计算机基础对于编程的重要意义

相信很多计算机专业刚步入社会开始编程工作的同学都有一个疑惑，大学四年学的计算机基础课程对编程来说几乎用不上，远远没有 C/C++、Java、Web、sql 有用。我当时也有这样的疑惑，因为大部分程序员刚开始工作的任务仅仅只是在别人搭好的框架下，在合适的地方添加上合适的代码，实现某个功能。在这个阶段有这样的疑惑是可以理解的，因为只是在山脚下看问题，不知庐山真面目。在这个阶段有些有心的同学可能会在编码过程中考虑下性能（方法级别的性能），合理使用集合类，合理采用一些算法，减少循环次数和时间复杂度。其中不乏仅仅为了完成任务而完成任务，只保证功能实现，不出语法错误。如果以后项目膨胀，且不提格式规范、性能、线程安全、内存管理等问题，就是耦合都可以造成致命问题，毒瘤程度可想而知。

对于前端来说，一个完全不懂程序是何物的人，几乎经过自己学习都能使用 html+css+js 编程实现页面。于是随着工作时间的增加（熟悉使用方法后），一些有追求同学开始追求实现原理了，而不限于会用了。懂得实现原理之后就有种想借鉴的想法，对于已经优化过了还是会遇到加载很慢的情形开始考虑继续优化的问题了，或者突然在 github 上见到某些代码结构（设计模式）特别优美。

基础课程

综述

有些偏题了，回到正题来。这里大致列了一下我学的计算机课程，对于科班出身的建议直接跳过：

基础相关：计算机概论、数据结构、操作系统、计算机组成原理、计算机网络、计算机体系结构。

编程相关：数据库、软件工程、编译原理、计算机安全与密码学、计算机图形学、微机原理与接口技术、算法基础。

编程语言：C++、Java、JavaWeb、C#高级编程、汇编语言。

数学相关：高等数学、线性代数、概率统计、离散数学。

其他跨学科：大学物理、大学英语、数字电路、模拟电路、计算机英语。

研究生课程：组合数学、数据仓库与数据挖掘、机器学习、人工智能、算法导论、专家系统、智能仿生算法、模式识别

数据结构

相信很多前端程序员最初写代码的时候开始真正思考的时候都是对对象、对于数组，直接使。数组中的常用方法（push ,pop,shift,unshift 等等）用着太方便了；用 set 还是数组；这个提前装进 map 不用每次查询会不会更好；对于递归的栈实现；在数据库中数据怎么表示成树结构，树结构数据如何查找、如果涉及到树结构某条数据的增加和删除怎么维护，这就涉及到了简单的数据结构知识。排序查找实现，虽然数组中提供了相应的实现方法（sort），但是如果需要特定场景的话，需要自己去实现。在数据量足够大的数据里查找一条数据，怎么查找比较快呢，二叉排序树的概念产生了，为了提高查找效率，二叉平衡排序树（AVL）降低了树层次。数据库索引就借鉴了这个，只不过数据库是 N 叉平衡排序树的变形。图的用法也很经典，只是目前自己接触面有限，还没有接触过，如果做

