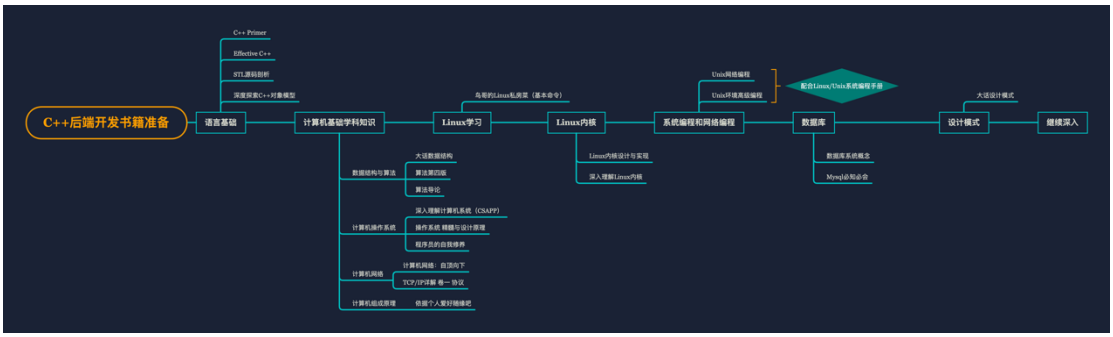

这里从个人视角介绍一下 C++技术栈后端开发的岗位主要学习内容，线路以及企业部署的技术点以从宏观层面对其进行掌握。

如果是本科阶段的同学，我想说，放弃学校的课，自学自己制定的路线吧（数学除外，不过有能力也建议自学）。学校的课最后 1-2 个月突击即可，题目都很套路，问题不大，千万不要被本科课程牵着鼻子走。

大多数初学的同学有一个问题，就是对于一个语言的区别和作用不是很明确。这里先谈一下 C/C++和 Java 的区别。

C/C++和 Java 确实不太一样。C 语言和 C++，尤其是 C++，语言密度更细，机制多，性能虽然高，但是就语言本身来说，包袱太重。所以也就称之为“造轮子”的语言，但是也正是因为他性能好，密度细，所以什么都能做。而 Java 本身是服务于互联网软件开发（后端开发、客户端开发）的语言，他有一个明显的生态圈的概念，所以应用领域非常清晰。因为 Java 是纯应用层的，所以相对而言 C++学习更加困难一些，对于程序员能力的要求要更高一些。

对于 C++的岗位

C/C++属于造轮子的语言，几乎什么都能做。不过一般来说还是做后台（服务端）开发比较多。比如：

- 通信公司后台开发
- 互联网公司后台开发
- 游戏公司后台开发等等。

后台开发又回分出很多的细节，比如

- 做数据处理和分析
- 做基础协议和通信
- 做服务端底层应用优化
- 做后台系统驱动和内核等等。

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。

学习建议

在我们初步学习的时候，首先从以下方面进行考虑。

- C++基础
- 数据结构与算法
- 操作系统
- 计算机网络
- 数据库
- 设计模式

对于技术岗位，软件基础知识可以说是个人的硬实力，是你能通过面试的一个大前提，即使后面的东西都还没有学习，基础部分肯定是要完成的，这也是后

面所有应用框架学习的基石。反之，在应用框架的学习时如果感觉吃力，可能非常有必要回过头来再巩固对应的基础知识。然而基础的确不是一蹴而就的，确实需要一定的反复和回炉。面试也主要是这些基础，公司校招一般看基础咋样，可塑性如何，这才是重点。之后才是项目和一些花里胡哨的。

有些很有意思的方向，可以作为进阶学习，对于大部分岗位面试而言，是加分项，而不是必选项。

- 分布式系统
- 编译原理
- Functional Programming
- 云原生、容器、服务编排等。

首先，对于软件基础知识的学习，建议看经典的英文书籍（上进的青年还是啃生肉吧，毕竟原汁原味），看第一遍的时候可以不用过于细致，快速的看，重点了解知识框架，在把握了整体的逻辑框架之后，第二遍可以详细学习各个章节的细节，这个时候可以根据自己的情况，选择性的看一些重要章节，。对于一些经典技术书籍，只看一两遍很难完全理解其中的精髓，之后可能还要多次进行回顾，也可能还需要在实际应用中再回顾书中的理论知识。例如深入理解计算机系统这本书，内容非常经典，很多地方当你再看一遍又会有不一样的理解和体会。总是学习是螺旋上升的复杂过程。

先学习语言和数据结构基础，然后并行学习网络、操作系统和刷题。

准备时间比较短的话，不用将每个知识板块的所有书籍看完，可以先看完基础书籍，例如 C++看 Primer，数据结构看大话数据结构，网络看谢希仁的计算机网络，操作系统看 CSAPP，数据库对于 C++岗考察不多，优先级放到最低，

