> 本文 GitHub https://github.com/rd2coding/Road2Coding 已经收录，里面有我整理的**6大编程方向(岗位)的自学路线+知识点大梳理**、**面试考点**、**我的简历**、**几本硬核pdf笔记**，以及**我的程序员人生**。

小伙伴们，大家好。

关于C语言和C++的学习路线终于梳理完了。当然我也只能从我曾经近3年的通信公司后台开发经历和目之所及的世界，跟大家聊聊这个话题。

这块的东西很多很杂，不过梳理和总结之后，回过头来看，其实学习思路、学习路线应该还是比较清晰的。由于个人认知有限，不足的地方也欢迎大家评论里补充。

先聊几个有趣的问题

这几个问题都是私信里常被问到的，也是我当时学习过程中的一些疑惑。

问： 为啥我学完了C语言或者C++，却还是啥东西也做不出来?

> **答：** 编程语言学完了就能做出东西那也真是天才哇！应该说语言学得就算再精通，它其实也只代表完成了“最小的”那一部分，和实际上手干活之间还是有一个非常大的**鸿沟**，这个鸿沟就表现为下文即将详述的 **编程基础四大件** + **应用实践编程**。

问： 为什么C/C++写出来的东西都是运行于黑乎乎的命令行？这玩意真有用吗?

> **答：** 嘿嘿，谁说黑乎乎的命令行里运行的程序就没有用！咱大名鼎鼎的Linux系统都以命令行跟用户交互呢，而且Linux里面很多强大的工具都是运行于黑乎乎的命令行！

问： 从技术学习和实际运用的角度来看，C/C++和Java到底区别在哪？

> **答：** C/C++，它和Java确实不太一样。C语言和C++，尤其C++，语言粒度细、机制多，性能虽然高，但语言本身的包袱也确实重，我们更愿意称它“造轮子”的语言！也正是因为C语言和C++性能好、粒度细，所以什么都能做。
>
> 而Java本身就是一种服务于互联网软件开发（后端开发+客户端开发）的语言，它有一个明显的“生态圈”的概念，所以应用领域非常清晰。我个人觉得C语言和C++编程比Java还是要难一些，Java毕竟大多是应用层的，C语言和C++则对程序员能力的要求要更高一些。

岗位分析

了解一下岗位，知道以后能做什么，这个也有利于自己树立学习目标。

C语言和C++属于“造轮子”语言，几乎什么都能做。不过一般来说，C语言和C++主要还是做后台（服务端）开发比较多，包括：

- 通信公司后台开发
- 互联网公司后台开发
- 游戏公司后台开发
- ……

当然这个后台开发具体职责又有很多细分，比如：

- 有做数据处理和分析的
- 有做基础协议和通信的
- 有做服务端底层应用优化的
- 甚至还有做后台系统驱动和内核的
- ……

不管怎样，下面即将要介绍的这些学习路线和内容适用于以上所有情况。所以呢，下面就来讲讲具体的学习路线。

语言本身的学习

C语言：

- 除了最最基础的语法：变量、条件、循环、字符串、数组、函数、结构体等之外
- C语言最最最最最重要的那就是：指针、内存管理，以后企业里开发就靠它俩吃饭，这也是检验C语言掌握情况的两大标准

> **参考书籍**：《C Primer Plus》、《C和指针》、《C专家编程》、《C陷阱和缺陷》等

C++：

- C++和C语言确实是不同的语言，但是C++确实是对C语言的延伸，可以理解为在C语言里加入了面向对象的特性。因为只有面向对象特性的加持，代码才能更好的**复用**、**扩展**和**工程化**，这是大型项目的必备要素
- 除了C语言所有的底子之外，还需要学习C++的面向对象（封装、继承与多态）特性、泛型、模板、STL等等

> **参考书籍（有先后顺序）**：《C++ Primer》、《Effective C++》、《C++ 标准程序库》、《STL源码剖析》《深度探索C++对象模型》等

