跟涛哥一起学嵌入式 14：Autotools发展史

文档说明	作者	日期
来自微信公众号：宅学部落(armLinuxfun)	wit	2018.10.26
嵌入式视频教程淘宝店：https://wanglitao.taobao.com/		
联系微信：brotau(宅学部落)		

在linux下面撸过代码、做过开发的，想必都听说过Makefile。对，是 Makefile，不是 make love。如果你看成了后者，只能说：同志，你的三观有问题，需要格式化你的硬盘！

在 Linux 开发程序，没有集成开发环境 IDE(integrated development environment),没有 VC++6.0，只有Makefile 和冰冷黑漆漆的 shell 窗口，寒冷的夜，考验着每一个工程师疲惫的心。

Makefile 语法复杂、难以维护。对于一个小项目还好，对于大型的项目和开源项目，现在流行使用一些工具自动生成 Makefile，可以大大减轻软件开发人员的负担。

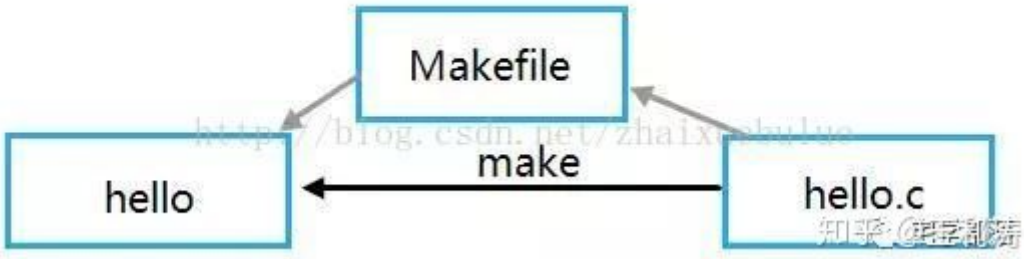
比较常见的工具有 GNU Autotools、CMake、QMake、SCons、Ant等。今天讲讲 GNU Autotools，这个工具在开源软件和大型项目中被广泛地使用。

Autotools 主要有 Autoconf、automake、libtool 等软件包工具组成，我们可以称为 Autotools 三剑客。

使用这几个工具，虽然能帮助我们自动生成 Makefile，但是命令过程复杂，中间会生成大量的各种配置文件和脚本。很多人往往会觉得很麻烦、搞不懂里面错综复杂的关系，今天以这些工具的发展过程，给大家理理里面的关系。

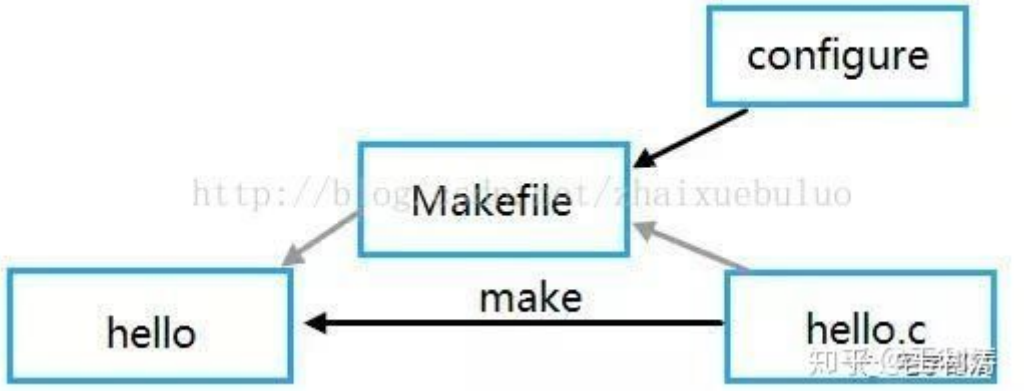
一、手工 makefile 时代

早期我们在 Unix、linux 环境下开发软件，makefile 都是手写的。通过 makefile，我们就可以使用 make 命令直接去编译我们的源代码。



后来，随着 Unix 版本越来越多，各分支差异越来越大，我们写的 makefile，有时候在其它 Unix 平台上可能就编译失败，比如库的问题等。不同的操作系统版本，库和头文件的路径可能不一样。怎么办？

