的预设变量。下面列出了其他的一些预设变量，这些都是些小巧精致的东西（参看 7.2 节，AC_ARG_VAR）。

4.7.1.1 CFLAGS

C 编译器的调试和优化选项，如果 configure 运行时没有设置，缺省的值是 AC_PROG_CC 的输出（当然前提是如果你调用了 AC_PROG_CC，否则就是空值）。当开始编译程序时，Configure 会使用这个变量来测试 C 特征。

4.7.1.2 Configure_input

一个说明的语句，内容大概是“此文件时由 configure 自动根据某 input file 生成”之类含义。AC_OUPT 会在创建的每一个 makefile 开始处增加一个包含此变量值的注释行。对于其他的文件，你应该在输入文件的开始处使用注释行来引用这个变量。例如，一个输入的 shell 脚本文件可以如下示：

```
# /bin/sh
# @configure_input@
```

这个注释的存在，可以提醒正修改此文件的用户：在使用之前，这个文件需要由 configure 来处理。

4.7.1.3 CPPFLAGS

这个变量用来指明头文件搜索路径（-Idir）以及 C 和 C++预处理器和编译器的其他各式选项。如果 configure 运行时为配置，缺省值为空。当编译或者预处理程序时，Configure 将用这个变量来测试 C 和 C++的一些特征。

4.7.1.4 CXXFLAGS

针对 C++编译器的调试和优化选项。如果 configure 运行时未设置，缺省的值是 AC_PROG_CXX 的输出（当然前提是如果你调用了 AC_PROG_CXX，否则就是空值）. 当编译程序时，Configure 将用这个变量来测试 C++的一些特征。

4.7.1.5 DEFS

为 C 编译器传递'-D'选项。如果调用了 AC_CONFIG_HEADERS，confgiure 将使用 -DHAVA_CONFIG_H 替换@DEFS@（参看 4.8 节）。configure 执行测试的时候，不会设置这个变量，只有在创建输出文件的时候，这个变量才被设置。参看 7.2 节，了解如何检查之前测试的结果。

4.7.1.6 ECHO_C ECHO_N ECHO_T

对于一个问题-答案消息对来说，如何禁止换行？ 这些变量能够提供一个方法：

```
echo $ECHO_N “And the winner is … $ECHO_C”
sleep 10000000000
echo “${ECHO_T}dead.”
```

某些老的不太普遍的 echo 实现，不提供方法来实现这个效果，在这些情况下 ECHO_T 杯设置为 tab。你可能不想使用它。

4.7.1.7 FCFLAGS

针对 Fortran 编译器的调试和优化选项。如果 configure 运行时没有设置，缺省的值是 AC_PROG_FC 的输出（当然前提是如果你调用了 AC_PROG_FC，否则就是空值）。当编译程序时，Configure 将用这个变量来测试 Fortran 的一些特征。

4.7.1.8 FFLAGS

针对 Fortran77 编译器的调试和优化选项。如果 configure 运行时没有设置，缺省的值是

AC_PROG_F77 的输出（当然前提是如果你调用了 AC_PROG_F77，否则就是空值）。当编译程序时，Configure 将用这个变量来测试 Fortran77 的一些特征。

4.7.1.9 LDFLAGS

Stripping('-s'),路径('-L'),以及其他的针对连接器的各样选项。请使用 LIBS 来为为连接器指定库（-l），而不要使用这个变量。如果 configure 运行时未配置，缺省值为空。链接时，configue 使用这个变量测试 C,C++和 Fortran 的特征。

4.7.1.10 LIBS

为连接器传递'-l'选项。缺省时是空值，但是一些 autoconf 的宏会预先设计附加的库，如果这些库被发现并且提供了必须的函数（参看 5.4 节），这些库将被加入这个变量。链接时，configure 将使用此变量检查 C,C++和 Fortran 的特征。

4.7.1.11 builddir

严格等于“.”，仅仅用来保持一致和对称。

4.7.1.12 abs_builddir

builder 的绝对路径。

4.7.1.13 top_builddir

相对路径，指向于当前编译树的最顶级目录。在顶级目录，这个值与 buiddir 一致。

4.7.1.14 abs_top_builddir

top_builddir 的绝对路径。

4.7.1.15 srcdir

相对路径，指向包含了 makefile 使用的源代码的目录。

4.7.1.16 abs_srcdir

srcdir 的绝对路径。

4.7.1.17 top_srcdir

相对路径，指向顶级源码目录。在顶级目录，与 srcdir 相同。

4.7.1.18 abs_top_srcidr

top_src 的绝对路径。

4.7.2 安装目录变量

下面的变量指定软件安装时使用的目录。这里讲详细讲述着什么时候，怎样使用这些变量。更多相关的信息可以参看《The GNU Coding Standards》的第“Variables for installation Directories”节。

4.7.2.1 bindir

用来安装可执行文件的目录。

4.7.2.2 datadir

用来安装只读的与体系结构独立的数据文件的目录。

4.7.2.3 exec_prefix

体系结构相关文件安装目录前缀。缺省时，与 prefix 相同。你应该避免直接在 exec_prefix 安装任何东西。但是包含体系结构相关文件的目录应该相对于 exec_prefix。

4.7.2.4 includedir

用来安装 C 头文件的目录。

4.7.2.5 infodir

用来安装 info 格式的文档的目录。

4.7.2.6 libdir

用来安装目标代码库的目录。

4.7.2.7 libexecdir

这个目录用来安装其他程序的可执行文件。

4.7.2.8 localstatedir

用来安装可修改的单机文件的目录。

4.7.2.9 mandir

这个目录用指定，用来安装 man 格式文档的顶级目录

4.7.2.10 oldincludedir

这个目录用来为非 GCC 编译器安装 C 头文件。

4.7.2.11 prefix

所有文件通用的安装前缀。如果 exec_prefix 指定了一个不同的值，prefix 则仅用于体系结构无关的文件。

4.7.2.12 sbindir

系统管理员使用的可执行文件安装目录。

4.7.2.13 sharedstatedir

用来安装可修改的体系结构无关文件的目录。（比对 localstatedir）

4.7.2.14 sysconfdir

用来安装只读的单机文件的目录。相对 datadir

这些变量大多从 prefix 或 exec_prefix 的值来。这是经过仔细考虑的以保障输出变量的目录不被扩大：典型的用法是'@datadir@'会被${prefix}/share 替换，而不是/usr/local/share。

