

linux下的动态库和静态库

linux下的动态库和静态库

- 1. 动态库的生成
- 2. 动态库的使用
- 3. 在makefile中生成动态库
- 4. 库的标准目录结构
- 5. 静态库的创建和使用
- 6. 静态库和动态库混用
- 7. c函数和c++函数
- 8. 动态库的手工加载

1. 动态库的生成

《C/C++ 学习指南》Linux篇

7.1 动态库的生成



作者：邵发 官网：<http://afanihao.cn>

背景问题

(1) printf定义在哪里？

我们#include <stdio.h>里面只有printf的声明。 . .

(2) 你编制了一个有价值的算法、或有价值的功能模块，但想给别人使用、但又不想公开代码？如何做到？

作者：邵发 官网：<http://afanihao.cn>



库

在C/C++里，使用库(Library)的技术，可以将编译好的符号提供给第三方使用。

库分为两种：

- (1) 共享库 Shared Library
- (2) 静态库 Static Library

通常共享库也称为动态库

本章介绍Linux下的动态库和静态库的使用方法

作者：邵发 官网：<http://afanihao.cn>



动态库的生成

使用g++命令来生成动态库

编译，生成.o文件 (编译选项 -fPIC)

`g++ -c -fPIC example.cpp -o example.o`

链接，生成目标 .so文件 (链接选项 -shared)

`g++ -shared example.o -o libexample.so`

(PIC: Position Independent Code位置无关代码)

作者：邵发 官网: <http://afanihao.cn>



动态库的生成

linux下动态库的规范命名:

libxxx.so

前缀 lib

后缀 .so

其中 xxx是库的名称

例如， libexample.so



查看符号

使用nm命令查看库中的符号

`nm libexample.so`

U: 代表没有定义



交付物

交付物：

example.h

libexample.so

同时，告之此so文件适用的平台（跨平台使用时可能会有问题）

例如，目标平台：centos 6.6 32bit



作者：邵发 官网：<http://afanihao.cn>

代码示例

```
//example.cpp
#include <stdio.h>
#include "example.h"
int example(int a, int b)
{
    printf("example library: a=%d, b=%d \n", a, b);
    return 0;
}
//example.h
int example(int a, int b);
```

2. 动态库的使用

《C/C++ 学习指南》Linux篇

7.2 动态库的使用



作者：邵发 官网：<http://afanihao.cn>

交付物

example.h

libexample.so

平台： centos6.6 32bit

如何使用这个库呢？



动态库的使用

代码:包含头文件, 调用里面的函数即可

```
#include "example.h"
int main()
{
    example(1,2);
    return 0;
}
```

作者: 邵发 官网: <http://afanihao.cn>



注意-L中的点表示当前目录。

注意两点:

1. 在链接时候, 一定要指定需要链接的动态库, 像这样: `-lexample`, 否则不知道你需要链接什么库, 会报错。但是如果动态库已经放在 (`/lib`, `/usr/lib`, `/usr/local/lib` 或者 `LD_LIBRARY_PATH` 环境变量) 里, 则不用指定动态库的具体位置。
2. 在程序运行的时候, 若动态库已经放在动态库已经放在 (`/lib`, `/usr/lib`, `/usr/local/lib` 或者 `LD_LIBRARY_PATH` 环境变量) 里, 则不用指定动态库的位置。

动态库的使用

编译:

```
g++ -c main.cpp -o main.o
```

链接:

```
g++ main.o -o helloworld -L. -lexample
```

链接选项:

-lexample 使用libexample.so这个库文件

-L. 指定库文件的位置



程序的运行

`./helloworld`

通常会提示无法运行程序：

`libexample.so:: No such file or directory`

问题： `libexample.so`就在当前目录下，为什么操作系统会提示找不到该文件呢？



程序的运行

操作系统默认从**标准位置**寻找相应的库
`/lib /usr/lib /usr/local/lib`

如果没有找到依赖的库文件，则从
`LD_LIBRARY_PATH`环境变量里寻找。。。。

也就是说，库文件**要么放在标准位置，要么放在**
`LD_LIBRARY_PATH`**指定的位置，才能被操作系统**
找到。。。



程序的运行

先使用**export**命令设置环境变量, 然后再运行程序。

```
export LD_LIBRARY_PATH=.
```

```
./helloworld
```

