理解 pkg-config 工具

By Robin On 2011年03月22日 · 1 Comment · In OS, 信息世界 | IT

你在 Unix 或 Linux 下开发过软件吗？写完一个程序，编译运行完全正常，在你本机上工作得好好的，你放到源代码管理系统中。然后，告诉你的同事说，你可以取下来用了。这时，你长长的出了一口气，几天的工作没有白费，多么清新的空气啊，你开始飘飘然了。

“Hi，怎么编译不过去？”你还沉浸在那种美妙的感觉之中，双臂充满着力量，似乎没有什么问题能难倒你的。正在此时，那个笨蛋已经冲着你嚷开了。

“不会吧，我这边好好的！”表面上你说得很客气，其实，你心里已经骂开了，真笨，不知道脑子干嘛用的。也许，你想的没错，上次，他犯了一个简单的错误，不是你一去就解决了吗。

他喊三次之后，你不得不放下你手上的工作，刚才那种美妙的感觉已经消失得无影无踪了，要不是你把情绪控制得很好，一肚子气就要撒在他身上了。你走到他的电脑前，键入 make，优雅的按下回车。怎么可能出错呢？你信心十足。然而，屏幕上的结果多少有点让人脸红，该死的，libxxx.so 怎么会让不到呢?

你在/usr目录中查找 libxxx.so，一切都逃不过你的眼睛。奇怪，libxxx.so 怎么在 /usr/local/lib 下，不是应该在 /usr/lib 下的吗？这你可不能怪别人，别人想安装在哪里都行，下次还可能安装到 /lib 目录下呢。

以上的场景并非虚构，我都经历过好几次，明明在本机上好好的，在别人的机器上连编译都过不去。可能两人的操作系统一模一样，需要的库都安装上，只是由于个人 喜好不同，安装在不同的目录而已。遇到这种情况，每次都技巧性的绕过去了，用的补丁型的方法，心里老惦记其它地方能不能工作。

今天我们要介绍的 **pkg-config**，为解决以上问题提供了一个优美方

案。从此，你再也不为此担忧了。**pkg-config**提供了下面几个功能：

1. 检查库的版本号。如果所需要的库的版本不满足要求，它会打印出错误信息，避免链接错误版本的库文件。
2. 获得编译预处理参数，如宏定义，头文件的位置。
3. 获得链接参数，如库及依赖的其它库的位置，文件名及其它一些连接参数。
4. 自动加入所依赖的其它库的设置。

这一切都自动的，库文件安装在哪里都没关系！

在使用前，我们说说 **pkg-config** 的原理，**pkg-config** 并非精灵，可以凭空得到以上信息。事实上，为了让**pkg-config**可以得到这些信息，要求库的提供者，提供一个.pc文件。比如gtk+-2.0的pc文件内容如下：

```
prefix=/usr
exec_prefix=/usr
libdir=/usr/lib
includedir=/usr/include
target=x11
gtk_binary_version=2.4.0
gtk_host=i386-redhat-linux-gnu
Name: GTK+
Description: GIMP Tool Kit (${target} target)
Version: 2.6.7
Requires: gdk-${target}-2.0 atk
Libs: -L${libdir} -lgtk-${target}-2.0
Cflags: -I${includedir}/gtk-2.0
```

这个文件一般放在 /usr/lib/pkgconfig/ 或者 /usr/local/lib/pkgconfig/ 里，当然也可以放在其它任何地方，如像 X11 相关的pc文件是放在 /usr/X11R6/lib/pkgconfig 下的。为了让pkgconfig可以找到你的pc文件，你要把pc文件所在的路径，设置在环境变量 PKG_CONFIG_PATH 里。

使用 **pkg-config** 的 –cflags 参数可以给出在编译时所需要的选项，而 –libs 参数可以给出连接时的选项。例如，假设一个 sample.c 的程序用到了 Glib 库，就可以这样编译：

$ gcc -c `**pkg-config** –cflags glib-2.0` sample.c

然后这样连接：

$ gcc sample.o -o sample `**pkg-config** –libs glib-2.0`

或者上面两步也可以合并为以下一步：

$ gcc sample.c -o sample `**pkg-config** –cflags –libs glib-2.0`

可以看到：由于使用了 **pkg-config** 工具来获得库的选项，所以不论库安装在什么目录下，都可以使用相同的编译和连接命令，带来了编译和连接界面的统一。

使用 **pkg-config** 工具提取库的编译和连接参数有两个基本的前提：

- 库本身在安装的时候必须提供一个相应的 .pc 文件(不这样做的库说明不支持 **pkg-config** 工具的使用)。
- **pkg-config** 必须知道要到哪里去寻找此 .pc 文件。