这些行为由 GNU 编码标准所约束，所以当用户使用如下命令时：

'make' 可以在执行 configure 时，指定一个不同 prefix。在这种情况下，如果确实需要，软件包可以硬编码来制定一辆关系。

'make install' 可以指定一个不同的安装位置。这样接你结案包还是仅依赖编译所在的目录（make install 执行时并不重编译）。这是一个特别重要的特性，因为很多人希望把软件的所有文件安装在一起，然后使用链接 link，将最终的位置指向那里。

为了支持这些特性，将 datadir 定义成为'${prefix}/share'使之依赖于当前 prefix 的值，就显得非常必要。

一个结论是：除了 makefile 之外，不要在其他地方使用这些变量。例如，你应该在 CPPFLAGS 中增加“ -DDATADIR="$(datadir)" ”，而不是在 configure 中执 datadir（例如：AC_DEFINE_UNQUOTED(DATADIR,"$datadir"）来实现 makefile 文件的硬编码。

同样的你不应该依赖 AC_OUTPUT_FILES 来替换你的 shell 脚本或其他文件中的 datadir，而应该让 make 来管理他们的替换规则。例如：autoconf 载入自己以.in 结束的 shell 脚本的模板，使用一个 makefile 的片断如下：

```
edit = sed \
    -e 's,@datadir\@,$(pkgdatadir),g' \
    -e 's,@prefix\@,$(prefix),g'
autoconf : Makefile $(srcdir)/autoconf.in
    rm –f autoconf autoconf.tmp
    $(edit) $(srcdir)/autoconf.in > autoconf.tmp
    chmod +x autoconf.tmp
    mv autoconf.tmp autoconf
autoheader : Makefile $(srcdir)/autoheader.in
    rm –f autoheader autoheader.tmp
    $(edit) $(srcdir)/autoconf.in ?autoheader.tmp
    chmod +x autoheader.tmp
    mv autoheader.tmp autoheader
```

如下细节值得注意

注意点	含义
@datadir\@	反斜杠阻止 configure 在 sed 的表达式中替换@datadir@
$(pkgdatadir)	不要使用"@pkgdatadir@"！使用匹配的 makefile 变量
,	在 sed 中不要使用 '/'，因为你使用的变量很可能就包含这样的字符
依赖 Makefile	因为 edit 使用的值依赖配置设定的值（例如：prefix），而不仅仅依赖 VERSION 等，所以输出依赖 makefile 而不是 configure.ac
分离依赖 当个后缀规则	你不能这样用。上面的片断不可以写成这样（影响移植性） autoconf autoheader : Makefile .in: rm –f $@ $@.tmp $(edit) $< >$@.tmp chmod +x $@.tmp mv $@.tmp $@ 更多信息参看 10.11 节
"$(srcdir)"	确保指定了到 source 的路径。否则不能支持个别模块独立的编译

4.7.3 编译目录

你可以用同一份源代码同时支持好几个体系结构的编译。每个体系结构的目标文件都存放在他们自己的目录下。

为了支持这个特性，make 使用 VPATH 变量在源码目录中定位这些文件。GNU Make 和目前大多数其他的 make 都支持这个特性。早期的 make 不致吃 VPATH，使用他们时，源码必须与目标文件所在目录一致。

为了支持 VPATH，每个 makefile.in 应该包含如下两行：

srcdir=@srcdir@

VPATH=@srcdir@

不要将 VPATH 设置为其他的变量，例如 VPATH=$(srcdir)，因为一些版本的 make 并不对 VPATH 使用变量替换。

configure 可以在生成 Makefile 的时候，使用正确的值替换 srddir。

除非在隐式规则中，你不应该使用 make 的变量$<（这个符号会被展开为源码目录中的文件名）。因为有些版本的 make 在显式规则中并不替换$<为源文件，而是一个空值。（默认规则的用法如下：.c.o 指定如何使用一个.c 文件来生成一个.o 文件）。

更好的做法是，自 makefile 的命令行中总应该引用一个冠以$(srcdir)/的源码文件。例如：

time.info:time.textinfo

$(MAKEINFO) $(srcdir)/time.textinfo

4.7.4 自动重编

可以在 maekfile.in 的开始处，加入一些规则，让你的软件包在配置文件改变后自动更新配置信息。下面将介绍一个例子，它包含了所有可选的文件，如 aclocal.m4 和一些相关的配置头文件。你可以忽略掉你的软件包中不适用的规则。

因为 VPATH 机制的限制，$(srcdir)/作为前缀被包含进来。

为了避免不必要的重新编译，”stam-‘文件时需要的，因为如果 config.h.in 和 config.h 的内容没有改变他们的时间戳也不会变。在软件包中包含‘stamp-h-in’，这样 make 会认为 config.h.in 是最新的。（相对 touch 来说，使用 echo 更好。因为 date 可能造成不必要的差异，如 CVS 冲突等，参看 10.10 节）

```
$(srcdir)/configure:configure.ac aclocal.m4
    cd $(srcdir) && autoconf
#autoheader might not change config.h.in,so touch a stamp file
$(srcdir)/config.h.in:stamp-h.in
$(srcdir)/stamp-h.in:configure.ac aclocal.m4
    cd $(srcdir) && autoheader
    echo timestamp > $(srcdir)/stamp-h.in
config.h:stamp-h
stamp-h:config.h.in config.status
    ./configure.status
Makefile:Makefile.in config.status
    ./config.status
Config.status:configure
    ./config.status --recheck
```

(如果你直接把上面的内容复制到你的 makefile 中，你要主要把那些对齐的空格替换为 TAB)

