看前必读

作者介绍

大家好，我是帅地，2020本科毕业生，校招提前批拿腾讯等多个大厂研发 Offer，毕业半年通过技术变现，挣到人生第一个 100 万，2020 年CSDN 博客之星，单篇文章破百万阅读.....

2018 年开始写技术文章，很多文章都火遍全网，比如写的算法文章在知乎单篇破万赞，这几年专注于**校招垂直**领域，帮助无数个学生拿到大厂 offer，可以说影响了一大批想要认真学习的小伙伴。

我相信很多读计算机专业的同学也和我一样，可能家境也不是特别好，也希望自己毕业后能够找到一份好的工作，帮忙改善自己的家庭情况...**总的来说就是，让自己可以在毕业之后，可以为自己感到自豪，而不是回首过去的思念，满怀后悔**。

所以呢，帅地写了这本比较短地小册子，并这这本册子命名为《帅地玩编程》（与公众号同名），希望能够帮助大家，找到一个可行的学习方向。

未来我们一起，加油！

小册有哪些内容?

这本 PDF 把帅地这几年写过的精华文章以及学习心得进行了更加细化的整合，比如**学习路线，校招攻略，个人学习经历啊**，可以说，从 0 到 1，这个校招求职的内容都覆盖了，并且这份 PDF，未来还会一直维护下去，就是基于最新的校招情况，不断更新迭代，所以帅友们可以大胆拿去参考，所有内容，都是帅地，**精心挑选**，质量绝对 有保障。

当然这份 PDF 不仅仅包括校招，也把自己的一些人生经历，抉择都写了，如果你有任何疑问，都可以来翻一番这份 PDF。

当然，后续内容更新可能无法及时在 PDF 更新，那么你可以来帅地维护的校招专属网站这里找：www.playoffer.cn

小册子阅读指南

如果你是一个相对迷茫或者没有明确规划的小白，那么我觉得你可以先阅读过来人的一个经历，也就是先阅读**大学之路**这一部分，看完之后有了一些自己的思考，那么你可以选择一条属于自己的学习路线来学，也就是看**前后端学习路线**这一部分；学习这些东西，更多的是以**校招**为标准，所以你在学习的过程中，可以先去了解校招相关的东西，也就是看**互联网校招准备**这一部分。

事实上，当你看完了这几个部分，**你就知道自己 应该干啥了，对自己也有了一个定位**，而这，也是小册子的目的。

这个时候如果你想选择就业不考研，那么就可以按照我说的知识去准备校招，如果想要考研，那么就把精力放在保研/考研上吧。

校招垂直网站

网站地址：www.playoffer.cn，网站的昵称是**PlayOffer**，翻译过来其实就是**玩offer**，代表轻松拿捏编程，也就是帅地玩编程，这也是帅地正在努力打造一个**垂直的校招**网站，并且所有的内容都是帅地经过严格的筛选，保证超高质量，毕竟网上各种杂东西太多了，容易误导或者浪费大家的时间。

不过呢，该网站还在更新中，很多内容还没有上架，你们可以关注官方配套公众号（PlayOffer），后面有更新内容都会在公众号发通知，大家也可以关注公众号哦。

另外就是，这里也有一个积累了很多面经到网站：https://www.iamshuaidi.com，也是我在维护的网站哦，这个网站主要专注于面经这块。

专属交流群

为了把各位热爱学习的小伙伴聚集起来，我也给大家创建了一个微信群，可以给大家交流分享，毕竟大家都是未来的人才

如果你想要进去，可以关注下微信公众号：**PlayOffer**，然后发送「**加群**」暗语，就可以获得加入方式了。扫码直达

如何找到帅地?

帅地有两个搜索公众号，你们可以关注，微信直接搜索对应名字就可以了：
每日问帅地：帅地每天会发布读者提问的一些问题

帅地玩编程：帅地会在这个公众号发布大学学习规划等文章

网站：https://www.playoffer.cn，所有的内容会沉淀在这个网站上

我的大学之路

除了大一之外，我觉得我的大学应该比大部分人都要有规划，事实证明一个有规划的大学，可以让你在毕业的时候，拥有更多的选择，我不大喜欢对别人说教，更多的还是喜欢讲诉自己的故事以及自己对事情的处理态度来给大家作为一个参考。

所以呢，这一部分，主要简单讲解下我在大学的一些事情，如果你觉得自己的大学比较迷茫，或许你可以看一看我的这些经历，可能就拥有自己的方向了。

我的秋招：从双非到上岸腾讯

历经两个月的秋招总算是结束了，从七月份开始复习秋招相关知识，到八月多开始笔试、面试，到九月下旬的秋招结束，在笔试面试的这两个月里，还是挺累的。这篇文章就说说秋招这段时间的收获以及给对于明年要参加秋招的同学的一些建议吧。

一、最后去的公司

对于我来说，这次秋招算是满意的吧，找到了想去的**城市（深圳）以及公司（腾讯）**，我投的岗位都是**后端开发**。在之前春招找实习的时候，人生的第一次献给了腾讯，那时候没啥面试经验，感觉傻傻的，没看过的可以看我之前写过的文章嗯，春招两次腾讯面试都挂二面了，分享下我失败+傻傻的面试经历。

在秋招，腾讯也是我第一家面试的公司，感觉还是挺有缘的，8 月 14 号接到了腾讯面试官的面试预约，当时突然有点后悔，感觉自己应该晚一点投，因为腾讯的提前批是 9 月 12 号才结束，正式批 9 月 26 号开始。感觉当时还有很多没复习，想晚一点再面试。**后来，我才知道，我错了，真的是越早投越好，千万别等到正式批或者提前批即将结束才**

投，那个时候投，真的会错过很多机会（至于为什么，后面会说）。17 号开始了秋招的第一场面试，到 8 月底面完了所有流程，9 月下旬出才收到面试结果。下面谈谈这次秋招的感受吧。

二、关于我

可能没看过之前我的文章的，很多人还不知道我。这里我简单介绍我的背景吧。

我今年大四，大一学的专业是**木材科学与工程**，后面转专业到**软件工程**，老家是广东的某个 5 线城市，在广州这边读大学，当然，是某个**双非大学**，至于是哪个？学校里有养**神兽**的就是了。

可能看我文章的读者中，很多人觉得我很厉害，说实话，其实我还是挺菜的，在校期间没有参加过任何比赛，没有拿过任何奖金，扎心了。所以这次能够拿到大厂的 offer，我觉得得归功于我之前对**计算机基础知识**以及**算法**学习，所以对这两方面，个人觉得比大部分同年级的人强，想拿大厂 offer，**基础知识 + 算法**必须重视。

还有就是，我有一个非常重要的点，就是能够把各类知识串联起来讲，当时入职腾讯时，面试官问我是不是**面霸**，因为面试官问的东西，好像我都能行云流水去扯一样，当时面腾讯时，第一面聊了 90 多分钟，喉咙都聊痛了。

三、基础 + 算法 + 项目

1、关于基础知识

秋招的竞争还是非常激烈的，如果你想要在秋招中拿到满意的 offer，那么从现在开始，就要把**计算机基础**（操作系统、数据库、计算机网络）、**算法**学好，特别是算法，不容易临时抱佛系，是一个长期积累的过程。

对于大厂，比起项目，它更加主要你的基础能力是否扎实吧。记得腾讯一面的时候，面试官就**哈希表**这个问题问了我有二十分钟，从刚开始让我用 C 语言来设计一个哈希表，后面问我如何设计 hash 哈希，怎么样设计更高效，怎么样设计能够最大程度减少碰撞，是否要动态扩容等等。一系列问题，我都按照自己的理解回答了，有些引用 redis 、 hashmap，并且我都举了一些例子。这个问题回答之后，感觉面试官有些惊讶，问我是否研究过 redis 这些框架的源码等。感觉这个问题回答之后，面试官对我更加感兴趣了。

所以我觉得，对于秋招，理解常见数据结构的相关设计，为什么要这么设计，实在是太重要了，可能很多人都知道链表、树、哈希表等，但被深入一问，可能就不懂，不知道为什么要这么设计了。

这次秋招，被问的最多的就是操作系统、计算机网络、MySQL了，虽然我面试的是 Java 工程师，但是很多公司并没有问我 Java 相关知识（ಠ_ಠ），不过这和一个公司的技术栈相关吧，像我面试的 腾讯，字节跳动，shopee，小米等，公司的主要开发语言不是 Java，所以这几个公司的面试，一个 Java 相关的知识点都没有问过我，反正我是哭了。不过这并不影响我的回答，因为这些计算机基础知识，我很早就在准备了。

所以对于要参加面试的同学，千万别把自己吊死在某个语言上，语言只是一门工具，而应该多花一些时间在一些**通用的知识**上，例如 **sql + Linux + 算法 + 操作系统 + 计算机网络**。

当然，如果公司的主要语言是 Java 的，还是会问很多 Java 相关知识的，例如我面试京东，蘑菇街，阿里的时候，就问了很多 Java 的知识，像京东，蘑菇街，cvte 就没问过我计算机网络、操作系统这些知识。

所以说，不同公司，侧重点还是不大同的，但是，对于 BAT 这些大公司，**基础知识 + 算法** 是必问的。

2、关于算法

如果算法学的差，会错过非常多非常多的面试机会（学历好除外），会很难过**笔试**这一关，秋招的笔试，反正我一直被虐，感觉笔试的难度还是很大的，自己一个人做笔试，想要全 a，还是非常难的。笔试题目一般是**选择题** + **编程题**，但有些公司没有选择题，全是编程题（不过现在提前批不用笔试了，直接面试）。不过无论是否有选择题，编程题做的差，就凉了，一般编程题占**60%**的分值。

反正我有挺多笔试环节就挂了的，有些我编程题全 A了，然而并没有收到面试通知，估计是我学历一般吧，有些公司还是看学历的，毕竟面试成本挺高的。

有人说，leetcode 的前 500 道题刷了，笔试稳吗？说实话，还真的不稳，得看你的掌握程度，像 leetcode 那些题，一看就知道是什么题型，应该用哪种算法。而笔试题完全不一样，很灵活，可能是多种算法的结合。而且，有时候题意还得看十几分钟才看懂要我们干嘛。不像 leetcode，就几十个字，简单明了。反正 leetcode 中挺多 hard 级别的题我都会做，不过笔试的难度有些并没有 hard 高，却做不出来。因为时间也是挺紧的....大概一道题只有 30 分钟的时间给你做吧。

所以，那些经常刷 leetcode 的，我的建议是，如果你要拿下笔试，那么应该彻底搞懂这道题的算法思想，力求**最优解**，但是说实话，很多人的情况是，面试中的算法都没搞好，笔试中的就更加不用说了，在时间不够的时候，这里还是建议别再笔试花太多时间。

对了，还有一点，建议大家在刷题的时候，直接在网页那里打代码，别跑到 IDE 里写了，因为面试手撕代码的时候，并不会给你 IDE 写，而是在笔记本手撕算法，如果你不熟悉的话，估计代码会经常写过，而且排版可能也会很乱。反正我春招面试阿里的时候，让我在笔记本做算法题，我哭了，调用库函数的时候，方法名啥的全忘了怎么写，而且代码也老是写错。因为平时在 idea 会提示，在笔记本没提示，特别不习惯。

最后就是，关于算法，笔试我们不做要求，面试则至少刷完常见的 easy + medium 的题的，帅地基于《剑指Offer》也整理了一份攻略，你们可以跟着我这份攻略刷完大概 70 道高频题吧：对于目标是中大厂的，这应该是最低要求了

3、关于项目

基础、算法很重要，进大厂缺一不可。那么对于一个参加秋招的学生来说，项目重要吗?

答是**非常重要**，我秋招最大的弱点是项目经验不好，这也让我在很多公司直接一面就凉了。我自己没有脚踏实地着去做一个项目，都是看视频速成的，而且自己也没有好好跟着视频打代码，自己尝试去做一个项目，和跟着视频去做，还是有所不一样的，毕竟跟着视频，很多东西不是自己想的，所以不深刻。

不过后面为了弥补自己的软肋，也是看了很多书，掌握项目中的一些理论，特别是场景题，感觉这块我答的还是可以的，主要得益于我对很多原理的理解，可能让我更加有思路去分析一些东西。

记得蘑菇街一面的时候，面试官一上来就让我讲项目，然后我就讲牛客网学的哪个项目，面试官让我讲**线上**的项目，别讲**练手**的项目，我哭了，因为我没有线上的项目，因为我的暑假实习，实际上就是去培训，并不像其他人去公司实习，可以参与到完整的项目流程。这个时候，我就随便说了培训期间水的一个项目（几天时间快速水的），然后我就被面试官怼死了，，，然后就没有然后了，一面挂。

然后节点 cvte 面试的时候，一面二面全程怼项目，全是我的弱项，我也哭了。可以说，秋招我最大的弱点是项目，多次被怼告诉我，**秋招，一定要有一个项目，这个项目不需要多高端，但需要你真正动手做过，研究过**。

所以说，项目非常重要，可以打打增加面试的成功率，特别是中小型公司。当然，我觉得对于有些大厂，如果基础理论非常强，算法也可以，也一样能进，因为有些公司并不看重你的项目，例如我面试过的腾讯，字节跳动，shopee，小米等，基本没怎么问项目（可能对我的项目不感兴趣，哈哈）。

不过在未来，我能预期到，**项目实践能力会越来越重要**，因为现在网上八股文太多了，大家早早准备，

4、总结

所以我觉得，只要把基础打好，算法学扎实，并且弄些项目经验，进大厂还是有机会，当然，竞争越来越大，进大厂也越来越难，所以也需要足够的决心去做，可能才有机会进大厂，毕竟毕业年薪就是三四十万，确实非常不容易，如果认真卷三年能够进大厂，还是值得的，对于普通家庭来说。

不过我觉得随着网上八股文越来越多，项目经验会变的越来越重要，进大厂不进要能被八股文，还得融会贯通，让面试官觉得你真的理解了，总之就是，要多思考吧。

四、关注我，助你搞懂面试必考点

我已经有好几个没写文章了，不过在接下来的日子里，帅地会基于最新的校招面试，出一系列相关的校招攻略教程，大家可以关注我的公众号「**帅地玩编程**」，也可以关注我的校招网站：https://www.playoffer.cn，同时也可以加入 QQ 群，小册子开头那里有些加入渠道。

普普通通，我的四年大学

前阵子有些读者问我大学期间的学习路线，说他自己现在有点迷茫。说实话，对于学习路线这种文章，一抓一大堆，我也不大喜欢去建议别人究竟该怎么学习，学习顺序之类的。不过对于大学，很多人进入大学的时候，可能都是同一个起点，大学四年过后，却是完全不同的人生轨迹。正好我也想记录下自己从高中进入大学这几年的学习与变化，我的大学经历，可以说是非常普通，没有参加任何竞赛，也没拿过奖学金（当然，国家助学金得拿，哈哈）。也正是因为普普通通，我才要分享，因为我相信你们都看过很多充满各种牛逼的大学经历。

所以这篇文章，就以时间的顺序，记录下这几年的学习、想法、看过的书等，或许，普通的大学经历，更加有参考性也不一定哦，嘻嘻。

一、非计算机专业的大一

在我高考分数下来的时候，看了下分数。我去，比预想中了至少少了几十分，真心想不通我的数学和英语为啥能考这么差，平时数学、物理是我的强项，算是经常全班第一，不过高考数学得比预想中的少了几十分，处于及格边缘，而且，英语也并预想的少了二十多分，当时还是很郁闷的，高考这么重要的场合，居然考这么低。

不过说实话，当时我并不伤心，因为我觉得，学校虽然挺重要，但也没有那么重要，我相信自己要干的事，在哪个学校都能干，而且在我看来，**技术与成绩，只是大学的一部分而已**，虽然是比较重要的一部分。当然，能上个牛逼一点的学校当然比较好了，至少还能装个逼，哈哈。

当时我是想读**计算机类**专业的，不过以我的分数，如果选择省内一本学校的话，我会读不上计算机专业，只有被调剂的份（当时没有考虑省外）。不过我是一定要读计算机专业的，并且也想尽量进个一本的学校（不是说看不起其他批次学校哈），当时我就查哪些学校转专业比较容易，所以后来我是报读了一个**容易转专业的学校**，也就是选择被调剂，之后在转专业到自己相对喜欢的专业。当然，这里存在转专业失败的风险。

后来我是被调剂到**木材科学与工程**这个专业，也算是意料之中，在开学的前几天，我看过一本对我影响挺大的书，叫做《李开复自传》，书里面有一句话一直影响着我，大致的意思就是『**当你把大学所学的知识全部忘光时，剩下的，便是教育的本质**』，说实话，我还是很喜欢这句话的，当时我解读为，在大学，**提升自己的思维、学习能力**才是最核心的。所以在之后的学习中，我都是比较在意自己的学习能力的，对于一件事情，我也会去认真做好，因为我觉得，**认真去做一件事情，非常重要，它可以间接着去提高你的学习能力**，尽管这件事情可能和你所学的专业没有任何关系。

在木材科学与工程里，有一门科目，就做《工程制图》，我去，我真的是被这门课给折磨了，画各种图。例如根据一张螺丝的图片，画出它是三视图，这种东西对我来说，太难了。后来我就想，我认真去学习下我就不信我会搞不定，后来事实证明，**人真的是各有所长**，对于制图，我真的不行，虽然认真学能搞懂，但是花是时间多，并且效果也不大好。

从那时起我就觉得，虽然**学习能力**是通用的，但是，**有些技能，你还真的不大适合**，因为**兴趣、以往的技能积累影响着你对这门技能的擅长程度**。选择自己比较擅长的技能去学习，还真的挺重要，**从而也更加坚定了我要转专业的想法。**

大学第一学期的国庆，我开始自学 c 语言，主要是看书。当时自己跟着书本打 demo，感觉挺有意思，好像七天时间，我学到了结构体部分，感觉自己还学的挺快的，当时可以说是不求甚解（就是感觉自己稍微理解了就接着下一章），接着学指针，我去，难度顿时加大，感觉遇到了瓶颈，之后就随便看了下没继续看了。

稍微学了点 C 语言之后，就像写个稍微好玩一点的程序，然后并不知道 C 语言能干什么，书上也没有啥有趣程序的案例。

后来我就去水百度贴吧，发现有好多人发了游戏程序，不过好多有些设计到其他知识，我也不懂。最后找到了个**贪吃蛇**的程序，这应该是我见过最简单的贪吃蛇了，只有黑色的界面以及把一个方块自己当做**蛇头**，不过我当时还是挺感兴趣的，就跟着源码打程序，之后在自己理解了**原来游戏是这样弄的**，**画面的动态效果是这样搞出来的**，把程序运行起来之后，稍微添加了一些自己的东西。

讲这个贪吃蛇的故事我是想说，**兴趣和喜欢探索**我觉得是非常重要的，当时在没有任何人教的情况下，自己去折腾，去搜索，最后写了个简单的贪吃蛇，我还是非常开心的。这个过程之中，也对我学习能力的提升很重要，在一个自己完全陌生的环境中，如何快速去适应，**愿意折腾很重要，但是，懂得利用搜索引擎，也非常非常重要**。很多人经常问我一些搜索引擎就能解决的问题，说实话，我是懒的回答的。有些人让我帮他解决下 bug，我笑笑不说话，不是我不帮你，是我帮不了你，也不大愿意帮你，bug 这种东西，你是自己搞出来的，你自己才是解决的最佳人选，解决的过程中，你真的能学习到好多东西，而且这些东西，是潜意识的，看不见的。

二、入坑数据结构与算法

在大一的寒假，了解到转专业考试是在 3 月份，我也在寒假赶紧重新学习指针，并且也学习了链表，说实话，对于当时的我来说，链表太他妈难了，真的被折腾了好久，因为当时并不大懂**内存地址**之类的，直到参加考试，也是对链表似懂非懂。说实话，如果你学习了好几天都搞不懂链表，别怀疑自己的智商，对于初学者来说，真的挺难，特别是对于我这个 C 语言只自学了大致十几天的非科班。

> 这里多说一些建议，对于没学过 C 语言的，如果有时间，我还是挺建议大家学习下 C 语言的，特别是指针那部分，也很欢迎大家把 C 语言中作为入门语言，我觉得，学习了 C 语言，以后学习别的语言，可以帮你理解的更深刻，而且也可以很快就入手其他语言，例如 Java 中的引用啥的，这不就是地址吗。反正，我觉得 C 语言如果有时间，是必须语言。当然，只是我个人的看法。

后来转专业通过了，巨开心，终于可以肆无忌惮着学习编程了。**当时是真的对学习充满兴趣**（这句话意味着后面学习不上心）。大一第二学习，学校开了一门**数据结构与算法**的课。这门课我觉得算是比较难的一门课了，相信很多人也都是被这门课折磨过。

但是，我想说的是，这门课，也就链表难，如果你学会了链表，后面我觉得一点也不难，由于我自己折腾过链表，所以我很快就上手这门课了。我也没有看学校的教材，**学校的教材是清华大学初版的严蔚敏的书**，说时候，这本书个人感觉真的不适合初学者，反正我看不下去，可能是我比较菜吧，当时自己另外卖了本《数据结构与算法分析—C 语言描述版》。说实话，这本书我很喜欢，感觉看着特别舒服，如何你要学习数据结构与算法，那么可以选这本，有 C 语言版，也有 Java 版，不过如果可以，我建议大家用 C 语言来实现那些数据结构。

在数据结构与算法书里，对链表的讲解比较详细，加上我之前看过一些链表的文章，所以很快就理解了链表，也跟着书上一点点去实现链表的基本操作（增删查改），这里我必须说下，**千万别自己理解了就不动手写代码，一定要动手写**，因为写的过程，你一定会遇到很多 Bug，通过解决这些 bug，你会对链表理解的更加深刻。

对于数据结构的学习，我基本是没听课的，因为我觉得老师讲的不适合我，当时老师链表还没讲完的时候，我自己已经学到了**图**那部分了。反正那本书涉及到的算法，我基本都学了，也都实现了，大致有：链表、队列、栈，二叉树、图，图学的比较久点，感觉设计的算法比较多，如深度遍历，广度遍历，最小生成树，拓扑排序，最短路径，我觉得，这几个图算法都很实用，也很重要，不懂的建议大家学。

