C++ 教程 & Pytorch Lightning

菠萝芝士焗饭

2023 年 4 月 10 日

目录

1 C++ 如何工作

首先看一段非常简单的程序：

```
#include <iostream>

// 如果找不到这个函数在其他文件的位置，就会出现链接报错
void Log(const char* message);
// 任何一个C++的程序都需要main函数，是程序的入口
// 注意到mian函数返回了一个整数，当你什么也不写的时候默认返回0
// 但这只对main函数适用，对于其他函数是一定要有对应的返回值
int main()
{
    // <<是一个重载运算符
    // 将hello world推送到cout，然后打印到控制台上，endl是输出回车
    std::cout << "Hello, World!" << std::endl;
    // get函数是等待我们输入，也是一个暂停函数
    std::cin.get();
}
```

第一句 include 是预处理，因为他在编译之前就处理完了，所有的预处理语句都会用 # 开始。include 表示把后面的文件复制到当前 cpp 文件上。这个文件也叫头文件，因为他写在开头。

main 是这个整个程序的入口，默认返回 0，如果你不写她就默认返回 0，但对于其他函数是需要返回和类型相同的。«表示一个重载运算符，把 hello world 写到输出流，在控制台

显示，endl 表示回车。get 表示等待用户输入，可以作为暂停的一种方式。
cpp 文件的处理可以分成几步：

- 预处理阶段，也就是执行头文件，在这里的主要操作就是把这些头文件复制到当前文件中
- 编译阶段，这个阶段会把 cpp 代码编译成机器执行的代码，这些代码会变成.o 的文件形式
- 然后链接，将这些 o 文件链接成一个 exe 可执行文件

通常，我们不会把一个函数写在一个代码里面，比如上面，把 log 函数单独拿了出来。需要在 test.cpp 文件上面单独声明这个函数，程序会自己去找这个函数在哪里。我把 Log 函数写在了 Log.cpp 文件上，那么在第二步编译阶段的时候，就会出现两个 o 文件，这个时候链接程序就会找到这个 Log 函数放进 test 程序里面。
首先编译：g++ -c test.cpp log.cpp ，会生成.o 文件，然后链接，g++ test.o log.o -o main，就是 test.cpp 中找到这个 log 函数的过程包含在其中，最后生成 main.exe，执行即可。

1.1 编译

这个过程的具体表现就是生成.o 文件，主要做了两件事：

- 预处理代码
- 创建抽象语法树

首先是预处理的代码，预处理这部分就是复制，把头文件复制到当前文件中去。看一个例子

```
#include <iostream>

int Log(const char* message)
{
   std::cout << "Logging ..." << message << std::endl;
```

少了两个关键部分，return 和}，我们可以在头文件补上。新建一个 my.h 文件，里面补上缺失的内容：

```
   return 1;
}
```

在代码后面补上

```
#include "my.h"
```

就可以运行了。然后就是生成 obj 文件，这个文件可以用 visual studio 可视化，但是我用的命令行直接 g++ 编译看不到，里面都是一些汇编代码，也就是说编译器会把这些代码变成汇编指令，让机器执行。

1.2 链接

链接的主要的目的是寻找每个符号和函数是在哪里，链接本身还是比较好理解的，这一节主要解决的是一些链接的问题。

1.2.1 函数重载

按照视频的内容，返回值不同，那么函数就不同，代码应该是能识别出来的。但实际上，在我的测试代码里面，仅仅只有返回值不同的函数是无法进行函数重载的，这个标准应该是新的标准，我以前学的时候返回值不同还是可以的。所以**仅仅返回值不同的函数是不能重载的**。声明 void Log(const char* message)，如果你的引用代码里面是有 int Log(const char* message)，她是会链接到的。

1.2.2 Ambiguity

这个问题是出现在存在多个可以链接到的函数，程序混乱了导致的。比如在 log.cpp 我们存在两个除了返回值其他都一样的函数，那么模型是不知道链接哪一个函数的。当然，这个错误只会出现在我们链接的时候，编译的时候是不会出现的。
以下有几种特殊的 ambiguity 情形。首先我在头文件 my.h 定义了一个 test 函数，然后在 test.cpp 和 log.cpp 引用这个函数，需要引用我们就需要声明，我们直接用

```
#include "my.h"
```

