编译、链接、静态库、动态库

目录

编译、链接、静态库、动态库
- 目录
- 前言
- 从.cpp到.exe —— C/C++程序的构建过程
  - 编译工具
  - 1 预处理 ⭐
  - 2 编译 ⭐
  - 3 汇编 ⭐
  - 4 链接 ⭐
  - 静态库和动态库 ⭐
    - 静态库
    - 动态库
- 写在最后

前言

在程设课上，我们运行一个C++程序的步骤通常是这样的：打开Visual Studio 2008，在文件中写好程序，然后点击“开始调试”或者“开始执行（不调试）”，一个黑色的方框就会弹出来。

实际上，从C++源代码文件到可执行文件的过程是十分复杂的，Visual Studio等现代化的IDE（Integrated Development Environment，集成开发环境）掩盖了程序构建的复杂流程。本节我们就以linux平台上的C++程序为例，简略介绍C++工程中的一些概念。

important!! 为了获得更好的实验体验，建议大家使用linux操作系统（虚拟机或WSL）来运行本节的程序。

从.cpp到.exe —— C/C++程序的构建过程

C/C++程序生成一个可执行文件的过程可以分为4个步骤：**预处理（Preprocessing）**、**编译（Compiling）**、**汇编（Assembly）**和**链接（Linking）**。

- 预处理：在编译器处理程序之前完成头文件的包含，宏扩展，条件编译，行控制等操作。
- 编译：通过词法分析和语法分析，在确认所有的指令都符合语法规则之后，将源文件代码翻译成等价的汇编代码。
- 汇编：将汇编语言代码翻译成目标机器指令，生成目标文件。
- 链接：将有关联的目标文件（以及库）相组合为一个可执行文件。

接下来，我们将通过演示实例介绍每一步发生的故事。

编译工具

针对不同的应用场景和平台，各大厂家设计了不同的C++编译工具。

- MSVC（Microsoft Visual C++）：MSVC是微软公司开发的C++开发工具，我们程设课上使用的Visual Studio就内置了MSVC。
- GCC（GNU Compiler Collection）：GCC是由GNU（GNU's Not Unix）开发的一套编译工具，支持C、C++、Fortran、Go等一系列语言。本教程中我们使用的编译工具就是GCC。

  GCC提供给用户的前端程序为 `gcc`（针对C）和 `g++`（针对C++）。它们的区别详见gcc vs g++。

  在linux(Ubuntu)平台上，可以使用以下指令安装上述工具：

  ```
  $ sudo apt-get install gcc g++
  ```
- 此外还有Clang、NVCC等编译工具。不同的编译工具对C++的支持不尽然相同，此处不再赘述。

1 预处理⭐

C++程序在预处理阶段会执行以下操作：宏的替换、头文件的插入、删除条件编译中不满足条件的部分。

```
$ g++ -E invsqrt.cpp -o invsqrt.i
```

2 编译⭐

C++程序在编译阶段会将C++文件转换为**汇编文件**。

```
# from .i file
$ g++ -S invsqrt.i -o invsqrt.s
# from .cpp file
$ g++ -S invsqrt.cpp -o invsqrt.s
```

3 汇编⭐

汇编语言文件经过汇编，生成**目标文件**.o文件（二进制文件，机器码），每一个源文件都对应一个目标文件。

```
# from .s file
$ g++ -c invsqrt.s -o invsqrt.o
# from .cpp file
$ g++ -c invsqrt.cpp -o invsqrt.o
$ g++ -c main.cpp -o main.o
```

> 生成的 `invsqrt.o` 和 `main.o` 文件不能直接打开，你可以使用 `readelf -a <object file>` 阅读其信息。

4 链接⭐

将每个源文件对应的目标.o文件链接起来，就生成一个**可执行程序文件**。

```
$ g++ invsqrt.o main.o -o main.exe
```

当然，如果想要使用.cpp文件一步到位生成可执行文件，可以使用以下指令：

```
$ g++ invsqrt.cpp main.cpp -o main.exe
```

> 实际上在linux系统上，可执行文件一般是没有后缀名的。此处为了方便说明添加了 `.exe` 文件。

> **语法总结**
>
> `g++` 和 `gcc` 工具中使用的一些命令行参数：
>
> - `-E` 只进行预处理
> - `-S` 只进行编译
> - `-c` 只生成目标文件
> - `-o <file>` 指定输出文件的名称。我们约定： `.i` 为预处理后的文件， `.s` 为汇编文件， `.o` 为目标文件。

静态库和动态库⭐

出于便于复用、封装细节或防止源码泄露等原因，在实际应用过程中，我们需要把C++源码封装为库(library)。

根据其行为不同，可以将库分为静态库(static library)和动态库(shared library)。

静态库

静态库的代码在编译的过程中，会被直接载入到可执行文件中。这样做的好处是：可执行文件在执行时，不再需要静态库本身。但缺点也显而易见：生成的可执行文件的体积会比较大。

linux平台下静态库的后缀通常为 `.a` ，命名方式通常为 `libxxx.a` ;windows平台下静态库的后缀通常为 `.lib` 。

在linux平台上生成静态库，并使用动态库链接形成可执行文件的方法为：

```
# generate static lib
$ ar crv libinvsqrt.a invsqrt.o
# link to generate the executable file
$ g++ -static main.cpp -L . -linvsqrt -o main_shared.exe
```

动态库

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。这就带来了一个明显的好处：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，减小了各个模块之间的耦合程度，也减小了可执行文件的体积。然而，这也要求用户的电脑上需要同时拥有可执行文件和动态库，也有可能因为版本不匹配等问题发生DLL Hell等问题。

linux平台下静态库的后缀通常为 `.so` ，命名方式通常为 `libxxx.so` ;windows平台下静态库的后缀通常为 `.dll` 。

在linux平台上生成动态库，并使用动态库链接形成可执行文件的方法为：

```
# generate shared lib
$ g++ invsqrt.cpp -I ./ -fPIC -shared -o libinvsqrt.so
# move the shared library to system
$ sudo mv libinvsqrt.so /usr/local/lib
# refresh
$ sudo ldconfig
# link to generate the executable file
$ g++ main.cpp -L . -linvsqrt -o main_shared.exe
```

写在最后

由于时间所限，还有很多有趣的内容我们没有涉及：

- gcc/g++有着丰富的命令行参数设置，比如程序优化、C/C++语言标准设置等。
- 在本节中，我们只介绍了如何在linux平台上生成和使用静态库、动态库。实际上，利用Visual Studio也可以便捷地在windows平台上生成静态库、动态库。
- ...

略过上述内容不会对我们的教学产生太大影响。感兴趣的同学可以参考以下文档：

- GCC官网
- learn cpp 一份新手友好的C++入门文档。
- 演练：使用Visual Studio创建并使用静态库
- 演练：使用Visual Studio创建并使用动态库