后来我们通过手写一个配置脚本 configure 来解决这个问题：针对你当前的 Unix 平台，通过脚本对 makefile 进行配置，就可以在当前平台上愉快地编译了。



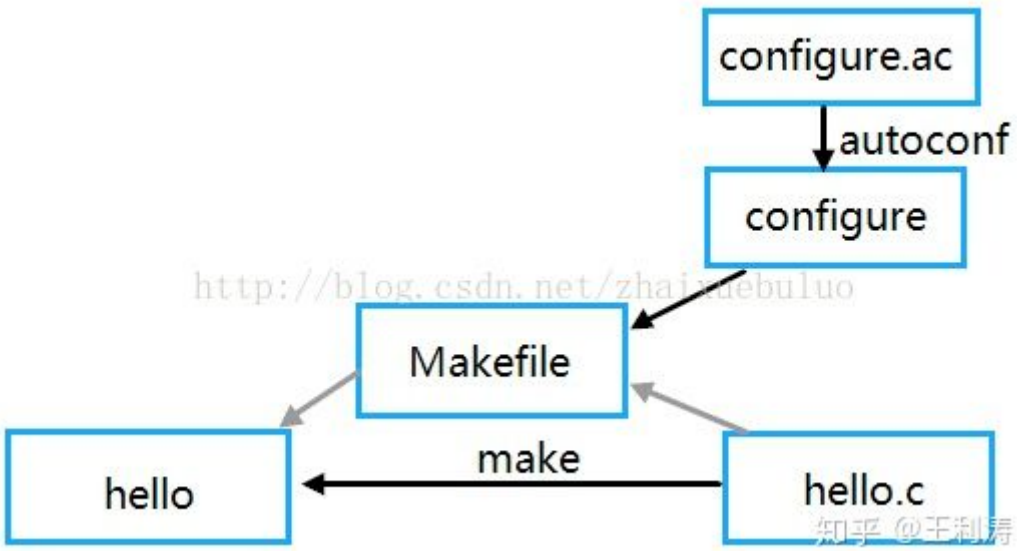
二、Autoconf 时代

后来 Linux 操作系统问世，后续的版本也越来越多，各种发布版本错综复杂，比如 Redhat 系列、Debian 系列等。差异越来越大，甚至包括操作系统的接口都出现差异。这时候别说 makefile 能不能正确编译的问题了，就连我们编写的应用程序，即使编译正确，也有可能在其它的平台上运行不起来。这个问题，大家都知道，后来出现了 POSIX API 标准，就是可移植的操作系统接口。就是无论你是发布什么版本的 Unix、Linux，OS 的接口要遵循这个标准，这样我们编写的应用程序才能在 Linux、Unix 等系列版本的操作系统上畅通无阻地运行。

而对于 makefile 来说，为了适配操作系统的更多版本，只能不断地添加代码，这就导致 configure 脚本越来远大，导致后来开发人员再也受不了了，维护成本越来越高。

