大型项目使用Automake/Autoconf完成编译配置

使用过开源C/C++项目的同学们都知道，标准的编译过程已经变成了简单的三部曲：configure/make/make install, 使用起来很方便，不像平时自己写代码，要手写一堆复杂的Makefile，而且换个编译环境，Makefile还需要修改（Eclipse也是这样）。

这么好的东东当然要拿来用了，但GNU的Autotool系列博大精深，工具数量又多，涉及的语言也多，要是自己从头看到尾，黄花菜都凉了，项目估计早就结束了；上网搜样例倒是有一大堆，但都是“hello world”的样例，离真正完成大型项目的目标还差得远。

没有办法，对照网上的样例，再找几个开源的源码，然后参考各种Autotools的手册，花了2天时间，终于完成了一个基本可用的Autotools。为了避免其他XDJM也浪费时间，因此将过程总结下来，就算是新年礼物，送给大家！！

**提纲挈领：使用Autotools其实很简单**

大家不要看到那么多工具，其实使用起来很简单，总结起来就是两部分：

1. 按照顺序调用各个工具；
2. 修改或者添加3个文件；

整个操作顺序如下图：



听到我这么讲，大家是否觉得有信心了？好的，下面我们来看具体如何操作：

1. 源码根目录调用**autoscan**脚本，生成configure.scan文件，然后将此文件重命名为configure.ac(或configure.in,早期使用.in后缀)
2. 修改**【configure.ac】**，利用autoconf提供的各种M4宏，配置项目需要的各种自动化探测项目
3. 编写**【自定义宏】**，建议每个宏一个单独的\*.m4文件；
4. 调用**acloca**l收集configure.ac中用到的各种非Autoconf的宏，包括自定义宏；
5. 调用**autoheader**，扫描configure.ac(configure.in)、acconfig.h（如果存在），生成config.h.in宏定义文件，里面主要是根据configure.ac中某些特定宏(如AC\_DEFINE)生成的#define和#undefine宏，configure在将根据实际的探测结果决定这些宏是否定义（具体见后面例子）。
6. 按照automake规定的规则和项目的目录结构，编写一个或多个【**Makefile.am】**(Makefile.am数目和存放位置和源码目录结构相关)，Makefile.am主要写的就是编译的目标及其源码组成。
7. 调用**automake**，将每个Makefile.am转化成Makefile.in，同时生成满足GNU编码规范的一系列文件（带-a选项自动添加缺少的文件，但有几个仍需要自己添加，在执行automake前需执行touch NEWS README AUTHORS ChangeLog）。如果configure.ac配置了使用libtool(定义了AC\_PROG\_LIBTOOL宏（老版本）或LT\_INIT宏)，需要在此步骤前先在项目根目录执行libtoolize --automake --copy --force，以生成ltmain.sh，供automake和config.status调用。
8. 调用**autoconf**，利用M4解析configure.ac，生成shell脚本configure。以上几步完成后，开发者的工作就算完成了，后面的定制就由开源软件的用户根据需要给configure输入不同的参数来完成。
9. 用户调用configure，生成Makefile，然后make && make install。

整个过程步骤有9步，但其中有6步你只需要简单的敲一个命令即可，只有剩下的三步需要你动手写一些东西，对应上面步骤中的蓝色黑体字部分，而本文的重点就是如何在大型项目中完成这三歩。

**步步为营：三步完成编译配置**

【**第2步：修改configure.ac文件**】

从上面的步骤可以看到，使用autoscan工具扫描后就会生成一个简单的configure.ac文件，这已经是一个完整的configure.ac文件框架了，但还不足以达到我们的要求，因此我们要在框架里面添加一些东西：

1.1 添加AM\_INIT\_AUTOMAKE宏

在AC\_INIT 宏下一行添加AM\_INIT\_AUTOMAKE([foreign -Wall -Werror])，中括号里面的选项可以根据需要来修改，具体请看automake手册关于这个宏的说明。

