

10 | 预处理命令（上）：必须掌握的“黑魔法”，让编译器帮你写代码

2020-01-28 胡光

人人都能学会的编程入门课

[进入课程 >](#)



讲述：胡光

时长 14:48 大小 11.87M



你好，我是胡光，欢迎回来。今天是大年初四，春节的气氛依然很浓厚，春节玩得开心吗？但也别忘了咱们的继续学习哦。今天还在看专栏，依旧没有忘记学习的你，我必须赞叹一声：学会编程，非你莫“鼠”！

之前我们学习的编程知识，都是作用在程序运行阶段，也就是说，当我们写完了一段代码以后，只有编译成可执行程序，我们才能在这个可执行程序运行后，看到当初我们所写代码的运行效果。而你有没有想过，存在一些编程技巧，是作用在非运行阶段的呢？这就是我们今天要学习的内容。



今天呢，我们将来讲解整个语言基础篇的最后一部分：预处理命令。那么什么是预处理命令呢？它又为什么被称为程序设计中的“黑魔法”呢？让我们开始今天的学习吧。


任务介绍

这次这个任务呢，我们将分成两节来讲解，这是因为，想要掌握程序设计中的这门“黑魔法”，真的急不来，咱得慢慢来。

本次这个任务呢，和输出有关系：请你实现一个打印“漂亮日志格式”的方法。你可能想用 `printf` 直接打印，别着急，听我详细说完这个打印日志的功能介绍以后，你可能就知道什么叫做“魔法般的方法”了。

首先我们先说“日志”的作用，程序中的“日志”，通常是指在程序运行过程中，输出的一些与程序当前状态或者数据相关的一些信息。这些信息，可以帮助程序开发人员做调试，帮助运营人员做数据分析，帮助管理人员分析日活等等。总而言之，一份合理的日志信息，是非常有价值的信息。而我们今天呢，接触一种最简单的日志形式，就是程序运行过程中的调试信息。


请你实现一个参数形式和 `printf` 函数一样的 `log` 方法，用法如代码所示：

 复制代码

```
1 #include <stdio.h>
2
3 void func(int a) {
4     log("a = %d\n", a);
5 }
6
7 int main() {
8     int a = 123;
9     printf("a = %d\n", a);
10    log("a = %d\n", a);
11    func(a);
12    return 0;
```

你会看到上述代码中，有一个和 `printf` 名字不一样可用法完全一样的方法叫做 `log`，而这个 `log` 的输出结果，和 `printf` 可不一样。

具体如下：

 复制代码

```
1 a = 123
2 [main, 10] a = 123
```

```
3 [func, 4] a = 123
```

你会看到 log 的方法，虽然和 printf 函数的用法一致，可在输出内容中，log 方法的输出明显比 printf 函数的输出要多了一些信息。

首先第 1 行，是 printf 函数的输出，这个就不用我多说了，想必你已经很熟悉了。第 2 行和第 3 行都是 log 方法的输出，一个是主函数中的 log 方法，另外一个是在 func 函数中执行的 log 方法。

你会看到，log 方法的输出中，会输出额外的两个信息：一个是所在的函数名称信息，在主函数中的 log 方法就会输出 main 主函数的名称，在 func 函数中的 log 方法，就会输出 func 函数的名称；除了函数名称信息以外，另一个就是多了一个 log 函数所在代码第几行的信息，第一个执行的 log 在代码的第 10 行，就输出了个 10，第二个 log 执行的时候，在代码的第 4 行，就输出了个 4。

正是因为 log 方法比 printf 函数多了这些信息，才使我们更清晰地知道相关调试信息在源代码逻辑中所在的位置，能够帮助我们更好的去理解，以及分析程序运行过程中的问题。哦，对了，这里再加一个小需求，就是设计完 log 方法以后，请再给这个 log 方法提供一个小开关，开关的作用是能够很方便的打开或者关闭程序中所有 log 的输出信息。

现在你应该清楚了本次的这个任务吧，那么如何完成这样的一个任务呢？跟我来一起开始预处理命令相关的学习吧。

必知必会，查缺补漏

1. 认识预处理命令家族

先来让我们认识一下今天课程的主角：预处理命令家族。在真实世界里面，有很多家族，每个家族都有自己的姓氏，例如：数学圈里面的伯努利家族，伯努利就是这个家族的统一的符号。而预处理命令家族，也有自己的特殊符号，那就是以 # 作为开头的代码。