三、选择不玩 ACM

没进入计算机专业之前，我还是非常想进学校的集训队的，感觉 acm 很牛逼，自己也经常去学校的 OJ 刷题，买了本算法入门的书籍《算法设计与分析基础》，这本书我觉得非常适合入门，用伪代码实现的，很简单，我好像一个星期就看完了。后面也买了本《挑战程序设计大赛》的算法书，我去，当时感觉这本书讲的挺不错，这本书主要是**刷题**，学的不亦乐乎。虽然学的不亦乐乎，不过我感觉那些竞赛的题太他妈难了，一道题有时搞了一天才懂，少则几个小时，并且还是看着答案来理解的。当时还是刷了挺多题的，不过当时感觉要是自己去打 acm 的话，凭借着我这种程度，感觉拿不了啥大奖，也感觉自己确实对这些竞赛题并不怎么敏感，不能一眼就看出解法，而且一道题几个小时，自己实在搞不来，所以我也就放弃了参加学校 acm 的打算了。

不过那几个月对算法的学习，也为了打下了不错的基础。后面我也自己想做点东西，不过 C 语言写的界面黑乎乎的，不大喜欢。所以后面学了些 HTML，这种即学即可以看出成果的技能，感觉学的比较有趣，学了几天，把大致的知识点学完，后面发现单单 HTML 不行，还得学下 JavaScript。然后花了十几天学了 JS，然后就没有然后，因为我移情别恋了。

> 不过这让我懂得了一些前端知识，说时候，我觉得就算你以后不学前端，那么也要应该懂点前端的知识，这是必须的，就算是做后端，也是经常需要用到前端的知识的。

四、瞎搞的暑假

1、windwos 编程

大一结束后的暑假，我挺想写个游戏，可是 C 语言又没有界面，怎么办？后面我发现 windows 编程可以画界面，画图之类的，也是用 C 语言实现的，所以当时我就看小甲鱼的视频学习 windows 编程，自己也顺便买了本《windows编程》的书，那本书 900 页，呵呵。我觉得 windows 编程还是挺难的，各种句柄啥的，大概学了差不多一个月，900 也大概看了 600页，各种键盘事件，鼠标事件，反正好多好多。后面靠着这些知识，写了个计算器，我去，当时还是很开心，还告诉我姐，让他试试这个计算器。然后就没有然后了，我又移情别恋了，哈哈。

不过 windows 编程的折腾，也让我学到了很多，例如知道了我们平时鼠标移动是怎么回事，知道了我们文本编辑器的底层是如何实现的，感觉知道了很多相对底层的东西，也让自己的学习能力更强了。

2、安卓的学习

因为 windows 编程写出来的程序，在手机不能运行，根据没人用我的程序，所以我转行学 Android，不过 Android 听说用 Java 实现的，所以我入门了 Java，好像是看了几天的视频入门的，只学了点皮毛就去写 Android了，买了本书，也是学的不亦乐乎，大概折腾了二十几天，跟着书本写了个天气预报，学习的过程中，由于自己太多的不懂了，遇到了很多 Bug，也是好几次把自己搞奔溃。后面还写了个《快乐学数字》的 app 发到应用宝上，当然，这些程序是跟着书本写的，哈哈。而且还几十个人下载了，当时也是挺开心。然后又没有然后了，我又移情别恋了，，，，

五、步入正轨

说实话，大一真的折腾了很多东西，那时候学什么都充满干劲，虽然学的很多知识都是后面用不上的，但我觉得很值，因为我觉得，大一大二这段时间，千万别问别人干学啥学啥，如果你有感兴趣的，就去折腾，全心全力的折腾，真的能学到很多东西，而这些东西，远远不是那些具体的知识点，更多的是，你的**学习能力，折腾能力，逻辑思维**。

到了大二，我慢慢开始去关注一些公众号，慢慢关注一些大佬的学习，可以说，从大二开始，慢慢从**沉浸在自己的世界里走出来**，了解到了很多技术之外的东西，比如工作求职，比如公众号挣钱，等等。

在大二主要学习了《离散数学》，这本书可以让我们学到听说算法与数据结构的知识的，后面学了《Java编程思想》，不过我觉得这本书对新手不友好，看不懂，期间看了尚学堂Java300集，感觉讲的很好，哔哩哔哩可以搜索到。看了这个视频之后，我感觉自己的 Java 很牛逼了，懂了很多东西。接着再来看《Java编程思想》，我去，感觉这本书讲的太好了，推荐学习Java的看。反正从这个时候开始，我就决定学习 Java 体系了。

六、开始有目标着学习

对于 Java，我就觉得看了 尚学堂 300 集 + 《Java 编程思想》感觉自己掌握的差不多了，后面听说学习 Java 进阶得看虚拟机，于是就买了本《深入理解Java虚拟机》这本书看。之后感觉自己的Java挺强了，不过上面这些，对多线程的知识讲的不多，关于Java的多线程，这里推荐《Java并发编程艺术》和《Java并发实战》，感觉讲的不错。我也就看了上面这些书而已，当然，零零散散看了挺多其他书的，这里不列举，因为我觉得上面这些，差不多够了。

不过，语言只是一门工具，我觉得我们需要选择一门语言，并且深入学习它，这里我选择了 Java，不过这还不够，了解到大厂面试，非常看重计算机基础，于是自己学习了计算机网络，当时在哔哩哔哩看韩老师的《计算机网络原理》这个视频，感觉讲的很好（现在的话，推荐中科大的吧），之后跟着学校的课程学习了**操作系统**，看的是《操作系统：精髓与设计原理》，也学了《计算机组成原理》。如果你想学习计算机基础，我觉得这两门课都要学下，。

不过这些知识学了后由于很少用，容易忘记，不过我觉得这并不重要，因为学习的过程中，你会学到很多设计、算法思想等。到了大三，学校开了门计算机网络的课程，所以我再次学习了计算机网络，当时看的是《计算机网络：自顶向下》这本书，我觉得讲的很好，并没有看学校的教材。当然，期间也学习了数据库等知识，这些课程的掌握、推荐书籍等，我不说了，后面会在自己的网站+视频中详细介绍吧。

> 由于自己写的有点长，字数有点多了，所以接下来可能就写的简洁点了，好像有点有头没尾？哈哈

七、微信公众号的开通

在大三这一年，我开通了自己的公众号，并且也开始写文章，说实话，我开通公众号的目的有三个：

1、挣钱

2、通过技术博客给面试加分

3、训练自己的软能力

事实证明，公众号给我带来了很多收获，认识了很多人，也挣了些钱。不过说实话，弄公众号，真的花了我很多时间，写一篇文章需要几个小时，还要排版等各种，我觉得我要是不弄公众号，现在学习的东西，肯定多很多，技术肯定牛逼很多。在弄公众号的这一年里，虽然少学了很多东西，不过我并不后悔，因为我说了，**技术和成绩只是大学的一部分**，玩公众号，我也学了很多其他的东西，例如运营，也接触了这几年兴起的所谓的**知识付费**，很多人也考这个挣了不少钱。

并且通过公众号，我也实现了经济独立，再也不用向爸妈要钱了，当时真的挺开心，后来依靠自己的努力买了属于自己的一加 6T，Mac 笔记本，还给爸妈寄了几次钱。

玩公众号到现在一年3个月，大概挣了**七八万**吧，虽然这些钱不值一提，不过对于还没毕业的我来说，真的很开心。虽然在公众号花了不少时间，不过秋招还是凭借着自己扎实的基础找到了想去的**大厂**。并且靠着公众号，在没毕业前，实现了月入过万。对于公众号这方面的折腾，我觉得我可以再写一篇长文了，大家感兴趣的话，之后可以写一篇关于公众号的历程。

> 大家千万别看到公众号这么好挣钱就去玩公众号哈，很多东西看着容易，弄起来还是很不容易的，有多少人是写着写着就放弃的，我当时也中途放过，只能说，公众号还是给我带来了很多收获的，而不止是期间挣到了钱。

八、烂尾

感觉这篇文章要烂尾了，哈哈。核心写了自己的大一大二的折腾，一不小心就写了六千多字了，关于步入正轨的学习介绍的很少，主要是因为步入正轨之后，感觉没啥好说的，主要就是那些**计算机基础知识**的学习，如果要详细写的话，感觉要分三篇文章写，所以我选择了烂尾。后面再写吧，估计这篇近 7000 字的文章写了三四个小时，不知道有多少人是看完的，所以还是不写那么长了，不然就没人看了，嘻嘻。大学这三年，总结下来就是 **大一的折腾与探索，大二基础知识的积累，大三公众号的运营以及秋招面试的准备**。

当然，上面说的只是技术方面的学习与折腾点，还有很多人生的感悟没写，后面再来写了，虽然你们可能并不感兴趣。好吧，不知道有多少人是看到这里的，看到这里的绝对是真爱。

九、最后的最后

虽然自己的大学普普通通，平平凡凡，但自己还算满意，至少有了自己的公众号，没事可以来扯扯淡，找到看自己想去的公司，也有了一帮天天给我点赞的读者，哈哈哈哈。

未来，我们一起冲！

写作15个月，说一说这一路上的感受

之前写了一篇自己大学经历的文章普普通通，我的三年大学，里面说到了公众号，之后就有很多人问我公众号要怎么开通，怎么运营，各种问题。说实话，要是我详细跟你说，估计得话我很多时间，而且我的公众号也不算是运营的很好，但也不算特别差，未来会如何，也很难预测，我也不是多牛逼的人，所以也不大喜欢给别人各种建议，所以这篇文章，我就说说**从开通公众号的那一刻起，我是怎么运营、怎么平衡自己的时间的、怎么折腾的，学习到了什么，收获了什么，同时失去了什么**。然后供你们参考，相信看完，你一定有所收获！

一、写公众号的目的

我是去年暑假，也就是即将步入大三的暑假开始写博客的，当时我这人不习惯做笔记，高中，大学也基本没做笔记，突然去写博客来记录知识？？所以我写博客是带着目的的，因为有某种目的，才有驱动我去认真写的动力，但是目的很简单，就是听说**写博客，以后面试、简历可以加分**，所以我就在 csdn 写了，但是写了几篇，太他妈花时间了，而且阅读个位数，顿时写不下去了。

> 注意，我写博客有个毛病，就是不仅仅记录知识，更多的是要讲给别人听，让别人能听懂，所以一篇要写好久

那时我也有关注一些公众号，有一些号主再倡导我们写公众号，并且我发现公众号到达 5000 粉丝之后，可以开通底部广告卡片

有人点击的话，大概一个点击一块钱，然后我就想着，毕业后肯定能达到 5000 粉丝，那样每天叫一些人给我点点，读者也点点，那我一天不就有几十块钱了？于是有了一个新的驱动力：**5000 粉丝，然后开通底部广告卡片**，所以最初的话，写公众号有哪个目的：

1、面试加分

2、开通底部广告挣零花钱

> 所以，大家没事，偶尔给我的底部小卡片，多戳一下，嘻嘻。那时是曾经的巨大动力。不过，现在是 500 粉丝就可以开通了，呜呜。

二、困难重重（排版篇）

1、让人扎心的排版

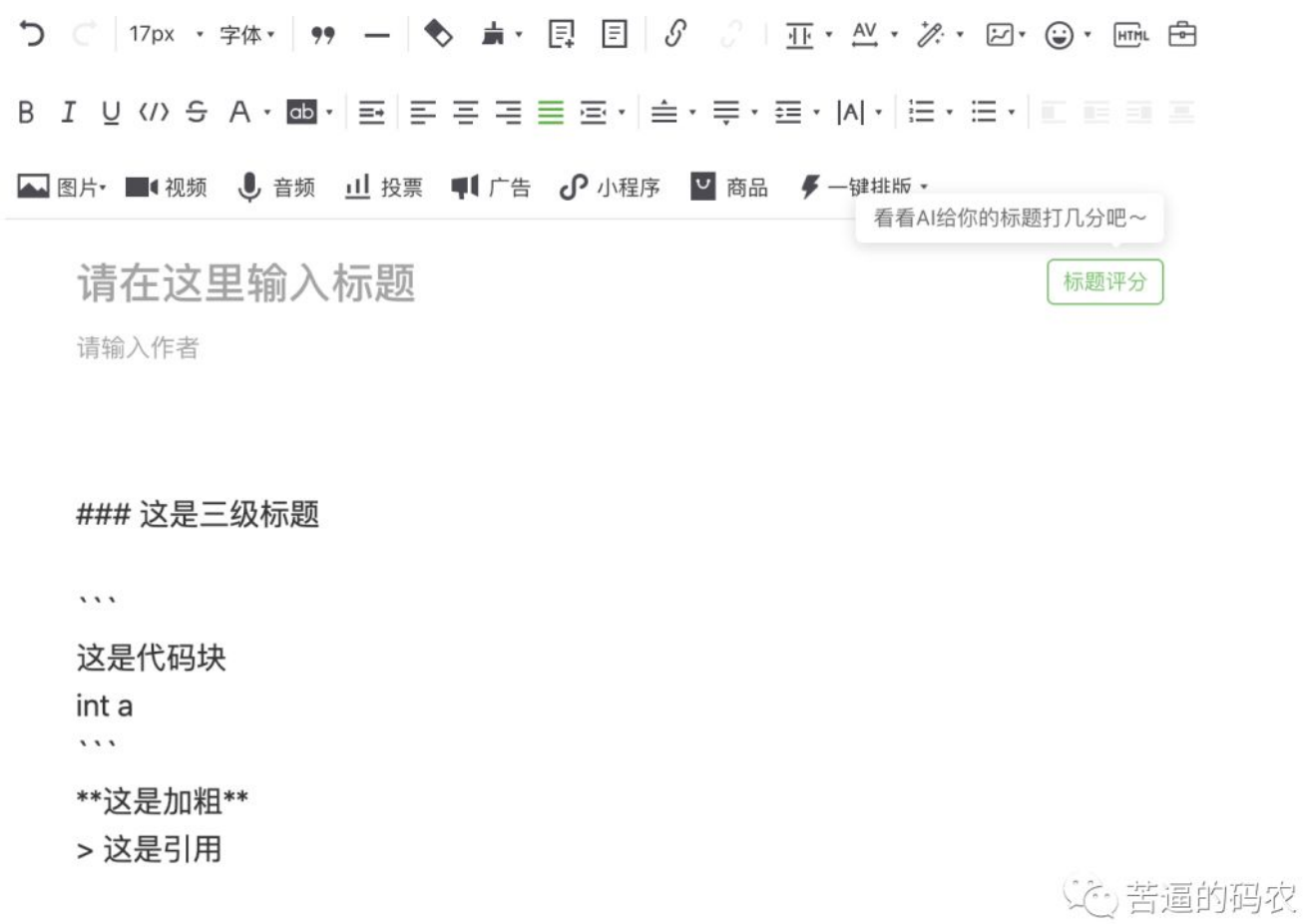
满怀信心写了一篇文章，大概写了 4 个小时，刚开始很难憋出字来，但是，我说第一篇文章排版用了 5 个小时，你相信吗?

因为在之前，公众号的后台编辑太不给力了，我折腾了好久，，，，刚开始我通过别人推荐是用**秀米**，然后感觉还行，把写好的文章排出来用了二三十分钟（排的过程中各种不懂），但是当时弄的花花绿绿的，就不大喜欢，于是听别人说用 markdown 语法写，之后再通过一些工具渲染。于是一边看着语法一边用 markdown 排版，排好之后，怎么渲染到公众号？？？

> 这里我先给写公众号的朋友一个建议，我强烈推荐大家使用 markdown 语法写的，现在大部分博客平台基本都支持 Markdown，而且很有笔记本也是，例如有道云笔记，印象笔记等等。所以有了一份，可以很方便复制粘贴到其他平台

关于怎么渲染的，我折腾了好久啊，跟着百度别人的介绍去找工具，但是不止为啥，我怎么就找不到渲染入口呢？后来搞了好久，听说谷歌浏览器有个 markdown 插件叫 **Markdown here**

只要你在公众号台点击一下这个插件，它就帮你渲染好，例如本来的如下：



点击 markdown here 插件一下，就会变成这样：

而且，这些 css 样式，你可以自己定制，反正我是自己折腾定制的。

我以为这下排版稳了，但是问题来了，由于公众号比较特殊，导致用 markdown here 的话，**图片**会渲染不过来，会丢失，这就很难受了，而且代码样式也不怎么好看，有时会乱。不过没办法，就自己手动调整以及粘贴图片吧。

所以，第一篇文章，真的是在排版折腾了好久啊。

2、找到心怡的排版

用过各种排版，期间的折腾就不写了，直接说我喜欢的排版以及推荐的排版工具吧。后来通过别人的介绍，用了 **md2all** 这个网页渲染工具，我去，太好了，代码样式好看，有多种选择，图片也不会丢失，也支持自己定制 css 样式。（PS：现在用的是 mdnice 了）

所以我就果断入坑了，不过这个工具也有一些问题，就是如何没有自己的图床的话，不能支持直接粘贴进去

不过这个问题不大，我平时文章在**掘金**写，写好之后在来 md2all 渲染，也会保存一份在**本地**。

名称	修改日期	大小	种类
md格式文章	2019年10月19日 上午11:08	--	文件夹
封面	2019年7月27日 上午11:35	--	文件夹
计算机基础	2019年8月4日 上午9:40	--	文件夹
不小心执行了rm -f.txt	2019年5月19日 下午12:33	2 KB	纯文本文稿
电脑的 ip 是...有配置过.md	2019年5月24日 下午10:07	7 KB	Markdown
关于三次握手...后悔系列.md	2019年4月25日 下午1:05	10 KB	Markdown
记一次面试：...死记硬背.md	2019年7月7日 下午10:51	6 KB	Markdown
数字签名是什么	2019年8月4日 上午9:40	5 KB	Markdown
我去，这两个...系列）(1).md	2019年7月13日 上午8:43	4 KB	Markdown
一句话撸完重...后悔系列.md	2019年4月25日 下午11:14	10 KB	Markdown
一文读懂一台...计算机的.md	2019年3月5日 上午10:14	13 KB	Markdown
经验	2019年10月16日 下午12:11	--	文件夹
【核心整理...么，怎么学?	2019年10月7日 上午8:52	16 KB	Markdown
历经两个月...+后台+腾讯）	2019年10月6日 上午9:30	17 KB	Markdown
嗯，春招两次...面试经历.md	2019年5月1日 下午12:54	21 KB	Markdown
秋招结束了...我的三年大学	2019年10月16日 下午12:11	20 KB	Markdown
送点福利	2019年9月26日 下午10:26	1	Markdown
玩公众号的经验.txt	2019年6月11日 上午11:45	3 KB	纯文本文稿

所以通过 md2all + 掘金，解决了排版问题（PS：现在无法用掘金作为图床了，自己用的腾讯有COS，不然一被禁止，图片就没了），但是公众号每篇文章都需要封面，并且封面的长宽有严格的比例，例如头条是 1：2.35，次条是 1：1，所以写完文章找封面也花了不少时间，后来通过**壹伴**这个插件解决了寻找封面的问题

所以当时我的写作 + 排版是采用**掘金 + md2all + 壹伴**，然后具体想怎么调样式，例如字体大小啊，间距啊，行距啊各种，就看你自己的爱好了，反正我认为：**简洁，是最美的**。最开始我也弄个很多骚的排版。

> PS：现在就用 mdnice 这一个吧，自己在本地typora写，用Mdnice渲染。

三、我放弃了

坚持难吗？不是很难；一直坚持难吗？说实话，太难了；一直没有正反馈的坚持难吗？卧槽，这太他妈难了。

从写一篇文章到发布，对我来说，至少三四个小时，多则五个小时，这还没有算上平时找素材的时间，**因为我想把最好的带给大家**，看过我的文章应该能感受到，我写的还是很认真的，也写的挺不错，被挺多号转载。

我**花三四个小时写一篇文章，而且十几个阅读量，根本没啥人读**，内心还是挺难受的，因为公众号是个封闭的平台，不像其他博客平台。然后我想，这三四个小时，我看书的话，感觉可以看半本书了，至少几十页吧。然后我在想，我这样写下去值不值得？？

> 可能很多人把文章发到自己的朋友圈还是能获得一百多阅读量，但是我坚持不发，因为我不想让身边的朋友知道我在写公众号，只有当我的公众号粉丝多起来，量起来了，能挣钱了，我再让他们知道。

然后写了大概四五篇，我放弃了，放弃了之后，感觉特别轻松，把多出来的时间学学习，玩玩游戏，看下视频，**舒服**~。

> 相信也有挺多写着写着就放弃的小伙伴，这一点也不丢人，毕竟坚持、自律太他妈难了，更何况没什么正向反馈的。

四、重拾笔墨

当时我也有在微信认识一些一起玩的伙伴，有的和我一样放弃了，有的依然坚持，我放弃一个月后，看到那些还在坚持的伙伴，好像有些搞的还不错，有一两百阅读量了。唉，**能坚持的总是少部分人啊**。

然后我就带着坚定的心情，再次来写文章了，并且一边劝自己，写文章就算没有人看，还是可以锻炼自己的一些技术之外的能力的，然后就坚持了几个星期，当时加了一些群，我把文章发群里，发现**每次写一篇文章，能涨 10 个左右的粉丝**，而且评价还不错，感觉收到了一些正反馈，当时还是挺开心的。后来认识了挺多和我一样写公众号的博主，感觉不在那么孤独。

1、学习和写文章的时间分配

到了九月，就开学了，也没有那么多时间写文章了，但是我当时不知道哪来的激情，可能是因为每次写一篇文章发到几个群里能涨 10 个粉丝，哈哈。

然后当时是打算每天写一篇，，，，但是一篇要花三四个小时，然后我在想怎么挤出时间，不过我发现我这人有个毛病，就是晚上到了 11 点，就不想学习的了，想玩学习之外的，然后就打算每晚十一点之后写文章，我有个特点，就是每次一篇文章一定要一次性写完，不然很难受，于是那阵子每晚写到一两点，不过感觉挺有激情。就是平时可能懒的写文章，但是一写就会很来劲，而且感觉时间过的特别快。

不过后来一天一篇顶不住，而且每天早上睡的很晚，可以说早课基本不去上的了，因为比较晚睡，起的很晚。后来我改变计划，每周写三四篇吧。至于时间的分配，一般是晚上写，平时该学啥还是学啥，但是说实话，这个写文章，还是让我花了不少时间，少看了很多书。

2、开始多平台引流

我发现有些人是多平台同步文章的，于是我想，我用 Markdown 写的，其他也支持这个样式，于是我也多平台，当时一股气注册了至少七八个，包括：开源中国，掘金，博客园，csdn，知乎，segment，51cto 等等。把文章同步过去，文末介绍下我的公众号。

但是，好像大部分平台的个位数阅读量，更新了十几篇我就放弃了，更新这么多，说实话，也挺累的。最后只留下了掘金、博客园、知乎、csdn。不过也引流不了多少，估计一天几个都没有，但是有些文章会突然 100+ 左右的阅读，就勉强更新了。可见有多苦逼！