1.2 如果需要，添加AC\_CONFIG\_HEADERS([config.h])宏

添加这个宏很简单，但关键是“如果需要”，什么情况下需要这个宏呢？

这个宏的目的是输出config.h，这是一个C的头文件，里面主要是包含很多宏定义#define，说到这里其实就很明确了，输出这个文件的目的就是提供各种相关的宏，而宏在代码中的作用就是#ifdef，也就是说：如果你的代码需要用到宏开关进行控制，那么就要输出这个文件。具体的使用方法如下：

1. 首先确定代码中需要使用什么宏来进行开关定制，确定宏的名称，编写和宏相关的代码，且要包含config.h的头文件；
2. 在configure.ac中的各种处理（例如AC\_CHECK\_\*\*\*，AC\_ARG\_\*\*\*）中使用AC\_DEFINE宏定义C/C++的宏，名称和上面的相同；如果是使用AC\_CHECK\_HEADERS，会自动添加宏定义；
3. 执行完第7歩后，Autoconf就会自动生成config .h文件

1.3添加编译链接需要的程序

编译、链接需要用到的程序需要添加在# Checks for programs.注释后面。对于C/C++来说，最常见的就是gcc, g++, 静态库编译、动态库编译，对应的选项如下：

AC\_PROG\_CXX

AC\_PROG\_CC

AC\_PROG\_RANLIB

如果使用libtool编译，则选项如下，注意使用了libtool则需要将AC\_PROG\_RANLIB去掉

LT\_INIT

1.4 在configure.ac代码中各个部分添加自己的检测处理

这一步是我们的主要工作，需要根据自己的项目具体情况来编写，常见操作对应的宏和样例请参考本文后面的“【**常见操作对应的宏**】”：。至于具体添加在哪个地方，configure.ac中的注释已经清楚的告诉你了，例如：

# Checks for libraries.

# Checks for library functions.

1.5 在AC\_OUTPUT上一行添加AC\_CONFIG\_FILES宏

添加这个宏的目的是制定Autoconf输出哪些文件，常见的文件就是Makefile文件，config.h在AC\_CONFIG\_HEADERS宏里面指定了，这里不需要再次指定。例如：

|  |
| --- |
| AC\_CONFIG\_FILES([Makefile tools/Makefile common/Makefile worker/Makefile]) |

【**第3步：编写自定义的Autoconf宏**】

Autoconf虽然提供了很多内置的宏，但在实际项目中，这些宏不可能满足所有的要求，有的处理还是要自己完成。虽然在configure.ac文件中可以直接编写各种处理代码，但这样做有几个缺点：

1. 很不美观：打开configure.ac文件，密密麻麻的一大段花花绿绿的Shell代码，看着眼花缭乱；
2. 修改起来很麻烦：要找半天才能找到要修改的位置，一不小心就改错了；

就像写C/C++代码要进行封装一样，Autoconf的处理也需要进行封装，这个封装就是自定宏，定义完成后在configure.ac中调用，看起来很清爽，修改也很简单。

下面我们来看如何自定义宏：

2.1 新建一个单独的目录，用于存放自定义宏，一般定义为m4

2.2 新建自定义宏文件

建议每个宏一个文件，文件必须以.m4结尾，文件名就是宏名（当然如果你非要不这么做也可以，文件名随便取）

2.3 编写Autoconf宏

具体的编写方式请参考Autoconf的手册第10章节，最好边看手册边对照一个开源软件的样例，这样效果最好了。这里说明几个需要注意的地方：

1）m4宏不是shell，请不要直接在文件中写shell代码，而要在宏的各个部分里面写代码；

最常见的就是if-else判断，如果要在代码中编写if-else判断，需要使用AS\_IF宏，或者在其它宏里面写，例如AC\_ARG\_WITH, AC\_CACHE\_CHECK；

