[**libtool工具的使用**](http://www.cnblogs.com/jingzhishen/p/3640406.html)

**http://blog.sina.com.cn/s/blog\_602f87700100fc8t.html**

**libtool作用：**

libtool 是一个通用库支持脚本(/usr/bin/libtool)，将使用动态库的复杂性隐藏在统一、可移植的接口中。

可以在不同平台上创建并调用动态库，我们可以认为libtool是gcc的一个抽象，也就是说，它包装了gcc或者其他的任何编译器，用户无需知道细节， 只要告诉libtool说我需要要编译哪些库即可，并且，它只与libtool文件打交道，例如lo、la为后缀的文件。

**libtool生成一个抽象的后缀名为la高层库libxx.la（其实是个文本文件），并将该库对其它库的依赖关系，都写在该la的文 件中。该文件中的dependency\_libs记录该库依赖的所有库（其中有些是以.la文件的形式加入的）；libdir则指出了库的安装位 置；library\_names记录了共享库的名字；old\_library记录了静态库的名字。**

**libtool使用方法：  
1. 编译为一个object文件：**

以BTX 为例：

源文件为：BTX.c. 头文件在 ../include/BTX.h

#libtool --mode=compile gcc -g -O -I../include -c BTX.c

用gcc编译一切正常：

mkdir .libs  
gcc -g -O -I../include -c BTX.c  -fPIC -DPIC -o .libs/BTX.o  
gcc -g -O -I../include -c BTX.c -o BTX.o >/dev/null 2>&1

#libtool --mode=compile mipsel\_linux\_gcc -g -O -I../include -c BTX.c  
显示需要--tag.

libtool支持多语言，以下是语言与tag的对应：

Language name      Tag name  
C                  CC  
C++                CXX  
Java               GCJ  
Fortran 77         F77  
Windows Resource   RC

#libtool --mode=compile **--tag=CC** mipsel-linux-gcc -g -I../include -c BTX.c

则正常了。

libtool: ignoring unknown tag mipsel-linux-gcc  
 mipsel-linux-gcc -g -I../include -c BTX.c  -fPIC -DPIC -o .libs/BTX.o  
 mipsel-linux-gcc -g -I../include -c BTX.c -o BTX.o >/dev/null 2>&1

--mode:表示此次动作是 compile(编译)。

--tag: 如果是交叉编译器，则需要用--tag指定编译器。

gcc -g -O -I../include -c BTX.c： 很清楚。

然后libtool作了3件事：

1.创建.libs

2.编译了一个与位置无关（-fPIC）的obj文件。 放在 .lib中。

3.编译了一个普通obj文件在本地。

另外，它还创建了一个lo文件。用于描述上面2个obj文件和他们的位置。格式与la文件类似。

**2. 将Object文件编译成可执行文件：**

Object的描述文件为 \*.lo文件，而libtool只与lo,la文件打交道，

#**libtool --mode=link --tag=CC mipsel-linux-gcc main.lo -o test**

则利用先前编译出的main.lo编译出结果: 可执行文件test.

**3. 编译一个库文件：**

#libtool --mode=link --tag=CC gcc -g -O -o libBTX.la -rpath /usr/local/lib BTX.lo

显示结果为：

rm -fr  .libs/libBTX.a .libs/libBTX.la .libs/libBTX.lai .libs/libBTX.so .libs/libBTX.so.0 .libs/libBTX.so.0.0.0  
gcc -shared  .libs/BTX.o   -Wl,-soname -Wl,libBTX.so.0 -o .libs/libBTX.so.0.0.0  
(cd .libs && rm -f libBTX.so.0 && ln -s libBTX.so.0.0.0 libBTX.so.0)  
(cd .libs && rm -f libBTX.so && ln -s libBTX.so.0.0.0 libBTX.so)  
ar cru .libs/libBTX.a  BTX.o  
ranlib .libs/libBTX.a  
creating libBTX.la  
(cd .libs && rm -f libBTX.la && ln -s ../libBTX.la libBTX.la)

link出两个共享库，一个是static，一个则是dynamic；需要注意的是，-rpath必须有才能产生dynamic库来，如果用-static，则只创建static库。

libBTX.la则指出静态库和动态库的名字和位置。

ranlib的作用：  
在一些旧版本的系统上，ranlib负责把静态库转换为其他的某种格式，使得新的库能够更快的链接；现在ar命令已经包含了上述功能；  
为了兼容性，在makefile中还是保留ranlib

#mipsel

**#libtool --mode=link --tag=CC mipsel-linux-gcc  -O -o libBTX.la -rpath /usr/local/lib BTX.lo**

当这样设置时，libtool会使用gcc,Sam 怎么也没法改正。所以只好临时在

/usr/bin/libtool中修改CC="mipsel-linux-gcc" 。这样就可以了。

mipsel-linux-gcc -shared  .libs/BTX.o   -Wl,-soname -Wl,libBTX.so.0 -o .libs/libBTX.so.0.0.0