五、开始真正发育

从 7 月玩，到了九月中旬，大概我也就只有 300 多粉丝，期间付出了多少，你们懂的，不过，**努力不一定成功，但成功总少不了努力**。注意，下面我要开始开挂了哈

1、投稿给我带来了第一批粉丝

后来我发现可以投稿，有些公众号接收投稿，于是我开启了投稿之路，但是，不是说你去投人家就愿意发，很多情况下人家根本不会发，例如：

1、你写的文章太冷门了，别人发了估计阅读量会很低，别人肯定不愿意发，因为对于很多大号，阅读量还是很重要的。

2、你投的文章别人已经发过好多类似的了，这时他可能不会接收你的。

3、你写的文章，和它的公众号定位不同，例如人家是写 Java 的，你去投 Python 的文章。

反正，我投了很多，一直期待他们发，然而，大家都没法，投稿路上还是挺困难的，**因为没有任何人来指导你，一直以来，都是自己在探索**。再说，要找到合适的公众号，并且它愿意接收投稿的，也很难找啊。

后来我觉得，要获取别人的转载，我得写一些**别人愿意转载**的文章，当时漫话技术文很流行，于是我就花很多时间，从找漫话人物，怎么排版漫话等等，写了第一篇漫话技术问：【漫话】什么是外部排序？

这篇文章，自我感觉写的挺好，然后去投稿，由于是算法类型的，比较通用，好几个公众号转载了，并且，好几个公众号主动来找我开白名单了。那篇文章，给我带来了差不多 1000 个粉丝。让我尝到了甜头。

> 关于投稿：当时数据结构于算法类我主要投的公众号：数据结构于算法、算法爱好者、码农有道、程序人生、CSDN。

后面我就以算法类文章为主，开始写漫话，例如写了平衡树、https 等，这些文章，由于上一篇文章评价不错，所以去投稿都被接收了，从 9 月多 10 月多，我的公众号从 300 多粉到 3000 多粉。但是我不得不吹一些，我写的这些文章，评价、点赞都很高！

再后来，我没去投稿了，有些公众号我给他们开了白名单，他们主动来转载。

这里我必须说一写，对于没人带的小号，我觉得定位太重要了，不然你去写 php，C++，C，嵌入式等，肯定很难发育，可以投稿的公众号也很少的。所以我的定位是**算法**、**Java**、**计算机底层知识**。

2、投稿的瓶颈

说实话，可以投稿的公众号感觉并不怎么多，有些个人号，它也不会一直转你的，有些转多了，该来的粉丝都来的，之后就涨粉特别少，从最开始一篇涨 200 多，到最后涨几十个，甚至十几个。也就是说，**我投稿遇到瓶颈了**。

3、折腾其他平台

后来我就开始折腾其他平台，研究他们的一些机制，例如**博客园**，我发现可以发首页，并且有**24小时阅读量排行榜**，**48小时排行榜**，点赞榜，评论榜。但是，人人可以发首页，并且首页的文章排序是按照时间来排序的，所以在 24 小时、48小时上榜很重要。否则你的文章就排在很后，根本没人看到。

然后我发现，上午看文的人多，如果文章标题不错，应该有机会上榜，于是我找一些不错的文章同步了，然后好几篇上榜了，有些还被编辑推荐了。我去，又看到了一个引流的不错平台。

还研究了知乎，掘金，csdn，大家也可以去研究研究，主要就是研究一些他们的一些推荐算法机制，编辑人员喜欢啥文章之类的，这里我就不说太多了，我觉得这几个平台的研究经历都可以写个万字长文了。例如之前还研究 csdn 刷量啥的，博客专家啥的。最近几天 csdn 倒是给我带了不少粉丝

我之前已经放弃过 csdn 的了，也一直吐槽，发现 csdn 改版了，变友好了，哈哈。然后最近研究了一下，然后发了几篇，都上推荐了，不过说个实话，根据我看了好几个博主被推荐博主的文章：**发之前我有预感会被推荐**。

4、总结

所以对于小号，我觉得最重要的就是**内容**，然后去投稿，因为很多大号都是靠转载的，接着就是多平台同步，并且不是简单的同步，还要去研究一些机制。

并且，文章内容定位也非常非常重要，写出通俗易懂的文章也非常重要，有时你感觉自己写的挺好，然而大家可能并没有看懂你写的，，，，，所以可以多观察别人是怎么写的。等粉丝量、阅读量有了一定的起色，可以参加一些**互推**，就是几个公众号之间的相互推荐，相当于资源互换。

六、这一年来的收获

1、挣钱

虽然各种折腾花了很多时间，但是我的公众号运营的也还不错，我三千多粉丝那会，靠着公众号挣到了第一笔钱：300 元，当时是真的很开心，太开心，靠自己写文章也能挣钱，然后那个月，大概挣了 900 元，解决了自己的生活费，再也不用向家里要钱的，真的很兴奋。这算是很大的正反馈，也让我更加有动力了，**从写作到今天，我算是还没毕业前，靠自己的折腾，挣了人生的第一个 10 万**。但是我不得不说，这些钱，挣的真心不容易，你们也看到了，这些离不开几个月坚持、挣扎，而且也有好多写了几个月放弃了。

后面我的粉丝越来越多了，挣的钱也越来越多，所以有时会接一些广告，不过这些广告都是通过自己的筛选的，觉得不会乱接。能够挣钱，也是我一直更文的最大动力来源，所以呢，希望读者能够多多支持，我也会多写一些优质的文章供大家阅读。

2、扩大了人脉

写公众号以来，也让我认识了很多人，包括一些大佬。而且我发现，好多写公众号的，本身的工资也都好高，然而他们比我还努力，这里真的不得不说，**比我们优秀的人，比我们还努力**，所以大家也别一直在吐槽环境啥啥了，什么工作多艰难了，先看看自己平时都在干嘛，所以我觉得，年轻人，还是要努力一点，脚踏实地，先把技术学好，技术是我们的立足之本，接着在分配点时间学习别的。

> 不过我这里不得不说一句，就是你认识了一个比你牛逼很多的大佬也是没啥用处的，因为很多时候，更多的是一种资源的交换，如果你们的差距太大，那么很难产生资源交换的，进而您们的交集也是很少的。当然，感情相好的除外。

3、拓展了自己的思路

说实话，写作一年以来，基本是自己从 0 开始，没有任何人带过我，自己一点点折腾过来，其中付出，可能只有搞过的人才会明白。这让我学习到了很多**技术之外的东西**，有人可能会问，那你说说技术之外的东西是啥，说实话，我也不好三言两语描述。例如，当我去一个陌生的网站时，我可能回想，这个网站主要通过什么盈利，如果我来这个网站玩，怎么样才能引流，推荐机制啥的各种.....。

总之，折腾了一年多，投入了挺多时间，不得不说，这个事情给我的人生带来了很大的变化，也让我对未来更加有信心。

> 很多人说我大四就这么厉害啊，说自己都工作几年了还.....，或者也和我同样大四了，还连秋招都不知道.....其实不是我多厉害，而是我行动了,也不是你不行，是你没有行动，没有坚持。行动和坚持不一定能行，但是不行动不坚持，一定不行。可能有些人觉得这是鸡汤，但是，对于绝大数的人来说，这确实是事实。

最后

其实本来要说的有很多的，不过一不小心，就写了这么多字了，今天手机玩了很久，打了太多的字了，手指干嘛打字有点不舒服了，所以就先写这么多了。其实想要说的话还有很多，不过后面再和大家分享了。

不过上面所说的，是我的一些经历以及见解，也并不一定是对的，所以供大家参考，不过我必须要说的是，**如果你是学技术的，那一定要把学技术放在第一位，因为技术，才是我们目前的立足之本**，我写作这个，算是副业吧。最后希望大家持续关注我勒，**你的一举一动，我都看在眼里。记在心里**！

如何让自己变的更加优秀?

这是 2020 年公众号的第一篇原创，文章的标题本来是要命名为『如何让自己变的更加优秀』，这看起来有点像鸡汤文?

说实话，我这个人基本不写鸡汤文，公众号发文几百篇，基本没有写过一篇鸡汤文，更多的是谈自己的**真实经历**，**真实感受**，所以有些文章可能在你们看来是鸡汤文，但是却是真实发生在自己的身上，不然我是不写的。形如这些文章：

1、我建议你这样度过自己的大学
2、给大学生的几点建议等等。

这种文章说的话，基本属于通用文，对谁都合适，但是又对谁都不合适。

今天这篇文章，**是有感而发**。最近不是年终了吗？这两天看了挺多人的年终总结，觉得有必要写一篇鸡汤文，同时也把这篇鸡汤文送给 2020 年的自己。

那么如何让自己变的更加优秀?

2019 年给我的感受就是：**多接触优秀的人，多接触正能量的人，远离负能量的圈子**。

0、多接触优秀的人，真的可以影响你

说实话，我玩公众号有个非常大的收获，就是认识了一群优秀的人，他们的优秀，让我也逐渐变的优秀起来（当然，我并没有说自己多优秀，只是觉得自己比以前优秀了）。

以前，我从来没觉得自己能够写文章挣钱，但是后面我认识了一群人，说实话，他们并没有多优秀，也不算聪明，他们也是普普通通，不过我们都在一个群里，后来他们开始玩公众号，那时候我才大二，不过把他们的公众号都关注了，等到大二暑假那会，发现他们的公众号玩的不错，还听到他们说挣了一些钱。

我可能比较势利，听到**钱**，我突然也想着自己是否也能像他们一样，也来去玩一玩，挣几个零花钱，想着他们可以，我觉得我可能也是可以的。于是，我走上了公众号这条路，刚开始是被他们远远甩在乡村路口，不过我时刻关注他们，他们的一举一动也经常激励着我，我也在偷偷模仿他们。

现在，我之前关注的那批人，都取得了不错的战绩，而我也因为玩公众号，收获了非常多。他们对我最大的影响估计就是：**看到他们可以，我觉得我可能也可以；看到他们再折腾，我也有了折腾的目标**。

也就是说，这种优秀的圈子，并不是每个人多聪明，每个人都非常牛逼，相反，是大家的背景都挺普通，大家都一直在探索，也会带动我去探索。不得不说，在一个正能量的圈子，都的很重要。

例如，本来今天想要赖床不想去练车的，自己有种在逃避的感觉。然而在知识星球看到大家的年终总结，顿时浑身充满了能量，觉得自己不能这样逃避下去了，马上起床，早上开着摩托车跑去练车了，练完车回来，感觉特别舒服。

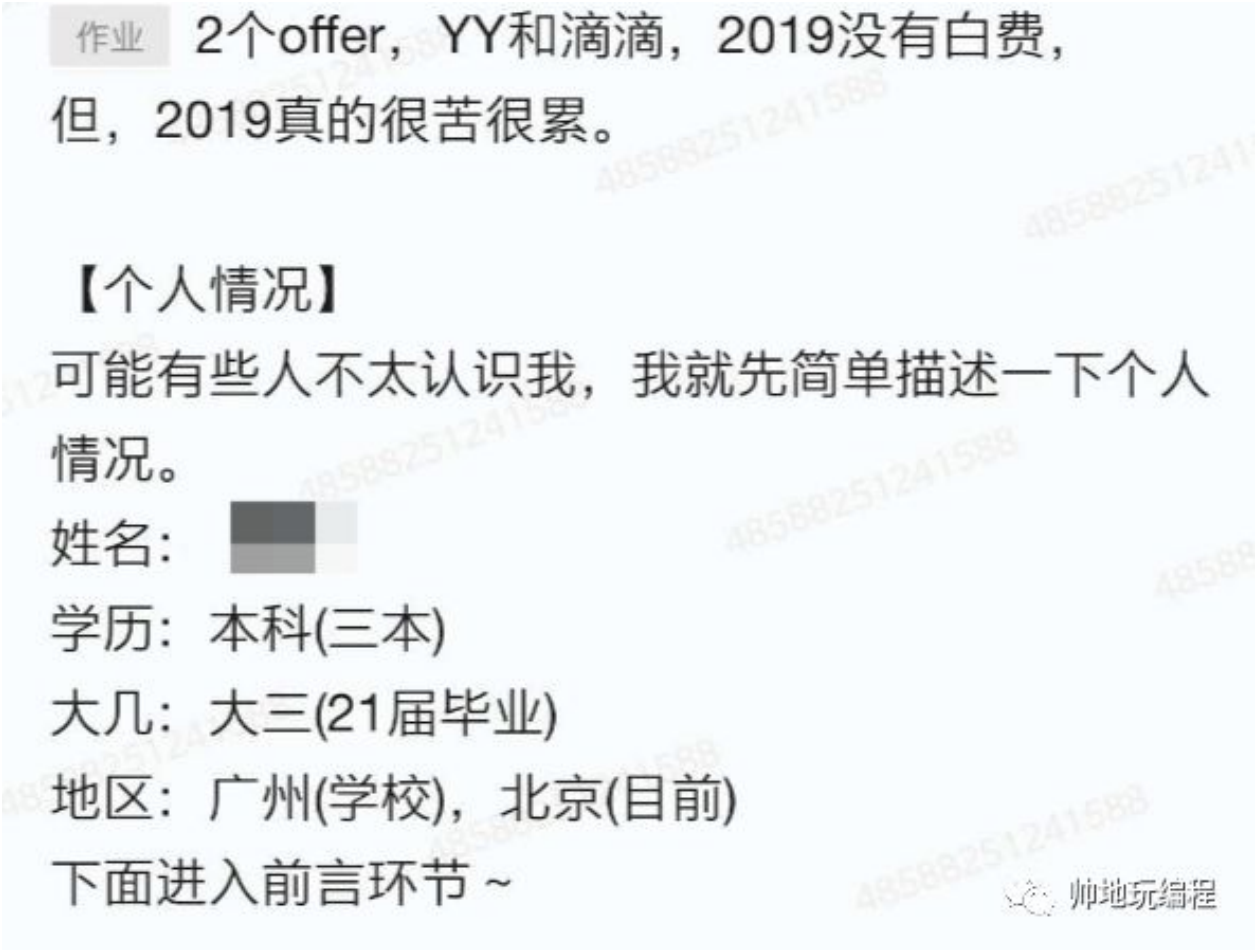
1、正能量的圈子，真的可以大大的激励你

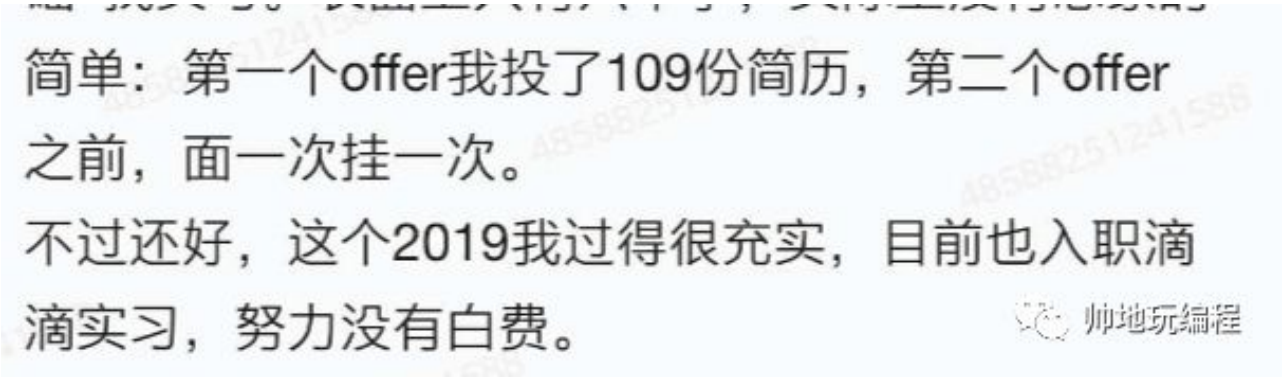
说实话，我不大喜欢负能量的人，负能量的圈子。例如有些人天天哭天喊地，天天说生活太难了，天天抱怨生活等等。你去认真做一件事，他们可能还会嘲笑你，搞的自己本来想认真去干的，结果在他们的影响下，自己也敷衍了事得了。我说的这些，觉得不是鸡汤，自己也确实发生过这样的事情。

前阵子我写了一一篇文章普普通通，我的三年大学。这篇文章激励的很多人，也有挺多人来加我，说自己不能再这样下去了，说自己也想进大厂，然后也开始努力学习了，有些人还经常来问我推荐书籍，看到我的文章激励了别人，我还是挺开心的，因为确实，很多人在大学都在玩，就算你现在的奋战之后没有进入理想的公司，那么比起以前，你也是更加优秀了，学到更多了。

我举个例子，如果你是三本院校的（这里不是看不起三本院校之类的哈，因为确实存在这样的情况），你想要认真学习，想要进大厂，可能你的同学会笑笑不说话，你可能会孤身一人在奋战。但是说实话，三本，二本进大厂的，还真的不少，**实际比例可能挺少，但是我在身边的圈子，真的见过不少**，并且还有挺多人是在同一个圈子的，大家都很正能量，也都失败过很多次，但是你看看我，我看看你，并且也有挺多过来人在鼓励他们，大家一下子又充满能力，继续干了。

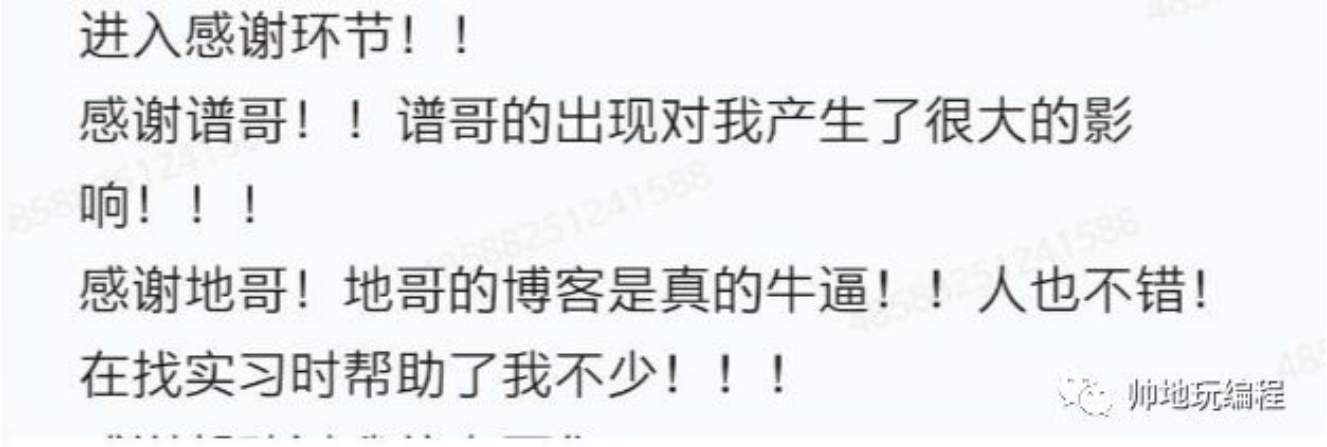
例如，我昨天在一个知识星球，就看到一个三本的大三的年终总结了，这个人我也经常有和他聊天，偶尔鼓励他，最后取得了不错的战绩，截图给大家看





如果你是三本的，本来你也有一颗奋斗的心，但是如果你身边好多负能量的，可能你会觉得进大厂太难了而放弃。不过如果你周围有挺多优秀且正能量的人，你可能并不会放弃，而且会充满能力，一直干下去，并且后面还有曾经奋斗过的人在鼓励你。那么你，觉得可能比之前更加优秀。

我觉得，这位同学能够拿到大厂 offer，真的离不开这个正能量的圈子， 当然，最重要的，还是他自己的努力，感觉自己有希望实现自己的目标，学习起来真的会很香。他投了 109 份简历，可能有些人 投了几十分简历被刷，估计都在**奉劝**自己的学弟学妹说大公司多难多难，我们三本学历 0 机会的了。最后这位同学也夸了我一下，也截个图



大家面试的时候，记得多看我汇总的文章，我比较**势利**，很多真的都是面试有考到。

在这样一个正能量的圈子里，和一群人优秀的人在一起，自己真的也会变优秀起来。

2、我比以前更加敢于尝试了

我举个例子：如果在以前，你说关于**理财**、**保险**相关的，我可能会给你一个**白眼**，因为我没钱，学习这东西也没啥用，而且感觉这东西害人。但是现在不同了，看到大家有时候都在尝试这东西，而且大家也都不是很有钱，更多的是在**学习**，而不是想着挣钱，算是为自己以后做准备，顺便探索下这方面的水。

但是现在的我，业余时间也会学习学习，因为我的观念变了，有了更多自己的想法，感觉接触下这东西或许以后对我有帮助。而且我可以告诉你，一旦你去认真接触一样东西，你会学习到很多其他知识。

> 当然，我这里并不是推荐大家去学，而是举一个发生在自己身上的例子

这样子说吧，如果你周围的人都在尝试、折腾各种东西，你本来没有什么 idea 的，但是看到大家都在尝试这，尝试那，你也会慢慢有自己的 idea。

3、那么如何找到这样的圈子?

那么问题来了，道理我都懂，但是我身边很难找到这样的圈子，咋办?