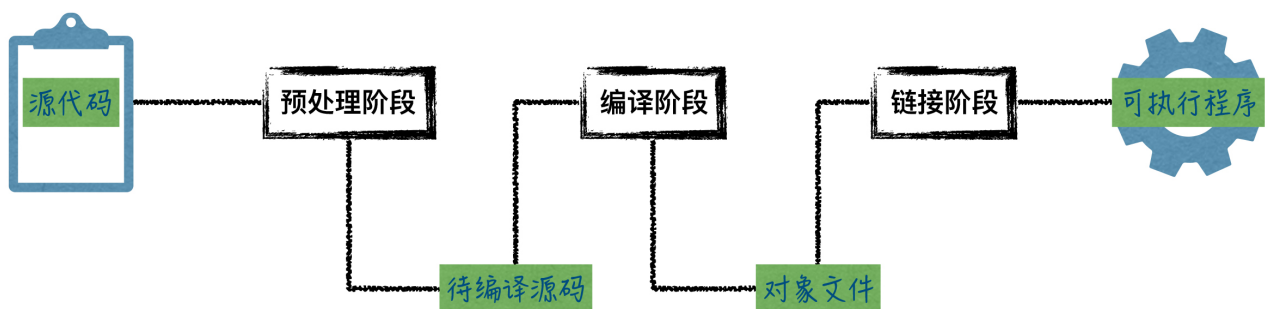
说到这个特征，你能想到什么？你之前其实就见过这个家族的成员，只不过那个时候，我们没有特殊的提出来过。敏锐的你，可能想到了，#include 不就是以 # 作为开头的代码么？

没错，`#include` 就是预处理命令家族中的一员，对于它的认知，你可能觉得，是用来做功能添加的，当我们写了 `#include <stdio.h>` 以后，程序中就有了 `scanf` 函数或者 `printf` 等函数的功能了，这种认识没有错，不过还是不够精准。

为了更精准地认识预处理命令的作用，我们得先来说一下 C 语言程序从源代码到可执行程序的过程，为了让你能够更聚焦地进行学习，我挑了三个重要的环节来展示给你，理解了这三个环节，也就能够理解 C 语言在编译过程中所报出来的 90% 的错误原因。

这三个环节就是：**预处理阶段**，**编译阶段**，**链接阶段**。三个阶段从前到后依次执行，完成整个 C 语言程序的编译过程，上一个阶段的输出就是下一个阶段的输入。说到这里，你可能发现了，原来我们之前所说的编译程序，是这个复杂过程的简称。

为了让你更清楚地了解三个阶段的关系，我给你准备了下面的一张图，帮助你理解：



程序编译流程图

在上图中，有两个概念，你是熟悉的，一个是**源代码**，就是你所编写的代码，另外一个**可执行程序**，就是你的编译器最终产生的，可以在你的环境中运行的那个程序。windows 下面，就是产生的那个后缀名为 `.exe` 的文件。

剩余两个概念，你可能比较陌生，一个是**待编译源码**，另外一个**对象文件**。关于这两个概念，今天我将重点给你介绍的就是**待编译源码**，也就是预处理阶段输出的内容，同时也是编译阶段的输入内容。

关于**对象文件**的相关知识，我会在后面给你留个小作业，不用担心，现阶段，你即使不理解**对象文件**是什么东西，也不会影响你之后的学习。如果你想搞懂什么是**对象文件**，那我建议你，先搞懂“声明”和“定义”的区别，这种学习路线，会更加有效一些。

2. 预处理阶段

下面呢，我们就来说说预处理阶段。首先先来看预处理阶段的输入和输出内容，输入内容是“源代码”就是你写的程序，输出内容是“待编译源码”。之所以叫做“待编译源码”，那是因为这份代码才是我们交给编译器完成后，续编译过程的真正的代码。它是由预处理器处理完“源代码”中的所有预处理命令后，所产生的代码，这份代码的内容跟“源代码”相比，已经算是面目全非了。

咱们下面就拿一个最简单的例子，来说明这一点。刚刚我们说过了，`#include` 是我们所谓的预处理命令家族中的一员，它真正的作用，是在预处理阶段的时候，把其后所指定文件中的内容粘贴到相应的代码处。

例如：`#include <stdio.h>` 这句代码，在预处理阶段，预处理器就会找到 `stdio.h` 这个文件，然后把这个文件中的内容原封不动的粘贴到 `#include <stdio.h>` 代码所在的位置。至于 `stdio.h` 这个文件在哪里，编译器是怎么找到它的，这个问题不是我们今天所讨论的重点，所以你可以先忽略它。这样呢，我们对于预处理命令 `include` 就有了更清晰的认识了。

下面呢，我们就围绕着 `include` 预处理命令设计一个小实验，来说明“源代码”和“待编译源码”的区别。

首先呢，我们准备两个文件，两个文件一定要在同一个目录下，一个文件的名字叫做 `my_header.h`，另外一个叫做 `test_include.c`，两个文件中的内容呢，如下所示：

```
1 //my_header.h 文件内容
2 int a = 123, b = 456;
```


 复制代码

```
1 //test_include.c 文件内容
2 #include <stdio.h>
3 #include "my_header.h"
4 int main() {
5     printf("%d + %d = %d\n", a, b, a + b);
6     return 0;
7 }
```

