Linux 下编译静态库和动态库

假设当前目录下有这些源文件：[main.c　　func.c　　func.h]，其中 main.c 要调用 func.c 中的函数。

【1】生成静态库：

```
$ gcc -c func.c -o func.o
$ ar rcs libfunc.a func.o
$ gcc main.c -o main -static -L. -lfunc
$ ./main
```

【2】生成动态库：

```
$ gcc -fPIC -c func.c -o func.o
$ gcc -shared -o libfunc.so.1.0.0 func.o
$ ln -s libfunc.so.1.0.0 libfunc.so
$ gcc main.c -o main -L. -lfunc
$ export LD_LIBRARY_PATH=$(pwd)
$ ./main
```

如果将 so 文件 copy 到系统 lib 目录(如/usr/lib)，则最后 2 步就不用了。

最后还有 3 个小知识：

【1】nm 命令：列出目标文件或 2 进制文件的所有符号。

【2】ldd 命令：列出为了使程序正常运行所需要的所有共享库。

【3】/etc/ld.so.conf 文件：除了标准目录（/lib 和/usr/lib）之外，链接器和加载器搜索共享库时要

检查的其他目录，和这个文件相关的一个命令是：ldconfig 。

静态库

在 linux 环境中，使用 ar 命令创建静态库文件.如下是命令的选项:

d -----从指定的静态库文件中删除文件
m -----把文件移动到指定的静态库文件中
p -----把静态库文件中指定的文件输出到标准输出
q -----快速地把文件追加到静态库文件中
r -----把文件插入到静态库文件中
t -----显示静态库文件中文件的列表
x -----从静态库文件中提取文件

还有多个修饰符修改以上基本选项,详细请 man ar 以下列出三个:

a -----把新的目标文件(*.o)添加到静态库文件中现有文件之后
b -----**********之前

v -----使用详细模式
ar 命令的命令行格式如下:
ar [-]{dmpqrtx}[abcfilNoPsSuvV] [membername] [count] archive files...
参数 archive 定义库的名称, files 是库文件中包含的目标文件的清单, 用空格分隔每个文件.
比如创建一个静态库文件的命令如下:
ar -r libapue.a error.o errorlog.o lockreg.o
这样就了 libapue.a 静态库文件, 可以用 t 选项显示包含在库中的文件
创建库文件之后,可以创建这个静态库文件的索引来帮助提高和库连接的其他程序的编译速度.使

用 ranlib 程序创建库的索引,索引存放在库文件内部.
ranlib libapue.a
用 nm 程序显示存档文件的索引,它可以显示目标文件的符号
nm libapue.a | more
如果是显示目标文件的符号:
nm error.o | more
如何使用呢?如下所示:
gcc -o test test.c libapue.a
这样就可以在 test.c 中调用在 libapue.a 中的函数了.
动态库
1.创建共享库
gcc -shared -o libapue.so error.o errorlog.o
这样就创建了共享库!
2.编译共享库
假设共享库位于当前目录(即跟程序文件相同的目录中)
gcc -o test -L. -lapue test.c
这样就编译出了不包含函数代码可执行文件了,但是但你运行时会发现 linux 动态加载器打不到

libapue.so 文件.
可以用 ldd 命令查看可执行文件依赖什么共享库:
ldd test
如何才能让动态加载器发现库文件呢?有两种方法可以解决:
LD_LIBRARY_PATH 环境变量
/etc/ld.so.conf 文件
1.环境变量
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:."
2.修改/etc/ld.so.conf 文件.位于/etc/ld.so.conf
一般应用程序的库文件不与系统库文件放在同一个目录下,一般把应用程序的共享库文件放

在/usr/local/lib 下,新建一个属于自己的目录 apue,然后把刚才 libapue.so

复制过去就行了
同时在/etc/ld.so.conf 中新增一行:
/usr/local/lib/apue
以后在编译程序时加上编译选项:
-L/usr/local/lib/apue -lapue
这样就可以使用这个 libapue.so 共享库了!!