可能有些人觉得接下来我会打广告，介绍某个圈子。然而我想告诉你的是，这样的圈子哪有那么容易找。。。。。。

不过我想说的就是，这样的圈子，周围往往很难找，更多的是存在于网上，因为这种圈子往往就是由一批志同道合的人组合起来的。所以需要你自己去探索，去寻找适合你的圈子。

当然，我也想过建议一个知识星球，把大家汇聚起来，不过，我觉得目前的我，还不行，目前的我还太菜了，等我以后强大点了，再来创建吧。

4、想删文的总结

这应该是我第一次写这种类型的文章，昨晚本来有好多话想和大家说的，不过写到文章发现好难写啊，本来想要写的有感染力点的，发现写不出来。搞的不想把这篇文章发出去，，，，，

写这篇文章可以说是很走心的吧，主要就是想要告诉你们：

1、要相信自己、多去尝试，或许你真的可以的。

2、多接触优秀的人，远离负能量的人，或许，你真的可以变的更加优秀。

如果你觉得你身边没有这样的人，那么请记住，还有我，你们的博主-帅地。我觉得我还是挺正能量滴，当然，如果你有加我微信好友，你会发现我的朋友圈发的一点也不正能力，，，，可能是，我比较热爱生活，朋友圈更多的是反应我的真实生活，平平凡凡，普普通通。

不过后面可能会创建一些微信群，把有志同道合的人聚集在一起。之前、现在也有挺多人叫我创建的，不过我还不知道怎么维护好这些群，，，，，所以一直迟迟没有建。后面创建了，再拉筛选那些想要让自己变的更加优秀的人进去，**相信一群志同道合的人在一起，真的可以让自己变的更加优秀**！

毕业了，大学这四年，说几点感受

2020.6.6，应该是帅地最后一次以学生的身份进入学校的一天了，以后再次回学校，便是以**校友**的身份了。

是的，帅地毕业了，在这四年大学里，可能和绝大部分学生一样，每天因为早课太困而日常逃课；上课趴桌子上睡觉，还特么每次都坐第一排；给老师面子假装认真听课；期末临时抱佛脚，一个星期学完一个学期的课……

还没有毕业的时候，帅地就已经在构思毕业那天要写一篇什么样的文章了：在毕业的那一天，我一定要写一篇万字长文，把我自己的大学经历、心得体会毫无保留着展示一遍，给各位还没有毕业的小伙伴作一个参考。不过，我发现我已经写过大学经历了：

普普通通，我的四年大学

所以今天，就不写我大学详细的学习经历了。对于还没有本科毕业的同学来说，我算是你们的学长，作为你们的学长，今天就说几句真心话供学弟学妹们作为一个参考吧，当然，居然是真心话，更多的会是我自己的**主观想法**，可能你听着会不舒服也不一定，但没办法，真心话就是这样。

说实话，我也不是多么牛逼的人，和大家一样，一枚普普通通的本科生，所以我说的事，不是多么高大上的事，更多的是一些**鸡毛蒜**大小的事勒。

学习知识永远是首要任务

基本每一个读大学的人都会经历过高考，高考有多辛苦，大家也应该有所体会，在高考的时候，是多么的向往大学生活，因为大家都跟你说，到了大学，你就自由了，你就可以做你自己喜欢做的事了.....

然而事实还真是这样，很多人到了大学，还真的**放飞自我**了，熬夜，逃课，开黑，各种社团活动.....这特么也太忙了吧，都没时间学习，只能挤出一点时间来看一看课本，抄一抄作业。

说实话，居然我们是学生，那么我觉得，无论在哪里，**学习知识，必须我们的首要任务**，其他的，都不是必选项，也就是说，其他的，你也可以玩，例如参加下社团活动，宿舍熬夜开黑，刷剧，这些完全是可以的，我当年也是这么过来的，但是，**要适当**，这些都是我们的**副业**；**多学习多看书，才是我们的主业**，切勿把主次颠倒了，当然，学习绝不仅仅是指学习**专业知识**，可以学可以看的书实在太多了，但是，学习与**专业有关**的知识却是我们在学习中的首要任务。

之所以说这个事，是因为，确实有挺多人把**主次**颠倒了，反正我的微信经常收到这样的信息：地哥，我大学有点废了，因为 xxxxx，所以 xxxx，接下来我该怎么学啊？有没有两三个月就把这些 xxxxx 学完的方法啊?

所以我希望关注了我的读者，在大学，能够好好学习，毕业之后，才能拥有更多选择的权利。

迷茫时其实也可以看看传记

说实话，大一大二那会，除了东野圭吾的悬疑小说以及专业书之后，我看过最多的书，应该就是**人物传记**了。我相信绝大部分人都会有迷茫的时刻，当然，也包括我。在我有点**迷茫**或者想**寻求支持**的时候，我觉得看人物传记，能帮助我的内心安静下来，能够给我带来一些启发或者动力。

因为我在学校，基本都是自学的，绩点差的一批，学校的书基本也没看，都是自己在干自己的事，有时候就会怀疑自己：这样真的可以吗？导致内心有多乱，或者突然就没了动力，不过我每次看了大佬们的传记之后，例如乔布斯啊，李开复啊等人物传记之后，又莫名其妙有动力了，而且感觉自己会变的更加有自己的想法。

所以呢，在你有点迷茫或者没动力或者想要寻求支持的时候，不妨可以找本传记看看。

说实话，我并不支持大家去做兼职

对于做兼职这个事，我说说我的**主观看法**吧，居然是主观想法，你们看看就好。

可能大部分在大学都有做过兼职，但是，如果你做的工作，是无法**辅助你的主业，对你的个人成长帮助不怎么大的**，那么我是不建议你去做兼职的，例如发海报、送水果、监考、家教等。

因为在我看来，学习才是最主要的任务，有人可能会问：帅地，天天学习学习，这特么还有大学体验吗？

那必须有的，如果你是想体验一下兼职的感觉，那必须可以去体验，多去体验各种工作，我还是挺支持的，但是如果你是为了挣钱，那我一点也不支持。实不相瞒，我大学没有做过兼职，我估计把做兼职的时间花在网上冲浪上了，所以我很早就了解清楚了春秋招，了解了一些大厂招聘时对校招生的要求。

有人可能会说，没办法，没钱，穷，只能压榨自己的时间去做兼职，来维持下自己的生活费用，不然谁愿意去做啊，辣么辛苦。

我知道很多人去做兼职，是真的为了挣钱，为了尽量少向父母要钱，因为家里确实比较穷。但是，我还是不支持你去做兼职，除非你做的兼职是对你的成长比较有帮助的。

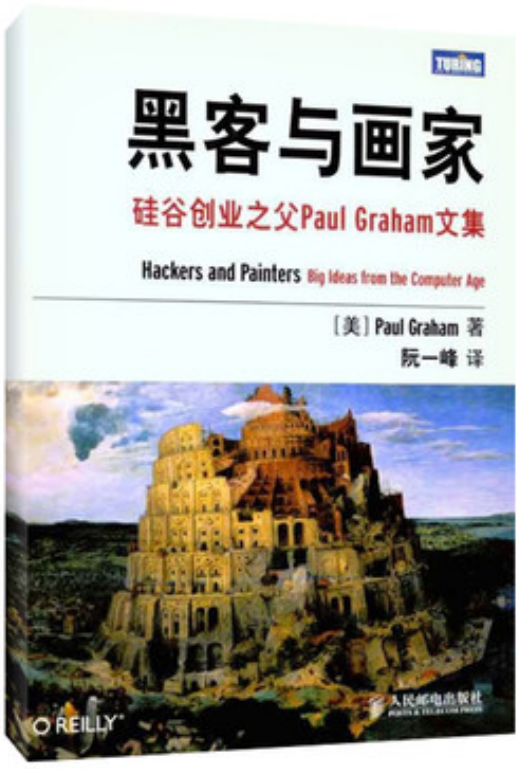
当然，我平时在学校，同学为了挣钱去做廉价劳动力的兼职，讲道理我是要说一说大道理，让他不要去做的，毕竟有些同学利用午休时间去做兼职，还是挺累的。然而我并没有去劝说，我怕被别人给我来个**反问句**，例如：家里没钱啊，没办法，不像你xxxx。

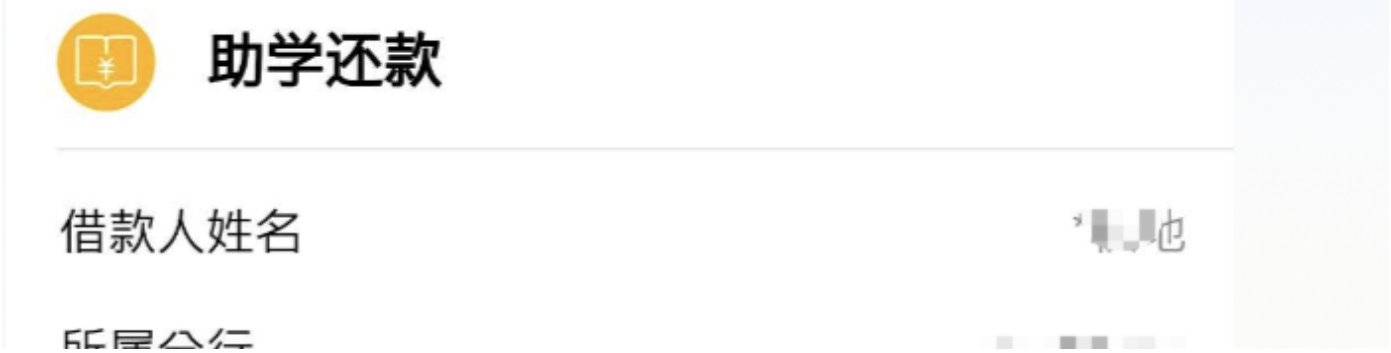
说出来你们可能不信，我家里比你们更穷，我四年大学全部贷款的，家里欠债十几万，有时候在家，父母还会吵架，这让我特么难过，因为在我看来，只要家庭和谐，其他都不是事，就算欠债百万也不慌，以后我慢慢还就是了，然而事实并非这样，有时候经济不好，就会容易出现矛盾，反正每次听到我爸们因为钱的事吵架，我是真心难过。

难过归难过，我还是照旧向家里人要钱，没钱就只能借了，我家是种务农的，有时候忙了好几个月，暴风雨一来，或者市场价格一变动，呵呵，几个月白忙了，忙了几个月还特么卖不回本钱。我家还借过几次高利贷，没办法，因为穷。

好吧，暴露了一波自己的家境，，，我可不是来求同情的话，我只是想跟大家说，**如果你是为了挣钱，并且做的工作对你的未来没什么提升，那么我不建议你去做兼职**，关于挣钱这种事，其实我也了解过一些，并且我觉得这些对你的未来成长是有帮助的，当然，对成长有帮助的，前期往往挣不到钱，往往需要积累，不然一来就可以上手的话，那就是廉价劳动力了。后面有时候我来写一篇对应的文章吧。

当然，很幸运，我虽然在大学没有做过兼职，但由于我自己在大一大二的积累，通过写作，通过输出自己的专业知识，让我挣的钱比兼职多的多的多，前几天我就把自己四年的贷款的三万多元一次性还了





说实话，能够在毕业前一次性还完，还是非常开心的。不过关于写作挣钱这一块，后面有机会在写一两篇文章说说，或许有些人也挺感兴趣滴。

热爱生活并享受生活

如果你打开我的朋友圈，可能会看到这几个字：**热爱生活，并享受生活带来的一切，无论喜怒哀乐**。

是的，我觉得无论处于什么阶段，生活很重要，未来我可能无法变的很牛逼，但简单、开心的生活，我觉得很有必要。实不相瞒，自由、保持好心情、乐观，从初中开始，这些关键词就一直伴随着我了，记得我初一时还建了一个 QQ 群，群名就叫**自由/快乐/开心first**。

当然，这种心态会受生活的各种因素影响，不是说自己想开心就能开心的，这是一个需要时间来慢慢培养的过程吧。我也希望你们无论遇到什么事，都不要太焦虑，享受生活带来的一切吧，然后努力着从物质、精神上提升自己的生活。

其实想对大家说的话，还有挺多，不过文章已经三千多字了，有些话，留着以后说吧，当然，我的这些话，对挺多人来说，可能也是**废话**了，但只要能给部分人带来帮助或者思路，那便足矣！

献上一波自演自拍的毕业照吧

今年的毕业季，不像往年那么热闹，并没有亲朋好友来一起和你拍照，也没有穿上学士服来拍个集体照，不过运气也还算好，昨天广州上午刚好没下雨，让我和几个同学有机会拍了几张照片，学校也给我们定制了一件T恤衫。

本来说要理发完再去拍的，然后学校也太特么严格了，进了学校就不能出去了，出去了就不能进来了，难顶，不能成为这条街最靓的仔了

1、学校在宿舍门口挂了一副图像，刚好可以当作背景

2、丁颖教授，一位对社会贡献颇多的人物

3、难顶，没有毕业服，只能将就当作是自己的毕业服了

4、拥有百年历史的建筑，不过最近刚翻新了一下

5、华南农业大学面积是真的大，一片绿

6、最后献上一张被朋友圈 n 人吐槽骚气的照片（够大方了吧？）

其实学校里面还有很多地点没去拍，听说女生可以在同一个地点拍上百张照片，，然而我们几位男生拍了几张照片就找各种借口不去拍了，最后回去还被雨淋成了狗,,,

最后

我们这一届，注定是不平凡的一届了。

2004年读小学，那年正好小学的教材改版了。

2013年中考，那年刚好数英语分数从120改成150。

2016年高考，那年正好高考试卷从广东卷改成全国卷。

2020年大学毕业，这一年，刚好赶上了云毕业。

最后祝自己毕业快乐勒

毕业半年，人生的第一个100万

大家好，我是帅地。

说起来我已经毕业一年了，但可能比绝大多数毕业一年的人都要经历的多，接触的多，其实在我年初的时候，我就挣到了人生的第一个一百万了，那会还在朋友圈截图发了年支出。

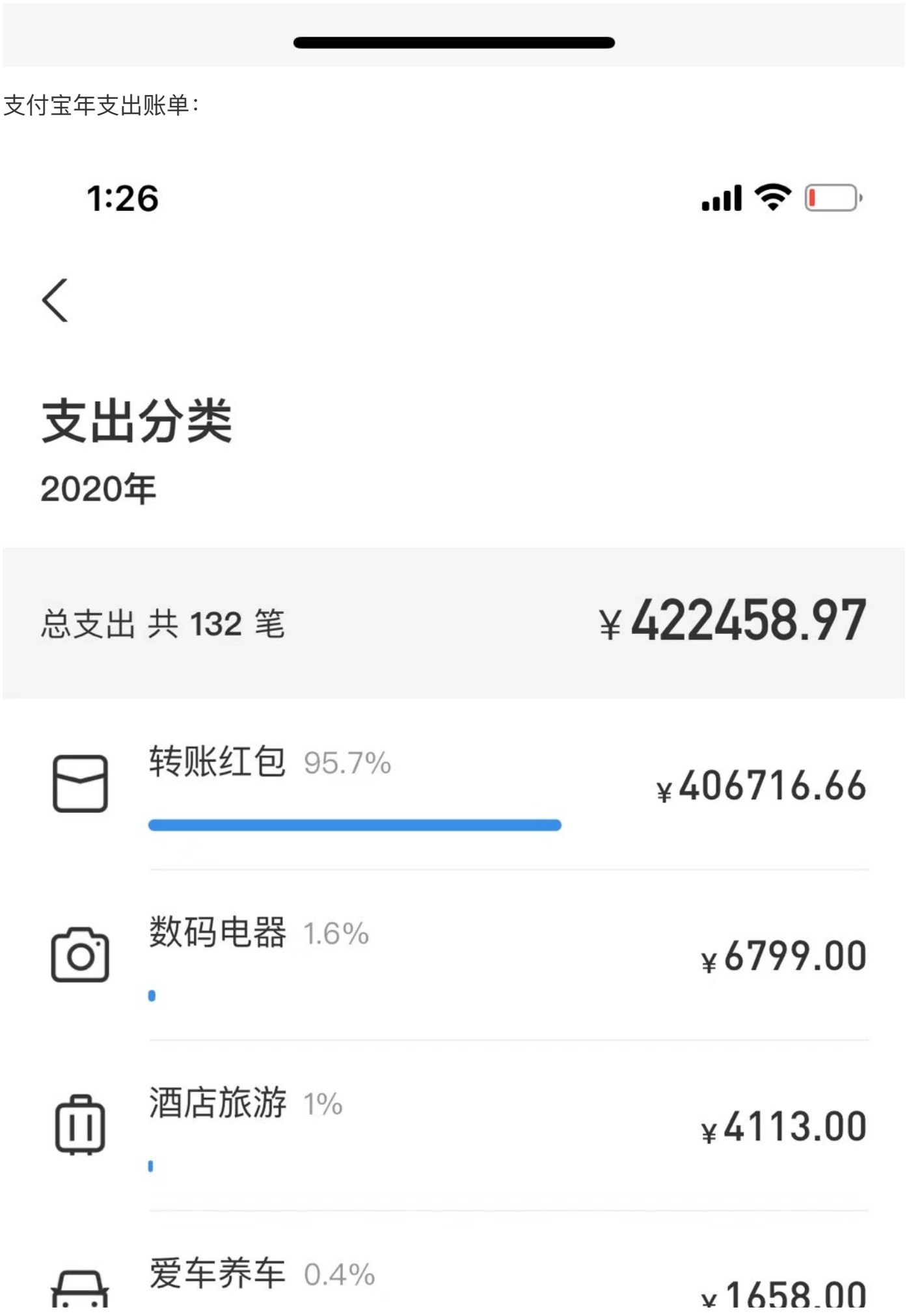
微信年支出账单：

¥510423.38

支出构成

转账		¥429425.00
消费支出		¥73641.86
发红包		¥4104.93
支付群收款		¥3238.27
赞赏		¥13.32

支付宝年支出账单：


1:26
支出分类
2020年
总支出 共 132 笔
¥422458.97
转账红包 95.7%
¥406716.66
数码电器 1.6%
¥6799.00
酒店旅游 1%
¥4113.00
爱车养车 0.4%
¥1658.00

那会刚好距离 2020 年 6 月份毕业半年，当然，不是我在这半年挣了一百万，而是我从大三到毕业半年这期间挣了一百万，年账单支出这么多，是因为我完成了毕业后的第一个目标，**给在老家的爸妈，盖了一栋房子**。

当然，我说这个不是为了吹牛批，更不是为了炫耀，因为这没啥，我只想在 **我是帅地** 这个号里，分享自己的经历和感悟。

帅地作为一个和大家一样的普通人，一路摸索爬滚，基本是没有大佬相助的，所以这里也具有**运气**成份。不过，如果你什么也不懂，那么再好的机遇在你面前，你也把握不住。

从 0 挣到人生的第一个一百万，我想说几点感悟，每个人在每个阶段的感悟是不同的，几年后我再来看这篇文章，可能我的观点改变了也不一定，趁着现在还没变，就真心和大家聊一聊我的感悟。

不躺平，持续学习

我发现我身边那些混的不错的，不少都是家庭背景一般，但个人非常努力，校招也收个了很多 offer，总之就是，肚子里有点硬货，当然，仅仅拥有这些，可能还不够，但是连这些都没有，那么懂再多的道理，如果自己什么也不会，那也是不行的。

我之所以能够挣到一百万，绝大部分得益于我大三开始写文章，能够写出不错的文章，是因为我大一大二积累了很多专业知识，后面我写的东西有人愿意看愿意听，是因为我校招拿了一些大厂 offer，所以写出来的东西，都是实打实的。

有自己的真实经历写出来的东西和只看大道理写出来的东西，给人的感觉是完全不同的。

所以我觉得，你得有一技之长，当你有了一技之长，那么你肯定有一份自己独特的学习经历，而这份独特的学习经历，以后可以给你带来很大的用处，至于以后是啥时候，我也不知道，这算是一种铺垫吧。

至于一技之长指啥?

其实不一定是专业技能，也可以是其他的，例如可以是视频制作，也可以是考研考公研究出的各种技巧等等，核心就是，你通过它，获得了成长，有了自己一份独特的经历。

现在互联网时代，人人都有机会去输出自己的价值观，人人都可以去打造个人品牌，但是想要有所输出，就得肚子里 有东西可以输出，并且这些输出得是对一些人来说是用价值的，这些东西需要你提前去学习，而不是等到机会来了再去学习，不然，你会抓不住机会

所以，如果你还是在校生，那么趁着有时间，好好学，而不是，一直埋怨太卷了，学别人躺平。

不要仅仅只提高了「认知」

最近几年，在我身边，貌似「认知」这个词还挺火的，很多人通过看一些大 V 的文章或者买了一些专栏，提高了自己的认知，开拓了自己的眼界。

「认知」这个词在有些人看来有点虚，但确实，很多人也确实是一点认知也没有，格局和见识小仅仅停留在自己的小圈子里。

也有部分人看了很多大 V 的感悟之后，眼界确实变高了，也自我感觉良好，这些人也买了很多付费产品，但是呢，除了认知提高了之外，好像自身其他方面并没有得到多大的成长，有种**懂了很多道理，却依旧过不好这一生**的感觉。

其实我是想说，**自身的硬实力**很重要，就是你感悟了之后，你一定要沉下心来，去提升自身的硬实力，逼迫自己静下心来，去弥补自己在硬实力的不足，不然单单只是提高了「认知」，很难让你真正成长，这些「认知」可能给了你一些方向，之后你需要花一阵时间去死磕，去提升自己的实力，不然这些所谓的「认知」，就成了心灵鸡汤了。

做自己想做的事

有时候选择一条路，别人可能并不会看好，周围的人也会否定你，这个时候可能你就是孤军奋战，虽然你内心有自己的想法，但这个想法你不能对他们说，可能是担心别人嘲笑你。

例如很多二本三本的一些读者就和我反馈过，如果在班里你说你的目标是一线互联网大厂，可能班里的人觉得不可能，他们还会劝你不要学那么认真了，没用的，有一些真心想好好学习的人，可能还得躲着他们学习。

我记得我有个读者，拿了腾讯实习 offer，想逃课来实习，部分同学还说要举报他，不但不支持，还要举报他逃课，，，，

我刚开始写公众号的时候，其实也是躲着别人在写的，每次被别人发现，我就说我是无聊写着玩玩的，在我没做成之前，我肯定是不敢说自己写文章的真实原因的，感觉很虚。

如果没有在一个**对**的环境里，你又想**向上生长**，那么请坚持自己的执着，去做自己想做的事，在网上，你可以找到很多志同道合的小伙伴，例如帅地。

不要佛系

我一直觉得，做什么事，都要认真去做，不要抱着可有可无的态度去做，**内心可以佛系，但行动上，不要佛系**。

如果抱着佛系的心态去做，你会得不到什么成长，如果你抱着，**我一定做成它**的这种态度去做，就算你最后失败了，但是你可以明白一些事情，而一直佛系对待的话，会有种你是局外人的感觉，**你有时候感觉自己懂了，其实你什么也不懂**。

就拿写文章，运营公众号的事来说吧，我觉得有些人很佛系，但我觉得，你带着野心去写文章，去运营公众号，你会获得更多的成长。

所以我的观点是，如果你想要获得更大的成长，那么请不要佛系，而是抱着野心去做。

总结

当然，我说的这些观点，可能部分人有身同感受，部分人一脸懵逼，这很正常，每个人在不同的阶段，感受是不同的。

在现在这个时代，买房可能很难，但衣行吃住还是不难的，假如你有想做的事，并且你觉得这件事的结果是值得期待的，那么就去做，不要被一些东西所束缚。

因为我始终觉得，按自己喜欢的方式去生活去奋斗，会过的比较开心，打从我懂事开始，我就一直告诉自己，要活的高兴，而我也确实在按照自己的方式在生活。

坚持做一件事

帅地从 2018 写文章到现在有 3 年了，这三年认识了非常多优秀的人，也看到了很多逆袭的同龄人，工作副业两开花。

从很多身边的案例中，我发现了一个事：**就是无论做什么事，积累到一定的数量很重要**。

例如说写公众号吧，如果你去写 leetcode 题解，你写几篇，可能没啥人看，也啥效果，写个几十篇，可能也没啥用，这个时候很多人往往放弃，但是当你写了几百篇，这个时候就完全不一样了，无论是对你还是对外界，这几百篇题解都会产生一定的影响。