最后一个小建议是：语言部分的学习建议不要拖太久，一定要规划好时间，一鼓作气，高强度给它压下来，否则容易把自己搞泄气。

编程基础四大件

基础四大件包括：**数据结构和算法**、**计算机网络**、**操作系统**、**设计模式**

这跟学什么编程语言、后续从事什么编程方向均无关，只要做编程开发，这四个计算机基础就无法避开。可以这么说，这基础四大件真的比编程语言重要！！！

1、数据结构和算法

可以说这个直接决定了面试的成败！几种基础数据结构类型得烂熟于心，比如：字符串、链表、二叉树、堆、栈、队列、哈希等；基本的几大算法也要了如指掌，比如查找、排序、动态规划、分治等等。建议LeetCode多刷题。

> **参考资料：** 《大话数据结构》、《算法》、《剑指offer》、《LeetCode刷题》等

2、计算机网络

此处的计算机网络指的就是TCP/IP协议栈，可以说它是当下互联网通信的基石，无论如何一定要对TCP/IP的协议栈了如指掌，主要就是学习和掌握原理，包括：**ARP协议、IP协议、ICMP协议、TCP和UDP协议、DNS协议、HTTP协议、HTTPS协议**。

> **参考书籍：** 《TCP/IP详解》等

3、操作系统

该部分重点包括：进程和线程的相关原理（原子性、并发、锁）、内存相关原理（内存分布、内存调度）

> **参考书籍：** 《深入理解计算机系统》等

4、设计模式

倒不需要23种设计模式全部记住，常见的几个如：**单例模式、工厂模式、代理模式、策略模式、模板方法模式**建议熟练于心。

> **参考书籍：** 《大话设计模式》、《设计模式之禅》等

应用实践编程

这部分会涉及到一些工具、编程环境、和具体编程实践，应该说这一部分学完，自己应该能做点东西出来、或者说能看懂一些实际项目的代码。

实践这部分，我也是结合我之前在通信公司的实际工作经历和所听所见，来聊一聊。

1、Linux操作系统的使用

很多人初学C语言、C++（包括我）都是在Windows环境上进行的，而实际企业级开发几乎不可能，所以Linux系统必须要会，我们别无选择。先谈使用。

包括：**常见的Linux操作命令**、**基本的Shell编程**。

> **参考书籍：** 《鸟哥的Linux私房菜》

2、编译/调试工具

首先是跟编译相关的：**编译工具！**

我们知道很多人学C语言、C++都在类似Visual Studio这种集成IDE里进行代码编译，这个其实也用了编译器，只不过是微软自家的MS编译器，而且所有操作均可视化。而企业里开发很少会基于Windows系统，所以Linux平台上的编译器更为重要，最典型的当属`GCC`，甚至有些公司有自己定制过的交叉编译工具，但没关系，只要`GCC`熟悉，其他问题都不大。

其次，大家自学C语言、C++，都借助类似VS这种IDE，点按钮即可对源文件编译。而企业里实际项目的编译动作叫`make`，编译的实际动作和过程都是写在`makefile`文件里，所以makefile的书写规则建议学习！
最后说到**调试**，Linux平台上的`GDB`调试工具要熟练使用，会借助于它进行调试。

> **参考资料：** 这部分没有书可推荐，英文好的同学可以直接看`GNU`官网关于`GCC`和`GDB`的文档，中文材料可以看：《debuging with gdb》（中文版）和陈皓先生的《跟我一起写makefile》

3、Linux环境编程

还是那句话，企业里C语言和C++几乎都是基于Linux平台的，这一部分我们没办法避开

（1）Linux系统编程

众所周知，Windows系统编程里有各种各样的Windows API，同理Linux系统API的使用就更加重要。

（2）多线程编程

此处指的是多线程编程实践相关的东西，一般包括：线程、资源、信号、同步、互斥、锁等等一些具体的编程方法。

（3）网络编程

此处的网络编程主要指的是具体Linux系统上的网络编程`API`和`IO`函数的编程实践。

> **参考书籍**：这三部分综合在一起，推荐必看书籍包括《Unix环境高级编程》、《Linux高性能服务器编程》、《POSIX多线程程序设计》

写在最后

联系我，直接微信扫码，给我私信即可↓

> 本文 GitHub https://github.com/rd2coding/Road2Coding 已经收录，里面有我整理的**6大编程方向的自学路线+知识点大梳理**、**我的简历**、**面试考点**、**几本硬核pdf笔记**，以及**我的程序员人生**，欢迎star。

每天进步一点点，Peace！