链滴

Autotool-- 简述

作者：ReyRen

原文链接：https://ld246.com/article/1564492082490

来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20190102.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
Autoconf, Automake 和 Libtool 是一些列的包, 主要的目的就是让你的软件可移植性性更高, 编
简化. 软件的可移植性和高效的编译系统是极其重要的, 尤其是对于现代软件.

Autoconf -- 这个工具的主要功能就是在进行包编译之前发现出系统的特性, 然后源码就能适应
些系统之间的差异.
Automake -- 这个就是产生 Makefile 的, 用来指导编译什么东西. Automake 是指上就是简化了
包组织结构以及性能和源码间依赖的描述过程.
Libtool -- 这是个命令行工具(对于编译器来讲), 主要就是使静态库和共享库的链接过程简化并且
用考虑平台性.

我们接下来所说的 Autotool 就是说 Autoconf, Automake 和 Libtool. 主要说明的就是这三者
间是怎样进行写协作的.
<h3 id="编译系统的前世今生">编译系统的前世今生</h3>
随着 Unix 操作系统的诞生及发展，首先出生的是 Autoconf.
Unix 的诞生是在 1969 年的贝尔实验室由 Dennis Ritchie 和 Ken Thompson 创造的. 但是在整个 70
年代, 这款 Unix 都不进行商业的销售. 只低价卖给一些大学. 这时候加州伯克利在 Unix 的源代码的基
上加入了自己的东西. 这就衍生出来就是 BSD.
到了 80 年代, AT&T 终于同于进行商业销售了. 这款进行销售的系统就是 System III.
随着 80 年代 Unix 系统的兴起. 好多公司自己进行 Unix 的改造然后形成不同的系统(Unix 版本). 比如
un Mircosystem 的 SunOS, HP 的 HP-UX 等等.
虽然这些不同 Unix 版本的基石是类似的, 但是都多多少少有些细微的差别的. 比如头文件, 系统库. 最
著的不同领域是终端处理和任务控制.
POSIX 的出现, 减缓了这些差异的产生, 但是 POSIX 的引入也会引入不少新的特性. 并且同一系统对 P
SIX 的遵循在不同的时期导致分出了更多的版本.
举个简单例子, <code>memcpy</code> 并不是任何系统上都能用. 在 BSD 系统上有个类似的函数
<code>bcopy</code>, 但是参数和顺序都保留了.
基于上面的背景, 一个程序的作者当然是想的能在多个平台上运行, 但是这样的话, 每个程序的作者都
要了解他想要支持的平台的信息, 并且还要指导同一款系统不同版本之间的差异, 这个工作量对软件的
发是极大的折损.
虽然可用 <code>#ifdef</code> 进行系统的限定, 但是这样的话, 很难得知什么版本有什么样的特性.
所以需要我们引入更加好的方法来处理平台的差异.
到了 1992 年, 四种帮助解决源代码可移植性的系统崭露头角:

Metaconfig
Cygnus "configure"脚本和 GCC 的"configure"脚本. 这两者是很类似的, 他们的开发团队都经
相互交流.
GNU Automake
Make
这些系统的共同点都是将编译一个程序分割成了两步走: 配置阶段和编译阶段. 编译阶段是用 Unix 的
ake, 遵循着一些在 Makefile 下的规则. 配置阶段就是生成 Makefile.
Metaconfig 和 Automake 都是使用功能测试来看系统兼容性. 它们都是使用 Bash 的脚本语言来运
各种各样的测试来看什么样的系统支持.
Cygnus 和 GCC 的"configure"也都是 bash 脚本.
Metaconfig 和 Autoconfig 是用于程序的开发者, 他们制造出一个脚本文件伴随着的是程序的源码.
于用户来说直接运行其脚本文件, 这样就能为未源码营造出一个能兼容当前系统的编译环境.
Cygnus 和 GCC 的"configure"脚本是支持交叉编译, 都支持编译一个交叉编译器.
Metaconfig 是交互式的默认情况下. Gygnus 和 GCC 的"configure"脚本和用 autoconf 生成的脚本,
make 生成的都是自己决定一切了.
这两个"configure"都是需要针对于不同的 Unix 发行版而进行不断的更新支持的, 或许对于作者来说
是件难事儿, 但是对于用户来说很是不方便. Metaconfig 和 Autoconfig 因为会用测试用例去检测, 这
生成的脚本是运行很顺利的在不同的 Unix 版本间. 在之后这个作者还和 gcc 的 configure 合作, 支持
用系统指定的头文件和 Makefile. 并且支持跨平台编译. 至此, gcc 也是用了 Autoconf 消除了 gcc 的"

onfigure"脚本. 之后有更多的都转换成用 Autoconf 了, 达到了编译巅峰.

<h3 id="Automake的前世今生">Automake 的前世今生</h3>
还远在 1994 年, Autoconf 是个坚实的框架处理不用的 Unix 发行版之间的差异. 然后程序员依需要写很大的"Makefile.in"文件, 染过 Autoconf 生成的"configure"脚本文件将"Makefile.in"转换为 Makefile"文件. 再用 make 进行最后的编译.
"Makefile.in"中描述的就是如何进行编译一个程序. 以为很多的程序在编译过程中很多是类似的步骤, 以回有大量的重复在"Makefile.in"中, 并且 Makefile 中的规则维护较为复杂.
也正是在这些原因的促使下, Automake 得以飞快的诞生. 开发人员需要写"Makefile.am", 这个语法比于 Makefile 简单很多. automake 通过 makefile.am 产生 makefile.in, 然后 autoconf 产生的脚将 makefile.in 转换为 makefile.
在 Makefile.am 中只需要写编译时需要的文件就行了, automake 会在生成 Makefile.in 的时候添加要的规则. automake 也会添加一些 GNU 的 makefile 规则进去.
<h3 id="Libtool的前世今生">Libtool 的前世今生</h3>
第一个共享库的诞生时在 System v release 3. 那会儿的贝尔实验室时真 :chicken: 儿的强. 共享的思想被广泛的推广，很快在很多 Unix 发行版上都开始用了, SunOS, HP-UX, AIX 等. 就这样, 对共库实现的平台差异性也随之而来.
1996 年的时候 Libtool 开始产生, 也是一些列的脚本, 用于处理不同系统平台上对共享库的产生和使的差异. 这个和 Automake 紧密相连, 虽然它们各自独立.
到这里对与 Autotool 的简介就结束了, 接下来会进行详细通俗的讲解.