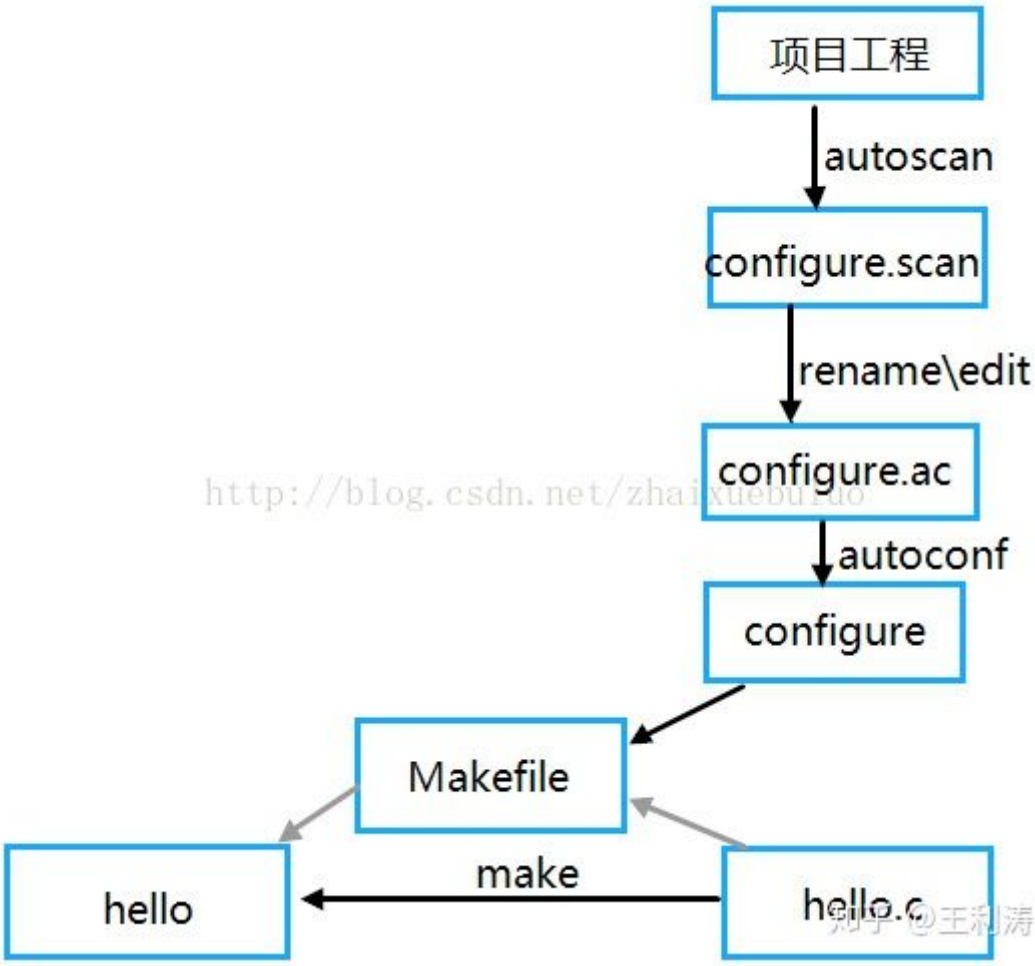
1991年，David Mackenzie 开发了 Autoconf 工具，用来自动生成 configure 脚本。因为对于大多数用户来说，对于不同版本的差异、库的版本、底层的细节，鬼才懒得管。他们关心的就是库文件、头文件的位置、软件最终的安装路径是在哪里。所以这个工具的出现，大大解放了开发人员的时间，给广大程序员带来了福音。

用户只需要定义几个宏，表示我们关心的配置选项，保存在 configure.ac 文件里，然后使用 Autoconf 工具就可以帮我们自动生成 configure 脚本了！



Autoconf 工具真是比大姨妈还贴心，为了减轻程序员的负担，configure.ac 文件也不用我们手写了。Autoconf工具包里有个 autoscan 工具，可以自动扫描我们的项目，生成一个 configure.scan 文件，里面自动添加了很多宏，省得我们再手工添加。

