

- **【SpringMVC】SpringMVC搭建框架** (<https://itnewbee.org/%e3%80%90springmvc%e3%80%91springmvc%e6%90%ad%e5%bb%ba%e6%a1%86%e6%9e%b6/>)
- **NPM使用方法** (<https://itnewbee.org/npm%e4%bd%bf%e7%94%a8%e6%96%b9%e6%b3%95/>)
- **Go优秀开源项目推荐** (<https://itnewbee.org/go%e4%bc%98%e7%a7%80%e5%bc%80%e6%ba%90%e9%a1%b9%e7%9b%ae%e6%8e%a8%e8%8d%90/>)
- **心态崩了，我怎么知道实际生产环境的 B+ 树索引有多少层？** (<https://itnewbee.org/%e5%bf%83%e6%80%81%e5%b4%a9%e4%ba%86%ef%bc%8c%e6%88%91%e6%80%8e%e4%b9%88%e7%9f%a5%e9%81%93%e5%ae%9e%e9%99%85%e7%94%9f%e4%ba%a7%e7%8e%af%e5%a2%83%e7%9a%84-b-%e6%a0%91%e7%b4%a2%e5%bc%95%e6%9c%89/>)
- **SpringCloud升级之路2020.0.x版**

CMake和静态库顺序

发布于：2021年6月11日 上午1:27

作者：risingsun (<https://itnewbee.org/author/risingsun/>)

类别：default (<https://itnewbee.org/category/default/>)

目录

目录 1 (https://blog.csdn.net/Aquester/article/details/84881574?utm_medium=distribute.pc_relevant.none-task-blog-title-6&spm=1001.2101.3001.4242#_Toc20456)

前言 1 (https://blog.csdn.net/Aquester/article/details/84881574?utm_medium=distribute.pc_relevant.none-task-blog-title-6&spm=1001.2101.3001.4242#_Toc13163)

方法 1 (https://blog.csdn.net/Aquester/article/details/84881574?utm_medium=distribute.pc_relevant.none-task-blog-title-6&spm=1001.2101.3001.4242#_Toc1614)

附1：链接静态库的顺序问题 2 (https://blog.csdn.net/Aquester/article/details/84881574?utm_medium=distribute.pc_relevant.none-task-blog-title-6&spm=1001.2101.3001.4242#_Toc5071)

附2：再议GCC编译时的静态库依赖次顺问题 3 (https://blog.csdn.net/Aquester/article/details/84881574?utm_medium=distribute.pc_relevant.none-task-blog-title-6&spm=1001.2101.3001.4242#_Toc7551)

附3：gcc链接参数-whole-archive的作用 4 (https://blog.csdn.net/Aquester/article/details/84881574?utm_medium=distribute.pc_relevant.none-task-blog-title-6&spm=1001.2101.3001.4242#_Toc23491)

附4：让有些“-l”链接静态库，而另一些链接共享库？ 6 (https://blog.csdn.net/Aquester/article/details/84881574?utm_medium=distribute.pc_relevant.none-task-blog-title-6&spm=1001.2101.3001.4242#_Toc31017)

附5：相关博文 6 (https://blog.csdn.net/Aquester/article/details/84881574?utm_medium=distribute.pc_relevant.none-task-blog-title-6&spm=1001.2101.3001.4242#_Toc6610)

前言

C/C++程序的许多同学被静态库的依赖折腾，因为默认情况下要求被依赖的库放在依赖它的库后面，当一个程序或共享库依赖的静态库较多时，可能会陷入解决链接问题的坑中。如果对静态库不熟悉，需要结构nm等工具来解决顺序问题。

但也可以偷懒，不关心静态库的顺序问题，ld为此提供了start-group和end-group两个选项，让包含在这两者间的静态库顺序可以随意。

方法

以CMake为例，假设程序x依赖三个静态库：libX1.a、libX2.a和libX3.a，而libX2.a又依赖libX1.a，libX3.a依赖libX2.a和libX1.a，正常情况下的CMakeLists.txt格式如下：

```

add_executable(
    x
    x.cpp
)
target_link_libraries(
    x
    libX1.a
    libX2.a
    libX3.a
)