也就是说，当你积累到一定的量，就存在爆发性效果的可能，例如我认识的号主 **labuladong**，**代码随想录**，都是在刷题这件事，前期不愠不火，后期量到了，就爆发了。

在做公众号这一行，这类型的人，我认识的还有好几个，就是前期不能搞的太乱，需要在某个方向坚持，当你积累到一定的量，就会有意想不到的效果。

其实很多事情也一样，刚开始刷算法题，可能很难，很煎熬，很多人刷了 几十道，感觉不出效果，往往会放弃，不过当你坚持刷了 一两百道，不仅会越刷越快，可能还爱上了；刚开始学习计算机基础，可能觉得没啥用，当你把一整套都咬牙学了，就突然发现拿大厂 offer 并非那么难了。

然而我发现不少人，在学一门知识的时候，特别喜欢变动，例如学习数据结构，有些人买了一本书，学到中途，突然感觉很吃力，就又换了一本书看，不过换一本书，往往需要在重头看，有些人可能还会又在中途放弃，跑去看视频，又得重头学……

反正，我是见过不少，就是学着学着，遇到了点困难或者是自己心思不定，就频繁变更赛道，以至于经常在学习重复的东西，成长的很慢。

这其实不大好，我觉得，应该咬牙把接下来的啃过去，把该做的量给做到位了，或许你会 get 很多心得。

无论是在学习上还是副业上，这种案例我都见过不少。

当然，也有很多坚持了之后，仍然无法得到期望结果的案例，毕竟，这不是充分条件，而且必要条件。

我有阵子特别想看书，不过经常因为一些事情，就把看书这事给耽误了，就是看起来断断续续的，看了前面，过一阵子再来看时，都忘了前面讲了啥，这个时候就特别纠结是从头看还是继续往下看，无论是效果还是效率，都很低。

所以我现在是，看书时会坚持把一本书完整看完，尽量不要断断续续，一次性看完，比较容易 get 到一些感悟。

帅地现在在做的一些事情，也是基于这个心得在做的；方向和技巧很重要，但**数量**，也不能少。

总的感悟就是，做事不能三心二意，也别老是怀疑自己，如果这个方向的结果是值得期待的，不妨先做到一定的量，在这个过程中，反复复盘和打磨，或许，会有

认知与行动

今天跟各位小伙伴聊一聊「认知」与「行动」，这其实是我这几年根据身边的朋友总结出来的一种感悟，其实也有不少球友有这种现象，所以我想聊一聊。

对于「认知」这个词，其实我是在大二看了一些大 V 的文章后，才接触到一个词，简单的来说，它有点是「无知」的对立面。

「认知」这个词在有些人看来有点虚，但确实，很多人也确实是一点认知也没有，格局和见识小仅仅停留在自己的小圈子里。

在大学或者工作之后，一般有两种人：

一种是认知比较低的，例如他在大学只知道按部就班学习，对其他的什么都不懂，例如不知道校招，不知道为啥要学这门课，往往等到了毕业之后才后悔没有早点去了解，当然，这些例子只是「认知」的一部分。

显然，这种状态是不大健康的，然而大部分人也确实是处于这种状态，除了学校之内的事，对其他的一无所知。

还有另外一种，可能是无意中看了一些大 V 的文章，或者加入了一些社群，如知识星球，通过这些，提高了自己的认知，开拓了自己的眼界。

第二种总归是好的，至少知道了自己要的是什么，也知道自己应该去学习什么，最重要的是，和同学交流的时候，你会变的更加自信。

但是呢，对于第二种，其实很多人是处于这样一种状态：就是除了认知提高了之外，好像自身其他方面并没有得到多大的成长，有种「懂了很多道理，却依旧过不好这一生」的感觉。

我看星球里 + 我微信里，就不乏有这样的一些案例，加了很多大 V 的星球，看了很多大 V 的感悟之后，眼界确实变高了，但是自身的「硬实力」并没有提高。

这其实很不好，有点像：你收集了好多非常好用的资料，只要你把这些资料看了，感觉就能考上名校，而这些资料也确实非常有用。但是，你却从来没有去看过这些资料。

也就是说，你掌握了某些东西，领悟了某些道理，但是你却不去行动。

这其实很不好。

今天我想说的就是，不要只是提高了「认知」，这样很难让你真正成长，这些「认知」可能给了你一些方向，之后你需要花一阵时间去死磕，去提升自己的实力，不然这些所谓的「认知」，就成了心灵鸡汤了。

例如在咱门知识星球，对于很多小白，或者迷茫的人士，你肯定可以通过星球的资料摸清自己的方向，知道该学什么，但是我却很担心你不去行动，以至于错过了最佳的学习时间。

所以呢，帅地希望星球里的小伙伴，不要成为这种人，认准了方向，就一定要去死磕，你一定要沉下心来，去提升自身的硬实力，逼迫自己静下心来，去弥补自己在硬实力的不足。

只要这样，你才能得到真正的成长。

帅地就是在大二接触了这些，了解了很多之后，去一步一个脚印写文章，去准备校招，去摸索写作，一点点积累自己的硬实力，做出一些别人看得见的东西出来，只有这样，你说的「认知」，别人才愿意去认可，否则，就是空谈！

行动起来，居然找准了方向，你只需一心扑进去学，只有「认知」与「行动」并存，你才能获得最大的收益！意想不到的效果。

保姆级学习路线

我的学习路线是对标中大厂 offer 来写的，，大厂 offer 不敢保证，但是拿个知名公司 Offer 还是可以的，所谓知名就是这家公司很多人听说过，需要学的我给你们规划好，最后还得看你自己掌握的如何，当然，如果你是大专，建议升个本，因为很多公司的最低学历要求就是**本科**。

这里目前是详细写了 Java，C++，Go 和前端的学习路线，后面在补充其他学习路线吧，以后你们不知道怎么发力时，来看看就行了，我不想把自己的学习路线规划的太死板，我希望你们能从我规划的学习路线中，找到属于自己的学习路线，遇到纠结的在向我发起提问。

另外，有些人错过了最佳学习时间，可能无法按照学习路线把所有东西都学了，例如已经大四错过秋招的该怎么学等等，可以看下面「**关于不同学历不同学习时间几个问题**」，这里会对一些问题进行说明。

Java 学习路线

我先大致说一下需要学习的点，然后下面有归类了学习的顺序，大家先通读一遍，之后在自己规划下属于你自己的学习顺序。

一、Java 技术栈的学习

1、Java 基础入门

> 注意，下面所有推荐的书籍，大家都可以在这里下载到：计算机书籍免费下载（高清带目录完整PDF版）。

一开始肯定要了解一波 Java 语言的特性，很多人都关心一开始看什么资料，其实我不推荐入门阶段看的太多太杂，因为你不知道哪些是有用的，哪些是过时的，或者在目前阶段用不到的。

对于初学者尤其是没有一点编程基础的人来说，要渡过这个从零到一的过程，往其实是非常艰难的，挫败感会非常强，所以找到一份合适的资料，还是挺有必要。

下面这些学习资料都是我曾经读过或者看过的，没看过的我也不会介绍。还是那句话，资料和视频都是辅助，顶多领你入门，剩下的还是靠自己。

学习思路：首先 Java 基础没啥好说的，随便找个视频快速看就可以了，我推荐上学堂Java高琪第一季；不过我当时的看了第一季和第二季，这里其实也建议看到第二季，也就看到 242 集，然后 第一季在 147 ～161 集是一个小项目，这个项目不想做可以不用做，我当时没做，或者你想看书也可以看书，就看《Java核心卷1》吧。

看完基础的可以看《Java编程思想》，这本书也很好，解决了我很多疑惑，不过有点难度，我是选择性看的，具体这本书如何读 + 重点章节，说明：Java编程思想怎么读+重点章节

2、Java 进阶

想要靠 Java 来吃饭，单单靠看书刷视频肯定是不够的，咱们还得深入学习一波 Java 语言的特性，为了让大家学起来更加有针对性，我总结了如下核心知识：

集合模块（超重点）：主要是包括各种常用集合：例如 Set（包括HashSet，TreeSet），Map（包括 HashMap，HashTable），List(包括ArrayList，LinkedList)等等，最核心的就是 hashMap,concurrenthashMap,ArrayList,LinkedLis 这几个，这几个源码必须看，，这部分，主要是看文章 + 自己看 JDK 源码学习。

多线程(超重点)：synchronize，volatile，这两个优先学，比较简单，之后学习线程池，并发包（如lock等等），有点难度，推荐看《Java 并发编程艺术》+《Java并发编程实战》，这两本就够了，《Java并发编程艺术》感觉得看三四遍才行，第一遍看了就忘光了，第二遍会有其他的收获，第三遍就可以吊打面试官了，这本书重点章节 + 如何阅读星球有写了：《Java并发编程艺术》如何阅读+重点章节

各种文件流（不怎么重点）：file，inputStream，outputStream等等，反正就是各种文件流，看书时你们自然会看到，这部分必须多实践，只有时间，才能真的理解不详细介绍。

虚拟机（超重点）：虚拟机是必须学习的了，重点是GC部分，推荐看《深入理解Java虚拟机：JVM 高级特性与最佳实践》，一本就差不多够了，多看四五遍就行了，这本书如何阅读也已经写了：《深入理解Java 虚拟机》阅读及其重点章节指南

其他：还有很多，如反射，注解，异常等。反射要了解他的一些应用，异常也挺重要，注解就一般般。

3、JavaWeb 入门

学了上面那么多，啥程序、网站也没写出来，有点难受？没事，这个时候，我们就要开始撸网站了，如果是以 Java 作为开发语言的，那么 JavaWeb 是必须学的了。这个时候你可以入门学习下这些（**注意，不用深入，快速找个视频入门就可以了**）

我下面列举一下 Javaweb 都是指哪些知识：

1、mysql、html+css+js 、 tomcat、xml等。推荐看视频，自己去B站搜索 JavaWeb的视频，那么哪里不会补哪里，这些知识，几个小时就可以入门一门的了。至于 msyql，入门推荐看《mysql不知必会》，之后可以看书或者看视频，具体视频这里暂时没有推荐。

2、servlet+jsp 系列。

不建议直接学习框架，也是应该先学习 servlet，这些底层才是最重要滴。servlet可以跟着视频学，至于 jsp 其实很少用了，不过还是可以了解一下。

4、框架的学习框

Servlet 写太麻烦了，只有认真学过 servlet 的人才能吹框架的好处，所以必须先学 servlet 再来学习框架，主要有三大框架：Spring + SpringMVC + Mybatis。

其实 SpringMVC 也是属于 Spring 吧，MVC 只是一种思想，这里学习顺序是先建议学习 Spring，直接看视频入门学习就可以了，后面再买书，掌握到什么程度？最后是要能知道一些原理，例如IOC，AOP的原理，使用了什么设计模式等等。

SSM 的视频可以在 B 站找，学完就找个练手的项目，或者你不想练手 ssm 也可以了，问题不大。

不过现在搭 ssm 太麻烦，基本都是用 SpringBoot 了，所以 SpringBoot 也必须，可以学完 SSM，目再学 SpringBoot，之后得做一些 springboot 的项目。

5、中间件的学习

必须学的中间件：redis，Redis 基本面试必问，工作也基本的都会使用到，所以必须掌握，推荐书籍《Redis 设计与实现》、《Redis开发与运维》。两本书刷完，就差不多了，或者说时间不是特别多的，刷完《Redis开发与运维》也行，这本书的重点内容我写过：

其他的话，像消息队列啊，分布式等相关框架，有时间也可以学习一波，项目用到在学吧，然后关于消息队列相关的面试题，如何应付，就看这个资料：Redis，消息队列和分布式

6、2个完整的项目

把上面的 Java 基础以及框架学了，就要做个完整的项目了，我只能说，项目非常非常重要，因为面试的过程，一定会问项目，而且必须是你自己亲自做过的项目，假如你只是看视频，没有动手实践，很容易就会被问懵。所以一定一定一定要亲自动手做一个项目。

而且在未来，项目会被越来越看重，实践能力也会越来越看重，至于项目怎么找？这个大家可以花点钱买，或者找一找盗版也行，至于 B 站上的项目，很多适合练手，一般质量都比较一般吧。

技术栈的话，个人是推荐 Springboot + redis + mq，或者也可以自己造个轮子，比如 RPC，比如实现一个文件系统，或者实现一个关系型数据库等，这些都是不错的项目，另外如何做了很多人都做过的项目，建议一定要改个名字 + 自己加一些东西上去。

帅地这边的项目课程（有钱的或者可以考虑吧）：从0设计一个web项目

二、数据结构与算法的学习

大学第一学期你入门了某种语言之后，就要开始学习**数据结构与算法**。当然，不一定是第一学期，对于非科班的也一样，也是建议这个学习顺序。当然，你有自己的目标，完全可以按照自己的来。

我的文章是**主写数据结构与算法**的，我秋招也是靠着这个优势拿到 offer 的，所以我认为数据结构与算法是非常非常重要的，可能你会听到工作都基本没用到这些知识啊，或者库函数都帮我们封装好了啊，我们干嘛要学。

如果你这样想的，那你就错了。对于想要找工作的来说，这方面的知识是必考的，所以你得学；对于已经工作的来说，这方面的知识，可以让你学习到很多设计思想，所谓 **数据结构 + 算法 = 程序**，而且你学了这个，你会发现学习其他知识，上手的特别快。综上，数据结构与算法必学。

好吧，上面扯了这么多，就是告诉你，数据结构与算法的重要性，好了，下面我介绍下入门数据结构必学的有哪些，不过你跟着书本的学习顺序来就是了。

1、基础数据结构的学习

1、时间复杂度、空间复杂度

2、链表、队列、栈

3、树（二叉树，查找二叉树、AVL树，红黑树等）

4、图（图有好多种算法，深度/广度搜索，最短路径、最小生存树等）

如果你是科班的，那么这些我觉得你大一第二学期都学完是最好的了，没学完也问题不大，有些人可能是先教《离散数学》这本课，为数据结构与算法做铺垫。

书籍推荐：《大话数据结构》、《数据结构与算法分析:C语言描述版》，学哪一本？都可以，问题不大，我当时学的是第二本。大家记得根据自己的语言去学，我上面列举的，都是用 c 语言来实现的，这里我也写过如何阅读：《数据结构与算法分析-xx语言描述版》怎么读+重点章节

2、算法的学习

在大一大二，真心建议你们把算法基本功打好，后面就真的没啥时间刷题之类的了，因为无论你以后要学习哪个方向的，算法都是实用的，会一些算法，说话也都自信了。算法的学习，刷题是必须的了，但不建议盲目刷题，而是先学习一些**算法思想**，在找对应的题刷，要学的主要有：

1、十大排序算法

2、递归、贪心、回溯、动态规划、枚举等

推荐书籍：《阿哈算法》、《算法设计与分析基础》适合入门；《算法第四版》、《编程之美》适合进阶

视频：这种还是挺建议看书，我没看过视频，，这里就不介绍了。

3、保持算法的学习

算法的学习，真的是靠积累的，而刷题是必须的。而且学校都会举办一些比赛，这里还是比较建议大家去参见的，这样也能让你更加有激情着去学习。当然，每年都会有很多 ACM 编程大赛，要不要参加呢？这个看你了，也不是说参加就一定好，看你自己吧，具体可以参考我之前一篇文章说的：我是继续打 acm 还是把精力花到学习其他技能去？

大家可以在 leetcode 长期保持刷题，一天一两道，或者一个星期三四道都行。

另外，如果时间不多的，那就把《剑指offer》刷个一两遍就行了，只有 剑指offer 的刷题攻略，我也写过：剑指offer刷题攻略

三、计算机基础的学习

看过我文章的都知道，我一直强大计算机基础的重要性，所以这里必须列举要学的有哪些。刚才说了选择一门语言深入，你在深入学习的过程中，肯定也在学习学校开设的专业课，包括：计算机网络 + 操作系统 + 数据库 + 汇编 + 计算机组成原理 + 编译原理 等等。

> 其实给他们排优先级是不大好的，这样给他们排，感觉容易被大佬喷，不过没办法，对于小白来说，我还是想排以下顺序

1、计算机网络 + 操作系统

2、数据库 + 计算机组成原理（数据库相应你们可能已经先学过 MySQL 了）

3、汇编 + 编译原理

不过呢，比较重要的是计算机网络 + 操作系统 + 数据库，这三门我单独拿出来说一下重点吧。

四、计算机网络

无论你是从事啥岗位，无论是校招还是社招，计算机网络基本都会问，特特是腾讯，字节，shopee，小米等这些非 Java 系的公司，问的更多。这块认真学，**一个半月**就可以搞定了。

计算机网络就是一堆协议的构成，下面是一些比较重要的知识点，学的时候可以重点关注下。

物理层、链路层：

1. MTU，MAC地址，以太网协议。
2. 广播与 ARP 协议

网络层

1. ip 地址分类
2. IP 地址与 MAC 地址区别
3. 子网划分，子网掩码
4. ICMP 协议及其应用
5. 路由寻址
6. 局域网，广域网区别

传输层（主要就是 TCP）

1. TCP首部报文格式（SYN、ACK、FIN、RST必须知道）
2. TCP滑动窗口原理，TCP 超时重传时间选择
3. TCP 拥塞控制，TCP 流量控制
4. TCP 三次握手与四次挥手以及状态码的变化
5. TCP连接释放中TIME_WAIT状态的作用
6. SYN 泛洪攻击
7. TCP 粘包，心跳包
8. UDP 如何实现可靠传输
9. UDP 与 TCP 的区别
10. UDP 以及 TCP 的应用场景

应用层

1. DNS 原理以及应用

2. HTTP 报文格式，HTTP1.0、HTTP1.1、HTTP2.0 之间的区别
3. HTTP 请求方法的区别：GET、HEAD、POST、PUT、DELETE
4. HTTP 状态码
5. HTTP 与 HTTPS 的区别
6. 数字证书，对称加密与非对称加密
7. cookie与session区别
8. 输入一个URL到显示页面的流程（越详细越好，搞明白这个，网络这块就差不多了）

书籍推荐：零基础可以先找个视频看看，然后有了一定基础可以看《计算机网网络：自顶向下》这本书，这本书建议看两遍以及以上，还有时间的可以看《TCP/IP详解卷1：协议》。

五、操作系统

操作系统和计算机网络差不多，不过计算机网络会问的多一些，操作系统会少一些，总结起来大致：

1、进程与线程区别

2、线程同步的方式：互斥锁、自旋锁、读写锁、条件变量

3、互斥锁与自旋锁的底层区别

4、孤儿进程与僵尸进程

5、死锁及避免

6、多线程与多进程比较

7、进程间通信：PIPE、FIFO、消息队列、信号量、共享内存、socket

8、管道与消息队列对比

9、fork进程的底层：读时共享，写时复制

10、线程上下文切换的流程

11、进程上下文切换的流程

12、进程的调度算法

13、阻塞IO与非阻塞IO

14、同步与异步的概念

15、静态链接与动态链接的过程

16、虚拟内存概念（非常重要）

17、MMU地址翻译的具体流程

18、缺页处理过程

19、缺页置换算法：最久未使用算法、先进先出算法、最佳置换算法

书籍推荐：《现代操作系统》

六、MySQL

数据库一般主流的有 MySQL 和 Oracle，不过建议大家学习 MySQL 了，因为大部分公司都是使用 MySQL，也是属于面试必问，而且工作中 MySQL 也是接触的最多的，毕竟工作 crud 才是常态。

下面这些是我认为比较重要的知识点：

1、一条 sql 语句是如何执行的？也就是说，从客户端执行了一条 sql 命令，服务端会进行哪些处理？（例如验证身份，是否启用缓存啥的）。

2、索引相关：索引是如何实现的？多种引擎的实现区别？聚族索引，非聚族索引，二级索引，唯一索引、最左匹配原则等等（非常重要）。

3、事务相关：例如事务的隔离是如何实现的？事务是如何保证原子性？不同的事务看到的数据怎么就不一样了？难道每个事务都拷贝一份视图？MVCC 的实现原理（重要）等等。

4、各种锁相关：例如表锁，行锁，间隙锁，共享锁，排他锁。这些锁的出现主要是用来解决哪些问题？（重要）

5、日志相关：redolog，binlog，undolog，这些日志的实现原理，为了解决怎么问题？日志也是非常重要的吧，面试也问的挺多。

6、数据库的主从备份、如何保证数据不丢失、如何保证高可用等等。

7、一些故障排查的命令，例如慢查询，sql 的执行计划，索引统计的刷新等等。

对于 2-4 这四个相关知识，面试被问到的频率是最高的，有时候面试会让你说一说索引，如果你知道的多的话就可以疯狂扯一波了，具体可以看我写的 MySQL 学习路线：[肝完了，我的 MySQL 学习之路]

七、Linux 补充（选学）

八股文大家基本都知道是啥，不过很多人都没有把 Linux 写上去，其实 Linux 的范围很大，例如 Linux 内核啊，Linux 网络编程等等。

这种不同的岗位可能需要掌握的要求不同，所以我也不讲这些，今天主要帮大家捋一捋「Linux 基础」这块，就是面试中有可能会问到的，这块需要准备的还是比较少的，下面做一些汇总，方便帅友们到时候可以更加有针对性去复习。

另外，基础操作命令我就不说了，就说一些需要注意的点

基础

1、文件系统：主要就是要了解下 inode，block，superblock，block bitmap 这些关键词的含义。

2、目录的大致含义：例如像 proc 这个目录就放一些进程相关的信息，默认软件安装在 usr 目录等等，这其实是一种规范，这个要大致了解下。

3、文件一些属性：例如有些文件可读，可写，可执行，对应 rwx 权限这些， 通过 d 可以知道这个一个目录等等。

4、软连接与硬链接

5、VIM 的几种模式区别了解一一下

进程与内存等(这块是重点)

1、几个进程命令：ps，top，pstree，netstat，这几个进程要自己用一用，工作后经常用到的，重点学习了解下。

2、Linux 进程的一些状态，记得好几种：R，D，S，Z，T 这五种，要了解下。

3、孤儿进程，僵尸进程 要知道（重要），然后还得知道如何解决这种情况，例如出现大量僵尸进程怎么办

4、中断相关：软中断啊，不可中断状态啊

5、内存中的 Cache 和 buffer 的区别

6、了解 Swap 区

7、SiGCHLD 信号，waitpid() 和 wait() 函数

几种重要的 IO 模型

1、同步，异步。

2、阻塞与非阻塞

3、select，epoll

4、NOI与IO

一些命令

如果可以，find，grep 这几个搜索相关的命令学下，因为项目会打印日志，我们需要捞日志来分析，就会经常用到 find,grep 这些，到时候面试官问到你也可以说。

项目故障排查

这块其实就是和实际工作比较靠近，做项目有时候会用到啥的，上面的优先学的，有空了再学一学这些

1、通过工具来追踪进程的栈轨迹，就是分析异常的原因，例如用 top 和 jstakc 来定位

2、掌握一些分析进程 IO，CUP，内存的工具，例如 pidstat

3、用 jsp，jstack 来分析死锁原因，一般就是通过栈信息来查看。

学习资料指导

有时间 + 零基础的，就看下《鸟哥Linux私房菜》这本书吧，讲的很细，可以看下，跟着做实验，容易做的实验可以跟着做，麻烦的就不需要了，看前面基础的部分就行，后面的可以不看了。

但是呢，这本书只是入门，其实很多没讲的，例如 select，epoll，僵尸进程啥的啊，这种一般，你们哪个不同就看找对应的文章就行，通过文字来逐个了解突破哦。

暂时先说这么多，后面再慢慢补充了，我说的这些都是我自己当时学过的，可能不是那么全，但是核心的基本都说了。

八、学习的顺序

对于Java，算法，计算机基础的学习顺序，假如你时间多，例如你大一或者大二，那么我觉得可以按照这个学习顺序：

1、先选一门语言入门，例如 C 语言或者 Java

2、之后学习数据结构与算法

3、之后一边学习我上面说的 Java 技术栈，一边学习计算机基础，优先学操作系统+计算机网络+数据库

假如你很急，时间不多了，那么我建议你先说我上面说的 Java 相关的知识，然后做项目，之后再回过头来学习算法和基础知识。

课程推荐：当然，如果你觉得自己不缺钱，八股文内容那么多，你抓不住重点，可以看看这门课程：八股文考点全方位分析

九、总结

我觉得对于大学四年来学，上面这些是最核心的，也是必须学的。但是仅仅是上面这些还是不够的，上面的这些一两年就可以学完的。所以你还有很多其他时间，那么你可以学一些自己感兴趣的，多折腾，多抖鼓，而我上面学的，希望你都学。

最后，是希望各位还在校的学生，大一可以好好浪，但也要保持应有的学习时间，之后，就好好学习吧。不管你是名校还是非名校，我觉得你在只要这几年认真学，进大公司的几率，真的非常大，这绝对不是鸡汤。

C++ 学习路线

一般开发岗主流的就是 Java 后台开发，前端开发以及 C++ 后台开发，现在 Go 开发也是越来越多了，今天把 C++ 后台开发学习路线补上。

写之前先来回答几个问题

1、C++ 后台开发有哪些岗位?