作者: 邵发 官网: <http://afanihao.cn>



3.在**makefile**中生成动态库

《C/C++ 学习 指 南》Linux篇

7.3 生成动态库 - 补充说明



作者：邵发 官网: <http://afanihao.cn>

内容提要

- (1) 在Makefile中创建动态库
- (2) 在动态库中使用别的动态库
- (3) 在动态库中共享class类型



1. 在Makefile中生成动态库

修改标准Makefile Lv1.0

增加编译选项 -fPIC， 链接选项 -shared

① EXE=libexample.so

② g++ -shared \$(CXX_OBJECTS) -o \$(EXE)

③ g++ -c -fPIC -MMD \$< -o \$@

作者：邵发 官网: <http://afanihao.cn>

```
##### 标准Makefile Lv1.1 / 生成动态库 #####
EXE=libexample.so
SUBDIR=src
CXX_SOURCES =$(foreach dir,$(SUBDIR), $(wildcard $(dir)/*.cpp))
CXX_OBJECTS=$(patsubst %.cpp, %.o, $(CXX_SOURCES))
DEP_FILES  =$(patsubst %.o, %.d, $(CXX_OBJECTS))

$(EXE): $(CXX_OBJECTS)
    g++ -shared $(CXX_OBJECTS) -o $(EXE)
%.o: %.cpp
    g++ -c -fPIC -MMD $< -o $@
    -include $(DEP_FILES)
clean:
    rm -rf $(CXX_OBJECTS) $(DEP_FILES) $(EXE)
test:
    echo $(CXX_OBJECTS)
```

2. 动态库中使用别的动态库

在创建动态库时，里面可以调用其他动态库。默认地，可以调用标准C函数库和标准C++ STL库。

如果要调用其他第三方库，可以在链接时添加链接选项。。。。

```
g++ -shared $(CXX_OBJECTS) -o $(EXE) -lxxx
```



3. 动态库中共享class类型

没有什么特别之外，直接按普通方式把class的头文件和cpp文件写出来就行

演示：

作者：邵发 官网：<http://afanihao.cn>



4. 库的标准目录结构

《C/C++ 学习 指 南》Linux篇

7.4 库的标准目录结构



作者：邵发 官网：<http://afanihao.cn>

库的目录结构

Linux下面可能会经常使用各种库，有的是系统自带的库，有的是第三方库。

通常它们的目录结构是：

libxxx/

- lib/

- include/

其中，**bin**下为程序，**lib**下为库文件，**include**为头文件



第三方库的目录结构

系统自带库放在

/usr

-include /

-lib /

演示：把example库放在
/home/mytest/example/目录下



第三方库的目录结构

头文件：

使用编译选项-I参数来指定

```
g++ -c -I /home/mytest/example/include ...
```

库文件：

-L /home/mytest/example -lexample

或 -I /home/mytest/example/libexample.so

(全路径也是可以的)



Makefile中使用库

通常，在Makefile里定义两个变量

CXXFLAGS表示C++的编译选项

LDFLAGS表示链接选项

作者：邵发 官网：<http://afanihao.cn>

```
##### 标准Makefile Lv1.2 / 使用动态库 #####
EXE=helloworld
SUBDIR=src object

#CXXFLAGS:编译选项， LDFLAGS:链接选项
CXXFLAGS += -I/home/mytest/example/include/
LDFLAGS += -L/home/mytest/example/lib -lexample

CXX_SOURCES =$(foreach dir,$(SUBDIR), $(wildcard $(dir)/*.cpp))
CXX_OBJECTS=$(patsubst %.cpp, %.o, $(CXX_SOURCES))
DEP_FILES  =$(patsubst %.o, %.d, $(CXX_OBJECTS))

$(EXE): $(CXX_OBJECTS)
    g++ $(CXX_OBJECTS) -o $(EXE) $(LDFLAGS)
%.o: %.cpp
    g++ -c $(CXXFLAGS) -MMD $< -o $@
    -include $(DEP_FILES)
clean:
    rm -rf $(CXX_OBJECTS) $(DEP_FILES) $(EXE)
test:
    echo $(CXX_OBJECTS)
```