我们只需要将这个 configure.scan 文件重命名为 configure.ac 文件，再修修补补，就可以了。



Autoconf 工具大大减轻了程序员的负担，妈妈，再也不用担心晚回家吃饭了

三、Automake 时代

然而，随着项目越来越大，makefile也越来越复杂，尤其是大型项目，手写越来越困难，怎么办？

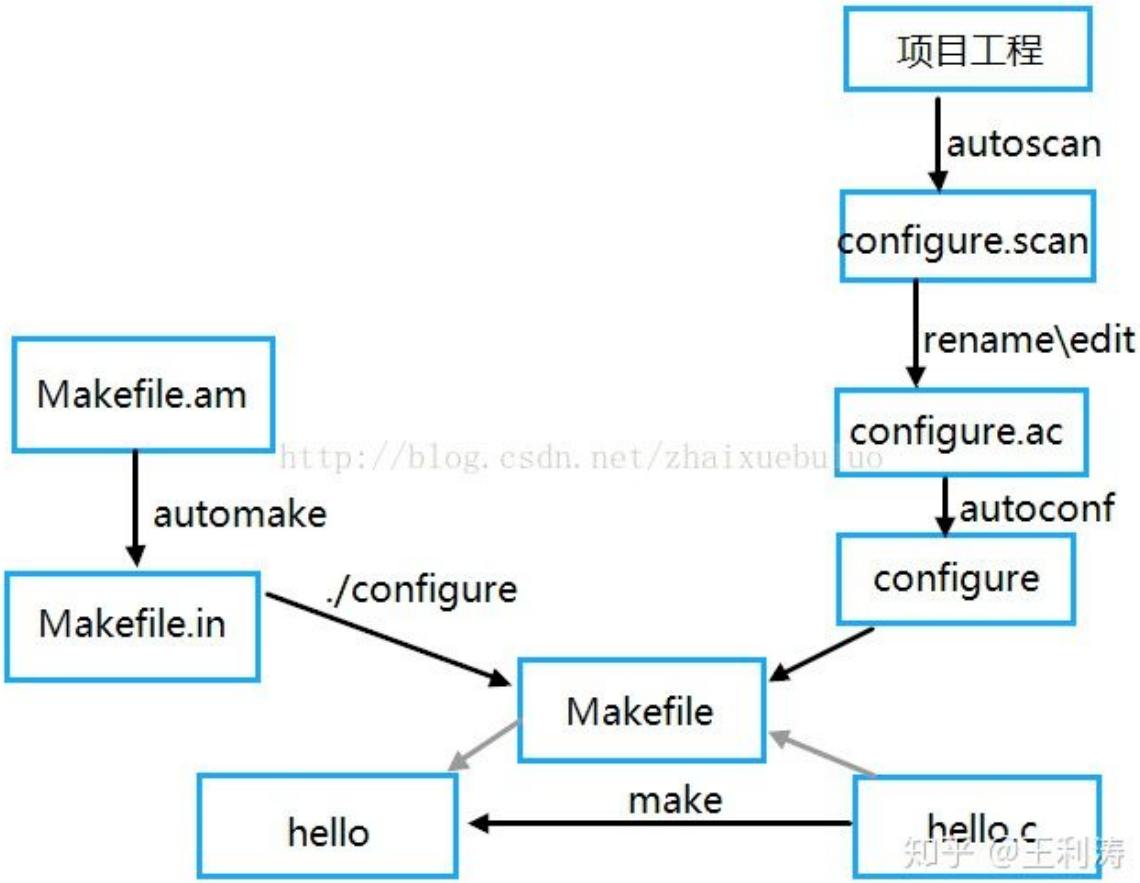
automake 工具这个时候闪亮登场了！

对于开发人员来说，我们关心的就是这个项目要生成什么可执行文件，需要编译哪些源文件，至于怎么编译的？底层的链接细节，鬼才懒得管。程序员的加班已经够多，内心已经够疲惫，我们拒绝makefile的肆虐和压迫！

使用 automake 工具，我们只需要手工编写一个 makefile.am 文件，在里面定义我们想要生成的目标、需要编译的源文件，然后使用 automake 工具就可以帮我们自动生成 makefile！