C++ 后台开发的岗位还是很多的，例如游戏引擎开发，游戏服务端开发，音视频服务端/客户端开发，数据库内核开发等等，而且 C++ 也能用来写深度学习，做硬件底层这些。

总之，C++ 后台开发的岗位，还是很丰富的，大家不用担心找不到合适的岗位。

2、C++ 后台开发岗位需求量大吗?

一般大公司大需求量会多一些，小公司需求量较少。

说到岗位需求量，那肯定是 Java 的岗位需求量是最大的，当然，学 Java 的人也是最多的，假如你要学习 C++，那我觉得你要定位大公司可能会好一点，进大公司反而会比 Java 容易。

假如你觉得自己实力很一般，够不着大公司，那我觉得你可以考虑学习 Java，因为大部分小公司，Java 岗位多一些。

但是呢，假如你是应届生，那么语言其实也不是特别重要，只要你 把计算机基础和算法学好，就算你是学 Java 的，也可以去面 C++；学 C++ 的也可以去面 Java。

我当时是学 Java 的，不过秋招那会还面了几个 C++ 岗位，直接和面试官说我不会 C++ 就可以了，他会问你其他的知识。

下面跟大家说一说 C++ 后台开发学习路线，为了方便大家做规划，每一个模块的学习，我都会说下大致的学习时间

一、C++ 基础

假如你有 C 语言基础，那么这块感觉花个三四个月就能拿下了，假如你是零基础的，估计还得学两三个月的 C 语言，也就是说，得花半年时间才行，没有 C 语言基础的可以B站找一个看，貌似翁凯老师讲的挺不错，可以考虑。

C++ 这块，重点需要学习的就是一些**关键字**、**面向对象**以及 **STL 容器**的知识，特别是 STL，还得研究下他们的一些源码，下面我总结一下一些比较重要的知识（其实是根据面试结果来挑选）。

1. 指针与引用的区别，C 与 C++ 的区别，struct 与 class 的区别
2. struct 内存对齐问题，sizeof 与 strlen 区别
3. 面向对象的三大特性：封装、继承、多态

4. 类的访问权限：private、protected、public
5. 类的构造函数、析构函数、赋值函数、拷贝函数
6. 移动构造函数与拷贝构造函数对比
7. 内存分区：全局区、堆区、栈区、常量区、代码区
8. 虚函数实现动态多态的原理、虚函数与纯虚函数的区别
9. 深拷贝与浅拷贝的区别
10. 一些关键字：static, const, extern, volatile 等
11. 四种类型转换：static_cast、dynamic_cast、const_cast、reinterpret_cast
12. 静态与多态：重写、重载、模板
13. 四种智能指针及底层实现：auto_ptr、unique_ptr、shared_ptr、weak_ptr
14. 右值引用
15. std::move函数
16. 迭代器原理与迭代器失效问题
17. 一些重要的 STL：vector, list, map, set 等。
18. 容器对比，如 map 与 unordered_map 对比，set 与 unordered_set 对比，vector 与 list 比较等。
19. STL容器空间配置器

等等。

根据书来学就可以了，然后学到一些重点，可以重点关注一下。

书籍推荐：

1、《C++Primer》，这本书内容很多的，把前面基础的十几章先看一看，不用从头到尾全啃，后面可以**字典**来使用。

2、《STL 源码剖析》，必看书籍，得知道常见 STL 的原理，建议看个两三遍。

3、《深度探索C++对象模型》，这本主要讲解**面向对象**的相关知识，可以帮你扫清各种迷雾。

面试题：学完之后可以通过面试题复习，这里整理了：C++面试题分类阅读指南(附答案)

除了语言部门，其他其实和 Java 的类似，例如计算机基础，算法这些。

二、数据结构与算法的学习

大学第一学期你入门了某种语言之后，就要开始学习**数据结构与算法**。当然，不一定是第一学期，对于非科班的也一样，也是建议这个学习顺序。当然，你有自己的目标，完全可以按照自己的来。

我的文章是**主写数据结构与算法**的，我秋招也是靠着这个优势拿到 offer 的，所以我认为数据结构与算法是非常非常重要的，可能你会听到工作都基本没用到这些知识啊，或者库函数都帮我们封装好了啊，我们干嘛要学。

如果你这样想的，那你就错了。对于想要找工作的来说，这方面的知识是必考的，所以你得学；对于已经工作的来说，这方面的知识，可以让你学习到很多设计思想，所谓 **数据结构 + 算法 = 程序**，而且你学了这个，你会发现学习其他知识，上手的特别快。综上，数据结构与算法必学。

好吧，上面扯了这么多，就是告诉你，数据结构与算法的重要性，好了，下面我介绍下入门数据结构必学的有哪些，不过你跟着书本的学习顺序来就是了。

1、基础数据结构的学习

1、时间复杂度、空间复杂度

2、链表、队列、栈

3、树（二叉树，查找二叉树、AVL树，红黑树等）

4、图（图有好多种算法，深度/广度搜索，最短路径、最小生存树等）

如果你是科班的，那么这些我觉得你大一第二学期都学完是最好的了，没学完也问题不大，有些人可能是先教《离散数学》这本课，为数据结构与算法做铺垫。

书籍推荐：《大话数据结构》、《数据结构与算法分析:C语言描述版》，学哪一本？都可以，问题不大，我当时学的是第二本。大家记得根据自己的语言去学，我上面列举的，都是用 c 语言来实现的，这里我也写过如何阅读：[《数据结构与算法分析-xx语言描述版》怎么读+重点章节]

2、算法的学习

在大一大二，真心建议你们把算法基本功打好，后面就真的没啥时间刷题之类的了，因为无论你以后要学习哪个方向的，算法都是实用的，会一些算法，说话也都自信了。算法的学习，刷题是必须的了，但不建议盲目刷题，而是先学习一些**算法思想**，在找对应的题刷，要学的主要有：

1、十大排序算法

2、递归、贪心、回溯、动态规划、枚举等

推荐书籍：《阿哈算法》、《算法设计与分析基础》适合入门；《算法第四版》、《算法导论》、《编程之美》适合进阶

视频：这种还是挺建议看书，我没看过视频，，这里就不介绍了。

3、保持算法的学习

算法的学习，真的是靠积累的，而刷题是必须的。而且学校都会举办一些比赛，这里还是比较建议大家去参见的，这样也能让你更加有激情着去学习。当然，每年都会有很多 ACM 编程大赛，要不要参加呢？这个看你了，也不是说参加就一定好，看你自己吧，具体可以参考我之前一篇文章说的：[我是继续打 acm 还是把精力花到学习其他技能去？]

大家可以在 leetcode 长期保持刷题，一天一两道，或者一个星期三四道都行。

另外，如果时间不多的，那就把《剑指offer》刷个一两遍就行了，只有 剑指offer 的刷题攻略，我也写过：剑指offer刷题攻略

三、计算机基础的学习

看过我文章的都知道，我一直强大计算机基础的重要性，所以这里必须列举要学的有哪些。刚才说了选择一门语言深入，你在深入学习的过程中，肯定也在学习学校开设的专业课，包括：计算机网络 + 操作系统 + 数据库 + 汇编 + 计算机组成原理 + 编译原理 等等。

> 其实给他们排优先级是不大好的，这样给他们排，感觉容易被大佬喷，不过没办法，对于小白来说，我还是想排以下顺序

1、计算机网络 + 操作系统

2、数据库 + 计算机组成原理（数据库相应你们可能已经先学过 MySQL 了）

3、汇编 + 编译原理

不过呢，比较重要的是计算机网络 + 操作系统 + 数据库，这三门我单独拿出来说一下重点吧。

四、计算机网络

无论你是从事啥岗位，无论是校招还是社招，计算机网络基本都会问，特特是腾讯，字节，shopee，小米等这些非 Java 系的公司，问的更多。这块认真学，**一个半月**就可以搞定了。

计算机网络就是一堆协议的构成，下面是一些比较重要的知识点，学的时候可以重点关注下。

物理层、链路层：

1. MTU，MAC地址，以太网协议。
2. 广播与 ARP 协议

网络层

1. ip 地址分类
2. IP 地址与 MAC 地址区别
3. 子网划分，子网掩码
4. ICMP 协议及其应用
5. 路由寻址
6. 局域网，广域网区别

传输层（主要就是 TCP）

1. TCP首部报文格式（SYN、ACK、FIN、RST必须知道）
2. TCP滑动窗口原理，TCP 超时重传时间选择
3. TCP 拥塞控制，TCP 流量控制
4. TCP 三次握手与四次挥手以及状态码的变化
5. TCP连接释放中TIME_WAIT状态的作用
6. SYN 泛洪攻击
7. TCP 粘包，心跳包

8. UDP 如何实现可靠传输
9. UDP 与 TCP 的区别
10. UDP 以及 TCP 的应用场景

应用层

1. DNS 原理以及应用
2. HTTP 报文格式，HTTP1.0、HTTP1.1、HTTP2.0 之间的区别
3. HTTP 请求方法的区别：GET、HEAD、POST、PUT、DELETE
4. HTTP 状态码
5. HTTP 与 HTTPS 的区别
6. 数字证书，对称加密与非对称加密
7. cookie与session区别
8. 输入一个URL到显示页面的流程（越详细越好，搞明白这个，网络这块就差不多了）

书籍推荐：零基础可以先找个视频看看，然后有了一定基础可以看《计算机网网络：自顶向下》这本书，这本书建议看两遍以及以上，还有时间的可以看《TCP/IP详解卷1：协议》。

五、操作系统

操作系统和计算机网络差不多，不过计算机网络会问的多一些，操作系统会少一些，总结起来大致：

1、进程与线程区别

2、线程同步的方式：互斥锁、自旋锁、读写锁、条件变量

3、互斥锁与自旋锁的底层区别

4、孤儿进程与僵尸进程

5、死锁及避免

6、多线程与多进程比较

7、进程间通信：PIPE、FIFO、消息队列、信号量、共享内存、socket

8、管道与消息队列对比

9、fork进程的底层：读时共享，写时复制

10、线程上下文切换的流程

11、进程上下文切换的流程

12、进程的调度算法

13、阻塞IO与非阻塞IO

14、同步与异步的概念

15、静态链接与动态链接的过程

16、虚拟内存概念（非常重要）

17、MMU地址翻译的具体流程

18、缺页处理过程

19、缺页置换算法：最久未使用算法、先进先出算法、最佳置换算法

书籍推荐：《现代操作系统》

六、MySQL

数据库一般主流的有 MySQL 和 Oracle，不过建议大家学习 MySQL 了，因为大部分公司都是使用 MySQL，也是属于面试必问，而且工作中 MySQL 也是接触的最多的，毕竟工作 crud 才是常态。

下面这些是我认为比较重要的知识点：

1、一条 sql 语句是如何执行的？也就是说，从客户端执行了一条 sql 命令，服务端会进行哪些处理？（例如验证身份，是否启用缓存啥的）。

2、索引相关：索引是如何实现的？多种引擎的实现区别？聚族索引，非聚族索引，二级索引，唯一索引、最左匹配原则等等（非常重要）。

3、事务相关：例如事务的隔离是如何实现的？事务是如何保证原子性？不同的事务看到的数据怎么就不一样了？难道每个事务都拷贝一份视图？MVCC 的实现原理（重要）等等。

4、各种锁相关：例如表锁，行锁，间隙锁，共享锁，排他锁。这些锁的出现主要是用来解决哪些问题？（重要）

5、日志相关：redolog，binlog，undolog，这些日志的实现原理，为了解决怎么问题？日志也是非常重要的吧，面试也问的挺多。

6、数据库的主从备份、如何保证数据不丢失、如何保证高可用等等。

7、一些故障排查的命令，例如慢查询，sql 的执行计划，索引统计的刷新等等。

对于 2-4 这四个相关知识，面试被问到的频率是最高的，有时候面试会让你说一说索引，如果你知道的多的话就可以疯狂扯一波了，具体可以看我写的 MySQL 学习路线：[肝完了，我的 MySQL 学习之路]

七、网络编程

网络编程这块，有些公司还是问的挺多的，特别是 IO 多路复用，同步非同步 IO，阻塞非阻塞啥的，当时面腾讯基本每次都问，，，，学习 C++ 这块还是要重视一下，下面我说一下比较重要的吧。

1、IO多路复用：select、poll、epoll的区别（非常重要，几乎必问，回答得越底层越好，要会使用）

2、手撕一个最简单的server端服务器（socket、bind、listen、accept这四个API一定要非常熟练）

3、线程池

4、基于事件驱动的reactor模式

5、边沿触发与水平触发的区别

6、非阻塞IO与阻塞IO区别

书籍：可以看一看《Unix网络编程》

八、Linux 补充

八股文大家基本都知道是啥，不过很多人都没有把 Linux 写上去，其实 Linux 的范围很大，例如 Linux 内核啊，Linux 网络编程等等。

这种不同的岗位可能需要掌握的要求不同，所以我也不讲这些，今天主要帮大家捋一捋「Linux 基础」这块，就是面试中有可能会问到的，这块需要准备的还是比较少的，下面做一些汇总，方便帅友们到时候可以更加有针对性去复习。

另外，基础操作命令我就不说了，就说一些需要注意的点

基础

1、文件系统：主要就是要了解下 inode，block，superblock，block bitmap 这些关键词的含义。

2、目录的大致含义：例如像 proc 这个目录就放一些进程相关的信息，默认软件安装在 usr 目录等等，这其实是一种规范，这个要大致了解下。

3、文件一些属性：例如有些文件可读，可写，可执行，对应 rwx 权限这些， 通过 d 可以知道这个一个目录等等。

4、软连接与硬链接

5、VIM 的几种模式区别了解一一下

进程与内存等(这块是重点)

1、几个进程命令：ps，top，pstree，netstat，这几个进程要自己用一用，工作后经常用到的，重点学习了解下。

2、Linux 进程的一些状态，记得好几种：R，D，S，Z，T 这五种，要了解下。

3、孤儿进程，僵尸进程 要知道（重要），然后还得知道如何解决这种情况，例如出现大量僵尸进程怎么办

4、中断相关：软中断啊，不可中断状态啊

5、内存中的 Cache 和 buffer 的区别

6、了解 Swap 区

7、SiGCHLD 信号，waitpid() 和 wait() 函数

几种重要的 IO 模型

1、同步，异步。

2、阻塞与非阻塞

3、select，epoll

4、NOI与IO

一些命令

如果可以，find，grep 这几个搜索相关的命令学下，因为项目会打印日志，我们需要捞日志来分析，就会经常用到 find,grep 这些，到时候面试官问到你也可以说。

项目故障排查

这块其实就是和实际工作比较靠近，做项目有时候会用到啥的，上面的优先学的，有空了再学一学这些

1、通过工具来追踪进程的栈轨迹，就是分析异常的原因，例如用 top 和 jstakc 来定位

2、掌握一些分析进程 IO，CUP，内存的工具，例如 pidstat

3、用 jsp，jstack 来分析死锁原因，一般就是通过栈信息来查看。

学习资料指导

有时间 + 零基础的，就看下《鸟哥Linux私房菜》这本书吧，讲的很细，可以看下，跟着做实验，容易做的实验可以跟着做，麻烦的就不需要了，看前面基础的部分就行，后面的可以不看了。

但是呢，这本书只是入门，其实很多没讲的，例如 select，epoll，僵尸进程啥的啊，这种一般，你们哪个不同就看找对应的文章就行，通过文字来逐个了解突破哦。

暂时先说这么多，后面再慢慢补充了，我说的这些都是我自己当时学过的，可能不是那么全，但是核心的基本都说了。

九、项目

项目是必须要做的了，Java 的项目教程满天飞，不过 C++ 的会少一些，至少没那么多培训机构视频可以白嫖，不过大家可以跟着书，或者 github 上找或者自己花点钱买一个付费视频吧。

我感觉至少做一个吧，比如做一个聊天系统也行，反正都行吧。

十、学习顺序

我建议有时间的，可以先入门下 C++ ，然后就是开始学习数据结构与算法，算法这块长期保持刷题，然后一边深入学习 C++，之后学习计算机网络，操作系统，在之后学习网络编程，项目这块放到最后面。

如果时间比较紧的，算法这块可以放松一点，C++ 和项目可以优先，计算机基础可以突击学习，通过视频或者别人总结的笔记突击。

总之，这一套学下来，感觉需要一年了，当然，这个不好衡量，还得看你自己掌握了哪些基础。

前端学习路线

虽然帅地的技术栈是后端开发，不过后端和前端显然是一对的，还是经常要和前端打交道，所以在之前也学习过不少前端的知识，下面总结的这套前端学习路线，是我参考了别人大量的学习经历 + 自己的思考 + 咨询前端大佬后提取出来的，可能不会像别人一样写一大堆，但你按照这个路线学习之后，应该也不算太差。

一、入门前端三剑客

前端和后端相比，需要学习的知识还是要少很多，如果你要入门后端，你可以要学习一大堆只是，但是你入门前端，只要把 HTML + CSS + JavaScript 这三门知识学习了，就基本差不多入门前端了，并且可以利用这三门知识，写出很多漂亮的交互页面，下面讲一讲这三门知识的学习。

1、**HTML**

html 学起来还是挺简单的，无论你是否有编程基础，我觉得都可以快速入门，对于新手，我推荐找个入门的视频看一下，然后跟着视频打代码就可以了，入门教程随便在慕课网啥找个免费的视频就可以了，随便搜索「html入门」即可，或者看菜鸟教程的一个入门教程也行.

看完视频，也可以看一下文字版的教程，不知道大家有没有看过阮一峰写过的教程，我觉得他写的教程都很棒，所以这里我也推荐下大家看一下阮一峰写的这份 HTML 入门教程，可能可以让你理解的更加透彻：https://wangdoc.com/html/

大家切勿眼高手低，一定要跟着视频或者书籍上的案例打代码，写代码的编辑器，我推荐 VS-Code，反正选一款你喜欢的编辑器就行，现在这些编辑器都有很多插件，像 VS-Code 这些编辑器，用好一款就可以写各种语言的代码了。

2、CSS

没啥好说的，和 HTML 相辅相成，你在学习 HTML 的过程中，其实也一直都在接触 CSS，所以你学了 HTML 之后，感觉几个小时就可以学完 CSS 了，还是一样，推荐找个视频快速入门，不多说。

html 是一门标签语言，里面有各种各样的标签，很多初学者学了之后可能就把这些标签忘了，**有人可能会问，老是忘了怎么办?**

忘了就忘了，没事的，不需要强行记住这些标签，你需要的是：脑子里有个印象，当你这实现某个功能的时候，你知道 HTML 有某个标签可以实现这样的功能就可以了，然后翻开对应的教程，你能够根据教程使用这个标签即可。

用的次数多了，也就记住了，所以学完 HTML 和 CSS，一定要多多实践，随便打开一个网页，对着葫芦画瓢，自己写一个和它类似的就可以了。

3、JavaScript

比起 HTML 和 CSS，JavaScript 会难一些，不过如果你有其他编程语言基础，例如学过 C 语言，Python 或者 Java 啥的，那学期 JavaScript 也是分分钟的事。

前面的 HTML 和 CSS，我的推荐大家找个视频快速入门即可，但是对于 JavaScript，假如你时间不是很紧，那么我推荐你用书籍系统学一下，有些知识，趁着有时间，一定要系统学，这样可以打下很深的底子，如果你觉得难的话，也可以先用视频快速入门，之后再回过头来钻研书籍，系统过一遍，通过系统学习，你会明白很多原理，学到很多设计思想，我看过一本《JavaScript 高级程序设计》，感觉还好，就推荐这一本吧。

学了 JavaScript 之后，可以学一学 ES6，面试貌似也经常会问到，可以看一看阮一峰写的一份入门教程：https://wangdoc.com/es6/

三大件的面试也整理了：

CSS 基础面试题阅读指南（必看）

HTML 基础面试题阅读指南（必看）

JavaScript基础面试题阅读指南（必看）

二、框架

目前前端用的比较多的主要有 Vue 和 React ,在学习框架之前，一定要先用 HTML + CSS + JavaScript 这些做一些项目，因为这些框架的底层实现，其实就是 JavaScript 实现的，然而，居然还有人知道如何使用 Vue，但没学过 JavaScript，这显然不合适，只有你体验过 HTML + CSS + JavaScript ，你才能更好着明白 Vue 和 React 的好处。

不过对于初学者，可以先学习 Vue，Vue 可能更好入门一些，之前实习的时候，被迫学习了几天的 vue，我是在慕课网先快速入门看的，学起来不难，就是细节容易忘，入门课程直达： https://www.imooc.com/coursescore/980

不过你看了课程之后，你去做项目的话，其实还是会遇到好多问题的，特别是在网络请求那块，在解决问题的过程中，你就能更加理解一些原理了。

学到什么程度?