加在这两个 cpp 文件里面，这个时候编译就会出现

> multiple definition of ‘test()’

的问题，**这个问题就要回到预处理语句的定义上，include 是把预处理头文件复制到当前文件中，所以相当于在两个 cpp 文件定义了两个相同的函数，这样就出现了定义的问题。**
如何解决？有两个方法：

- 把 test 函数隐藏起来，对其他文件不可见。使用 static 关键字。static int test() 就没问题了，虽然他们会被复制到各自的 cpp 文件中，但是 static 关键字使得这些函数是只能自己看到，对其他文件来说不存在。
  - 另一种方法是用 Inline 关键字，这样程序并不会把 test 函数当成是一个函数，她会把引用这个函数直接替换成这个函数的内容。比如 test.cpp 引用了 test()，而 test()return 1;，那么她会把 test.cpp 的 test() 换成 return 1; 这样没有了函数调用自然就没有了链接。
- 另一种方法是通过头文件引入。把方法写到另一个 cpp 文件中，然后把这个方法引入到.h 头文件。其实就是用函数声明，只不过把这个函数声明分到了另一个文件里面。

所以.h 的文件里面很多都是用 static 关键字，就是为了反正多重定义。

2 C++ 变量

在程序运行的过程中，我们需要为存储在变量里面的数据命名，这个写数据就存储在变量里面。这一节比较简单

- 变量之间的区别就是大小，他们的大小不同导致他们在计算机里面的区别
- 程序中默认了一些约定俗成的规则，比如 char 一般用于存储字符。但实际上字符的存储也是数字，和 ascii 码表对应

常见的几类 char, short, int, long, longlong, bool 等。他们占用的内存都是字节，因为内存只能以字节为单位查询。虽然 bool 只需要一个比特就能完成查询，但是模型查询的最小单位是字节，所以只能分配字节给 bool 变量。

3 函数

函数是用来执行特定任务的代码块。使用函数的目的是希望代码的可读性更高，希望代码块的重用性更高。

- 每次调用函数的时候，编译器会生成 call 指令，也就是说编译器会为每一个函数创建一个 stack 结构，把这些变量都推送到 stack 中。
- 所以在运行函数的时候程序会不断在内存中跳跃执行

所以不要创建过多的函数。另外，如果函数在定义的时候明确要求返回值，那么这个函数就一定要返回。

4 头文件

主要了解几个问题

- 为什么需要头文件
- 头文件是干什么的
- 什么情况下需要

头文件通常是用来声明某些函数，这样可以在程序中使用。当我们使用一个定义在其他文件的变量的时候，我们需要声明这个函数，而头文件就是用来存储这些函数声明的地方。当然你也可以直接在你需要用到这个函数的地方进行声明，但如果你有 100 个函数要引用，你就要写 100 次，而且如果你需要在其他的 cpp 文件里面也引用这些函数，那么你又要写 100 次。如果使用头文件只需要把这 100 个函数声明写进去，再引用这个头文件就行，只需要写一次。
再头文件开始的时候还加上了一段代码：

```
#pragma once
```

意思就是头文件里面的内容只会被转换成一个编译单元，这是为了防止你再头文件里面定义了某个函数或者类，然后重复引入同一个头文件导致的定义重复。
另一种复制重复定义的方法是

```
#ifndef _LOG_H
#define _LOG_H

#endif
```

ifndef 表示如果后面这个变量没有被定义，那么就编译后面的代码。如果定义了就不编译。
简单来说 **头文件是用于声明代码，而 cpp 文件是实现这些声明。**
最后还有两个补充的点：

- <> 和"" 的区别，<> 表示从系统目前搜索，然后再搜索环境变量列出的目录，**但是她不会查找当前目录，差不多就是只会查找系统目录。**"" 是会查找当前目前目录下的头文件，然后再去搜索系统目录，所以"" 是会搜索几乎所有可能存在头文件的地方。所有 <> 一般用来引入系统文件，“” 一般用来引入自己写的文件。
- 在写 C++ 的时候引入系统都文件 iostream，这个文件后面是没有扩展名的，这是为了和 C 语言区分开。