```

上面的写法libX1.a、libX2.a和libX3.a的顺序不能变，只能按上面的先后顺序。如果去掉顺序的烦恼和痛苦，可以采用如下的写法：

```

target_link_libraries(
    x
    -Wl,-start-group
    libX1.a
    libX3.a
    libX2.a
    -Wl,-end-group
)
或
target_link_libraries(
    x
    -Wl,-start-group
    libX3.a
    libX2.a
    libX1.a
    -Wl,-end-group
)

```

都可以，完全不用关心顺序。

前面说了start-group和end-group是ld的选项，是链接选项，不是gcc/g++的编译选项，直接命令行或其它编译方式也可以使用，比如命令行方式：

```
g++ -g -o x x.cpp -Wl,-start-group libX2.a libX1.a libX3.a -Wl,-end-group
```

-21.Spring Cloud
LoadBalancer简介
(//itnewbee.org
/springcloud
%e5%8d%87%e7
%ba%a7%e4%b9
%8b%e8
%b7%af2020-0-
x%e7%89%88-21-
spring-cloud-
loadbalancer
%e7%ae%80%e4
%bb%8b/)

日历

2022年 3月

一 二 三 四 五 六 日

1	2	3	4	5	6	
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			
« 8月 (/itnewbee.org /2021/08/)						

附1：链接静态库的顺序问题

在链接静态库时，如果多个静态库之间存在依赖关系，则有依赖关系的静态库之间存在顺序问题，这个在使用静态库时需要注意，否则会报符号找不到问题。举例，libb.a依赖于liba.a，而可执行文件test只直接依赖于libb.a，则链接选项应当为“-b -a”，而不是“-a -b”，否则会报liba.a中的某些符号找不到。

```
gcc -c a.c
ar cr liba.a a.o
gcc -c b.c
ar cr libb.a b.o
```

虽然libb.a使用到了liba.o中的一些函数，但并不会将它们的定义包含进来，所以在链接test时需要指定这两个库。

另外，在编译libb.a时是不指定liba.a的，因为编译一个静态库不会使用到链接选项，而只需要指定需要依赖的头文件路径即可。

-Wl的使用：

-Wl表示后面的参数传递给链接器，其中l是linker的意思。

链接时指定共享库的搜索路径（类似于设置LD_LIBRARY_PATH）：

-Wl,-rpath=/usr/local/abc:/data/abc

以上也可以分开写：

-Wl,-rpath=/usr/local/abc -Wl,-rpath=/data/abc

部分库链接它的静态库，部分库链接它的共享库：

-Wl,-static -lb -Wl,-call_shared -la -lz

指定链接器：

-Wl,-dynamic-linker /lib/ld-linux.so.2 -e _so_start

指定导出的符号：

-Wl,-export-dynamic,-version-script,exports.lds

exports.lds的格式可以为：

```
{
global:
foo;
};
```

指定共享库的soname：

-Wl,-export-dynamic,-version-script,exports.lds,-soname=libqhttpd.so

-rpath 增加共享库搜索路径

—retain-symbols-file表示不丢弃未定义的符号和需要重定位的符号

—export-dynamic 创建一个动态连接的可执行程序时，把所有的符号加到动态符号表中

附2：再议GCC编译时的静态库依

赖次顺问题

假设有如三个源代码文件：

```
$ cat a.cpp
void a()
{
}

$ cat b.cpp
extern void a();
void b()
{
    a(); // 调用a.cpp中的a()
}

$ cat x.cpp
extern void b();
int main()
{
    b(); // 调用b.cpp中的b()
    return 0;
}
```

对应的Makefile文件：

```

all: x

liba.a: a.o

libb.a: b.o

x: x.o liba.a libb.a # 问题出在这儿

    g++ -g -o $@ $^

a.o: a.cpp

    g++ -g -c $^

b.o: b.cpp

    g++ -g -c $^

x.o: x.cpp

    g++ -g -c $^

clean:

    rm -f a.o b.o x.o x

```

使用上面的Makefile编译，将会遇到如下所示的“undefined reference”问题：

```

g++ -g -c x.cpp
g++ -g -c a.cpp
g++ -g -c b.cpp
g++ -g -o x x.o liba.a libb.a # 改成“g++ -g -o x x.o libb.a liba.a”即可解决
libb.a(b.o): In function `b()':
/data/jayyi/gongyi/activities/phonebook/b.cpp:2: undefined reference to `a()'
collect2: ld returned 1 exit status
make: *** [x] Error 1