对于初学者，我的一半建议就是，跟着一份教程，过一遍，然后做对应的项目即可，之后遇到啥，就去搜索啥，不用纠结这个学到什么程度。

Vue 和 React 随便选一个重点学习即可，我推荐 vue，不过学了 Vue 之后，有时间的话，我建议可以了解下 React，快入通过视频了解，感觉几个小时或者一两天就够了。

另外，相关面试题帅地也有整理：Vue基础面试题阅读指南（必看）

三、数据结构

无论你是走什么岗位，数据结构都是必须学习的一门课程，从面试的角度来讲，面试基本比问，特别是校招；从个人的提升上来看，学好数据结构与算法，可以让你走的更远。

但是，数据结构与算法这玩意，可深可浅，不过我觉得至少得掌握如下基础知识：

1、时间复杂度、空间复杂度

2、链表、队列、栈

3、树：初级：二叉树，查找二叉树，进阶：AVL树，红黑树等，至少掌握初级吧。

4、图（图有好多种算法，深度/广度搜索，最短路径、最小生存树等），对于图，其实无论是面试还是工作，都挺少用到，学起来也有一定难度，假如你时间不多，我觉得可以先不学。

不过如果你是科班的，那么这些我觉得你大一第二学期把这些都学完是最好的了，没学完也问题不大，有些人可能是先教《离散数学》这本课，为数据结构与算法做铺垫。

书籍推荐：你学过 JavaScript，所以可以用 Javascript 来写这些数据结构，至于书籍，其实我也不知道推荐啥，网上根据目录找一本。

把基础数据结构学了之后，我觉得你要保持刷题，这个还是挺重要的，例如可以每天保持刷一两道，刚开始刷会挺吃力，但后面熟练了，就会快很多，不过很多人在吃力的那会，就放弃了，所以也就有了人与人之间的差距。

我觉得至少把《剑指 offer》刷完吧，刷完之后，可以去把 leetcode 中 top 100 的热门题做了就差不多了，前端对算法的考察，相对没有后端严格。

另外，如果时间不多的，那就把《剑指offer》刷个一两遍就行了，只有 剑指offer 的刷题攻略，我也写过：剑指offer刷题攻略

四、计算机网络

无论是前端开发还是后端开发，说到底都是数据通过网络在多台主机之间的交互，而且对于前端，计算机网络的知识，用的可能比后端还多，特别是 HTTP 这块，所以呢，计算机网络必须好好学，而且还得重点学。

入门我推荐《图解 HTTP》，不过看完这本我觉得还不够，可以看《计算机网络：自顶向下》这本书，多看两遍，以后面试就可以和面试杆上了。

一边看一边犯困怎么办？

我的建议是，硬着头皮死磕一边，因为根据读者的反馈，确实有挺多人跟我说这玩意看着好困，不过我当时学习的时候，看着好带劲，就是了解了很多原因，很爽，然而事实是，有些人，看着却是一种煎熬，，，，学计算机网络，就一条主线：理解一台计算机是如何找到另外一台计算机，并且把数据交付给他的，或者你可以看我这篇科普文章：[一文读懂一台计算机是如何把数据发送给另一台计算机的]

也写了一个极简教程：[计算机网络入门简介]

下面说一下需要重点掌握的内容吧：

物理层、链路层：

1. MTU，MAC地址，以太网协议。
2. 广播与 ARP 协议

网络层

1. ip 地址分类
2. IP 地址与 MAC 地址区别
3. 子网划分，子网掩码
4. ICMP 协议及其应用
5. 路由寻址
6. 局域网，广域网区别

传输层（主要就是 TCP）

1. TCP首部报文格式（SYN、ACK、FIN、RST必须知道）
2. TCP滑动窗口原理，TCP 超时重传时间选择
3. TCP 拥塞控制，TCP 流量控制
4. TCP 三次握手与四次挥手以及状态码的变化
5. TCP连接释放中TIME_WAIT状态的作用
6. TCP 粘包，心跳包
7. UDP 如何实现可靠传输
8. UDP 与 TCP 的区别
9. UDP 以及 TCP 的应用场景

应用层

1. DNS 原理以及应用
2. HTTP 报文格式，HTTP1.0、HTTP1.1、HTTP2.0 之间的区别

3. HTTP 请求方法的区别：GET、HEAD、POST、PUT、DELETE
4. HTTP 状态码
5. HTTP 与 HTTPS 的区别
6. 数字证书，对称加密与非对称加密
7. cookie与session区别
8. 输入一个URL到显示页面的流程（越详细越好，搞明白这个，网络这块就差不多了）

学完之后可以刷一刷对应的面试题，可以在帅地的网站看：计算机网络面试真题

五、浏览器工作原理

学前端，基本天天和浏览器打交道，因为网页上的各种界面，都是由浏览器来渲染的，所以还是非常有必要学习一下浏览器相关的知识。

如果你在浏览器按 F12，会出现一个「调试」的界面

里面有很多东西，例如各种网络请求数据，各种脚本数据，感兴趣的话，可以去研究研究。

那么具体要学习哪些呢?

我觉得至少得了解一下本地 cookie ，localStorage，SessionStorage 存储吧，还有就是，如何查看一个 http 的请求状态，浏览器关闭后会做哪些处理之类的。

总的来说，就是，从我们发起一个 http 请求，到页面展示如初，浏览器都经历了哪些逻辑处理?

六、进阶

学完了上面这些，可以学一些帮助我们更好着构建一个前端项目的工具，比较常见的有如下几种：

Node.js：这个必须学，主要就是可以帮助我们很快着构建出一个 web 项目，一条命令就搞定了，入门可以在慕课网看视频，我之前看过一个，顺便推荐一下：

进阶或者更甚层次了解，一般都得看书，自己网上搜一本吧。

Webpack：不同浏览器对 JavaScript 的特性支持的不一致，可以通过构建工具把 JavaScript 代码转换成浏览器能支持的。使用构建工具也能够做到性能优化，比如压缩代码，这个 webpack 可以了解一下，在以后做项目的过程中，还是经常用到 node.js 和 webpack 的，我觉得刚开始会使用就好，后面遇到问题了，在通过问题驱动的方式去深入了家。

七、项目

学了 HTML + CSS + JavaScript 的时候，可以做一些小小的项目，这里我这边暂时没有对应的项目，我 后面补充。

建议项目的话，可以做一做 vue 相关的项目。

八、学习顺序问题

这里讲一下学习顺序的问题，就是说学习了 html + css + javascript 之后，我是先学习前端的一些框架好啊，还是学习数据结构与算法好啊，还是学习计算机网络，浏览器工作原理好呢?

我觉得这个和你时间有关，假如你还是大一大二的话，学校会有数据结构，计算机网络的课程，我觉得跟着学校的顺序学就行，然后的话，像刷题，我觉得有些东西同时做并不会存在矛盾，例如我就建议刷题这个时期，长期保持，然后一边做其他的。

总的来说，我觉得也可以按照我说的这个学习顺序来学，然后算法那一块，当你学习了 Javascript 之后，就可以穿拆整个过程了。

Go开发学习路线

Go 学习路线拖了好久了，主要是 Go 不像 Java，有着丰富的学习资料 + 过来人的各种踩坑，Go 的话，网上的学习资料比较零散，而且很多还是培训机构的，罗列一大堆知识，可能初学者看了和没看一样。

不过嘛，既然要写，很多东西就得调研清楚，我先说下，我写的这个学习路线，基于**校招版**的哦，其实在校招中，语言的占比不大，所以一般我们深入去掌握一门语言就可以了，既然选择了 Go，那也得去掌握下 Go 的语言特性。

下面我先关于 Go 的就业岗位以及校招面试做一些问答。

Go 的岗位多嘛

Go 可以说是最近两年比较火的语言，虽然说比较火，但是岗位数量还是和 Java/C++ 没得比，当然，走 Go 的人也会少一些，至于 Java/C++ 和 Go 的就业比例，说实话，这个不好统计，我也不清楚。

只能说，Go 在大公司可能会多一点。

因为 Go 用在云**原生啊，容器虚拟化，分布式存储**啥的啊，可能会多一些，这这块，一般小公司做的比较少。

另外，如果很多公司之前用的架构不是 Go，你让它重新整理架构，这是一个大工程，并且 Go 的组件没有 Java 这些丰富，所以在很多公司中，Java 依然是主流，Go 在大厂中可能会多一些。

所以我觉得，大家在选择 Go/C++/Java 的时候，最好不要以岗位多不多来选择，如果是基于兴趣是最好的吧，并且在校招中，语言不是重点。

另外，目前我知道的，用 Go 最多的公司就是 **字节跳动** 和 **B站**了。

就业前景

这里我强调一下，不用关心就业前景；不用关心就业前景；不用关心就业前景...

前端，Go/Java/C++，这些都是目前需求量相对比较大且比较主流的岗位，只要你学好，那都可以很香，不用关心天花板这些问题，没啥用，你能走到天花板，那就是牛人了。

当然，如果你说你要走 PHP，net 这些，那我就真的不大建议，倒不是找不到工作，而是选择面会窄，并且一般公司会找有几年开发经验的人。

> 下面开始学学习路线

一、学 Go 之前

我觉得你学习 Go 之前应该也学过其他语言吧？如果你有学过面向对象的语言，再来学 Go，可能可以对 Go 理解的更加深刻。

我之前学过一阵子 Go，学起来还是不难，就是容易把一些语法忘了，所以在学的时候，我希望大家可以前期多做一些练习，主要就是属于语法的使用。

如果你是 0 基础的，有些学起来还是有点小难度，不过也问题不大，多看几遍就好了。

二、Go 入门

说实话，入门这块，没啥好说的，学习任何一门语言都一样，随便找个简易的教程，跟着学就行了，知识点也没啥强调的，反正都得掌握。

一般就是这几个：

1、函数：Go 的函数，还是和 Java 啥的有一点点不一样。

2、指针

3、结构体

4、面向对象的知识以及它的特性（注意和Java/C++的一些区别）

5、IO处理

5、还有一个非常重要的 goroutine 和 channel 这两个。

如果你喜欢看视频，可以在 B 站 或者木刻网找个播放量高的视频就行，这里我就不推荐了。

如果你喜欢看文字，那么可以在菜鸟教程这里找，快速入门还挺不错：Go 语言入门

另外就是关于 Go 的书籍，目前有一本用的人比较多的是《Go程序设计语言》，这本书我看过一部分，个人觉得不适合 0 基础的初学者哈，我还是建议大家入门就找个视频或者找个简易的教程，后面入门之后，再来看这些书。

三、Go web + 网络编程

学完 Go 入门，可以学一学 Go web 开发，看看 Go 都能干些啥，不过学习 Go web 开发之前，最好是有一点前端基础，比如简单看过 html + js + css，这样在学习 Go web 的时候，会容易一些。

Go web 的话，主要就是学习下网络编程，看看 Go 是怎么进行交互的，比如你可以用 Go 做一些小项目，比如 RPC，或者通过 Go web，学习它和 Mysql, redis 这些怎么整合。

关于 Go web 的书籍：《Go web 开发》，这本书不厚，可以跟着里面写一写例子。

另外就是关于 Go 框架的，Go 的框架不算多，不过我觉得你可以基于项目来学一些框架，比如你要做个 xxx 的项目，你一般是跟着教程做的，那么看看这个教程用的啥框架，那么就去跟着学。

一般 Go Web主流的框架有 Iris，Beego， Gin 这几个

四、Go进阶

其实很多语言的学习都是类似的，包括 Java，一般就是先简单入门，看看怎么使用，之后去学习它的一些源码，另外这里我需要强调一个点：

就是要先学 Go web 还是先学 Go 进阶?

我觉得这个和 Java 或者 C++ 类似，就是先学集合的源码之后再去做项目，还是先做项目再回过头来学习源码。

我的答案是：都可以。

按照你当时的节奏以及资料顺序来就行，比如有些人的资料，里面本身就安排好了，或者有些资料在学习 Go 入门的时候，就顺带讲了很多进阶的知识了。

也就是说，没有严格的顺序，都可以，你学的顺畅就行。

至于 Go 进阶，主要有以下需要去深入了解下

1、容器类：比如map，slice这些

2、并发：主要是 channel，goroutine 这些

3、标准库：主要是 context, reflect, unsafe 这些

4、内存管理：垃圾回收这些

这几个基本都是 Go 的核心知识，可以深入去学习下，如同 Java 里的并发，JVM，集合源码啥的。

学习资料：推荐这本电子书，之前学习 Go 时还看过他不少文章，链接：https://golang.design/go-questions/

书籍的话，就推荐：《Go语言设计与实现》，这是一本新出的书，这个人我看过他写的文字，也非常硬核。

五、计算机网络

无论你是从事啥岗位，无论是校招还是社招，计算机网络基本都会问，特特是腾讯，字节，shopee，小米等这些非 Java 系的公司，问的更多。这块认真学，**一个半月**就可以搞定了。

计算机网络就是一堆协议的构成，下面是一些比较重要的知识点，学的时候可以重点关注下。

物理层、链路层：

1. MTU，MAC地址，以太网协议。
2. 广播与 ARP 协议

网络层

1. ip 地址分类
2. IP 地址与 MAC 地址区别
3. 子网划分，子网掩码
4. ICMP 协议及其应用
5. 路由寻址
6. 局域网，广域网区别

传输层（主要就是 TCP）

1. TCP首部报文格式（SYN、ACK、FIN、RST必须知道）
2. TCP滑动窗口原理，TCP 超时重传时间选择
3. TCP 拥塞控制，TCP 流量控制
4. TCP 三次握手与四次挥手以及状态码的变化
5. TCP连接释放中TIME_WAIT状态的作用
6. SYN 泛洪攻击
7. TCP 粘包，心跳包
8. UDP 如何实现可靠传输
9. UDP 与 TCP 的区别
10. UDP 以及 TCP 的应用场景

应用层

1. DNS 原理以及应用

2. HTTP 报文格式，HTTP1.0、HTTP1.1、HTTP2.0 之间的区别
3. HTTP 请求方法的区别：GET、HEAD、POST、PUT、DELETE
4. HTTP 状态码
5. HTTP 与 HTTPS 的区别
6. 数字证书，对称加密与非对称加密
7. cookie与session区别
8. 输入一个URL到显示页面的流程（越详细越好，搞明白这个，网络这块就差不多了）

书籍推荐：零基础可以先找个视频看看，然后有了一定基础可以看《计算机网网络：自顶向下》这本书，这本书建议看两遍以及以上，还有时间的可以看《TCP/IP详解卷1：协议》。

六、操作系统

操作系统和计算机网络差不多，不过计算机网络会问的多一些，操作系统会少一些，总结起来大致：

1、进程与线程区别

2、线程同步的方式：互斥锁、自旋锁、读写锁、条件变量

3、互斥锁与自旋锁的底层区别

4、孤儿进程与僵尸进程

5、死锁及避免

6、多线程与多进程比较

7、进程间通信：PIPE、FIFO、消息队列、信号量、共享内存、socket

8、管道与消息队列对比

9、fork进程的底层：读时共享，写时复制

10、线程上下文切换的流程

11、进程上下文切换的流程

12、进程的调度算法

13、阻塞IO与非阻塞IO

14、同步与异步的概念

15、静态链接与动态链接的过程

16、虚拟内存概念（非常重要）

17、MMU地址翻译的具体流程

18、缺页处理过程

19、缺页置换算法：最久未使用算法、先进先出算法、最佳置换算法

书籍推荐：《现代操作系统》

七、MySQL

数据库一般主流的有 MySQL 和 Oracle，不过建议大家学习 MySQL 了，因为大部分公司都是使用 MySQL，也是属于面试必问，而且工作中 MySQL 也是接触的最多的，毕竟工作 crud 才是常态。

下面这些是我认为比较重要的知识点：

1、一条 sql 语句是如何执行的？也就是说，从客户端执行了一条 sql 命令，服务端会进行哪些处理？（例如验证身份，是否启用缓存啥的）。

2、索引相关：索引是如何实现的？多种引擎的实现区别？聚族索引，非聚族索引，二级索引，唯一索引、最左匹配原则等等（非常重要）。

3、事务相关：例如事务的隔离是如何实现的？事务是如何保证原子性？不同的事务看到的数据怎么就不一样了？难道每个事务都拷贝一份视图？MVCC 的实现原理（重要）等等。

4、各种锁相关：例如表锁，行锁，间隙锁，共享锁，排他锁。这些锁的出现主要是用来解决哪些问题？（重要）

5、日志相关：redolog，binlog，undolog，这些日志的实现原理，为了解决怎么问题？日志也是非常重要的吧，面试也问的挺多。

6、数据库的主从备份、如何保证数据不丢失、如何保证高可用等等。

7、一些故障排查的命令，例如慢查询，sql 的执行计划，索引统计的刷新等等。

对于 2-4 这四个相关知识，面试被问到的频率是最高的，有时候面试会让你说一说索引，如果你知道的多的话就可以疯狂扯一波了，具体可以看我写的 MySQL 学习路线：[肝完了，我的 MySQL 学习之路](

六、算法

算法冲击大厂必须学的了，这里不强调了，看我这篇文章： 算法学习路线

七、总结

除了语言和框架部分，计算机基础和算法，不过啥学习路线都是差不多滴。

当然，如果你的目标是面试小公司，那么计算机基础和算法可以少学一点，先把 Go 这块相关的技术栈先学起来哦。大家加油勒！

关于不同学历不同学习时间几个问题

我上面写的学习路线，是比较系统的，也都是重点，不过呢，针对不同的学习阶段，侧重点还是有点不一样的，例如学习很好的，学历差的，大一大二浪过去，大三才开始学习的，已经大四了，系统学习来不及的，等等，所以我这里做一个统一的答复，给大家一个参考。

1、学历比较好，985，头部211

对于学历比较好的，简历面还是比较容易过的，那我觉得你需要侧重把基础 + 算法打好，最重要就是算法，只要你算法学好，就是 offer 收割机，这也是学历带给你的一些光环，那么你没有比赛，没有日常实习，也相对容易在春招拿个实习 offer。

2、学历一般，二本，双非

学历不好，努力来补了，毕竟别人高中努力过了，你欠下的债，得补回来，那对于学历不好的，你需要付出更多努力，你不仅要在大一大二把算法和基础打好，如果有机会，还有参加一些诸如蓝桥杯，leetcode 周赛，项目相关的比赛，并且大三争取能不能去混一个日常实习，总的来说就是，你需要学的更多。

不然，你的简历没啥优势，你得通过别的方式来补，当然，也不要给自己太多压力，就一步一个脚印来。

3、大一大二浪过去了，系统学习来不及了

这个要看情况，如果你觉得我上面列的那些东西学不完，那么我推荐你就不要定位大公司了，而定位中小公司，那么能干活就显的非常重要，那你的学习路线可以先学语言=》框架=》做项目，之后有时间，优先学计网，OS 的高频面试题，然后能够把《剑指 offer 》刷完最好。

但假如你学习能力强，一天能投入七八个小时以上，那么半年时间，可以把我上面说的都学完。

总的来说就是，没时间学基础和算法的，优先保证自己能干活，也就是优先学做项目。

4、没时间，够不着大厂，定位小公司

这种就 很明确了，定位小公司，那就先语言，在框架，最后项目，毕竟不可能人人都进大公司，对于毕业去了小公司的，也不要气馁，如果你愿意，在小公司认真学两年，跳槽大公司还是很大概率的。

校招攻略

只要尽快了解校招，才不过错过关键的招聘节点 + 更加有针对性去学习，主要需要了解的就是：春秋招时间 + 意义=》简历投递方式=〉简历书写，总之，对校招不了解的，建议把这几篇文章看一下，看完大致知道怎么回事了。

关于春秋招的一切

对于日后需要找工作的同学来说，春招/秋招可以说是学生生涯中非常重要的一战，但是我发现很多人却对春秋招的了解甚少，有些人甚至以为是毕业之后再开始找工作，进而导致了找工作的最佳时间段。

说实话，要嘛错过春秋招，要嘛临时抱佛脚准备春秋招的人，我可以说是经常遇到，加上我公众号的读者里，学生较多，所以我觉得有必要写一篇关于春秋招的文章，供大家一个参考。

如果你对春秋招不怎么了解，或者不知道怎么准备春秋招，又或者想看看我是如何准备春秋招的，那么还是建议大家好好看一看这篇文章，我这篇文章会写的比较详细，大家也可以根据目录来看。

1、春招开启时间 + 春招的目的

春招开始的大致时间：每年的 3 月 ～ 5 月。

（1）春招是找实习的最佳时机

对于计算机专业的学生来说，如果学校允许，大部分人都会在大三/研二的暑假去各大公司实习，但是大家必须记住的是，**这个实习 offer 并不是在暑假期间找的，而是在春招期间找到**。

也就说，春招最重要的目的，就是**找实习**，基本各大公司都会在春招期间发出大量的实习 offer，你在 3 ～ 5 月份拿到实习 offer 之后，可以等到暑假再去公司实习。当然，你拿到 实习 offer 之后，也可以马上申请去公司实习，这主要取决于你（当然，公司不给提前去的话，那就没办法了）。

对于毕业后要直接工作的同学来说，我建议你一定一定要参加春招！

如果你能在春招拿到不错的实习 offer，**这不仅让你有了实习经历，甚至还有很大的概率直接在公司转正**，就算在春招没拿到 offer，那么也不亏，通过春招的多轮面试，可以让你积累的很多面试经验。