可以先暂时不看。看完基础书籍之后，会形成基本的知识，但是深度不够，这个时候再考虑在各个知识板块看一些进阶书籍，建议你至少有一方面学习深入（语言基础或者操作系统）。

学习路线

首先说明，这里的学习不完全按照顺序进行，有同时交叉进行的，同学可以自行选择。

1、 C++基础

软件学习中，语言肯定是最基础的。学习 C++之前最好先学好 C 语言，他是 C++的一个子集。推荐 C primer plus。这个先导课学完以后可以在学一下它的进阶：数据结构与算法分析，或者大话数据结构。在学习数据结构的同时，可以开始同步学习 C++语法知识。

由于 C++比较底层，语法非常灵活（非常适合写算法），这就导致了语法规则十分繁琐，而且涵盖了 C 语言的内容，所以学习 C++相对于学习其他语言来说成本要高很多。但是如果学好 C++，学习其他语言可以说是速成。

C++ primer C++基础语法学习相对最权威的书籍，以 C++11 来讲解，非常全面的讲解了 C++的语法以及 C++11 的新特性，至少要通读一遍，把我整本书的大体框架，然后结合个人情况选择性的区看一些重点章节。

Effective C++ 这本书主要讲解 C++程序的过程中需要注意的一些条款，有助于梳理在编写 C++程序的时候需要注意的一些常见的问题和注意事项。培养好的 C++编程习惯，也是面试常考的。如果我们认真读过了 C++ primer，那么

EC++读起来是非常快的。

STL 源码剖析 这本书讲解了 C++的底层实现，内容包括 C++底层的内存管理、各种容器的数据结构的实现、常见的算法实现等。个人建议列为必读书籍。这个可以帮助我们深入理解 C++底层，同时也可以当成我们数据结构的复习和巩固。而且里面也有很多面试常考的点。

深度探索 C++对象模型 这本书讲解了 C++面向对象的底层实现机制，如果可以的话直接读英文原版。翻译版本，内容比较晦涩，但是十分重要。尤其里面讲到了虚函数的实现机制。看完这本书，会对 C++面向对象的理解帮助极大。建议必读。

2、 数据结构与算法

数据结构是软件设计的精髓，一定要把基础打实。

我们在学习算法的时候，首先要了解的是算法的思想，算法实质就是在用巧妙的方法处理数据，既然是处理数据，他就有逻辑，所以我们要好好捋清楚他的思想逻辑，然后在用我们所学的语言去实现这里面的思想逻辑。

对于复杂度分析的部分可以尝试去看看，如果很讨厌公式的推导也可以直接跳过，但是一定要尝试推导一些常见算法的复杂度。当我们推导的多了自然而然心里就对复杂度有了认知了，当我们在看见一个算法的时候就能通过代码大体估算出他的复杂度。

一般来说我们达到使用一门语言熟练的复现出这些算法思想就可以了，如果想要在算法上下更多功夫，那么我们除了学习这些算法以及刷题，我们需要对于算法有进一步的了解，比如一个算法的来龙去脉，缺点优点，应用场景。当我们

对于这些算法有了这些了解。我们在心里就会有一个大纲和对比，那么如果把我们放在一个具体的应用场景里面的时候，我们才有可能在项目中有一个比较好的应用。

在学习了数据结构与算法的同时，可以同步学习 C++的语法知识。当两者的基础全部学习完之后，就可以开始踏上刷题的道路了。同样，刷题也可以和前后的学习同步进行。在刷题的过程中，开始学习 STL 源码剖析，有助于你更游刃有余的使用 STL，提高代码效率。

个人认为，我们不要去随意窥探别人解题的各种技巧。我们在没有一定的思维方式和算法基础的时候，随意去看别人的代码和逻辑，一来看不懂，二来我们还要在逻辑梳理上耗费大量的时间，甚至在没有一定积累的情况下，我们可能连逻辑都梳理不通，这样又会浪费我们大量的时间。

当我们把各种算法总结分类，有了一套自己的体系，这个时候我们可以开始尝试做一些项目，比如 JSON 库、正则引擎等等。

3、 操作系统

在语言和数据结构之外，作为学习 C++体系的同学，我们需要对操作系统有一个比较深入的了解，而且面试问到的操作系统都会比较深，比较他本身就是很深奥的一个东西。

学习操作系统，我们可以先看**深入理解操作系统（CSAPP）**，操作系统方面的神书。我们第一次接触到操作系统的时候，可能很多概念都难以理解，我们遇到非常难得概念的时候可以先去查找相关的内容，实在学不会就先跳过，继续后面的内容，没有必要按照安排的章节顺序学习。比如操作系统涉及的网络，可能

