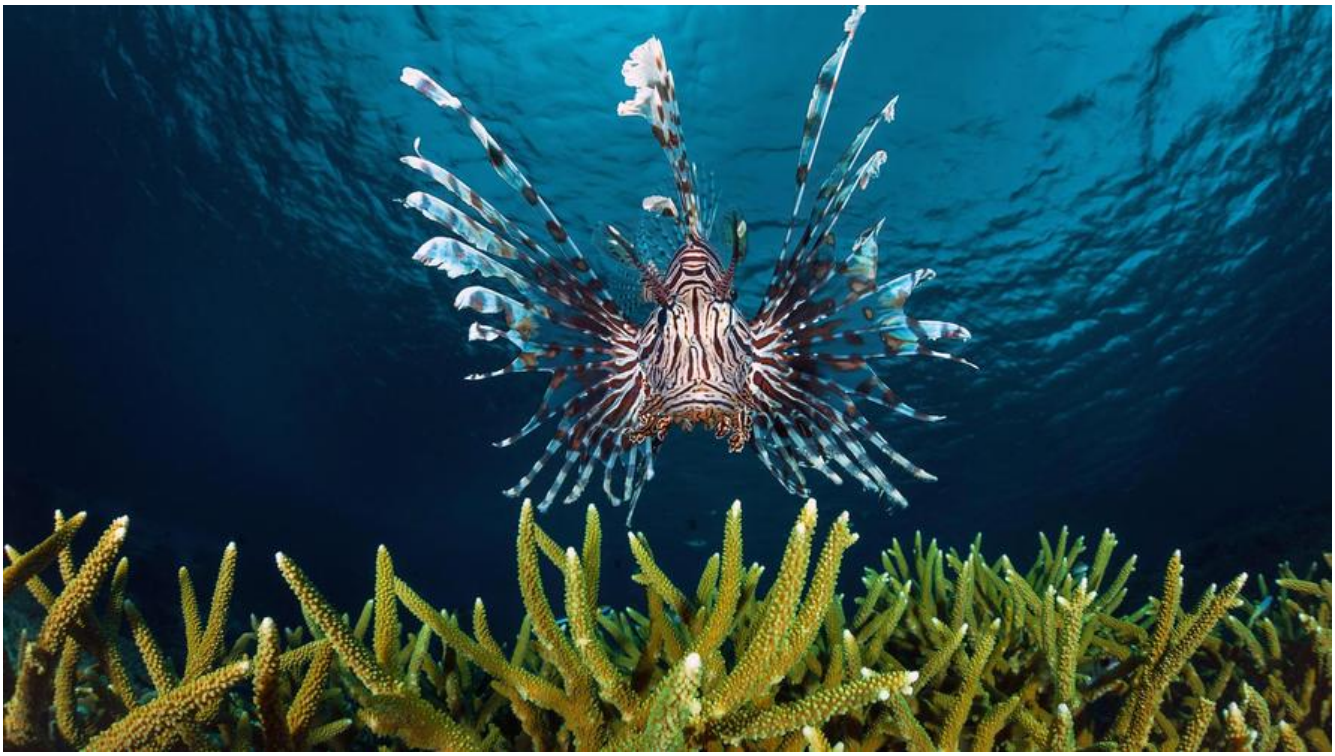
链滴

Autotool--configure 衍生物和 Makefile Target

作者：ReyRen

原文链接：https://ld246.com/article/1565875255843

来源网站：链滴



由configure生成的文件

当你调用configure后会发现会有一些列的文件在你的编译树中生成. 由configure构建的编译树的结和生成的文件的不同随着要构建的包的不同而有所不同. 以下是生成的文件及其说明:

config.cache

configure可以缓存系统测试的结果, 这样是有利于后续的检测的. 这个文件里存放的就是缓存的测试果的数据, 是普通文件, 如果需要是可以手动的进行修改和删除的.

config.log

随着configure执行, 在这过程中, 它会生成描述每个检测的结果和状况的. configure调用shell和一些具产生的输出信息是庞大的, 但是对用户是隐藏的, 这样是为了保证输出的可读性. 所有的输出信息都重定向(标准输出会出报错和一些结果)到"config.log"中. 当执行configure的时候出现一些天马行空输出信息的时候, 第一个想到的就是查这个文件. 很典型的一个场景是, 当在一个Solaris系统上运行configure后告诉你找不到工作的C编译器. 在"config.log"中会详细说明, Solaris默认"/usr/ucb/cc"是一个序, 也就是暗示用户可选择性的C编器器没有安装.

config.status

这是configure生成的一个shell脚本文件, 用来重新生成当前的所有配置. 也就是说, 重新生成的文件会部重新生成一波. 这个脚本也可以用来当"--recheck"设置后再次运行configue.

config.h

config.h这里我上传了一个我的项目中用configure生成的完整config.h. 有兴趣可以直接点击现在看看. 这里也罗列一些:

```
/* building in place */
#define BUILDING_IN_PLACE 1
/* Define to 1 if you have the `alarm` function */
#define HAVE_ALARM 1
/* Define to 1 if you have `alloca`, as a function or macro */
#define HAVE_ALLOCA 1
```

....

许多用configure的包都是用c或者c++写的, 一些configure运行的测试涉及到的是c和c++语言的变测试和它们的实现. 这样源代码就可以以可编程方式来处理这些差异性. 规避的方式是通过像#define样的预编译指令放在config.h中(configure运行后自动生成 的)， 然后源文件把config.h头加入到自的代码中. 就像这样:

```
#if HAVE_CONFIG_H
#  include <config.h>
#endif /* HAVE_CONFIG_H */

#if HAVE_UNISTD_H
#  include <unistd.h>
#endif /* HAVE_UNISTD_H */
```

虽然unistd.h似乎看起来也不错, 它是由POSIX.1标准定义的, 但是还是强烈建议永远使用config.h.