 复制代码

如果你编译运行 `test_include.c` 这个程序的话，你会发现，程序可以正常通过编译，并且会在屏幕上正确输出一行信息：


```
1 123 + 456 = 579
```

 复制代码

这个过程中，我们就重点来思考一个问题，为什么在 `test_include` 源文件中没有定义 `a`、`b` 变量，而我们在主函数中却可以访问到 `a`、`b` 变量，并且 `a`、`b` 变量所对应的值和我们在 `my_header` 头文件中对 `a`、`b` 变量初始化的值一样？

要解答上面这个问题，就要理解刚刚所说的 `include` 预处理命令的作用。回想一下，刚刚我们介绍 `include` 预处理命令的作用，就是在预处理阶段，把后面指定文件的内容原封不动的粘贴到对应的位置。也就是说，`test_include` 源代码文件经过了预处理阶段以后，所产生的待编译源码已经变成了如下样子，如下所示，其中我删掉了 `stdio.h` 展开以后的内容：

```
1 // 假装这里有 stdio.h 展开以后的内容
2 int a = 123, b = 456; // my_header.h 展开以后的内容
3 int main() {
4     printf("%d + %d = %d\n", a, b, a + b);
5     return 0;
6 }
```

 复制代码

在这份待编译源码中，你可以看到是存在变量 `a` 和 `b` 的相关定义和赋值初始化的。因为待编译源码是一份合法的代码，所以才能通过编译阶段，最终生成具有相应功能的可执行文件。

看完了这个过程以后，我希望你注意到一点，如果要分析最终程序的功能，不是分析“源代码”，而是要分析“待编译源码”，也就是说，是“待编译源码”决定了程序最终功能。

要想搞清楚待编译源码，就必须理解预处理阶段做的事情，也就是各种预处理命令的作用。这些预处理命令，会在编译过程中，帮你改变你的代码，更形象化一点儿，就是仿佛是编译器在帮你修改代码一样。

那么程序最终的功能呢，就是由这份编译器修改过后的代码所决定的，编译器就是预处理命令这个“黑魔法”背后，那股神秘而强大的力量。

思考题

今天呢，没有以往具体的要求，让你写出一个实现什么功能的程序。而是留了一个对你要求更高，更加考验你思考总结能力的问题。

这个课后自学作业，就是请你通过自己查阅资料，搞清楚对象文件的作用，并且用尽可能简短的话语在留言区阐述你的理解。记住：由简到繁，是能力，由繁到简，是境界。

课程小结

最后，我来给你做一下这次的课程总结。今天，只希望你理解以下三点即可：

1. C 语言的程序编译是一套过程，中间你必须搞懂的有：预处理阶段，编译阶段和链接阶段。
2. 程序最终的功能，是由“待编译源码”决定的，而“待编译源码”是由各种各样的预处理命令决定的。
3. 预处理命令之所以被称为“黑魔法”，是因为编译器会根据预处理命令改变你的源代码，这个过程，神秘而具有力量，功能强大。

下篇文章中呢，我将带你具体的认识几个预处理命令家族中的成员，带你真正的体会一下这个“黑魔法”的力量，并且我们会在下一篇文章中，解决掉今天提到的“打印漂亮日志”的任务。

好了，今天就到这里了，我是胡光，我们下期见。

人人都能学会的编程入门课

>>> 每天 10 分钟，轻松学编程

胡光

原百度高级算法研发工程师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 09 | 函数：自己动手实现低配版 scanf 函数

下一篇 11 | 预处理命令（下）：必须掌握的“黑魔法”，让编译器帮你写代码

精选留言 (3)

 写留言



一步

2020-01-29

有没有什么办法可以看到 预处理阶段后的 待编译源代码的内容？

作者回复: 可以的，如果使用Linux环境的话，编译的时候，gcc -E 源文件，用这个命令就可以看到预处理以后的代码。



2



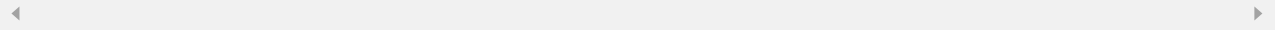
Geek_Andy_Lee00

2020-01-30

对对象文件的理解：待编译源码是高级语言，经过编译器编译之后就变成了等效的的汇编代码，也就是机器可以执行的机器指令。

展开 ∨

作者回复: 如果要是能从『定义』和『声明』的角度来解释的话, 就完美了。



一步

2020-01-29

像文章所说 预处理命令 `#include` 会把 `include` 的内容拷贝到 `#include` 处, 如果所有文件都这样处理, 那么每个文件就会出现好多重复的代码, 想知道C语言预处理阶段是怎么处理这样的问题的?

作者回复: 往后看, 看后面的条件编译。^_^

