# autotools 入门

授人于鱼不如授人于渔

推荐阅读：
autobook
autoconf手册
automake手册

基本上到这儿，就该结束了。剩下的就不保证绝对正确了。

# autotools的作用:

## 1. 使代码可移植

代码可移植不是自动的，代码自身必须保证其在任意平台可以运行。autotools只是告诉，这个平台什么可以用，什么不可以用。

## 2. 使安装分发很容易

# ./configure
# make
# make install

这个最初是gnu的规范，后来变成了开源的规范。

## 3. 看起来很专业的样子

专业，至少从看起来专业开始。但是切记，估计只有跑在虚拟机上的代码（例如perl），和精通各个平台的人，才能写出真正可移植的代码。

## 我对可移植的理解:

作为公司内部项目，不必追求可移植，在linux平台运行，（只要能）就应该榨干linux提供的每一份特性。

# info autoreconf
`autoreconf' runs `autoconf', `autoheader', `aclocal', `automake',`libtoolize', and `autopoint' (when appropriate) repeatedly to update the GNU Build System in the specified directories and their subdirectories (*note Subdirectories::). By default, it only remakes those files that are older than their sources.

autotools 通常说的是autoconf，automake，autolib，但是还有很多辅助的工具，包括autoheader，aclocal，autoscan。

# 这些工具的关系：

autoconf 根据configure.in(configure.ac)生成configure。
automake 根据 Makefile.am 生成Makefile.in。
autotools 是解决各个平台对.a，.so 的处理不一致的，我不懂这个，今天就不说了。
这些都是开发者要做的。

使用者运行 ./configure 时发生的事情：
configure 检测平台特性，生成config.h，config.h 定义了很多宏，这些宏标识了平台特性，例如：#define HAVE_MEMSET 1；说明该平台提供了memset函数；根据Makefile.in，生成相应平台的Makefile，不同的平台，Makefile规则不同。

上面所的平台特性,其实不准确，准确的说法是，检查特性。configure不会检查这个系统是Linux还是FreeBSD，它只检查某个函数是否得到支持。

注意:autoconf 并不依赖automake，NB的开发者可以手动写Makefile.in，例如apache就没有Makefile.am；configure 只是一个普通的脚本，NB的开发者可以手动写configure（只要它能完成上述功能），例如nginx的作者。

作为不NB的开发者，我们要完成configure.in文件和Makefile.am文件的编写。下面主要讲解着两个文件。

目录结构如下：

```
0> ls
aclocal.m4  config.h.in  configure.in  jobqueue.cpp  Makefile.am  mysqlqueue.cpp
config      configure    jobclient.pl  lib           Makefile.in  mysqlqueue.hpp

0> ls ./lib
conffile.cpp  conffile.hpp  gc.hpp  Makefile.am  Makefile.in  pidfile.hpp
```

主程序文件在顶层目录，子目录lib下存放库文件。

可以看到configure.in 文件， Makefile.am Makefile.in文件, lib/Makefile.am, lib/Makefile.in 文件。

使用autoscan可以生成一个configure.in的样板文件，下面讲解confiugre.in的常用指令。令人欣慰的是多数指令autoscan会帮我们生成。

使用>作为注释，这是方便讲解用的。

```
# perl -pi -e 's/^>.*$//' configure.in
```

这个命令去除所有以>开头的行

configure.in 由宏和shell脚本组成，宏由M4展开，shell脚本原封不动，拷贝到configure。

四个主要的环境变量：

| | |
|---|---|
| CXXFLAGS | C++ compiler flags |
| LDFLAGS | linker flags, e.g. -L<lib dir> if you have libraries in a nonstandard directory <lib dir> |
| CPPFLAGS | C/C++ preprocessor flags, e.g. -I<include dir> if you have headers in a nonstandard directory <include dir> |
| CFLAGS | C compiler flags |

在使用configure时，要区分这四个变量，如果是c++，就不要用CFLAGS。例如：
./configure CPPFLAGS="-I/usr/local/mysql5/include" CXXFLAGS="-g -O3" LDFLAGS="-lmysqlclient"

一些shell变量，用在Makefile.am中
$top_srcdir 源文件顶层目录
$srcdir 当前源文件目录

# -*- Autoconf -*-
# Process this file with autoconf to produce a configure script.

> AC_PREREQ (version), 要求autoconf的版本
AC_PREREQ(2.59)

> AC_INIT (package, version, [bug-report], [tarname], [url])
AC_INIT(jobqueue, 1.0, zhiyong1@staff.sina.com.cn)