INCLUDE路径

比较：

```
#include <example.h>
#include "example.h"
```

<example.h> : 仅从INCLUDE路径里查找此头文件

"example.h" : 先从当前目录下查找，如果不存在，再从INCLUDE路径里查找。

用-I选项来指定自定义的INCLUDE路径

标准INCLUDE路径： /usr/include /usr/local/include

作者：邵发 官网：<http://afanihao.cn>



5. 静态库的创建和使用

《C/C++ 学习 指 南》Linux篇

7.5 静态库的创建与使用



作者：邵发 官网：<http://afanihao.cn>

静态库

静态库， static library

标准命名： lib**xxx.a**

第一步： 编译， 得到*.o文件

第二步： 打包

```
ar -rcs libxxx.a file1.o file2.o ... fileN.o
```

(注： ar只是将*.o文件打个包而已， 并非“链接”)



静态库

演示：

```
g++ -c test1.cpp -o test1.o
```

```
g++ -c test2.cpp -o test2.o
```

```
ar -rcs libtest.a test1.o test2.o
```

查看静态库中的符号

```
nm libtest.a
```

作者：邵发 官网: <http://afanihao.cn>



交付物

静态库的交付物：

- 头文件 *.h
- 库文件 libxxx.a

另需注明，此静态库适用的平台
如centos6.6 32bit系统



使用静态库

由于libtest.a本质是就是test1.o和test2.o打个包而已，因此可以像.o文件一样使用

比较：

```
g++ main.cpp test1.o test2.o -o helloworld
```

```
g++ main.cpp libtest.a -o helloworld
```

也就是说，在命令行里直接加上libtest.a的全路径是可以的。



使用静态库

使用-l选项，指定静态库

```
g++ main.cpp -o helloworld -L..../K07_05A -ltest
```

编译器在..../K07_05A下面发现了libtest.a。。。。

(考虑：如果在..../K07_05A同时存在libtest.a和libtest.so，编译器会选哪一个？)



静态库和动态库的区别

使用静态库：

最终的程序(helloworld)里含有函数test1和test2的代码，所以helloworld在运行的时候不依赖libtest.a的存在

使用动态库：

最终的程序里没有相应代码，所以程序在运行的时候会寻找libtest.so

可执行程序的，体积有差别。。。

注：使用nm命令，查看是否含有符号的定义



作者：邵发 官网：<http://afanihao.cn>

6. 静态库和动态库混用

《C/C++ 学习指南》Linux篇

7.6 静态库与动态库混用



作者：邵发 官网：<http://afanihao.cn>

静态库和动态库同时存在

接上节课，如果目录下同时存在libtest.a和libtest.so。。。。

-build/

-libtest.a

-libtest.so

执行命令：

```
g++ main.cpp -o helloworld -Lbuild -ltest
```



静态库和动态库同时存在

查看它到底用的哪个库。。。。

nm helloworld

readelf -d helloworld



静态库和动态库同时存在

如果 libtest.so 和 libtest.a 同时存在。 。 。

则默认优先连接 libtest.so



强制使用静态库

当 libtest.so 和 libtest.a 同时存在，以下两种方式可以强制使用静态库：

① 使用全路径 【推荐】

```
g++ main.cpp build/libtest.a -o helloworld
```

② **-static** : 强制所有的库都使用静态库版本 【行不通】

```
g++ main.cpp -o helloworld -static -Lbuild -ltest
```

缺点：

所有的库(包括 libc , libstdc++)都必须提供静态库版本，少一个都不行。。

注： centos默认安装时不带 libc.a libstdc++.a ...

作者：邵发 官网: <http://afanihao.cn>



小结

注：除非你能提供所有库的静态库版本，否则你无法使用 - static参数

所以，当静态库和动态库混用时，通常就是直接指定静态库的全路径。 . . 。

(静态库就是一堆.o文件打个包而已。 . .)

作者：邵发 官网: <http://afanihao.cn>



7.c函数和c++函数

c++调用c的库需要添加external c.....详情请见课件。

8.动态库的手工加载

也就是动态库在代码中来加载，此部分略过。