另外 i，你可以使用“AC_CONFIG_FILES([stamp-h],[echo timestamp > stamp-h])”，这样 config.status 将保障 config.h 是最新的。关于 AC_OUTPUT 的更多信息，参考 4.4 节。

关于更多配置相关的依赖信息，请参考 14 章。

4.8 配置头文件

当软件包包含的使用 C 预处理符号的不是少量的测试信息时，通过命令行“-D”选项传递给编译器的信息就太长了。这会导致两个问题： make 输出的信息肉眼无法检查；更严重的是可能会超出一些操作系统的极限长度。相对于传递-D 选项给编译器的方法，congirue 也可以创建一个包含#define 指令的 C 头文件。宏 AC_CONFIG_HEADERS 用来选择这个方式的输出。这个可以紧跟在 C_INIT 之后。

软件包中相关文件，应该在任何头文件之前包含这个配置头文件，以保障声明的一致性。使用“#include <config.h>”代替“#include “config.h””，并传递编译器-I 选项（如-I.或者-I..，任何包含了 config.h 的目录）。这样的方法可以让编译目录能够找到知己的 config.h 而不是源码目录的 config.h（比如执行了一次发行 make 之后，源码目录自己可能已经配置，从而生成了 config.h）。

AC_CONFIG_HEADERS(header…,[cmds],[init-cmds])

如 4.5 节，这是一个例行化宏。让 AC_OUTPUT 创建 header（空格分隔符）指定的文件列表，这些文件包含了 C 预处理#define 语句，在生成文件中替换”@DEFS@”，使用“-DHAVE_CONFIG_H”替换 DEFS 的值。一般 header 的名字是 config.h.

如果 header 指定的文件已经存在，并且内容与 AC_OUTPUT 准备输出的一致，则保留。这样做的好处是，可以避免依赖于 header 的目标文件进行不必要的重新编译。

一般输入文件命名为 config.in；但是你可以修改这个配置，如下：

AC_CONFIG_HEADERS([config.h:config.hin])

AC_CONFIG_HEADERS([defines.h:defs.pre:defines.h.in:defs.post])

这样你可以让你的文件名可以在 MS-DOS 下可用，或者追加（以及在头部添加）样板文件。

4.8.1 配置头文件模板

你的最后的发行包中应该包含模板文件，这个文件与你希望的最终头文件看起来一致，包含注释，以及#undef 语句等。例如，假如你的 configure.ac 做如下调用：

AC_CONFIG_HEADERS([config.h])

AC_CHECK_HEADERS([unistd.h])

这样你将有类似如下代码（conf.h.in），在有 unistd.h 的系统中，configure 将定义宏#define HAVE_UNISTD_H 1 。在其它的系统，整行将被注释掉（避免系统重定义这个符号）。

/* Define as 1 if you have unistd.h*/

#undef HAVE_UNISTD_H

注意第一行的#undef，在 HAVE_UNISTD_H 后面没有任何东西（包括空格）。然后你可以使用预处理器解码配置头文件。

```
#include <conf.h>

#if HAVE_UNISTD_H
```

#include <unistd.h>
#else
/*麻烦来了*/
#endif

老格式的模板，使用#define 替换#undef。强烈建议不要这样做。类似的旧模板在同一行使用注释。在预处理宏增加注释从来不是好的主意。

因为这实在是一个枯燥的事情来维护模板头文件的更新，你可以使用 autoheader 来生成它。参看 4.8.2 节。

4.8.2 用 autoheader 来创建 config.h.in

4.9 运行任意配置命令

可以在 config.status 之前，之后和中间运行任意命令。下面三个宏收集这些命令，当他们被多次调用时，就执行这些命令。AC_CONFIG_COMMANDS 替换过时的宏 AC_OUTPUT_COMMANDS；详细参看 15.4 节。

4.9.1 AC_CONFIG_COMMANDS(tag…,[cmds],[init-cmds])

参看4.5 节。指定额外的字啊config.status之后执行的shell命令，以及为configure初始化变量的命令。命令与tag联合，因为cmds一般创建一个文件，tag应该是一个文件名。需要的时候所在的目录也会创建。

一个非真实的例子：

```
fubar=42
AC_CONFIG_COMMANDS([fubar],[echo this is extra $fubar,and so on.],[fubar=$fubar])
```

下面试一个稍好一点的 ：

```
AC_CONFIG_COMMANDS([time-stamp],[date>time-stamp])
```

4.9.2 AC_CONFIG_COMMANDS_PRE(cmds)

在 config.status 创建之前执行。

4.9.3 AC_CONFIG_COMMANDS_POST(cmds)

在 config.status 创建之后执行。

4.10 建立配置链接

你可能发现创建那些目标依赖检查结果的链接很方便。可以使用 AC_CONFIG_COMMANDS，但是当软件包的编译目录与源码目录不一致的时候，创建相对的符号链接则比较脆弱。

AC_CONFIG_LINKS(dest:source…,[cmds],[init-cmds])

让AC_OUTPUT创建链接名dest，指向存在的文件source。可以是创建符号链接，硬链接，甚至一个拷贝。Dest和source应该相对于顶级源码或者编译目录。参看4.5 节。例如：

AC_CONFIG_LINKS(host.h:config/$machine.h

Object.h:config/$obj_format.h)

在当前目录创建一个链接名为’host.h’的链接，指向’srcdir/config/$machine.h’，以及一个链接名为’object.h’的链接，指向’srcdir/config/$obj_format.h’。

诱人的”.”对于 dest 是无效的，因为 config.status 不可能猜测出道哪里创建链接。

接着，可以运行如下命令，创建链接：

./config.status host.h object.h

4.11 在子目录配置其他软件包

大多数情况下，使用 AC_OUTPUT 在子目录生成 Makefile 就足够了。如果 configure 脚本控制了不止一个独立的软件包时，可以使用 AC_CONFIG_SUBDIRS 来为位于子目录的其他的 软件包运行 configure.