2）AC\_DEFUN是定义autoconf的宏，AC\_DEFINE是定义C/C++的config .h里面的宏，不要混淆了；

2.4 运行aclocal工具，生成aclocal.m4

由于自定义宏是放在我们新建的目录中的，configure.ac并没有像C/C++那样的include语句可用，因此也就找不到这些宏，这时就需要aclocal工具了：aclocal会将自定义宏编译成configure.ac可用的宏，保存在和configure.ac同级目录下的aclocal.m4文件中，这样在configure.ac就能够直接使用了。具体的编译方法如下（m4就是你的目录）:

aclocal -I m4

同时需要在根目录下的Makefile.am中添加ACLOCAL\_AMFLAGS = -I m4。

还有一种方法是将所有的自定义宏都放入到一个acinclude.m4文件中，不过不推荐这种方法，原因是因为这种方法的缺点和直接将所有自定义宏放入configure.ac中没有多大差别。

【**第6步：编写Makefile.am文件**】

对于大型项目来说，代码一般都是分目录存放的，而不会像Hello world样例那样简单的就几个文件，因此写Makefile.am就麻烦一些，但其实主要是工作量增加了，原则都是一样的：

**原则1**：每个目录一个Makefile.am文件；同时在configure.ac的AC\_CONFIG\_FILES宏中指定输出所有的Makefile文件，例如：

AC\_CONFIG\_FILES([Makefile tools/Makefile common/Makefile worker/Makefile])

**原则2**：父目录需要包含子目录

在父目录下的Makefile.am中添加: SUBDIRS = 所有子目录，例如SUBDIRS=test tools

**原则3**：Makefile.am中指明当前目录如何编译

前两个原则很简单，这里就不多说了，重点说一下如何编写Makefile.am。

编写Makefile.am主要是完成3件事情：编译(make)、安装(make install)、打包(make dist)，下面我们一一来进行讲解。

3.1 编译安装

编译和安装的规则是绑定在一起的，通过同一条语句同时指定了编译和安装的处理方式，具体的格式为：安装目录\_编译类型=编译目标

3.1.1【安装目录】

例如：bin\_PROGRAMS = hello subdir/goodbye，其中安装目录是bin，编译类型是PROGRAMS，编译目标是两个程序hello, goodbye.

常用缺省的安装目录如下

|  |  |  |
| --- | --- | --- |
| 目录 | Makefile.am中的变量 | 使用方式 |
| prefix | /usr/local | 安装目录，通过--prefix指定 |
| exec\_prefix | ${prefix} | 同prefix |
| bindir | ${exec\_prefix}/bin | bin\_编译类型 |
| libdir | ${exec\_prefix}/lib | lib\_编译类型 |
| includedir | ${prefix}/include | include\_编译类型 |
| noinstdir | 无 | noinst\_编译类型，特殊的目录，表示编译目标不安装。 |

除了常用的缺省目录外，有时候我们还需要**自定义目录**，例如我们希望安装完成后安装目录下有一个配置文件目录config，同时将指定的test.ini拷贝到config目录，则config目录需要通过自定义目录方式定义，然后按照缺省目录的使用方式使用。例如：

在根目录下的Makefile.am中添加如下内容：

configdir=${prefix}/config => 定义一个自定义的目录名称config,注意dir后缀是固定的

config\_DATA=config/test.ini => 使用自定义的目录config，必须要有这句，否则目录不会创建, =号后面如果有对应的文件，安装时会将对应的文件拷贝到config目录下。

3.1.2【编译类型】

常见编译类型如下，没有自定义编译类型

|  |  |  |
| --- | --- | --- |
| 类型 | 说明 | 使用方式 |
| PROGRAMS | 可执行程序 | bin\_PROGRAMS |
| LIBRARIES | 库文件 | lib\_LIBRARIES |
| LTLIBRARIES (Libtool libraries) | libtool库文件 | lib\_LTLIBRARIES |
| HEADERS | 头文件 | include\_HEADERS |
| SCRIPTS | 脚本文件，有可执行权限 | test\_SCRIPTS（需要自定义test目录） |
| DATA | 数据文件，无可执行权限 | conf\_DATA(需要自定义conf目录) |

3.1.3【编译目标】

编译目标其实就是编译类型对应的具体文件，其中需要make生成的文件主要有如下几个：可执行程序\_PROGRAMS，普通库文件\_LIBRARIES，libtool库文件\_LTLIBRARIES,其它类型对应的编译目标不需要编译，源文件就是目标文件。