我们学完计算机网络和网络编程后，再回头看就非常简单了。

我们根据个人情况过完第一次以后，就可以开始学习 **操作系统 精髓与设计原理**，我们可以再将这本书结合 CSAPP 一起看，两本书互相补充。

看完前面的相关的内容，如果我们相对于编译链接的原理深入的学习，可以参考程序员的自我修养，这本书非常详细的讲解了程序编译过程中的具体细节。

另外，我们还需要针对 Linux 进行学习，可以先看 **鸟哥的 Linux 私房菜** ，这本书速看，主要就是了解一下 Linux 的基本命令。进一步开始学习系统编程，可以先看 APUE，其中对于系统编程常用接口的使用方法和对应的应用实例可以好好看看。由于这里实操较多，理论较少，可以配合 Linux 编程手册详解来看。

学完前面的操作系统相关内容以后，我们可以对 Linux 的内核展开深入的学习和钻研。这个时候我们可以看 Linux 内核设计与实现和深入理解 Linux 内核，前者以实践为主，后者以理论为主，我们可以先学习理论，再从宏观的视角理解各个知识板块的逻辑关系和实现。

4、 计算机网络

学习网络，可以先看计算机网络或者计算机网络：自顶向下，关于物理层和链路层可以快速过，了解一下基本原理即可。重点学习放在网络层、传输层和应用层，其中传输层的 TCP 和 UDP 非常重要。

网络的知识比较容易理解，但是知识点比较繁杂，所以我们要注意文档或者笔记的建立。

学完计算机网络，我们对于计算机网络的知识体系基本就建立起来了，这个时候我们还需要对网络有一定深度的了解，此时建议学习 TCP/IP 详解。

如果我们对网络编程继续进行实践，就可以继续学习 Unix 网络编程。

到这里为止，我们就基本就满足所有的要求了。剩下的事情就是我们还要丰富一下自己的项目经验，多做一些工程。比如当我们学习了网络编程以后，可以学习一下 Linux 多线程服务端编程 来学习 C++的一些实际应用。如果我们看完了常用的网络 IO 模型，还可以自己实现一些网络库，并基于此实现一个 http server。

5、 数据库

数据库是我们工作中肯定会用到的，我们首先把数据库的基础打好，可以看看数据库系统概念，了解数据库的基础知识，然后阅读 mysql 必知必会这本书，可以帮助我们快速学习 mysql 基本语法。

6、 设计模式

对于设计模式，我们目前没有大型软件的项目经验，这方面的能力很难体现出现，但是他是可以作为知识点扩展学习的，了解每个设计模式的思路就好。

项目

我们很多人学习 C、C++都是在类似 Visual Studio 这种即成 IDE 里面进行代码编译，这个其实用到了编译器，不过是微软自家的。但是作为 C++，更多是在 Linux 平台的编译器，所以我们还需要熟悉一下 Linux 平台的 GCC，甚至有的公司会自己定制交叉编译工具，但是无妨，只要我们熟悉 GCC，其他的问题都不大。

而且，大家在自学 C、C++，都是在借助 IDE，点击按钮就可以进行编译运行。而我们这个编译动作其实叫做 make，编译的实际动作和过程是写在 makefile 文件里面的，所以对于 makefile 的书写规则也建议学习一下。

然后说调试，Linux 平台的 GDB 调试工具要熟练使用，我们会借助它进行调试。可以直接看 GNU 的官网关于 GCC 和 GDB 的文档，或者看看 debugging with gbd 或者《跟我一起写 makefile》。

要知道 C 或者 C++写的项目是非常朴素的，没有 python、Java 那么美观的可视化界面、网页等等，大多都是运行于命令行那种黑乎乎的界面，甚至完全跑在后台的，所以学习起来可能也会比较枯燥。

进一步学习

这个时候如果想要继续精进我们的水平，就可以开始学习 C++并发编程的学习。

如果时间充裕，可以看一下《操作系统真象还原》，自己去实现一个简单的操作系统。

如果还有余力的话，接下来就要开始向分布式系统进发了。这里我看的视频是 MIT 6.824（Schedule：Spring），书籍方面个人推荐《Design Data Intensive Applications》。

为了变得更强，还需要看一看工业界中常用的东西，各种中间件，比如 Redis，可以看看它源码中感兴趣的部分。还有消息队列，这个也是特别值得学习的东西，了解它的原理，看它的源码，了解大神的 C++代码是怎样的，也是可以学习到很多东西。

例如我之前才学到的 clang 支持的 thread safety annotation。自己也可以实现一个单机版本的 kv 数据库，可以用最简单的 bitcask 的方案做存储，再加上自己写的网络库来对外提供服务，做起来也会很有趣。

再往后就是我做的一个详情图了，当我们有前面基础的积累，有了自己的编程思想和思考问题的方式，我们在学习后面的东西就很简单了，就不做过多描述了，可以直接看图示。