Makefile

configure的共同点之一就是生成Makefile和其他文件. 之前也提到过, Makefile就是由configure 从应的输入文件(Makefile.in) 生成出来的一个普通文件. 接下来会简单介绍一下如何用make去处理这个akefile.

比较常见的Makefile targets

到目前为止, configure生成了一些文件包括Makefile. 绝大多数的项目使用Makefile都会有一些列的targets. 这个targets就是后想要表现出来的任务的名字 -- 通常来说, 想要编译所属包下的所有的程序, 么这个targets就叫做all:

make all

编译所有期待编译的文件在所需要编译的包中.

make check

运行包所包含的自我检测

make clean

删除所有派生出来的文件.

当然也会有很多不是通用的targets可能呗重新组织, 尤其是一个包含"Makefile"的包遵循GNU "Makefile" 标准或者是由automake所生成的.

Configuration Names

GNU Autotool针对于各种各样的系统类型进行了命名. 这些命名都是有标准的格式规范的.

比如一些configuration names是'sparc-sun-solaris2.7', 'i586-pc-linux-gnu', 'i386-pc-cygwin'.

所有的configuration names都是由三部分组成的, 在一些文件中它们被称为configurtion三剑客. 这个部分就是**cpu-manufacturer-operating system**. 当今呢, 这有时候也分为了四部分, 区分kernel系统类型: **cpu-manufacturer-kernel-operating system**.

使用configuration names在configure工具中时, 通常是没必要指定全名的. 通常, 中间的名字(manufcturer)将会省略, 这样就会变成"i386-linux"或者"sparc-sunos". 脚本文件config.sub将会将其补全换为标准格式.

```
### Let's recognize common machines as not being operating systems so
```

```
### that things like config.sub decstation-3100 work.  We also
### recognize some manufacturers as not being operating systems, so we
### can provide default operating systems below.
case $os in
    -sun*os*)
        # Prevent following clause from handling this invalid input.
        ;;
    -dec* | -mips* | -sequent* | -encore* | -pc532* | -sgi* | -sony* | \
    -att* | -7300* | -3300* | -delta* | -motorola* | -sun[234]* | \
    -unicom* | -ibm* | -next | -hp | -isi* | -apollo | -altos* | \
    -convergent* | -ncr* | -news | -32* | -3600* | -3100* | -hitachi* |\
    -c[123]* | -convex* | -sun | -crds | -omron* | -dg | -ultra | -tti* | \
    -harris | -dolphin | -highlevel | -gould | -cbm | -ns | -masscomp | \
    -apple | -axis | -knuth | -cray | -microblaze*)
        os=
        basic_machine=$1
        ;;
    -bluegene*)
        os=-cnk
        ;;
    -sim | -cisco | -oki | -wec | -winbond)
        os=
        basic_machine=$1
        ;;
```

当然在绝大多数的Unix及类unix系统中, shell脚本config.guess

将会打印出正确的运行的configuration name. 它能达成这样的目的是通过运行标准的uname程序, 且通过测试一些系统其他的特征属性. 在一些系统上, "config.guess"是需要运行的c编译器或者汇编的.

因为"config.guess"是可以识别出及其的configuration name的, 所以对于用户或者开发者通常就会来在不是普遍的case中指定configuration name. 比如说当构建一个交叉编译器的时候.

接下来描述一下configuration name不同域的内容:

cpu

在系统上使用的处理器的类型, 这一般就是什么"i386"或者"sparc"啥的. 再具体点的变量也有可能是到, 比如"mipsel"表示的就是小端对齐的MIPS处理器.

manufacturer

这是一个很随意的域, 指的就是系统的供应商. 好多时候都是指定为"unknown". 其他一些公用的属性如"pc"指的就是IBM PC兼容系统. 或者是工作站厂商, "sun".

operating system

这个就是跑在系统上的操作系统的名字, 比如说"solaris2.5", "winnt4.0". 这对于系统版本号并没有严的限制.

configuration names可以用来描述各种各样的系统, 甚至是一些不运行操作系统的嵌入式系统. 在这情况下, 这个域通常用来指定目标文件的格式: elf或者coff.

kernel

这个主要是用在GNU/linux系统下, 一个典型的Gnu/Linux configuration name是"i586-pc-linux-nulibc1".可以看出来内核: linux和操作系统"gnulibc1"是分开的.

configure允许精确的控制二进制文件的格式. 为一个指定类型的机器编译包而需要在这个指定的机器是没有必要的, 相反, 交叉编译程序是可以使用的. 更进一步说, 如果你现在要build的包是本身可以在

叉配置下操作的, 那么编译系统就不是必须是交叉配置包所宿主的机器类型:
为GNU/Linux编译一个简单包:

host = build = target = ‘i586-pc-linux-gnu’

在GNU/Linux下交叉交叉编译一个包:

#运行在IBM AIX机器上
build = ‘i586-pc-linux-gnu’, host = target = ‘rs6000-ibm-aix3.2’

在GNU/Linux上编译一个Solaris为宿主的MIPS-ECOFF交叉编译器

build = ‘i586-pc-linux-gnu’, host = ‘sparc-sun-solaris2.4’, tar- get = ‘mips-idt-ecoff’

这一个章节就介绍道这里, 接下来会简单介绍一下Makefile了.