库

库是写好的、成熟的，可以复用的代码。现实中每个程序都要依赖很多基础的底层库，不可能每个人的代码都从零开始，因此库的存在意义非同寻常。

本质上来说库是一种可执行代码的二进制形式，可以被操作系统载入内存执行。库有两种：**静态库（.a、.lib）**和**动态库（.so、.dll）**。

静态库

之所以称为[静态库]，是因为在链接阶段，会将**汇编生成的目标文件.o与引用到的库一起链接打包到可执行文件中**。因此对应的链接方式称为静态链接。

> 试想一下，静态库与汇编生成的目标文件一起链接为可执行文件，那么静态库必定跟.o文件格式相似。

其实一个静态库可以简单看成是一组目标文件（.o/.obj文件）的集合，即很多目标文件经过压缩打包后形成的一个文件。静态库特点总结：

- 静态库对函数库的链接是放在编译时期完成的。
- 程序在运行时与函数库再无瓜葛，移植方便

- 浪费空间和资源，因为所有相关的目标文件与牵涉到的函数库被链接合成一个可执行文件

linux下使用ar工具、Windows下vs使用lib.exe，将目标文件压缩到一起，并且对其进行编号和索引，以便于查找和检索。一般创建静态库的步骤如图所示：

```
#pragma once
class StaticMath
{
public:
    StaticMath(void);
    ~StaticMath(void);
    static double add(double a, double b);//加法
    static double sub(double a, double b);//减法
    static double mul(double a, double b);//乘法
    static double div(double a, double b);//除法
    void print();
};
```

Linux下创建与使用静态库

Linux静态库命名规则

> Linux静态库命名规范，必须是”lib[your_library_name].a”：lib为前缀，中间是静态库名，扩展名为.a。

1. 创建静态库（.a）

Linux创建静态库过程如下

将代码文件编译成目标文件.o（StaticMath.o）

```
g++ -c StaticMath.cpp
```

注意带参数-c，否则直接编译为可执行文件。通过ar工具将目标文件打包成.a静态库文件

2. 生成静态库 libstaticmath.a

makefile文件CMake等管理工具能够生成静态库，输入多个命令太麻烦了

使用静态库

Linux下使用静态库，只需要在编译的时候，指定静态库的搜索路径（-L选项）、指定静态库名（不需要lib前缀和.a后缀，-l选项）

```
g++ TestStaticLibrary.cpp -L../StaticLibrary -lstaticmath
```

- -L:表示要连接的库所在目录
- -l:指定链接时需要的动态库，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.a或.so来确定库的名称

动态库

1、为什么还需要动态库

空间浪费是静态库的一个问题

静态库在内存中存在多份拷贝导致空间浪费

静态库对程序的更新、部署和发布页会带来麻烦

如果静态库liba,lib更新了，所以使用它的应用程序都需要重新编译、发布给用户

> 动态库在程序编译时并不会被连接到目标代码中，而是在程序运行是才被载入。不同的应用程序如果调用相同的库，那么在内存里只需要有一份该共享库的实例，规避了空间浪费问题。动态库在程序运行是才被载入，也解决

> 了静态库对程序的更新、部署和发布页会带来麻烦。用户只需要更新动态库即可，增量更新。

动态库特点总结

- 动态库把对一些库函数的链接载入推迟到程序运行的时期
- 可以实现进程之间的资源共享（因此动态库也被称为共享库）
- 将一些程序升级变得简单
- 甚至可以真正做到链接载入完全由程序员在代码中控制（显示调用）

2、Linux下创建与使用动态库

linux动态库的命名规则为： 动态链接库的名字形式为libxxx.so，前缀是lib，后缀名为“.so”

- 针对于实际库文件，每个共享库都有个特殊的名字“soname”。在程序启动后，程序通过这个名字来告诉动态加载器该载入哪个共享库
- 在文件系统中，soname仅是一个链接到实际动态库的链接。对于动态库而言，每个库实际上都有另一个名字给编译器来用

3、创建动态库（.so）

编写四则运算动态库代码

```
#pragma once
class DynamicMath
{
public:
DynamicMath(void);
~DynamicMath(void);
static double add(double a, double b);
static double sub(double a, double b);
static double mul(double a, double b);
static double div(double a, double b);
void print();
};
```

首先，生成目标文件，此时要加编译器选项-fpic

```
g++ -fPIC -c DynamicMath.cpp  #生成了目标文件 .o
```

-fPIC 创建与地址无关的编译程序（pic，position independent code），是为了能够在多个应用程序间共享

然后，生成动态库，此时要加链接器选项-shared

```
g++ -shared -o libdynmath.so DynamicMath.o
```

-shared指定生成动态链接库

也可以合并为一个命令

```
g++ -fPIC -shared -o libdynmath.so
DynamicMath.cpp
```

4、使用动态库

```
#include "../DynamicLibrary/DynamicMath.h"
#include <iostream>
using namespace std;
int main(int argc, char* argv[])
{
double a = 10;
double b = 2;
cout << "a + b = " << DynamicMath::add(a, b) << endl;
cout << "a - b = " << DynamicMath::sub(a, b) << endl;
cout << "a * b = " << DynamicMath::mul(a, b) << endl;
cout << "a / b = " << DynamicMath::div(a, b) << endl;
DynamicMath dyn;
dyn.print();
return 0;
}
```

引用动态库编译成可执行文件（跟静态库方式一样）

```
g++ TestDynamicLibrary.cpp -L../DynamicLibrary -ldynmath
```

在执行的时候是如何定位共享库文件的呢?

1. 当系统加载可执行代码时候，能够知道其所依赖的库的名字，但是还需要知道绝对路径。此时就需要系统动态载入器(dynamic linker/loader)
2. 对于elf格式的可执行程序，是由ld-linux.so*来完成的，它先后搜索elf文件的 DT_RPATH段——环境变量 LD_LIBRARY_PATH—/etc/ld.so.cache文件列表—/lib/,/usr/lib 目录找到库文件后将其载入内存

如何让系统能够找到它

- 如果安装在/lib或者/usr/lib下，那么ld默认能够找到，无需其他操作
- 如果安装在其他目录，需要将其添加到/etc/ld.so.cache文件中
    - 如果安装在其他目录，需要将其添加到/etc/ld.so.cache文件中
    - 如果安装在其他目录，需要将其添加到/etc/ld.so.cache文件中