但是此时的 makefile，是通用的 makefile，定义了程序编译、链接的通用规则，保存在 makefile.in 里面。

如果想在特定平台上成功编译，还需要我们前面的脚本 configure 配置一下，最终才生成我们项目的 Makefile。



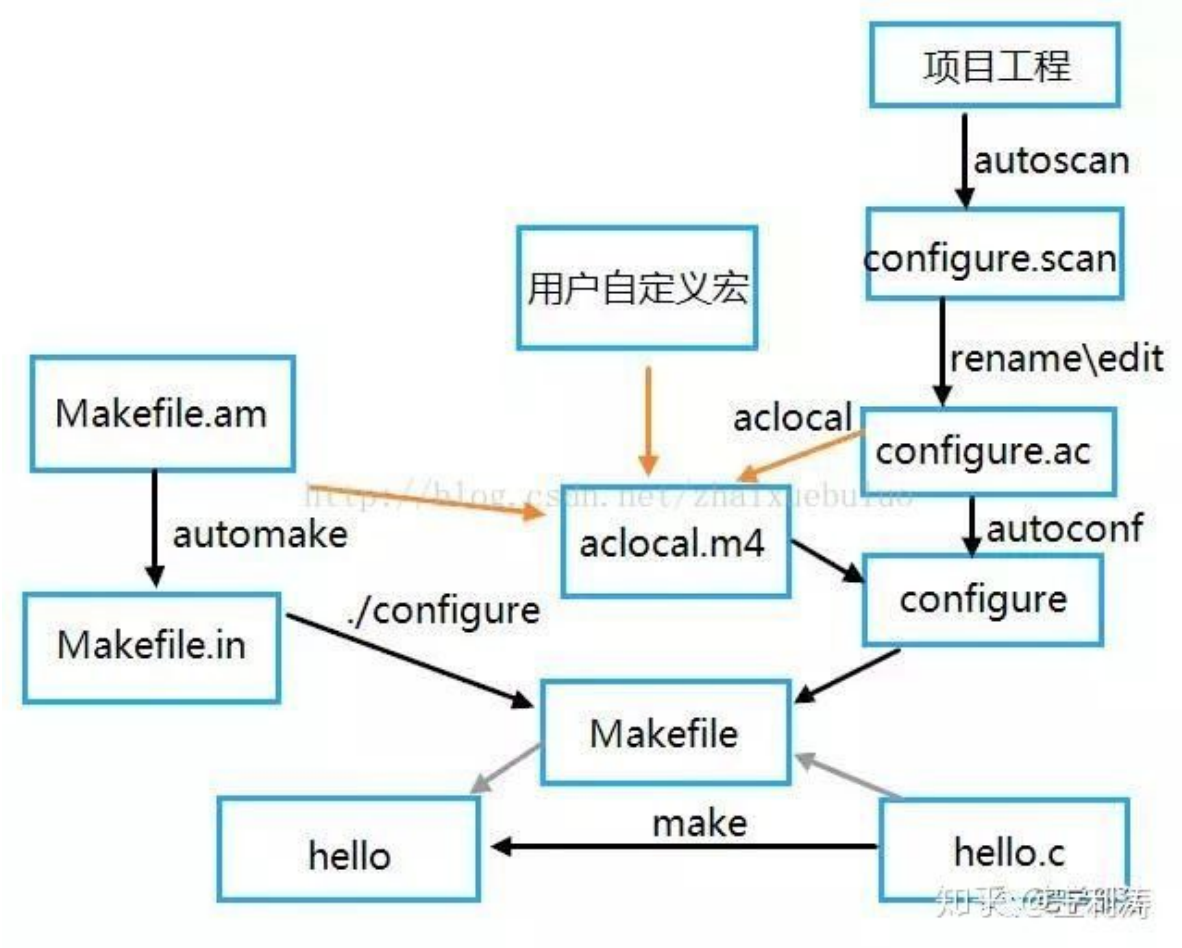
Autoconf 工具通过定义一系列的宏，提供给我们使用，让我们去设置一些需要的配置选项、去配置我们的makefile。

后来 Automake 工具出现后，自定义了一些宏，对这些宏进行了扩展。比如，Autoconf 以前都是单独使用的，现在要跟 automake 配合使用，要在 configure.ac 文件里添加一个 AM_INIT_AUTOMAKE 这个宏。

这个宏是在 automake 工具包里定义的，当我们运行 autoconf 命令的时候，就是出错，因为找不到这个宏的定义。怎么办，咋办？

后来在 configure.ac 同目录下，定义一个 aclocal.m4 格式的文件，里面存放用户定义的一些宏、或者 automake 的一些宏，这样，autoconf 运行的时候，就可以在这个文件里找到宏定义了。

