缺失的一课——编译

上海交通大学超算队 XFlops

2023年11月22日

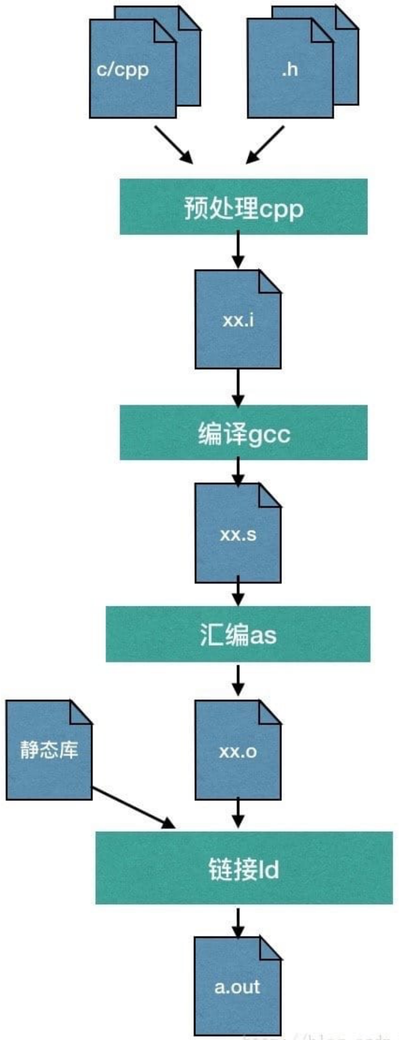
编译在做什么

预处理：处理宏定义，头文件内容等

编译：将源文件转换为汇编指令

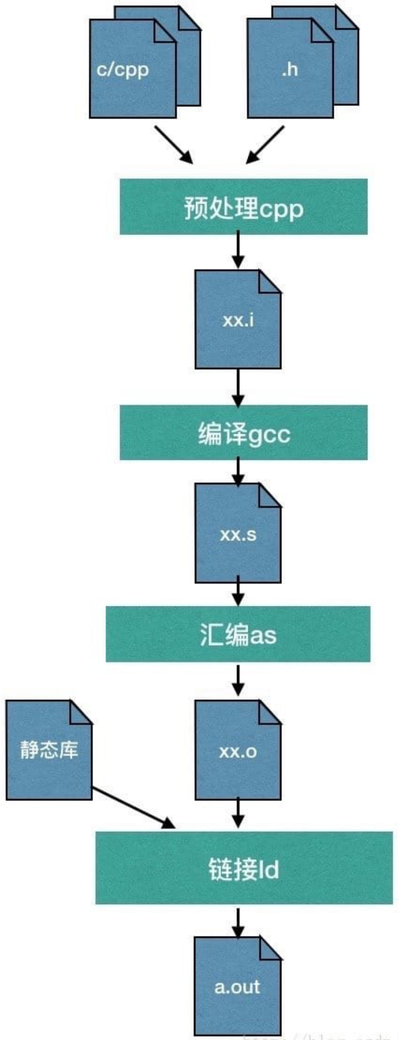
汇编：汇编指令到机器码

链接：将库文件，.o文件一起生成一个可执行文件



编译时需要注意的东西

- 预处理时的头文件
- 链接时的库文件
- 自己写的代码别出问题



编译器如何确定头文件的位置?

```
1   #include <iostream>
2   #include "b.h"
3   #include "c.h"
4   using namespace std;
5   int main() {
6       b();
7       c();
8       return 0;
9   }
```

- Iostream这类库一般放在编译器的默认搜索路径下，编译器可以自行找到。
- 但是b.h，c.h并不在默认路径下。

```
[asc@node016 Example]$ g++ a.cc
a.cc:2:10: 致命错误：b.h：没有那个文件或目录
 #include "b.h"
          ^~~~~
编译中断。
```

编译器如何确定头文件的位置?