GTK+ 及其依赖库支持使用 **pkg-config** 工具，所以剩下的问题就是如何告诉**pkg-config** 到哪里去寻找库对应的 .pc 文件，这也是通过设置搜索路径来解决的。

对于支持 **pkg-config** 工具的 GTK+ 及其依赖库来说，库的头文件的搜索路径的设置变成了对 .pc 文件搜索路径的设置。.pc 文件的搜索路径是通过环境变量 PKG_CONFIG_PATH 来设置的，**pkg-config** 将按照设置路径的先后顺序进行搜索，直到找到指定的 .pc 文件为止。

安装完 Glib 后，在 bash 中应该进行如下设置：

$ export PKG_CONFIG_PATH=/opt/gtk/lib/pkgconfig:$PKG_CONFIG_PATH

可以执行下面的命令检查是否 /opt/gtk/lib/pkgconfig 路径已经设置在 PKG_CONFIG_PATH 环境变量中：

$ echo $PKG_CONFIG_PATH

这样设置之后，使用 glib 库的其它程序或库在编译的时候 **pkg-config** 就知道首先要到 /opt/gtk/lib/pkgconfig 这个目录中去寻找 glib-2.0.pc 了（GTK+ 和其它的依赖库的 .pc 文件也将拷贝到这里，也会首先到这里搜索它们对应的 .pc 文件）。之后，通过**pkg-config** 就可以把其中库的编译和连接参数提取出来供程序在编译和连接时使用。

另外还需要注意的是：环境变量的设置只对当前的终端窗口有效。如果到了没有进行上述设置的终端窗口中，**pkg-config** 将找不到新安装的 glib-2.0.pc 文件、从而可能使后面进行的安装（如 glib 之后的 Atk 的安装）无法进行。

在我们采用的安装方案中，由于是使用环境变量对 GTK+ 及其依赖库进行的设置，所以当系统重新启动、或者新开一个终端窗口之后，如果想使用新安装的 GTK+ 库，需要如上面那样重新设置 PKG_CONFIG_PATH 和 LD_LIBRARY_PATH 环境变量。

这种使用 GTK+ 的方法，在使用之前多了一个对库进行设置的过程。虽然显得稍微繁琐了一些，但却是一种最安全的使用 GTK+ 库的方式，不会对系统上已经存在的使用了 GTK+ 库的程序（比如 GNOME 桌面）带来任何冲击。

为了使库的设置变得简单一些，可以把下面的这两句设置保存到一个文件中（比如 set_gtk-2.10 文件）：

```
export
PKG_CONFIG_PATH=/opt/gtk/lib/pkgconfig:$PKG_CONFIG_PA
TH
export LD_LIBRARY_PATH=/opt/gtk/lib:$LD_LIBRARY_PATH
```

之后，就可以用下面的方法进行库的设置了（其中的 source 命令也可以用 . 代替）：

```
$ source set_gtk-2.10
```

只有在用新版的 GTK+ 库开发应用程序、或者运行使用了新版 GTK+ 库的程序的时候，才有必要进行上述设置。

如果想避免使用 GTK+ 库之前上述设置的麻烦，可以把上面两个环境变

量的设置在系统的配置文件中（如 /etc/profile）或者自己的用户配置文件中（如 ~/.bash_profile）；库的搜索路径也可以设置在 /etc/ld.so.conf 文件中，等等。这种设置在系统启动时会生效，从而会导致使用 GTK+ 的程序使用新版的 GTK+ 运行库，这有可能会带来一些问题。当然，如果你发现用新版的 GTK+ 代替旧版没有什么问题的话，使用这种设置方式是比较方便的。

库文件在连接（静态库和共享库）和运行（仅限于使用共享库的程序）时被使用，其搜索路径是在系统中进行设置的。一般 Linux 系统把 /lib 和 /usr/lib 两个目录作为默认的库搜索路径，所以使用这两个目录中的库时不需要进行设置搜索路径即可直接使用。对于处于默认库搜索路径之外的库，需要将库的位置添加到库的搜索路径之中。设置库文件的搜索路径有下列两种方式，可任选其一使用：

1. 在环境变量 LD_LIBRARY_PATH 中指明库的搜索路径。
2. 在 /etc/ld.so.conf 文件中添加库的搜索路径。

将自己可能存放库文件的路径都加入到 /etc/ld.so.conf 中是明智的选择。（^_^）

添加方法也极其简单，将库文件的绝对路径直接写进去就OK了，一行一个。例如：

/usr/X11R6/lib
/usr/local/lib
/opt/lib

需要注意的是：第二种搜索路径的设置方式对于程序连接时的库（包括共享库和静态库）的定位已经足够了，但是对于使用了共享库的程序的执行还是不够的。这是 因为为了加快程序执行时对共享库的定位速度，避免使用搜索路径查找共享库的低效率，所以是直接读取库列表文件 /etc/ld.so.cache 从中进行搜索的。/etc/ld.so.cache 是一个非文本的数据文件，不能直接编辑，它是根据 /etc/ld.so.conf 中设置的搜索路径由 /sbin/ldconfig 命令将这些搜索路径下的共享库文件集中在一起而生成的（ldconfig 命令要以 root 权限执行）。因此，为了保证程序执行时对库