但这里，Sam发现libBTX.la中并没有显示出libBTX.so依赖于libbluetooth.so。这不是没有达到使用libtool的目的吗？

于是Sam又作了修改：

**#libtool --mode=link --tag=CC mipsel-linux-gcc BTX.lo -rpath /home/sam/work/current/BCM/BCM7405/ToolChain/crosstools\_hf-linux-2.6.18.0\_gcc-4.2-9ts\_uclibc-nptl-0.9.29-20070423\_20080702/mipsel-linux/lib -o libBTX.la -lbluetooth**

果然：在la文件中，可以看到**dependency\_libs=' -lbluetooth'**

则在编译为可执行文件时，可以不用写 -lbluetooth

**#libtool --mode=link --tag=CC mipsel-linux-gcc main.lo libBTX.la -o test -lpthread**

当然，也可以隐式的写：

**#libtool --mode=link --tag=CC mipsel-linux-gcc main.lo -lBTX -o test -lpthread**

结果也正常：

mipsel-linux-gcc .libs/main.o -o .libs/test  /home/sam/work/current/BCM/BCM7405/Bluetooth/Bluetooth\_App/Second\_New\_Version/BTX\_source/BTX\_Release/src/.libs/libBTX.so -lbluetooth -lpthread   -Wl,--rpath -Wl,/home/sam/work/current/BCM/BCM7405/ToolChain/crosstools\_hf-linux-2.6.18.0\_gcc-4.2-9ts\_uclibc-nptl-0.9.29-20070423\_20080702/mipsel-linux/lib  
creating test

libtool 从libBTX.la中取出相应的东西。

**4. 将object + so 编译出一个可执行文件：**

#libtool --mode=link mipsel-linux-gcc -L../resource main.lo libBTX.la -o BTX\_Test -lbluetooth -lpthread

显示为：

mipsel-linux-gcc .libs/main.o -o .libs/BTX\_Test  -L/home/sam/work/current/BCM/BCM7405/Bluetooth/Bluetooth\_App/Second\_New\_Version/BTX\_source/BTX\_Release/resource ./.libs/libBTX.so -lbluetooth -lpthread   -Wl,--rpath -Wl,/usr/local/lib

**背景知识：**

**背景知识1：libtool为何使用 -Wl,--rpath-link -Wl,**

使用libtool解决编译问题看上去没什么问题：库的名称、路径、依赖都得到了很好的解决。但下结论不要那么着急，一个显而易见的问题就是：并不是所有的库都是用libtool编译的。

例如：我用常规办法生成libBTX.so. 则需要编译连接成可执行文件时，就没有libBTX.la文件。

这样，libBTX.so依赖于 libbluetooth.so这个信息就无人得知。

**#libtool --mode=link --tag=CC mipsel-linux-gcc main.lo -L./ -lBTX -o test -lpthread**

则因为没有libBTX.la, 所以根本没有 依赖于 libbluetooth.so的信息。所以用到libbluetooth.so的东西全无法链接。

**选项“-Wl,--rpath-link –Wl,DIR”会把-rpath-link选项及路径信息传递给链接器。**

**相当于 --rpath-link /xxx/lib**

背景知识2：

是因为如果在本地编译的情况下，**link时**，gcc在命令行中找不到一个库（比如上面的liba.so）依赖的其它库（比如libb.so），链接器会按照某种策略到某些路径下面去寻找需要的共享库：  
  
1. 所有由'-rpath-link'选项指定的搜索路径.  
  
2. 所有由'-rpath'指定的搜索路径. '-rpath'跟'-rpath\_link'的不同之处在于,由'-rpath'指定的路径被包含在可执行文件中,并在运行时使用, 而'-rpath-link'选项仅仅在连接时起作用.  
  
3. 在一个ELF系统中, 如果'-rpath'和'rpath-link'选项没有被使用, 会搜索环境变量'LD\_RUN\_PATH'的内容.它也只对本地连接器起作用.  
  
4. 在SunOS上, '-rpath'选项不使用, 只搜索所有由'-L'指定的目录.  
  
5. 对于一个本地连接器,环境变量'LD\_LIBRARY\_PATH'的内容被搜索.  
  
6. 对于一个本地ELF连接器,共享库中的`DT\_RUNPATH'和`DT\_RPATH'操作符会被需要它的共享库搜索. 如果'DT\_RUNPATH'存在了, 那'DT\_RPATH'就会被忽略.  
  
7. 缺省目录, 常规的,如'/lib'和'/usr/lib'.  
  
8. 对于ELF系统上的本地连接器, 如果文件'/etc/ld.so.conf'存在, 这个文件中有的目录会被搜索.

从以上可以看出，在使用本地工具链进行本地编译情况下，只要库存在于某个位置，gcc总能通过如上策略找到需要的共享库。**但在交叉编译下，上述八种策略，可以使用的仅仅有两个：-rpath-link，-rpath**。这两个选项在上述八种策略当中优先级最高，当指定这两个选项时，如果链接需要的共享库找不到，链接器会优先到这两个选项指定的路径下去搜索需要的共享库。通过上面的描述可以看到**：-rpath指定的路径将被写到可执行文件中；-rpath- link则不会**；我们当然不希望交叉编译情况下使用的路径信息被写进最终的可执行文件，所以我们选择使用选项-rpath-link。