w05example > Example > inc > C b.h >

```h
#pragma once
#include <iostream>

void b();
```

w05example > Example > src > b.cc > ...

```cpp
#include "b.h"

void b(){
    std::cout << "func b" << std::endl;
}

```

- 编译器提供了–I选项来设置头文件的搜索路径

```
[asc@node016 Example]$ g++ a.cc src/b.cc -I ./inc
/tmp/ccne4oJc.o: 在函数‘main’中：
a.cc:(.text+0xa): 对‘c()’未定义的引用
collect2: 错误： ld 返回 1
```

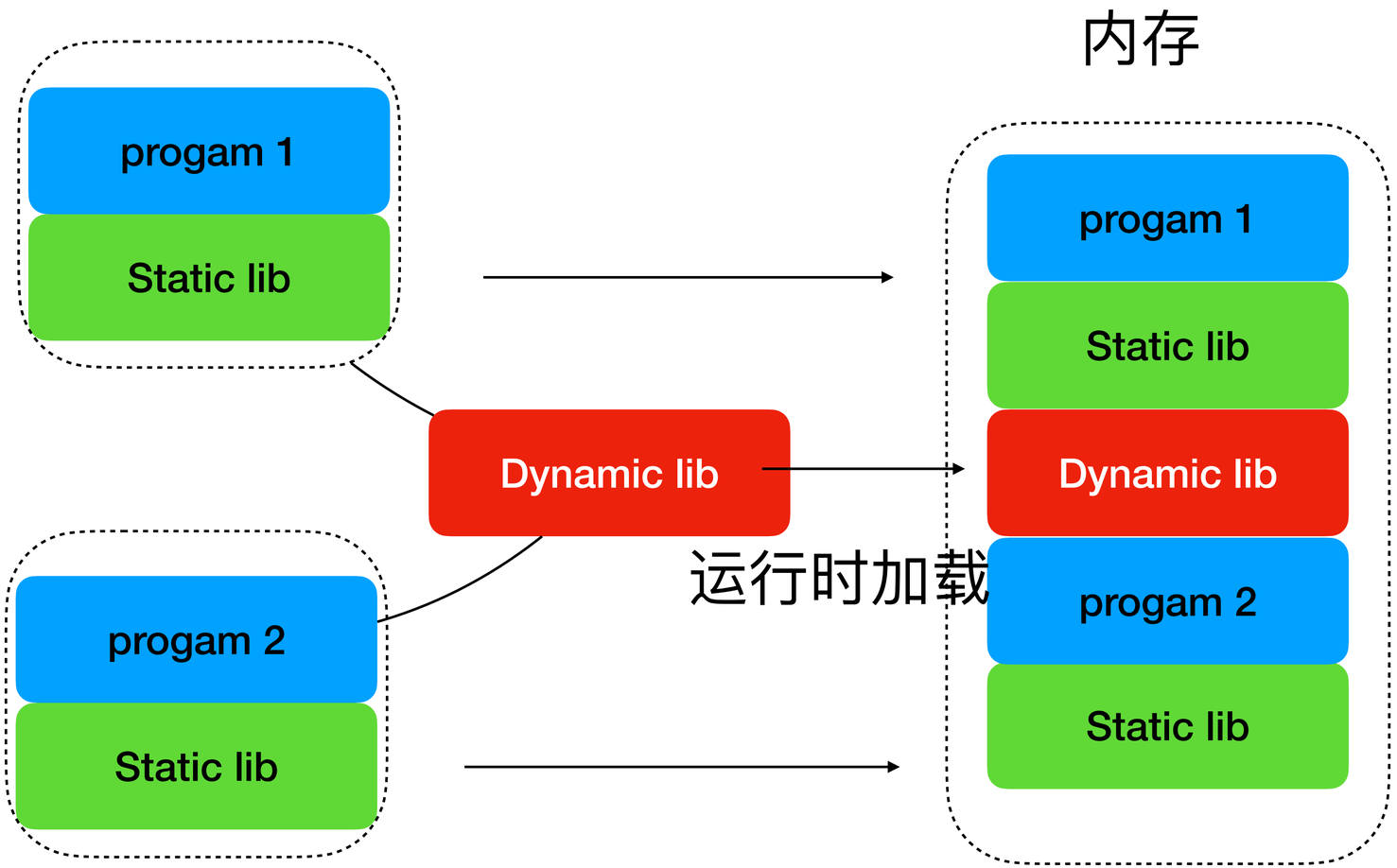
编译器如何确定库文件的位置?

```
[asc@node016 lib]$ ls
libtest.a  libtest.so
[asc@node016 lib]$ pwd
/dssg/home/acct-hpc/asc/shli/2023-tutorial/w05example/Example/lib
[asc@node016 lib]$
```