的定位，在 /etc/ld.so.conf 中进行了库搜索路径的设置之后，还必须要运行 /sbin/ldconfig 命令更新 /etc/ld.so.cache 文件之后才可以。
ldconfig ,简单的说，它的作用就是将 /etc/ld.so.conf 列出的路径下的库文件缓存到/etc/ld.so.cache以供使用。因此当安装完一些库文件（例如刚安装好 glib 或者修改 ld.so.conf 增加新的库路径)后，需要运行一下 /sbin/ldconfig 使所有的库文件都被缓存到 ld.so.cache 中，如果没做，即使库文件明明就在 /usr/lib 下的，也是不会被使用的，结果编译过程中抱错，缺少xxx库，去查看发现明明就在那放着，搞的想大骂 computer 蠢猪一个。 （^_^）
在程序连接时，对于库文件（静态库和共享库）的搜索路径，除了上面的设置方式之外，还可以通过 -L 参数显式指定。因为用 -L 设置的路径将被优先搜索，所以在连接的时候通常都会以这种方式直接指定要连接的库的路径。
前面已经说明过了，库搜索路径的设置有两种方式：在环境变量 LD_LIBRARY_PATH 中设置以及在 /etc/ld.so.conf 文件中设置。其中，第二种设置方式需要 root 权限，以改变 /etc/ld.so.conf 文件并执行 /sbin/ldconfig 命令。而且，当系统重新启动后，所有的基于 GTK2 的程序在运行时都将使用新安装的 GTK+ 库。不幸的是，由于 GTK+ 版本的改变，这有时会给应用程序带来兼容性的问题，造成某些程序运行不正常。为了避免出现上面的这些情况，在 GTK+ 及其依赖库的安装过程中对于库的搜索路径的设置将采用第一种方式进行。这种设置方式不需要 root 权限，设置也简单：

```
$ export LD_LIBRARY_PATH=/opt/gtk/lib:$LD_LIBRARY_PATH
```

可以用下面的命令查看 LD_LIBRAY_PATH 的设置内容：

```
$ echo $LD_LIBRARY_PATH
```

最后，我来总结一下，PKG_CONFIG_PATH 主要指明.pc文件的所在路径，这样 **pkg-config** 工具就可以根据.pc文件的内容动态生成编译和连接选项，比如 Cflags （编译用）和 Libs （连接用），如果使用的是动态链接库，那么程序在连接和运行时，一般 Linux 系统把 /lib 和 /usr/lib

两个目录作为默认的库搜索路径，对于处于默认库搜索路径之外的库，系统管理员可以设置 LD_LIBRARY_PATH 环境变量或在 /etc/ld.so.conf 文件中添加库的搜索路径。值得说明的是，使用 gcc 连接时的选项，如果不用 **pkg-config** 工具，需要显示的声明连接的动态链接库名。使用 gcc 的同学可以查看下面的注意事项。

Linux 系统中，为了让动态链接库能被系统中其它程序共享,其名字应符合 lib*.so.* 这种格式。如果某个动态链接库不符合此格式，则 Linux 的动态链接库自动装入程序（ld）将搜索不到此链接库，其它程序也无法共享之。格式中，第一个*通常表示为简写的库名,第二个*通常表示为该库的版本号。如在我的系统中，基本C动态链接库的名字为 libc.so.6，线程 pthread 动态链接库的名字为 libpthread.so.0 等等。如果没有指定版本号，比如 libmy.so ，这也是符合要求的格式。

gcc 命令几个重要选项:

- -shared 该选项指定生成动态连接库（让连接器生成T类型的导出符号表，有时候也生成弱连接W类型的导出符号，不用该标志外部程序无法连接。相当于一个可执行文件）。
- -fPIC：表示编译为位置独立的代码，不用此选项的话编译后的代码是位置相关的所以动态载入时是通过代码拷贝的方式来满足不同进程的需要，而不能达到真正代码段共享的目的。
- -L.：表示要连接的库在当前目录中。
- -lmy：编译器查找动态连接库时有隐含的命名规则，即在给出的名字前面加上lib，后面加上.so来确定库的名称（libmy.so）。

当然如果有 root 权限的话，可以修改 /etc/ld.so.conf 文件，然后调用 /sbin/ldconfig 来达到同样的目的，不过如果没有 root 权限，那么只能采用输出 LD_LIBRARY_PATH 的方法了。