> AC_CONFIG_AUX_DIR (dir), 指定辅助文件的目录
> ls config
> config.guess config.sub depcomp install-sh missing
> 这些文件是autoconf生成的，要随源码发布
AC_CONFIG_AUX_DIR(config)

> AC_CONFIG_HEADERS (header)
> 生成的配置头文件的名称
AC_CONFIG_HEADER([config.h])

> AC_CONFIG_SRCDIR (unique-file-in-source-dir)
> ./configure --srcdir=DIR 指定源码目录，configure 查找 unique-file-in-source-dir, 确定指定了正确的源码目录
AC_CONFIG_SRCDIR([jobqueue.cpp])

> AC_ARG_ENABLE (feature, help-string, [action-if-given], [action-if-not-given])
> ./configure --with_debug，自定义configure的参数
> 这里自定义了一个with_debug参数，如果configure带这个参数，设置shell变量
> with_debug的值为yes，否则为no
> 根据with_debug的值，设置不同的编译参数

> AS_HELP_STRING (left-hand-side, right-hand-side)
> ./configure --help
> --with_debug enable debug

```
AC_ARG_ENABLE(with_debug, AS_HELP_STRING([--with_debug],[enable
debug]), with_debug=yes, with_debug=no)
CXXFLAGS="$CXXFLAGS -g -Wall -Wno-comment -Wformat -Wimplicit -
Wparentheses -Wswitch -Wunused -Wshadow"
if test $with_debug = "yes"; then
    CXXFLAGS="$CXXFLAGS -O0 -DDEBUG -D_DEBUG_"
else
    CXXFLAGS="$CXXFLAGS -O2 -DNDEBUG"
fi
```

> AC_PREFIX_DEFAULT (prefix)
> 指定默认安装目录，如果不指定是/usr/local
AC_PREFIX_DEFAULT([/usr/local/jobqueue])

> AM_MAINTAINER_MODE([default-mode])
> automake使用，根据一定的规则，重新生成 configure Makefile.in
AM_MAINTAINER_MODE([enable])

> AM_INIT_AUTOMAKE([OPTIONS])
> automake的运行参数，指定foreign，automake仅检查必须的文件
> 例如，NEWS文件是gnu的标准，指定foreign将不检查NEWS文件是否存在
AM_INIT_AUTOMAKE([foreign])

# Checks for programs.
> 检查c++编译器
AC_PROG_CXX
> 检查c编译器
AC_PROG_CC
> 检查链接器
AC_PROG_LD
> 检查ranlib程序
AC_PROG_RANLIB
> 检查install 程序
AC_PROG_INSTALL

> AC_CHECK_LIB (library, function, [action-if-found], [action-if-not-found],
>                 [other-libraries])
> 指定libcurl中的一个函数(curl_easy_init)，检查是否有libcurl库
> 如果没有指定action-if-found，且找到了libcurl 库，则默认添加-lcurl到LIBS
> 如果没有指定action-if-not-found，且没有找到libcurl 库，则默认不退出configure
> 如果需要改变默认行为，用下面检查mysqlclient的方法

```
# Checks for libraries.
# FIXME: Replace `main' with a function in `-lcurl':
AC_CHECK_LIB([curl], [curl_easy_init])
```

> 检查mysqlclient库，如果存在，增加一个头文件查找位置（-I），
> 增加一个头文件查找位置的原因，下页介绍。
> 如果没找到AC_MSG_ERROR 报错，退出configure

```
# FIXME: Replace `main' with a function in `-lmysqlclient':
AC_CHECK_LIB([mysqlclient], [mysql_init],
        [LIBS="-lmysqlclient $LIBS"
         CPPFLAGS="-I/usr/local/mysql/include $CPPFLAGS"
         LDFLAGS="-L/usr/local/mysql/lib $LDFLAGS"],
        AC_MSG_ERROR([can't find mysqlclient]))
```

AC_CHECK_LIB的原理下面介绍

AC_CHECK_LIB 的检查原理是，生成一小段代码，在这段代码中调用指定的函数，如果这段代码通过编译，则这个库通过检查。

```
char mysql_init ();
int
main ()
{
mysql_init ();
  ;
  return 0;
}
```

这段代码就是AC_CHECK_LIB([mysqlclient],[mysql_init]) 生成的代码。可以看出mysql_init的声明和mysql.h中的不一样，并且也没有包含mysql.h头文件。只要gcc tmp.c -lmysqlclient 能编译通过就行了，它也不保证一定能找到mysql.h头文件。这也是我们在检查mysqlclient时，增加一个头文件包含目录的原因。