AC_CONFIG_SUBDIRS(dir …)

让 AC_OUTPUT 为每一个空格分割的列表指出的子目录运行 configure。其中 dir 应该是纯字面上的含义的。例如，不能这样使用

```
if test “$package_foo_enabled” = yes ; then
        $my_subdirs=”$my_subdirs foo”
fi
AC_CONFIG_SUBDIRS($my_subdirs)
```

因为这个会阻止”。/configure –help=recursive”显示软件包 foo 的选项。恰当的写法如下：

```
if test “$package_foo_enabled” = yes ; then
        AC_CONFIG_SUBDIRS(foo)
fi
```

如果指定的 dir 不存在，将会报告一个错误：如果目录是可选的，可以如下使用

```
if test –d $srcdir/foo ; then
        AC_CONFIG_SUBDIRS(foo)
fi
```

如果给定的 dir 包含 configure.gnu，将执行它，而不会执行 configure.这是因为这个软件包可能是使用的非 autoconf 的脚本 **Configure** ，这个脚本不能通过一个包装好的 configure 调用，因为在一些大小写不敏感的文件系统上，他们是一样的。同样的，如果一个 dir 宝航了 configure.in 而不是

configure，则使用通过 AC_CONFIG_AUX_DIR 参数指定的 Cygnus 的 configure 脚本。
子目录的 configure 脚本使用与给与当前 confiugre 同样的命令行选项，但是会做一些细微的改变，包括：
-为 cache 文件调整相对路径
-为源码目录调整相对路径
-衍生$prefix 当前的值，缺省情况下，包含顶级目录与子目录的差异。

这个宏也为 makefile 设置输出变量 subdirs，这样 make 可以使用这个变量确定需要递归的子目录。这个宏可以调用多次。

4.12 缺省前缀

缺省时，configure 设置 prefix 值为”/usr/local”。用户可以使用”--prefix”和”—exec-prefix”选项来指定不同的定义。有两种方式改变这个缺省值：在生成 confgiure 的时候指定，也可以在执行 configure 的时候指定。

有些软件可能不希望缺省安装在”?usr/lcoal”目录，而是缺省安装到其他的一个目录。为了实现这一点，定义了 AC_PREFIX_DEFAULT 宏。

AC_PREDFIX_DEFAULT(prefix)

设置缺省的默认安装前缀，取代”/usr/local”。

对用户来说，如果 configure 可以根据他已经安装了的相关的程序来确定目前的安装前缀，是比较方便的。如果你愿意提供这个功能，可以使用 AC_PREFIXX_PROGRAM。

AC_PREFIX_PROGRAM(program)

如果用户没有使用—prefix 选项指定一个安装前缀。从 PATH 找到 program 的路径，根据 program 的所在路径猜测一个值。如果找到 program，设置包含 program 的父目录，否则就是上面给出的缺省前缀。例如：如果 program 是 gcc，而且路径包含”/usr/local/gnu/bin/gcc”，则设置前缀为”/usr/local/gnu”。

5 存在性检查

这些宏用来检查特定系统上的一些被软件包可能需要或者希望用到的特性。如果你想检查一种特性，而这些宏都没有提供。你可能需要通过结合合适的参数调用一些更原始的宏来实现。(参看第 6 章)。

这些检查，将打印一些信息，告诉用户“正在做什么，发现了什么”。并将发现的结果 cache 起来，供未来 confgiure 使用（参看 7.3 节）

这些宏其中的一部分也会设置输出变量（怎样取这些值，参看 4.7 节)。下面都会使用短语“define name”作为 C 语言“#define name 1”语句的简称。在程序中怎样使用这些预定义符号，请参看 7.1 节。

5.1 通常行为

有很多努力是为了让 autoconf 能够很容易被使用者学会。显而易见要达到这个目的最好是坚持标准的接口和行为，尽可能的避免例外。不幸的是因为历史和一些过渡的原因，autconf 还是有太多的例外。但是这本节，我们只讲述一些通用的规则。

5.1.1 标准符号

所有普通的宏 AC_DEFINE 一个符号作为检查的结果，并将他们的参数转化为标准的字母表。首先，参数会被转化为大写，然后用“P”替换所有的“*”，余下的非字母和数字的字符使用下划线“_”替换。例如：

AC_CHECK_TYPES(struct $Expensive*)

如果检查成功，将定义一个符号”HAVE_STRUCT__EXPENSIVEP”。

5.1.2 缺省 Includes

个别检查依赖一组头文件。因为这些头文件不是所有地方都有效的，所以应该提供一组安全的 Include 检查。例如：

```
#if TIME_WITH_SYS_TIME
#   include <sys/time.h>
#   include<time.h>
#else
# if HAVE_SYS_TIME_H
#     include<sys/time.h>
# else
#     include<time.h>
# endif
#endif
```

除非你确切的知道你在做什么，你应该避免使用无条件的 include，应预先检查你的 include 的存在性。（参看 5.6 节）。

大多数普通宏使用下面的宏来提供缺省的 includes 集。

AC_DEFAULT_INCLUDES([include-directives])

如果定义了则展开到 include-directives。否则展开为：

```
#include <stdio.h>
#if HAVE_SYS_TYPES_H
# include <sys pes.h>
#endif
#if HAVE_SYS_STAT_H
# include <sys/stat.h>
#endif
#if STDC_HEADERS
# include <stdlib.h>
# include <stddef.h>
#else
# if HAVE_STDLIB_H
# include <stdlib.h>
# endif
#endif
#if HAVE_STRING_H
# if !STDC_HEADERS && HAVE_MEMORY_H
# include <memory.h>
# endif
# include <string.h>
#endif
#if HAVE_STRINGS_H
# include <strings.h>
#endif
#if HAVE_INTTYPES_H
# include <inttypes.h>
#else
# if HAVE_STDINT_H
# include <stdint.h>
# endif
#endif
#if HAVE_UNISTD_H
# include <unistd.h>
#endif
```