* 标准的编译配置

如果你熟悉gcc的编译命令写法，那么Automake的Makefile.am编译过程就很好写了。因为Automake只是将写在一行gcc命令里的各个不同部分的信息分开定义而已。我们来看具体是如何定义的：

\_SOURCES：对应gcc命令中的源代码文件

\_LIBADD：编译链接库时需要链接的其它库，对应gcc命令中的\*.a等文件

\_LDADD：编译链接程序时需要链接的其他库，对应gcc命令中的\*.a等文件

\_LDFLAGS：链接选项，对应gcc命令中的-L, -l, -shared, -fpic等选项

\_LIBTOOLFLAGS：libtool编译时的选项

\*\*FLAGS(例如\_CFLAGS/\_CXXFLAGS)：编译选项，对应gcc命令中的-O2, -g, -I等选项

举例如下：

|  |
| --- |
| #不同的编译类型只是第一句不一样，后面的编译配置都是一样的  bin\_PROGRAMS= myproject  myproject\_SOURCES = main.c  myproject\_LDADD = ./utils/libutils.a ./module1/libmodule1.a ./core1/libcore.a  myproject\_LDFLAGS = -L/home/test/local -lmemcached  myproject\_CFLAGS = -I./core1/ -I./module1/ -I./utils/ -O2 -g |

* 如何编译可执行程序

对于大型项目来说，代码基本上都是分目录存放的，如果是直接写makefile文件，一般都是将所有源文件首先编译成\*.o的文件，再链接成最终的二进制文件。但在Automake里面这样是行不通的，因为你只要仔细看编译类型表格就会发现，并没有一种编译类型能够编译\*.o文件，无法像常规makefile那样来编写，所以就需要采取一些技巧。

其实这个技巧也很简单：将非main函数所在目录的文件编译成静态链接库，然后采用链接静态库的方式编译可执行程序。

样例如下：

=================根目录Makefile.am======================

|  |
| --- |
| #对应Makefile.am原则2  SUBDIRS = tools common worker |

=================tool目录Makefile.am======================

|  |
| --- |
| #只是为了编译而生成的.a库文件，没有必要安装, 所以是noinst  noinst\_LIBRARIES=libtools.a  libtools\_a\_SOURCES=./urlcode.h \  ./stringtools.cpp \  ./stringtools.h \  ./urlcode.c |

===============common目录Makefile.am======================

|  |
| --- |
| #只是为了编译而生成的.a库文件，没有必要安装, 所以是noinst  noinst\_LIBRARIES=libcommon.a  libcommon\_a\_SOURCES=./iniparser.c \  (省略很多文件, 实际使用时要一一填写)  ./exception.h \ |

==============worker目录Makefile.am============================

|  |
| --- |
| bin\_PROGRAMS=worker  worker\_SOURCES=./workeralgorithm.cpp \  ./worker.cpp \  (省略很多文件, 实际使用时要一一填写)  ./worker.h  #通过\_LDADD告诉Automake需要链接哪些库  worker\_LDADD=../tools/libtools.a ../common/libcommon.a |

* 如何编译静态库

Automake天然支持编译静态库，只需要将编译类型指定为\_LIBRARIES即可。

* 如何编译动态库

需要注意的是：\_LIBRARIES只支持静态库(即\*.a文件),而不支持编译动态库(\*.so)文件，要编译动态链接库,需要使用\_PROGRAMS。除此之外，还需要采用自定义目录的方式避开Automake的两个隐含的限制：

1. 如果使用bin\_PROGRAMS, 则库文件会安装到bin目录下，这个不符合我们对动态库的要求；
2. automake不允许用lib\_ PROGRAMS

下面假设将utils编译成so，采用自定义目录的方式，修改Makefile.am如下：

|  |
| --- |
| mylibdir=$libdir #$libdir其实就是lib目录，请参考【安装目录】表格  mylib\_PROGRAMS= libutils.so  libutils\_so\_SOURCES = utils.c utils.h  libutils\_so\_LDFLAGS = -shared –fpic #这个就是gcc编译动态库的选项 |