事实上，不可奢求太多，configure怎么可能知道mysql_init的返回值呢？

聪明如你，一定会问，c++怎么办，我要包括的库就是一个类？没有函数？如何写AC_CHECK_LIB呢？而且，此时是一定要看到头文件的。

AC_CHECK_LIB 不支持c++类的检查。但是autoconf支持自定义检查代码。事实上，AC_CHECK_LIB 的原理，就是我根据下面的代码，猜出来的。

> AC_LANG (language)
> 指定检查代码的语言
AC_LANG(C++)

> AC_COMPILE_IFELSE (input, [action-if-true], [action-if-false])
> 编译input中的代码
> AC_LANG_PROGRAM (prologue, body)
> 生成代码段，prologue中的内容放在main外，body放到main里
```
AC_COMPILE_IFELSE(
   AC_LANG_PROGRAM([[#include <threadpool.hpp>]],
                   [ThreadPool pool(1, 1)]),
   [LIBS="-lthreadpool $LIBS"],
   AC_MSG_ERROR([can't find threadpool])
)
```

注意这段代码仅检查了能否“编译”，没检查“链接”，我检查编译通过，添加了相应的-l。检查链接用AC_LINK_IFELSE，并且要保证LIBS里面有相应的-l。

代码段如下:
```
#include <threadpool.hpp>
int
main ()
{
ThreadPool pool(1, 1)
  ;
  return 0;
}
```

> 检查dirent.h
AC_HEADER_DIRENT

> 检查标准c头文件
AC_HEADER_STDC

> AC_CHECK_HEADERS (header-file..., [action-if-found], [action-if-not-found], [includes])
> 检查相应的头文件是否存在。
AC_CHECK_HEADERS([fcntl.h stdint.h string.h sys/file.h sys/time.h unistd.h])

# Checks for typedefs, structures, and compiler characteristics.
> 检查一些typedef，数据结构
AC_HEADER_STDBOOL
AC_C_CONST
AC_C_INLINE
AC_TYPE_OFF_T
AC_TYPE_SIZE_T
AC_HEADER_TIME
AC_STRUCT_TM

> 检查库函数
# Checks for library functions.
AC_FUNC_FSEEKO
AC_FUNC_SETVBUF_REVERSED
AC_TYPE_SIGNAL
AC_FUNC_STRFTIME
AC_CHECK_FUNCS([dup2 ftruncate gettimeofday localtime_r memset strerror strstr])

> 如果库函数dup2有定义，#define HAVE_DUP2 1

> 到这些目录找Makefile文件
AC_CONFIG_FILES([Makefile lib/Makefile])

> configure.in 结束
AC_OUTPUT

下面讲解Makefile.am

> make的顺序，例如先编译基础库，再编译应用代码。
SUBDIRS         = lib .

> include 的目录
INCLUDES        = -I$(top_srcdir)/lib

> bin_PROGRAMS, 编译目标
bin_PROGRAMS    = jobqueue

> 目标jobqueue依赖的源文件
jobqueue_SOURCES = mysqlqueue.cpp mysqlqueue.hpp jobqueue.cpp
> 目标jobqueue需要链接的库
jobqueue_LDADD   = lib/libstdx.a

> 不需要编译，只需要分发的脚本
dist_bin_SCRIPTS = jobclient.pl
> 额外需要分发的其他东西
EXTRA_DIST      = jobqueue.conf
> 安装时需要执行的命令
install-data-local:
    mkdir -p $(DESTDIR)$(prefix)/conf
    mkdir -p $(DESTDIR)$(prefix)/logs
    $(INSTALL_DATA) $(top_srcdir)/jobqueue.conf $(DESTDIR)$(prefix)/conf

下面讲解src/Makefile.am

> 不需要安装的库
noinst_LIBRARIES  = libstdx.a

> 这个库依赖的文件
libstdx_a_SOURCES = conffile.cpp conffile.hpp gc.hpp

> 编译成.a的.o
libstdx_a_ADDLIB  = @LIBOBJS@

# 分发简介

打包源文件
# make dist

检查打包是否完整
# make distcheck

清理
# make distclean

# 生成
我们很容易弄糊涂，修改了configure.in的那些地方需要调用什么，修改了Makefile.am需要调用什么，为此推荐命令 autoreconf -ivf

# 调试
configure执行错误，主要检查是宏还是shell错误，宏错误多和宏的函数有关，shell错误和通常的shell错误没啥区别
make错误，主要看看Makefile.am的依赖有没有写对

# 结束
以上介绍，可能有不准确的地方，如有疑问，最好翻查相应的文档

# Q&A