如果使用了缺省的 includes，就会检查这些头文件的存在性和兼容性。你不需要执行 AC_HEADERS_STDC，也不需要检查“stdlib.h”等。

这些头文件会按照他们被 include 的顺序来检查。例如：在一些系统“string.h”和“strings.h”都存在，但是互相冲突。则只定义 HAVE_STRING_H，而不会定义 HAVE_STRINGS。

5.1.3 选择性程序

这些宏检查特定程序的存在或者行为。他们用来从几个供选择的程序中选中一个，并决定如何做。如果没有宏支持你需要检查的某个程序，并且你不需要检查他的任何特别的属性。你可以使用一个通用的程序检查宏。

5.1.3.1 特定的程序检查

这些宏检查特定的程序：他们是否存在，以及在某种情况下他们是否支持某种特性。

AC_PROG_AWK

按照顺序检查 gawk,mawk,nawk,以及 awk，并将首次发现的一个设置为输出变量 AWK 的值。为什么首先尝试 gawk，是因为这个是被证明为最佳的实现。

AC_PROG_EGREP

先后检查 grep –E 和 egrep，并将首次发现的设置为输出变量 EGREP 的值。

AC_PROG_FGREP

先后检查 grep –F 和 fgrep，并将首次发现的设置为输出变量 FGREP 的值。

AC_PROG_INSTALL

如果在当前 PATH 中找到一个与 BSD 兼容的 install 程序，则将输出变量 INSTALL 设置为此程序的路径。否则，设置 INSTALL 为“dir/install-sh-c”，（dir 通过检查默认目录或者 AC_CONFIG_AUX_DIR 设定的目录来确定）。同时设置输出变量 INSTALL_PROGRAM 和 INSTALL_SCRIPT 为“${INSTALL}”，设置 INSTALL_DATA 为”${INSTALL} –m 644”。

这个宏可以筛选出很多已知不可用的 install 实例。出于速度的考虑，他将优先使用一个 C 程序而不是 shell 脚本。除了 install-sh 之外，他还可以使用 install.sh，但是这个名字比较古老因为一些 make 程序在没有发现 Makeifle 的时候，会使用此文件生成 install。

你可以使用与 autoconf 一起提供的 install-sh。如果你使用了 AC_PROG_INSTALL 宏，你必须在你的发行中包含”install-sh”或者”install.sh”。否则即使你的系统由一个更好的 install 程序，configure 也将产生一个错误（没有找到这些文件）。这是一个安全机制，避免你把这个文件意外的排除，从而使在没有安装 BSD 兼容的 install 程序不能正常的安装软件包。

如果要使用你自己的安装程序，你当让不用使用 AC_PROG_INSTALL.直接在 Makefile.in 中写上你的程序名就好了。

6 编写测试

7 测试的结果

Configure 检测出某些特性存在与否之后，通过什么方法把这些信息记录下来呢？ 一般有如下四种：设置一个 C 预处理符号；在输出文件设置一个变量；在 cache 文件保存结果供 configure 后续使用；以及打印信息让用户知道测试结果。

7.1 定义 C 预处理符号

最普通的做法就是对检查出来的某特性结果定义一个 C 预定义符号。这是通过调用 AC_DEFINE 和 AC_DEFINE_UNQUOTED 来实现的。

默认情况下，AC_OUTPUT 通过这些宏把这些符号定义放在输出变量 DEFS 中，对于每个定义的符号都包含一个选项“-Dsymbol=value”。在 Autoconf 版本 1 中，configure 正在运行时，是没有这个变量 DEFS 定义的。确认 autoconf 是否已经定义了某个 C 预处理符号，可以通过检查相应的 cache 变量，如下例子：

AC_CHECK_FUNC(vprintf,[AC_DEFINE(HAVE_VPRINTF)])
if test “$ac_cv_func_vprintf” !=yes; then
    AC_CHECK_FUNC(_doprnt,[AC_DEFINE(HAVE_DOPRNT)])
fi

如果调用了 AC_CONFIG_HEADERS 宏，AC_OUTPUT 将不会创建 DEFS，而是用正确的值替换在模板文件中#define 语句。关于这类输出的更多信息参看 4.8 节。

AC_DEFINE(variable,value,[description])
AC_DEFINE(variable)

完全字面上的定义 C 预处理变量 variable 为 value。Value 不应该包含字面上的新行，如果当前没有使用 AC_CONFIG_HEADERS 也不应该包含字符“#”，因为 make 往往会“吃掉”它们。如果希望使用 shell 变量，请使用 AC_DEFINE_UNQUOTED。Description 只在你使用 AC_CONFIG_HEADERS 时有用，这时的 description 将作为宏定义之前的注释放在生成的 configure.h.in 中。下面的例子定义一个预处理变量：名字是 EQUATION，值是”$a > $b”：

AC_DEFINE(EQUATION, “$a > $b”)

如果既没有给出 value 也没有给出 description，某人未能的 value 是 1（不是空字符串，切记）。这是为了与老版本的 autoconf 兼容，但是这个用法是应该废止的，因为未来的版本可能会抛弃他。

AC_DEFINE_UNQUOTED(variable,value,[description])
AC_DEFINE_UNQUOTED(variable)

与 AC_DEFINE 一样。但是对 variable 和 value 将执行一次 shell 的三次展开：变量展开(‘$’)，命令替换(‘`’)，反斜杠转义(‘\’)。单引号和双引号字符没有特别的含义。当 variable 或者 value 是一个 shell 变量时使用这个宏。例如：

AC_DEFINE_UNQUOTED(config_machfile, "$machfile")
AC_DEFINE_UNQUOTED(GETGROUPS_T, $ac_cv_type_getgroups)
AC_DEFINE_UNQUOTED($ac_tr_hdr)

因为 Bourne Shell 的语法原因，在其他宏调用或者 shell 代码中，不要使用分号将 AC_DEINFE 或

者 AC_DEFINE_UNQUOTED 调用隔开，因为这会导致生成的 configure 脚本语法错误。请使用空格或者是新行。比如：

AC_CHECK_HEADER(elf.h, [AC_DEFINE(SVR4) LIBS="$LIBS -lelf"])