社区、物流、链路选择等直观情形，以及可以模拟出树结构的数据，这个可能会用到。可以说不理解数据结构就不算程序员。

操作系统

如果在修改数据的时候，是否同时有其他地方在修改这条数据，这样数据不就紊乱了吗？于是操作系统的多线程问题就出来了，线程同步和线程异步概念也开始清楚了，线程异步使用场景也很熟悉了。例如发送短信验证码、推送通知、文件 IO 等不影响主逻辑执行且相对较慢的操作，采用异步就行了。后来在操作系统上看到线程和进程从产生到消亡需要开销维护，如果同时有很多请求，对于每个请求要新建一个线程运行完后消亡，岂不是要开销很大资源吗？于是想到数据库里面有数据库连接池的概念，采用线程池来维护产生和消亡岂不是更好，但时新的问题会产生，如果用户大部分时间都是集中在某个峰值访问，线程池不够用怎么办，干脆排个队吧，在哪排呢？数据库/内存，数据库太慢了，在内存中万一服务重启了数据丢失了怎么办，正好想到系统中正在用的 redis 进行数据缓存，干脆也放进去排队吧。后来接到一个 OA 的任务，其中有个办公用品的发放与补给，听着很简单，但一思考觉得库存和消耗做起来怎么这么麻烦啊，思路总是补清晰，然后灵光一现，这不是操作系统上的生产者和消费者模式吗，于是问题迎刃而解，但是在编码中总会遇到一些困扰，直到了解了任务队列的概念。操作系统的虚拟技术使用场景等等。现在反观操作系统，发现操作系统简直就是最好的算法书，如果数据结构属于使用级别的，那么操作系统就属于创造级别的。在需要加载大量数据的缓存中，如何淘汰和替换缓存中的数据，分页淘汰算法提供了参考；如果编程到一定阶段再翻翻操作系统发现进程同步机制、进程通信机制、消息缓存机制、管道、死锁预防与处理、调度机制算法（FCFS、轮转法、多级反馈轮转法、优先级法、最短作业优先法、最高响应比优先法）、虚拟存储技术、静态/动态分区法、分区回收机制、覆盖和交换技术、分页管理、分页置换淘汰算法（随机淘汰、FIFO、LRU 等）、分段管理、文件物理结构逻辑结构、文件存取控制、数据传送控制方式（程序、终端、DMA、通道）等每一个都有在编程中的应用场景，还有关于 Linux 的基础知识等。难怪数据结构和计算机操作系统在计算机专业课考研中占据的比例是最大的。操作系统就是一本启发式的高阶算法书。

组成原理

计算机组成原理中数据的机器层表示能对数据存储有很深入的理解，对编程语言中的数据类型理解和使用起来会非常容易，数据运算过程和原理对于　大浮点数计算结果的误差也会理解，对数据溢出也会有比较深刻的理解，也能帮助定义数据类型时合理优化。指令系统能够了解计算机程序执行原理和过程，对某些底层编程优化起到优化指导作用。存储系统、CPU 系统如 cpu 运行机制、内存读写策略、内存缓存策略等能够在深层次上指导优化代码。外设中，如果理解磁盘的寻道方式，采用不同的循环嵌套查询出的速度会不一样的。输入输出以及中断、总线等有助于认识计算机系统。由于自己水平有限，还做不到在这个层次上对代码进行优化。总的来说，计算机组成原理对大部分人来说是了解工作流程的作用，让人知道我的程序是如何工作的，而不是一头雾水的去工作。当然其中有很多方面，对编程理解还是很有帮助的，在高层次的来说，计算机组成原理能起到指导优化代码的作用。

计算机网络

计算机网络：网络层次划分了解网络传输原理、以太网、点对点协议 PPP、IP 协议、ARP、IARP、VPN、NAT、ICMP、子网、路由选择协议、TCP、UDP、http 等传输

协议的理解，TCP 流量控制、TCP 拥塞控制、报文格式抓包分析等、TCP 连接和断开过程、DNS、FTP，其中一部分可以借鉴使用（流量控制和拥塞控制、路由选择），一部分可以分析数据（报文格式）、更多的是了解，一部分可以优化传输（数据包的传送），了解不一定记得但是最常使用的，如果对这些完全不知道，也只能在单机状况根据别人提供的框架下做一个码农。

数据库

常用的是 sql 语句以及数据库管理，这点重要性不言而喻。触发器、存储过程、视图在某些情况下能减轻编码的负担。数据表设计规范三大范式的要求对数据库设计很重要。

一个好的前端不仅仅是一个前端，不要只盯着眼前的一亩三分地，更要了解前端以外的知识。

一个项目启动前，通常会有一个需求讨论会。如果你不懂理论知识，当聊到数据库、服务器、并发等名词时，你就只能两眼一摸黑，插不上嘴，安安静静的坐在一边。但如果你学过这方面的知识时，你就能和他们一起指点江山，不再是外人。

在前端方向，理论知识也有用武之地。例如 `babel`，就需要用到编译原理；研究 `webgl`，还得用上图形学的知识；学了软件工程，你就明白测试、团队规范的重要性和必要性。说白了，懂计算机理论知识的前端和普通前端是站在不同维度上看问题的。

计算机已经发展几十年了，中间淘汰的技术数不胜数，前端还是最近几年才火起来的，说不定哪天这个职业就没了。如果发生这种情况，你还能干什么？

技术会过时，理论知识不会过时，只要冯诺依曼体系还在，你学的东西就一直有用。学好计算机理论知识，不干前端，还能干别的。