- 与头文件类似，我们需要提供库的路径给编译器，此时我们需要-L libpath -llibname，来告诉编译器库的路径以及名称。

```
[asc@node016 Example]$ g++ a.cc src/b.cc -o a_dynamic -I ./inc -L ./lib -ltest
[asc@node016 Example]$
```

动态库与静态库的区别



动态库是“共享”的

静态库则是每个程序一份

动态库以及静态库的编译方法

```
#dynamic lib
g++ src/c.cc -fPIC -shared -o lib/libtest.so -I ./inc
#static lib
g++ -c src/c.cc -o libtest.o -I ./inc
ar crf lib/libtest.a libtest.o
```

- -fPIC -shared 用于生成位置无关代码
- ar用于将.o文件转换成静态库.a

更便捷的编译方法——Make

我不想敲命令啦！

```
CC = g++
DIRPATH=$(CURDIR)
CCFLAGS= -O2
SRC=a.cc src/b.cc

LIBSRC=src/c.cc

.PHONY:dynamic static clean

dynamic: $(SRC) libtest.so
    $(CC) $(SRC) -o a_dynamic $(CCFLAGS) -I ./inc -L ./lib -ltest
static: $(SRC) libtest.a
    $(CC) $(SRC) -o a_static $(CCFLAGS) -I ./inc -L ./lib -l:libtest.a
libtest.so: $(LIBSRC)
    $(CC) $(LIBSRC) -fPIC -shared -o $(DIRPATH)/lib/libtest.so $(CCFLAGS) -I ./inc
libtest.a: $(LIBSRC)
    $(CC) -c $(LIBSRC) -o $(DIRPATH)/lib/libtest.o $(CCFLAGS) -I ./inc
    ar crf $(DIRPATH)/lib/libtest.a $(DIRPATH)/lib/libtest.o
clean:
    rm $(DIRPATH)/lib/libtest.* a
```

- 利用Target：conditions来确定源代码之间的依赖关系
- 告知编译器相关头文件，库文件的位置。
- 利用时间戳判断是否需要执行编译
- 使用方法 make [target] [options]

Makefile的Make——Cmake

我不想写Makefile啦！

```
cmake_minimum_required(VERSION 3.10)

project(TestCMake)

#set relative paths.
set(LIB ${CMAKE_SOURCE_DIR}/lib)
set(INC ${CMAKE_SOURCE_DIR}/inc)
set(SRC a.cc ${CMAKE_SOURCE_DIR}/src/b.cc)
set(LIBRARY_OUTPUT_DIRECTORY $(LIB))

add_executable(a ${SRC})
target_include_directories(a PUBLIC ${INC})

add_library(test SHARED ${CMAKE_SOURCE_DIR}/src/c.cc)
target_include_directories(test PUBLIC ${INC})

target_link_libraries(a test)
```

- 比makefile的抽象程度更高，通过给定目标文件名，源文件以及依赖来生成构建脚本（例如Makefile）
- 注意！cmake是用来生成脚本的，之后需要运行cmake —build或者对应的构建软件（例如make）来进行实际的编译。

感谢聆听