或者这样：

```
AC_CHECK_HEADER(elf.h,
    [AC_DEFINE(SVR4)
     LIBS="$LIBS -lelf"])
```

而不应该这样：

AC_CHECK_HEADER(elf.h, [AC_DEFINE(SVR4); LIBS="$LIBS -lelf"])

7.2 设置输出变量

另外一个记录检测结果的方法是设置输出变量，configure 输出时，这些 shell 变量的值将替换到相应的一些文件中。下面有两个宏是用来创建新的输出变量的。参看 4.7.1 节可获得通常情况下可用的输出变量列表。

AC_SUBST(variable,[value])

从 shell 变量创建输出变量。让 AC_OUTPTU 替换输出文件中的变量 variable（一般是一些 makefile 文件）。详细来说，当 AC_OUTPUT 调用时，将输入文件中"@variable@"替换为 shell 变量 variable 的值。Variable 不应该包含字面的新行符。

如果 value 被指定，将赋值给 variable。

AC_SUBST_FILE(variable)

让 AC_OUTPUT 将 shell 变量 variable 指定的文件内容插入（不是替换）到输出文件。如果不存在这样的文件，variable 将被设置为/dev/null。

这个宏在需要向 makefile 插入一个包含特定依赖，特定主机 make 指令或目标类型的 makefile 片段时，非常有用。例如，我们的 configure.ac 包含如下内容：

```
AC_SUBST_FILE(host_frag)
host_frag=$srcdir/conf/sub4.mh
```

然后在“Makeffile.in”中包含如下内容：

```
@host_frag@
```

在不同环境运行 configure 是极度危险的事情。例如：如果用户运行"CC=bizarre-cc ./configure"，然后 cache，config.h，以及很多其他输出文件将以 bezarre-cc 为 C 编译器建立依赖。然后，因为

一些原因，用户再次运行.configure，或者执行'./config.status --recheck'（参看 4.7.4 节，以及 14 章），这样配置就不一致了，隐藏的结果依赖于两个不同的编译器。

造成这种情况的环境变量，例如上面提到的“CC”，被叫做**敏感变量**，这些变量可以被 AC_ARG_VAR 定义。

AC_ARG_VAR(variable,description)

定义 variable 为一个敏感变量，并在“./configure --help”的变量区间中包含 descripton 的信息。

作为敏感变量意味着：

-variable 是 AC_SUBST 输出的

-当 configure 启动时，variable 的值是保存在 cache 中的（如果不是命令行指定，也可能是环境变量引入的）。事实上，如果用户使用"./configure CC=bizarre-cc"命令，configure 可以检查；但是不可能在用户使用"CC=bizarre-cc ./configure"命令时检查出来。不幸的是，后者确实用户经常干的。

我们强调被保存的 variable 的值是初始值，而不是 configure 执行过程中的值。事实上，指定“./configure FOO=foo”，让“./configure”判断 FOO 是 foo 是两个相当不同的运行过程。

-在两次 configure 运行时，将检查 variable 的值是否一致，例如：

$ ./configure --silent --config-cache

$ CC=cc ./configure --silent --config-cache

configure: error: 'CC' was not set in the previous run
configure: error: changes in the environment can compromise \
the build
configure: error: run 'make distclean' and/or \
'rm config.cache' and start over

如果 variable 没有设置，或者他的值被改变。都将看到同样的结果。

-variable 的值将保存下来，当自动重配置（参看 14 章）时，就像命令行参数传入一样，即使当时没有使用 cache：

$CC=/usr/bin/cc ./configure undeclared_var=raboof –silent
$./config.status –recheck
第二条命令将执行/bin/sh ./configure undeclared_var=raboof –silent \
CC=/usr/bin/cc –no-create –no-recursion

7.3 Caching 结果

为了避免对同样的特性重复进行多次的检查（执行不同的 configure 脚本，或者重复执行相同的 configure），configure 可以选择性的将许多检测的结果存放到一个 cache 文件中（参看 7.3.2 节）。如果执行 configure 时，激活了 cache 选项并找到了 cache 文件，它将优先从 cache 中获取结果，这样 configure 就可以执行得更快。

AC_CACHE_VAL

去任 cahce-id 标记的检查结果可用。如果从 cache 中渠道，并 configure 没有带”--quiet”或者”--silent”选项，则打印结果被 cache 的信息说明。否则执行 shell 命令 commands-to-set-it。如果 shell 命令用来执行确定它的值，这个值会在 configure 创建输出文件之前存入 cache 文件。关于如何命名 cache-id 的变量，请参看 7.3.1 节。

除开设置变量 cache-id 之外，Commands-to-set-it 必须没有副作用。下面将提到。

AC_CACHE_CHECK(message,cache-id,commands-to-set-it)

关注输出信息的 AC_CACHE_VAR 的一个包装。这是一个方便的缩写，它调用 AC_MSG_CHECKING 输出 message，然后 AC_CACHE_VAR(cache-id,commands-to-set-it)，最后用 cache-id 作为参数调用 AC_MSG_RESULT。

除开设置变量 cache-id 之外，Commands-to-set-it 必须没有副作用。下面将提到。

It is very common to find buggy macros using AC_CACHE_VAL or AC_CACHE_CHECK, because people are tempted to call AC_DEFINE in the commands-to-set-it. Instead, the code that follows the call to AC_CACHE_VAL should call AC_DEFINE, by examining the value of the cache variable. For instance, the following macro is broken:

```
AC_DEFUN([AC_SHELL_TRUE],
[AC_CACHE_CHECK([whether true(1) works], [ac_cv_shell_true_works],

                [ac_cv_shell_true_works=no

                 true && ac_cv_shell_true_works=yes

                 if test $ac_cv_shell_true_works = yes; then

                   AC_DEFINE([TRUE_WORKS], 1

                             [Define if ‘true(1)’ works properly.])

                 fi])

    ])
```

This fails if the cache is enabled: the second time this macro is run, TRUE_WORKS will not be defined. The proper implementation is:

```
AC_DEFUN([AC_SHELL_TRUE],
[AC_CACHE_CHECK([whether true(1) works], [ac_cv_shell_true_works],

                [ac_cv_shell_true_works=no
```

```
                    true && ac_cv_shell_true_works=yes])
  if test $ac_cv_shell_true_works = yes; then
    AC_DEFINE([TRUE_WORKS], 1
              [Define if 'true(1)' works properly.])
  fi
])
```

Also, commands-to-set-it should not print any messages, for example with AC_MSG_CHECKING; do that before calling AC_CACHE_VAL, so the messages are printed regardless of whether the results of the check are retrieved from the cache or determined by running the shell commands.