```

这个问题的原因是b.cpp依赖a.cpp，gcc要求（实际是ld要求）libb.a须放在liba.a前面，即需要改成：g++ -g -o x x.o libb.a liba.a，也就是被依赖的库需要放在后头。

这是最常规的解决办法，除此之外，只需要加入–start-group和–end-group两个链接参数，即可保持被依赖的库放在前头，也就是改成如下即可：g++ -g -o \$@ -Wl,–start-group \$^ -Wl,–end-group。

这里的“-Wl,”表示后面跟着的参数是传递给链接器ld的，gcc不关心具体是啥。“–start-group”表示范围的开始；“–end-group”表示范围的结束，是可选的。位于“–end-group”之后的仍然要求被依赖的库放在后头。

附3：gcc链接参数–whole-archive的作用

// a.h

```
extern void foo();
```

// a.cpp

```
#include <stdio.h>
```

```
void foo()
```

```
{
```

```
    printf("foo\n");
```

```
}
```

// x.cpp

```
#include "a.h"
```

```
int main()
```

```
{
```

```
    foo();
```

```
    return 0;
```

```
}
```

// Makefile

```
all: x

x: x.cpp liba.so
g++ -g -o $@ $^

liba.so: liba.a
g++ -g -fPIC -shared -o $@ $^
#g++ -g -fPIC -shared -o $@ -Wl,-whole-archive $^ -Wl,-no-whole-archive

liba.a: a.o
ar cru $@ $^

a.o: a.cpp
g++ -g -c $^

clean:
rm -f x a.o liba.a liba.so
```

\$ make

```
g++ -g -c a.cpp
ar cru liba.a a.o
g++ -g -fPIC -shared -o liba.so liba.a
#g++ -g -fPIC -shared -o liba.so -Wl,-whole-archive liba.a -Wl,-no-whole-archive
g++ -g -o x x.cpp liba.so
/tmp/cc6UYIAF.o: In function `main':
/data/ld/x.cpp:5: undefined reference to `foo()'
collect2: ld returned 1 exit status
make: *** [x] Error 1
```

默认情况下，对于未使用到的符号（函数是一种符号），链接器不会将它们链接进共享库和可执行程序。

这个时候，可以启用链接参数“-whole-archive”来告诉链接器，将后面库中所有符号都链接进来，参数“-no-whole-archive”则是重置，以避免后面库的所有符号被链接进来。

// Makefile

```
all: x

x: x.cpp liba.so

g++ -g -o $@ $^

liba.so: liba.a

g++ -g -fPIC -shared -o $@ -Wl,-whole-archive $^ -Wl,-no-whole-archive

liba.a: a.o

ar cru $@ $^

a.o: a.cpp

g++ -g -c $^

clean:

rm -f x a.o liba.a liba.so
```

附4：如何让有些“-l”链接静态库，而另一些链接共享库？

用“-Wl,-Bstatic”指定链接静态库，使用“-Wl,-Bdynamic”指定链接共享库，使用示例：

```
-Wl,-Bstatic -lmysqlclient_r -lssl -lcrypto -Wl,-Bdynamic -lrt -Wl,-Bdynamic -pthread -Wl,-Bstatic -lgtest
```

“-Wl”表示是传递给链接器ld的参数，而不是编译器gcc/g++的参数。

附5：相关博文

1) 链接静态库的顺序问题

<https://blog.csdn.net/Aquester/article/details/7780640> (<https://blog.csdn.net/Aquester/article/details/7780640>)

2) 再议GCC编译时的静态库依赖顺序问题

<https://blog.csdn.net/Aquester/article/details/48547685> (<https://blog.csdn.net/Aquester/article/details/48547685>)

3) 如何让有些“-l”链接静态库，而另一些链接共享库？

<http://blog.chinaunix.net/uid-20682147-id-5096676.html> (<http://blog.chinaunix.net/uid-20682147-id-5096676.html>)

4) 小心两个共享库共用同一个静态库

<http://blog.chinaunix.net/uid-20682147-id-3760647.html> (<http://blog.chinaunix.net/uid-20682147-id-3760647.html>)