* 如何编译libtool库

对于跨平台可移植的库来说，推荐使用libtool编译，而且Automake内置了libtool的支持，只需要将编译类型修改为\_LTLIBRARIES即可。

需要注意的是：如果要使用libtool编译，需要在configure.ac中添加LT\_INIT宏，同时注释掉AC\_PROG\_RANLIB，因为使用了LT\_INIT后，AC\_PROG\_RANLIB就没有作用了。

3.2 打包

Automake缺省情况下会自动打包，自动打包包含如下内容：

1. 所有源文件
2. 所有Makefile.am/Makefile.in文件
3. configure读取的文件
4. Makefile.am’s (using include) 和configure.ac’ (using m4\_include)包含的文件
5. 缺省的文件，例如README, ChangeLog, NEWS, AUTHORS

如果除了这些缺省的文件外，你还想将其它文件打包(一般包括静态库、头文件、配置文件、帮助文件)，有如下两种方法：

1. 粗粒度方式：通过EXTRA\_DIST来指定，指定文件就打包文件，指定目录就打包目录，例如：

EXTRA\_DIST=conf/config.ini test tools/initialize.sh

如果test是目录，那么会将test目录下所有的文件和目录都打包。

1. 细粒度方式：在“安装目录\_编译类型=编译目标”前添加dist(表示需要打包), 或者nodist(不需要打包)，例如：

|  |
| --- |
| #将data\_DATA= distribute-this打包  dist\_data\_DATA = distribute-this  #foo\_ SOURCES不打包  bin\_PROGRAMS = foo  nodist\_foo\_SOURCES = do-not-distribute.c |

【后记】

GNU Autotool工具博大精深，我也是结合项目的实际应用来使用的，并没有完整的研究所有的工具，因此难免存在瑕疵和纰漏，如果大家发现有疑问或者问题的地方，欢迎大家指正。当然，GNU自己的手册是最权威的，如果你有疑问的话，参考手册，以手册为准。

如果想了解autotools的工作原理和流程以及更高级的技巧，请参考胡华强写的《autoconf and automake介绍与典型应用.doc》。

【**常见操作对应的宏**】

1）**给./configure添加--with-package参数，例如：./configure --with-libmemcached**

**AC\_ARG\_WITH**，具体如何写请参考autoconf手册15.2章节，里面给了一个完整的样例。

2）**给./configure添加 –enable-feature参数，例如：./configure –enable-multithread**

**AC\_ARG\_ENABLE**，顾名思义，这个宏的意思就是打开开关，这个开关可以是编译开关，也可以是代码功能开关，如果是编译开关，则要配合AM\_CONDITIONAL宏来使用（样例请看automake手册20.1章节的AM\_CONDITIONAL宏说明）；如果是代码功能开关，则要配合AC\_DEFINE宏来使用（请参考autoconf手册15.2章节的AC\_ARG\_WITH宏的样例）

3）**在./configure的时候检查头文件**

**AC\_CHECK\_HEADER**: 检查一个头文件

**AC\_CHECK\_HEADERS**：检查一批头文件

4）**在./configure时检查库文件**

**AC\_CHECK\_LIB**：样例请参考autoconf手册15.2章节的AC\_ARG\_WITH宏的样例

5）**修改make行为**

如果你想修改默认的make行为，可以先使用AC\_ARG\_WITH或者AC\_ARG\_ENABLE添加./configure参数，再结合如下两个宏来完成：

AM\_CONDITIONAL：在./configure.ac中增加一个automake宏，在Makefile.am中使用if-else-endif来使用宏；

AC\_SUBST：在./configure.ac中直接修改automake的变量，例如AM\_CXXFLAGS，AM\_CFLAGS等编译链接。

【完整样例】



【参考资料】

1. 入门材料：<http://sources.redhat.com/autobook/autobook/autobook_toc.html> 。

2. autoconf手册：<http://www.gnu.org/software/autoconf/manual/autoconf.html> 。

3. automake手册：<http://sources.redhat.com/automake/automake.html> 。

4. libtool手册：<http://www.gnu.org/software/libtool/manual/libtool.html>

5. tutorial：<http://www.lrde.epita.fr/~adl/dl/autotools.pdf> 。