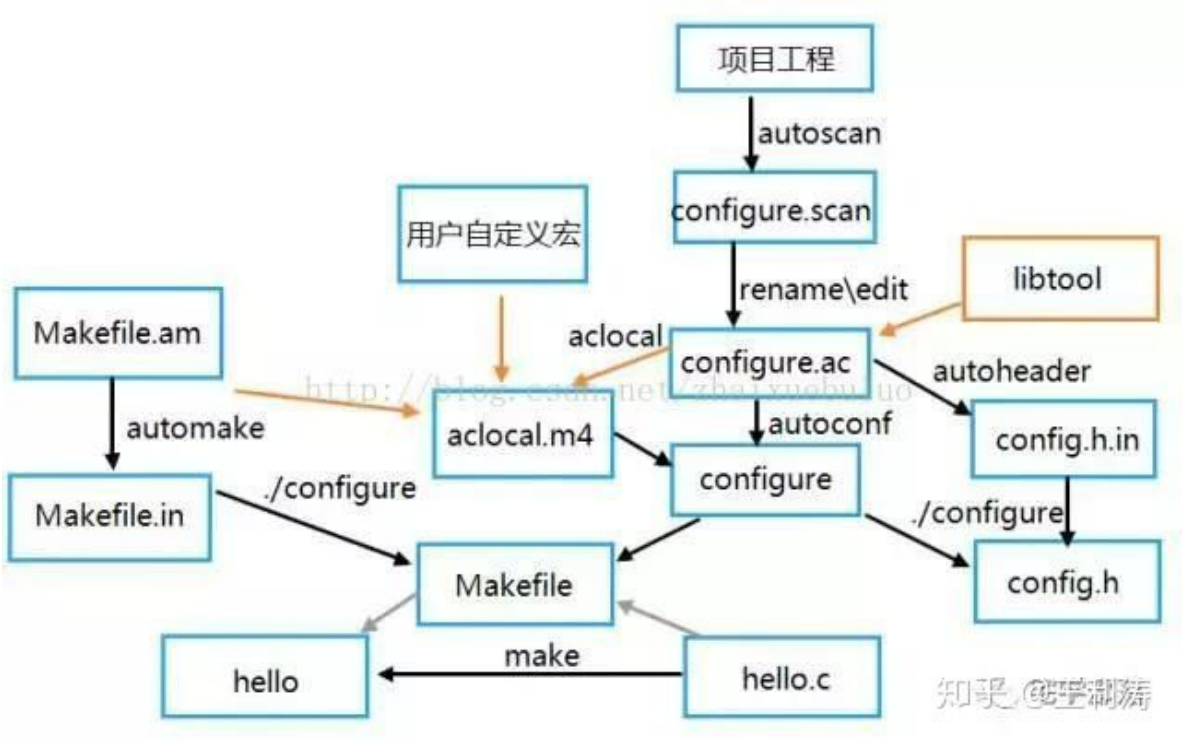
偷懒，是人类社会进步的最大动力。后来，为了进一步减少工作量，又出现一个 aclocal 工具，会自动将automake、autoconf 以及用户定义的所有宏统统放在 aclocal.m4 文件里。



为什么要保存在 aclocal.m4 这种格式的文件里？我也不知道....，m4，macro 宏后面4个字母，缩写就是 m4.