7.3.1 Cache 变量名

Cache 变量名字应该符合如下格式：

package-prefix _cv_value-type _specific-value _[additional-options]
for example, 'ac_cv_header_stat_broken' or 'ac_cv_prog_gcc_traditional'. The parts of the variable name are:

package-prefix

An abbreviation for your package or organization; the same prefix you begin local Autoconf macros with, except lowercase by convention. For cache values used by the distributed Autoconf macros, this value is 'ac'.

_cv_ Indicates that this shell variable is a cache value. This string must be present in the variable name, including the leading underscore.

value-type A convention for classifying cache values, to produce a rational naming system.

The values used in Autoconf are listed in Section 9.2 [Macro Names], page 103.

specific-value

Which member of the class of cache values this test applies to. For example,

which function (‘alloca’), program (‘gcc’), or output variable (‘INSTALL’).

additional-options

Any particular behavior of the specific member that this test applies to. For

example, ‘broken’ or ‘set’. This part of the name may be omitted if it does not apply.

The values assigned to cache variables may not contain newlines. Usually, their values will be Boolean (‘yes’ or ‘no’) or the names of files or functions; so this is not an important restriction.

7.3.2 Cache 文件

Cache 文件是一个 shell 脚本，保存了运行在同一个系统的 confgiure 测试结果。所有可以在不同的 configure 脚本或者多次的 configure 执行中共享。在不同的系统上，他没有什么作用。如果 cache 因为什么原因包含了所无的信息，用户可以删除或者编辑它。

7.3.3 Cache 检查点

如果你的 confgure 脚本，或者 configure.ac 调用的宏，突然从 configure 进程中退出。在关键点使用 AC_CACHE_SAVE 添加检查点是有效的做法。这样做可以节省时间，避免在纠正错误点后，重新运行错误点之前的检查。

AC_CACHE_LOAD

从 cache 文件加载 cache 的值。如果 cache 文件不存在则创建一个新的。AC_INIT 会自动调用它。

AC_CACHE_SAVE

将所有的 cache 变量输入到 cache 文件。AC_OUTPUT 会自动调用，但是在一些关键点主动调用 AC_CACHE_SAVE 是一个非常有效的做法。

例如：

. . . AC INIT, etc. . . .

```
  # Checks for programs.
  AC_PROG_CC
  AC_PROG_GCC_TRADITIONAL

   . . . more program checks . . .

AC_CACHE_SAVE

  # Checks for libraries.
  AC_CHECK_LIB(nsl, gethostbyname)
  AC_CHECK_LIB(socket, connect)

   . . . more lib checks . . .

AC_CACHE_SAVE

  # Might abort...
  AM_PATH_GTK(1.0.2,, [AC_MSG_ERROR([GTK not in path])])
  AM_PATH_GTKMM(0.9.5,, [AC_MSG_ERROR([GTK not in path])])

   . . . AC OUTPUT, etc. . . .
```

7.4 打印消息

Configure 脚本需要给用户提供给各运行时的信息。下面的一些宏对几种类型的信息给出合适的输出。他们的参数都是封闭的双引号引用，所以 shell 将对他们执行变量（$）以及反引号（`）的替换。

这些宏都从 shell 命令 echo 包装而来，configure 脚本应该不需要直接使用 echo 显示信息。使用这些宏比较容易改变输出信息（比如怎样以及什么时候输出每种信息；类似这样的修改，只需要修改宏的定义，然后所有的调用都自动修改了）。

用来诊断静态发行版本，例如，当 autoconf 被执行，参看 9.3 节。

AC_MSG_CHECKING(feature-description)

通知用户 configure 正在检查一个某个的特征。这个宏打印一个以 checking 开头，以…结尾的信息，注意没有新行。他必须跟着一个 AC_MSG_RESULT 的调用来打印检查的结果以及一个新行。Feature-description 可以是一些字符串，类似”whether the Fortran compiler accepts C++ comment ”或者是”for c89”。

当 congiure 运行时使用了--quiet 或者 --silent 选项时，这个宏不会输出任何信息。

AC_MSG_NOTICE(message)

AC_MSG_ERROR(error-description,[exit-status])

错误发生时，打印信息并调用 exit(exit-status)退出。

AC_MSG_FAILURE(error-description,[exit-status])

异常情况发生时，打印信息并调用 exit(exit-status)退出。

AC MSG WARN (problem-description)

告诉用户可能发生的问题，继续执行。

8 M4 编程

Autoconf 在两个基础上完成：M4sugar 和 M4sh。M4sugar 为纯 M4 编程提供了方便的宏支持。M4sh 提供了专著于 shell 脚本生成的宏。

到 autoconf 目前这个版本时，这两个基础支撑层还在实验，他们的接口可能会在未来改变。这样造成实际的问题是：“一定不要使用未纳入文档的东西”。

8.1 M4 引用

8.1.1 活性字符

8.1.2 一个宏调用

8.1.3 引用与嵌套宏

8.1.4 changequote 是邪恶的

8.1.5 四对符号

8.1.6 熟悉引用规则

8.2 使用 autom4te

8.2.1 调用 autom4te

8.2.2 定制 autom4te

8.3 M4sugar 编程

8.3.1 重定义 M4 宏

8.3.2 执行宏

8.3.3 强制模式

8.4 M4sh 编程

9 写 autoconf 宏

当你需要写一个供一个以上的软件包使用的特征检查的时候，最后的方法是把它包装成一个新的宏。这里是相关的一些操作规则和指导方针。

9.1 宏定义

9.2 宏命名

9.3 汇报消息

9.4 宏间依赖

9.4.1 必要宏

9.4.2 建议顺序

9.5 过时的宏

配置和可移植技术已经发展了几年。经常有更好的问题解决方法被开发，或者更系统的设计方案被纳入。这些在 autoconf 的很多部分都已经发生。遮掩的结果就是部分宏现在看来过时了。他们还在工作，但是已经不是更好的方法，因此他们应该被更现代的宏替代。理想情况下，autoupdate 会用新的实现替换旧的宏调用。