文本文件嘛，后缀名可以随便定义，比如我姓王，后缀名定义为.wang 也是可以的。不要纠结于这些细节，我们的征途是构建 makefile。

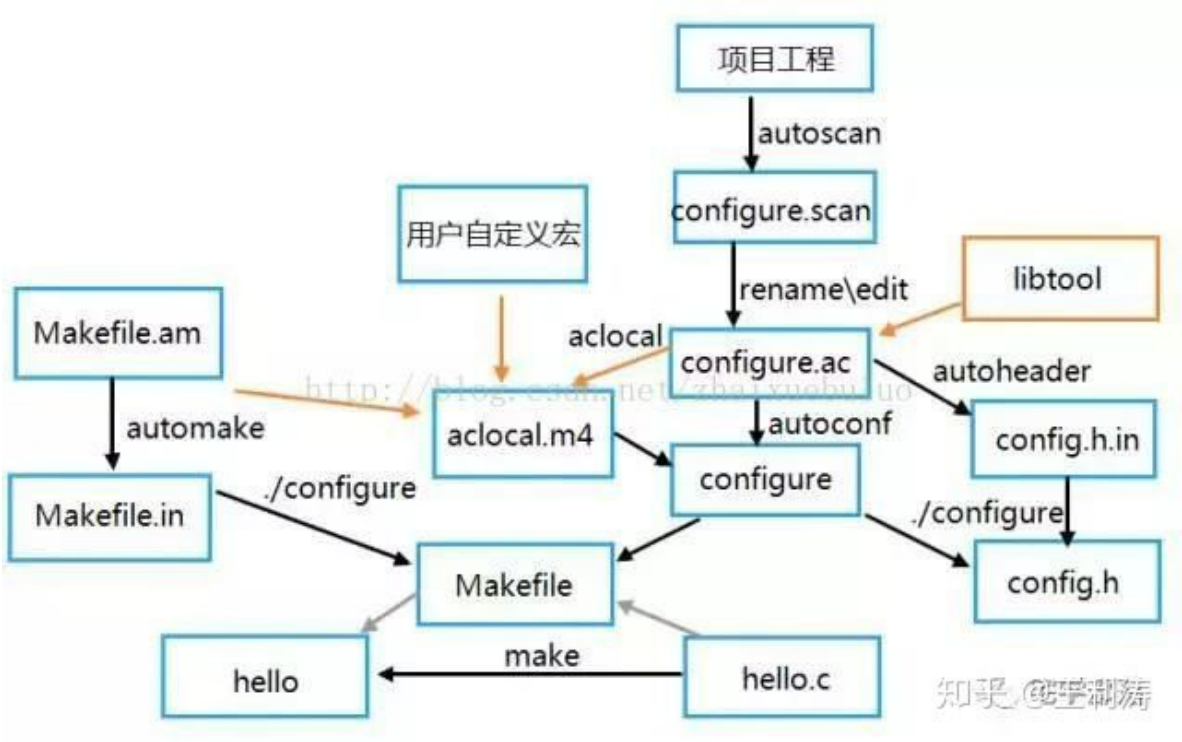
autoconf 工具包里还有一个 autoheader 工具，用来将 configure.ac 里面的宏配置转换为我们C语言格式的#define 形式，并保存在 config.h.in 文件里，当我们运行 ./configure 生成 makefile 的时候，顺便也会将config.h.in 转换为 config.h 文件，这样在我们的程序里，如果想使用这些宏，就可以直接 #include “config.h” 就可以了。



比如头文件里面定义的软件版本号 VERSION 宏，就可以在程序里直接使用，打印在程序里打印我们的软件版本号。

四、Libtool 时代

随着Unix、Linux 之间的差异越来越大，对动态共享库的管理差异也越来越大，比如有些共享库，使用 .so 格式，有的是 .a，有的是 .o 的形式。运行时对动态库的管理方式也一样，有的操作系统支持动态加载，有的就不支持。这就对我们 Makefile 带来了挑战。怎么办？Libtool 的工具出现就是为了解决这个问题的，它通过对生成的动态库进行抽象，统一生成 .la 的形式，可以支持十几种各种不同的平台。



具体的使用教程，

可以参考我发布的视频教程：《Linux三剑客》之Makefile工程实践第2季：使用Autotools自动生成Makefile。

专注嵌入式、Linux精品教程：https://wanglitao.taobao.com/
嵌入式技术教程博客：http://zhaixue.cc/
联系 QQ：3284757626
嵌入式技术交流QQ群：475504428
微信公众号：宅学部落(armlinuxfun)