Autoconf 提供了一个简单的方法来使一个宏过期。

AU DEFUN (old-macro, implementation, [message])

定义 old-macro 为 implementation。与 AC_DEFUN 唯一的区别是：用户将看到警告信息（old-macro 现在已经过期）。如果用户使用 autoupdate，则 old-macro 将被替换为 implementation. 此时 message 将被输出。

9.6 编码风格

Autoconf 宏遵循严格的编码风格。我们鼓励你遵循这些风格，特别是你希望发布你的宏时。无论是提供给 autoconf，还是其他。

10 可移植 shell 编程

当你写自己的检查时，为保证代码的可移植性，有一些 shell 脚本编程技巧应该避免。Bourne Shell 和向上兼容的脚本如 Korn Shell 以及 Bash 已经发展了好些年，但是为了避免麻烦。最好不要使用 UNIX version 7 之后加入的特性，大约 1977（参看 6.7 节）

10.1 Shellology

10.2 Here-document

10.3 文件描述符

10.4 文件系统惯例

10.5 Shell 替换

10.6 赋值

10.7 Shell 中的括号

10.8 特殊 shell 变量

10.9 Shell 内建的局限性

10.10 常用工具的局限性

可以期待在每台机器都可以找到的一个小的工具集，可能也包含一些局限，你必须了解到。

10.11 Make 的局限性

11 手工配置

少数特征不能通过测试程序自动监测。例如，目标文件格式的详细信息，传递给编译器和链接器的特殊选项等等。

11.1 指定系统类型

11.2 获取规范的系统类型

11.3 使用系统类型

12 场所配置

Configure 脚本支持好几种本地配置选择。用户可以有多种方法来指定：外部软件包所在位置，可选特定的包含或者排除，安装路径，以及设置缺省的选项值。

12.1 与外部软件一起工作

12.2 选择软件选项

12.3 让帮助信息更漂亮

12.4 场所配置细节

12.5 安装时转换程序名

12.6 设置场所缺省信息

13 执行 configure 脚本

下面是关于使用 configure 脚本如何配置软件包的指令，与包含在软件包的 INSTALL 文件一致。可以使用与 autoconf 一起带给你的纯文版版本的 INSTALL。

13.1 基本安装

这些是一般的安装指令。

Configure 脚本尝试为各种系统相关的变量推测正确的值。并使用这些值在软件包的各个目录创建一个 Makefile。它也会创建一个或者多个包含系统依赖相关定义的“.h”文件。最后，生成一个 shell 脚本 config.status（你可以在后面重建当前的配置），同时也生成一个 config.log 包含编译器的输出（通常用来 debugging configure）。

它也可能使用一个可选的文件（一般叫做 config.cache，同时使用—cache-file=config.cache，或者简写-C 激活了此选项）来加快重配置的速度。（cache 默认是关闭的，用以避免偶尔的时候使用到过期的 cache 文件）。

13.2 编译器和选项

13.3 多体系结构的编译

13.4 安装路径

默认情况下，make install 将把软件的文件安装到/usr/local/bin,/usr/local/man 等处。你可以使用”—prefix=path”指定一个安装前缀来改变安装路径。

可以分别指定系统结构相关的文件和体系结构无关文件的安装前缀。如果你使用”—exec-prefix=path”，软件包将使用这个路径安装程序和库。文档和其他数据文件则仍然使用普通的前缀（—prefix 指定的）。

13.5 特征选项

一些软件包关注—enable-feature 选项，feature 代表软件包的一个可选部分。

13.6 指定系统类型

13.7 缺省共享

13.8 定义变量

13.9 Configure 语法

Configure 识别下面的选项，这些选项控制它的操作。

选项	说明
--help -h	打印 configure 的选项摘要，然后退出
--version -V	打印生成 configure 脚本的 autoconf 的版本，然后退出
--cache-file=file	启用 cache:使用或者保存测试的结果。文件名习惯使用 config.cache。默认是 /dev/null 用来禁止 cache。
--config-cache -C	--cache-file=config.cache 的别名
--quiet --silent -q	仅输出错误信息，禁止常规输出
--srcdir=dir	到 dir 查找软件源码。一般 configure 可以自动确定目录。

还有更多其他的选项，configure 也支持，但是并不是广泛使用。可以通过 configure—help 看到更多信息。

14 重建配置

Configure 脚本建立的文件 config.status，是从模版文件实际完成配置和替换工作的“工作者”。同时它也会纪录最后运行时的配置选项，以满意重配置的需要。

命令格式：

./config.status option ... [file ...]

这个命令将配置 file 指定文件，如果没有指定，所有的模版都会被替换，指定的 file 不能带依赖关系，比如：

./config.status foobar

而不是：

./config.status foobar:foo.in:bar.in

15 过时的结构

15.1 过时的 Config.status 语句

15.2 “acconfig.h”

15.3 用 autoupdate 使 configure.ac 现代化

15.4 过时的宏

15.5 从版本 1 升级

15.6 从版本 2.14 升级

16 用 autotest 生成测试套件

17 常见问题及回答

18 Autoconf 的由来

19 附录 A 版权

19.1 GNU 自由文档许可

19.2 如何使用这个许可

20 附录 B 索引

20.1 环境变量

20.2 输出变量

20.3 预处理符号

20.4 Autoconf 宏

20.5 M4 宏

20.6 Autotest 宏

20.7 程序与函数

20.8 概念