实不相瞒，我在春招就很惨了，面了几家公司，都由于各种各样的原因被刷了，这让我积累的很多面试经验。我觉得自己能够在秋招的提前批就拿到 offer，春招的面试经历是立了很大功劳的。

今年的春招刚过不久，对于现在大二/研一的同学来说，如果要参加明年的春招，我觉得最好提前准备吧。

（2）春招也是最后一次找工作的机会

对于秋招没有找到工作，或者没有找到满意工作或者考验失败要去找工作的同学来说，那么春招，将是你最后一次以**校招**的身份去找工作了。

不过这个时候，公司只有少量的 offer 了，大量的 offer 都已经在秋招发出去了。当然，竞争的人也少了很多，很多大佬已经在秋招拿了满意的 offer ，不会来跟你竞争了，你的对手估计就是那些考研失败/没有找到工作/没有找到满意工作的学生，所以在最后的这次春招中，找到工作的机会还是非常大了。

所以，对于大四/研三的同学来说，如果在秋招没有找到工作，一定要抓住 3 ～ 5 月份的春招。

2、秋招开启时间 + 春招的目的

秋招开始的大致时间：每年的 7 月 ～ 10 月底。

是的，春招最主要的目的是找实习，并且面向大三/研二；而秋招最重要的目的就是，找**正式工作的 offer**，面向**准大四/研二**的你。

对于毕业后要找工作的同学，你可千万不能错过，各大公司将会在秋招派出大量的 offer，你也会在秋招期间被笔试虐，被面试官虐，还没面试就被刷简历……总之，秋招，真的很累，每天不停笔试面试，不停被刷……当然，大佬除外。

不过这里有个问题需要说下，**秋招 7 月份就开启了，然而我还在公司实习怎么办?**

还能怎么办，当然是一边实习一遍偷偷面试笔试，我那会七月底开始投递简历，八月一边实习一边疯狂参加笔试面试，大家都是这样的，不过大部分公司的笔试，往往都是在晚上，我觉得完全有时间参加；面试的话，有些你也可以和面试官商量面试的时间。

当然，如果你刚好在理想的公司实习，并且感觉自己转正的概率挺大，那么我觉得你还是要好好准备一下转正相关的工作，例如答辩啥的。如果成功转正的话，就舒服很多了，不需要跳入秋招这个火海。

3、春秋招大致流程

春招和秋招的流程其实差不多，大概都是：

（1）投递简历：一般去各大招聘网站投递/官网投递，后面说。

（2）笔试：大部分公司都会有笔试环节，对于很多互联网大厂来说，这个笔试成绩将直接影响你能否进入面试环节，所以说，笔试成绩非常非常重要，并且笔试中最重要的部分就是**编程题**，编程题绝大部份都是**算法题**，例如像字节、拼多多、腾讯这些大公司，笔试可能全部都是算法题，一般 3 ～ 5道，时间 120 分钟。

（3）面试：能够进入面试环节，其实也挺不容易，笔试成绩好，面试机会真的会多很多，笔试做的差，你的面试机会真的会少很多，面试一般是 2～3 轮技术面 + 1 轮 HR 面。并且大部分公司都支持远程面试，持续时间 1 周 ～ 1 个月，如果是现场面试的话，就很快，可能一次性面完技术面。

（4）意向书、录用通知书、签三方啥的，能到这个环节的话，那么恭喜你了，99% 稳了。

4、什么是提前批/内推

（1）内推

对于内推，在当时我也百度过什么是内推，也看过很多回答，然而对内推的含义以及流程啥的，依旧不是很懂。所以我今天就以回答问题的形式谈一谈对内推的理解。

什么是内推

显然，内推就是**内部员工**的推荐。指在求职中，不通过常规的简历投递渠道（包括但不限于网申、双选会、宣讲会现场投递）等方式，而是通过已经在某企业就职的内部员工，将各方面条件优秀的求职者的简历直接投递到HR或部门负责人手中的一种招聘手段。

内推和自己去官网投递有什么区别吗?

对于校招生来说，我认为区别不大。在我看来，内推只是另外一种简历的投递渠道而已，并不存在免笔试/走绿色通道的优势，当然，我指的是在大部分情况下是这样，有个别走绿色通道也是有可能。

到时候你可以去牛客网或者各大招聘公众号看，**内推可以说是满天飞**，基本没啥门槛，扫码/点击链接即可参加内推，我说这些数据只是想跟你说，**通过内推的渠道去投递简历与通过其他渠道去投递简历，区别不大**，千万不要觉得有了内推，感觉自己多了很多优势。

内推依然需要经简历筛选，有些依然需要笔试，是的，依然需要笔试。我秋招投递的简历，80% 都是通过内推链接填写的，但好些依然需要笔试，这个还得看公司，不过通过内推形式的话，你可以找内推人查询自己的简历状态，可能这也是内推的一种好处吧。

所以，总的来说就是，内推就是简历的另外一种投递方式，并不存在多大的优势，但也没啥坏处（除了把你的简历暴露给内推人知道了），所以如果有内推，还是可以参加。

（2）提前批

基本大部分公司都会有提前批，我还是非常建议大家参加提前批，如果你提前批失败了，那么依然不影响你参加正式批，并且各大公司会在提前批发出大量的 offer，我秋招时就是在提前批拿了 offer 之后，就早早结束秋招了。

例如正式批 9 月份开始的话，可能提前批 7 月份就开始了，建议大家不要错过。

有人可能会问，提前批失败了会影响正式批吗?

我认为影响不大，虽然你失败了，有些公司的系统会记录的面试情况，但我认为影响不大。我参加春秋招那会，有人面试阿里，在秋招失败了好几次，最后还是进了，腾讯也是有好多次面试机会，前面面的很差，后面还是进了。也就是说，提前批失败了，比起你多了一次面试机会，那么我会选择多一次面试机会。

怎么说呢，如果面评差的一塌糊涂，那肯定会有所影响，但是如果你觉得自己不至于那么菜，那么就大胆投递，因为我觉得，比起影响正式批，获得一次面试机会的收益是更大的，毕竟很多大公司，会至少把 1/3 的名额给了提前批，加上大部分名额给了实习转正，留给正式批的名额，不多了。

不过我必须说明的是，提前批会比正式批难

是的，提前批高手如云，很多大佬都会参加提前批，并且各大公司在提前批的要求可能也会高一些，毕竟名额有限，大家都想招优秀一点的人，那么提前批肯定会提高要求，反正提前批 offer 有剩的话，我可以在正式批发出，所以提前批会比较难，但这不能阻碍我们去参加提前批，

5、有哪些投递的渠道

对于投递渠道，其实我这还真的没啥渠道，一般就是：牛客网+招聘软件(boss直聘，智联等) + 官网。

这里我必须强调一个点，很多人跟我说，**大公司都没来我们学校宣传，我连投递大公司的机会都没有**。每次看到这样的人，我就知道这人对校招一点也不了解。

大公司的战场，不在校园宣传，而是在网上，基本各大公司，你都可以在网上进行简历投递。实不相瞒，学校的校园招聘，我一场都没参加过，我简历基本都是通过牛客网 + 公众号 + 官网投递的。

我觉得，牛客网 + 公众号，可以满足大部分人了，例如在牛客网上，在求职讨论区那里，会有非常多的内推，建议的多关注。并且里面也汇聚了各大公司的招聘时间等，而且也有很多大佬在那里分享面经。

而公众号的话，如果你对哪个公司感兴趣，完全可以去关注这个公司的公众号，例如你对腾讯感兴趣，可以关注腾讯招聘，例如你对字节感兴趣，可以关注字节招聘，这些公众号，会第一时间告知你校招的开启时间以及简历投递方式。

所以，对于投递渠道，我推荐牛客网（https://www.nowcoder.com/） + 微信公众号，至于一些其他的招聘 app，大家可以去百度搜索下。

6、我是如何准备的春秋招的

实不相瞒，春招我特么没准备过，所以我春招被刷了几家之后，我就意识到自己的问题的，于是从 6 月份我就开始准备了，我准备的是 **Java后端开发**的岗位，如果你也是面试开发岗，那么我的这些你可以做一个参考，我会屏蔽语言特性。

我认为在校招中，最重要的就是**算法 + 基础知识 + 项目**了。并且这三个模块，完全可以同时进行准备。

（1）算法

我刚才说过，对于很多大公司，笔试将直接影响你是否获得面试的机会，而笔试的大部分题目都是算法题，所以算法就显的非常重要了。

不得不说，算法可以说是很多人的软肋，并且算法不容易短时间提升，所以我建议你，一定要长期保持刷题，如果你是要准备即将到来的秋招，我建议你现在就可以刷了，那么刷哪些题呢?

我建议你先刷《剑指 offer》这本书的题，算是对标面试吧，不过有一些题其实也很冷门，不过小部分冷门也问题不大，当然，你也可以跟着我写的攻略刷：剑指offer刷题攻略

刷完剑指 offer，如果你想挑战难一点的，那么我给你推荐另外一本书《程序员代码面试指南:IT名企算法与数据结构题目最优解》，这本书有点难度，不过总结了非常多的题型，适合应付笔试。因为笔试中的题，比面试中的题，难多了。

不过如果你觉得自己刷这本书太吃力了，那么也可以去 leetcode 刷那些 top100 之类的题，反正我是不建议从第 1 题刷到第 n 题这种模式，当然，我这里不是教你怎么学算法，而是如果你不知道怎么准备校招的话，或许可以按照我说的这个来。

总的来说就是：先刷剑指 offer，不过剑指 offer 应付面试还好，应付笔试我觉得不行，而《程序员代码面试指南:IT名企算法与数据结构题目最优解》这本书，我觉得应付笔试挺不错，不过挺难。然后如果你 leetcode 那些热门题没刷过的话，我其实也建议你去看一看，一步一步来。

我自己的话，其实就刷了 剑指 offer，然后刷了牛客网那些 sql 的题，之后刷了部分《程序员代码面试指南:IT名企算法与数据结构题目最优解》的题。

（2）基础知识

基础知识我自己其实强调过很多次了，也没什么好说的，我觉得除了深入准备一门语言外，剩下的就是准备基础知识了，需要准备的还挺多的，这里就不多说了，因为我**学习路线**那里写过了。

（5）项目

项目还是非常重要，特别是对于中小公司，这里就不多说了，我前端，C++，Java 的项目也都有提供了哦。

校招简历书写指南

今天我们就来聊一聊在校招时，简历该如何写的问题。说实话，对于简历的书写方式，可能不同的人会有不同的见解，并且不同的面试官/HR在筛选的时候也会有所差异，所以在我看来，不存在一种绝对稳的简历模版。

我在之前校招时也问过很多过来人关于简历的问题，并且我也认识不少当过面试官的大佬，当时也都叫帮忙修改过简历，今天我就根据自己的经验以及大佬们的建议，跟大家说一说简历该如何写，或许可以给你带来一些有益的参考。

> 最后也会给出我自己参加校招时的简历模版 + 大佬们参加校招时的模版

先来回答一个问题：**简历是写一页好还是两页好呢?**

对于这个问题，可能你的师兄师姐会告诉你，最好写一页，面试官没那么多时间去看简历；也有人会告诉你，最好写两页，毕竟我们没拿过什么大奖，只能在简历上多点自己会的东西。

我的答案是，一页两页都可以，只要你把想要表达的东西都表达出来了，那么就可以了，但是不建议第一页写不下，第二页只写了 20%，然后其他留空白，个人建议最好不要留空太多。

简历里该有哪些东西?

对于校招生来说，简历里无非就个人背景、教育背景、基础技能、项目经历、个人荣誉、自我评价这几个模块，其中最重要的就是**基础技能**和**项目经历**这两个模块了，所以本次也重点讲下这两个模块的书写。

一、基础技能的书写

对于想要进大厂的同学来说，基础知识的掌握无非是最重要的一个点，因为大厂更看重你未来的可塑造性，而扎实的基础，便是一个很重要的证明。有人可能会问，什么是基础知识?

基础知识说来说去无非就是我们大学所学的那些课程，例如数据结构与算法、计算机网络、操作系统、数据库、Linux等等。

我看过好多人的简历，很多人在写基础技能的时候，基本都是写的很「简洁」，例如写计算机网络的时候，简历上就单单写了这一行字：掌握计算机网络。

说实话，计算机网络所包含的知识太多了，如果你不是多牛逼的人，我这里是建议大家在写基础技能的时候，写详细一点，例如把你掌握的一些协议写出来：

熟悉 OSI 七层模型和 TCP/IP 四层体系分层结构，掌握常见网络协议，如 ARP、ICMP、TCP、UDP、DNS、DHCP、https 安全工作原理等。

你把这些罗列出来，给人的感觉就是你好像真的掌握一样，并且也方便面试官挑选问题来询问，正常情况下面试官都会挑你懂的技能来问你是不是真的懂，而不是专门找一些简历里不存在的冷门知识来问你。

所以在写基础技能的时候，我建议大家，写的详细一点，把你自己掌握，想要被面试官问的知识罗列出来。

例如你懂数据结构与算法，那么千万别只写「掌握数据结构与算法」，可以把你觉得自己掌握的还不错的算法写出来，例如我就是这样写的：

扎实的数据机构与算法，例如掌握栈、队列、链表、二叉树、图、排序算法、递归、动态规划、分治、回溯等。

因为大家都学过数据结构与算法，可能大部分人都简历里都写着**掌握数据结构与算法**几个字，具体是掌握多少，不同的人是天壤之别的。

所以呢，在校招的简历上，如果你没有什么比较牛逼的，拿到出手的经历，那么我建议你一定要在基础知识这个方向上下功夫，写的时候尽量详细点，给大家截图下我简历上写的（我的简历是 1 页的，所以力求简洁）


说明：如果项目描写比较好，也可以写上去，如果技术比较多，也不用写太多篇幅，自己把握空间。

专业技能

- 熟练掌握 Java 基础、面向对象思想，熟练使用集合等基础框架。
- 熟悉 Java 多线程与并发底层原理，如 volatile、synchronized、ReentrantLock 等。
- 初步理解 JVM 原理，如类加载机制，内存结构，垃圾回收机制等。
- 熟悉掌握数据结构与常见算法，如链表、二叉树、哈希表、搜索算法、排序算法等。
- 熟悉计算机网络常见协议，掌握 HTTP、TCP、UDP、IP、DNS、ICMP 等网络协议。
- 熟悉 MySQL 数据库的基本原理与使用，如索引原理、锁机制等。
- 了解缓存中间件 Redis 基本内容,如基本数据结构,缓存常见问题等。
- 了解消息队列 MQ，会使用 SpringBoot、SpringMVC、Mybatis 等框架

说明：写了的就一定要负责，写的实话尽量整齐，一个知识点写样就行，不要出现有些贼长有些贼短，也不用写太多，个人觉写 6~8 行就行。

我在秋招那会，面试官问基础技能时，也大部分从我的简历上有列举的知识进行提问，然后延伸。这里需要强调的是，简历上写出来的，一定自己要懂。之前我 Linux 的一些命令写在简历上，结果被面试官一问，特么我刚好那个命令还真没用过，把我尴尬死了。

二、项目书写

之前我看过几个要参加春秋招小伙伴的一些简历，经常看到这样的情况：

1、有些人在写项目的时候，项目背景说了一大堆，然后自己做了什么，却是一笔带过，我竟无法从他的项目经历中找出他究竟做了啥。

2、也有很多人问我，自己没有实习项目或者没有真实的线上项目怎么办?

对于校招生来说，其他很多人项目都接触的比较少，并且很多大公司在这方面也不会对你要求太苛刻，所以我觉得，对于一个项目最重要的不是这个项目有多高大上，而是你在这个项目中**自己**做了什么，用了什么技术解决了什么问题。

并且对于一个项目，我们也可以做一些适当的包装，然后自己准备好相应的面试题，例如你的项目是跟着视频做的，如果觉得自己掌握的不错，其实也是可以包装成自己的实习项目。

在写项目的时候，比较重要的几个点就是：

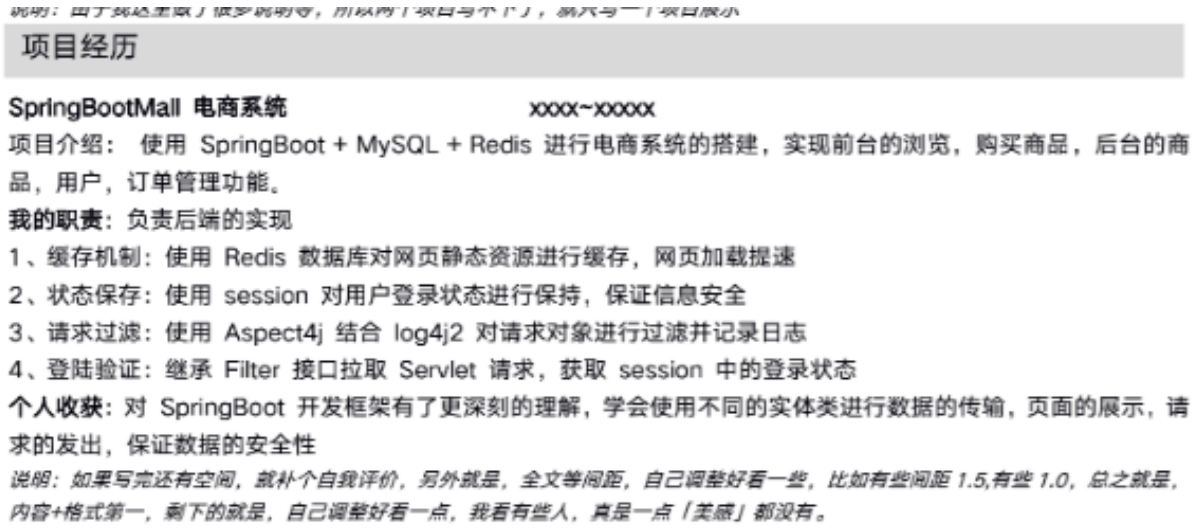
（1）、使用技术：把自己在项目中用到的技术写出来，例如是否用了 redis，还是用了 spring，springcloud 等等。

（2）、简洁介绍下项目：我猜大多数人的项目都是比较简单的，所以在介绍的时候，可以简单介绍了项目的功能、背景之类的，这方面别写太多。

（3）、我的职责：我觉得这个是最重要的，就是你干了什么，用了什么技术解决了什么问题，并且面试官大概率会从你解决的这些问题中问你问题。例如当时我在项目中就说了用 redis 来充当消息队列，解决异步问题，然后自己也准备的相关的面试题，之后面试官刚好就这个消息队列的问题问了挺多，然后就……

也就是说，在写项目的时候，用到的技术，以及是自己做了什么，解决了什么问题，是最重要的一部分吧，因为简历就是用来展示我们自己的，所以必须要让面试官看到我们会什么，做了什么。

这里我也展示一下我写的模版



三、个人荣誉

如果你有获得什么好的证书或者比赛，那么可以在这一栏中写进去，例如你的英语六级成绩好不错，那么可以把自己通过六级考试这个事情写进去。但是如果你像我一样，只考了四级，并且四级成绩还贼差，那么就不建议把四级写进去了，因为**通过英语四级**这个事情，是个本科应该大部分都有这个要求？所以还是不写吧。

还有那些你参加过一些社交活动啊，知识答辩这些证书啊，也没必要写了。写上去的那些荣誉证书啥的，建议和你的专业挂钩，例如你拿到软考证书，你在学校参加 ACM 拿过奖，自己在网上参加比赛拿过的奖等。

四、自我评价

很多人都会在最后来个自我评价，并且大部分人有写和没写一样，因为他是这样写的：

1、本人热爱编程，喜欢学习，喜欢看书，喜欢分享。

2、喜欢探索、喜欢挑战。

说实话，我觉得这些写了和没写一样，因为凡是个人都具有这样的特性，所以在自我评价这一个模块，如果你有写的话，我建议你写具体点。

例如，你说自己喜欢看书，可以顺便举例下看过的一些书；你喜欢分享，可以说下自己写了多少篇文章；喜欢学习，可以说一说你经常去那些社区，例如 csdn，博客园，stackoverflow。

最后

如果你的简历已经写好了，那么可以根据我这篇文章看看有没有可以改进的地方，如果你的简历还没写，或者想要找个简历来参考。

为啥要做项目 + 项目怎么学 + 问什么 + 怎么写

一、为什么要做项目

如果你觉得自己很牛逼，比如学校非常牛逼，或者拿过什么牛逼的将，那么其实没有项目，也问题不大，即使你的简历比较空，也能获得很多面试机会。

但如果你觉得自己两者都没有，那么项目就显的很重要的。

比如你没有项目这一块，你的简历将会非常空，可能连半页的内容都没有，简历一般比较占位置的就是

1、项目

2、专业技能

所以一个简单的原因就是，项目可以让我们的简历看起来更加丰富。

如果你没有做过项目，突然让你来做个项目，我相信你会一脸懵逼，不知道从哪里做起，但是如果你做过项目，那就不一样了，有些项目虽小，但五脏俱全：表的设计，Dao层，Control层，缓存啊....总之就是，基本各个知识都用到了，而且这些，在真实的企业中，也是有这种类似规范的。

所以你做过项目，意味着，你至少掌握了这些知识的使用，而不是空泛的描述，并且也懂得项目的一些基本架构。

也就是说，做了项目，至少能够证明你是真的学过这些技术的，尽管你可能不知道一些原理。

所以呢，对于普通人来说，做项目，是非常有必要的，特别是小公司，因为小公司希望你能够尽快去干活，而你做过项目，上手会更快。

二、项目主要考察什么

一般来说，大部分人做的项目，特别是 Java 这块的，基本都是跟着视频敲代码，或者跟着书上的一些案例敲代码，毕竟绝大部分人，都是没有实习经历，很难接触到其他的真实代码。

不过我可以和大家说，对于初学者，去企业实习几个月接触的项目，并不一定会比自己跟着视频做的项目学的多，很多时候就是去打杂。

所以大部分人是没有真实那种线上项目的，而这些，大部分面试官是知道的，毕竟面试官也是过来人，所以面试官在问你项目的时候，更多的还是问你一些「理论」相关的知识，也就是说，大部分项目的考察，本质上是对基础知识的考察。

比如你项目用到了 Redis 缓存，那么面试官就会问你 Redis 有几种数据类型啊，缓存过期了怎么办啊，数据库与缓存一致性怎么解决啊，为啥要使用缓存啊，等等。

如果你回答的不错，那么面试官可能会往更深的问，比如给你出一些难题，问你项目的数据量大了如何处理啊，运行项目的进程突然挂了怎么办啊。

所以呢，项目的考察一般有两种：

1、项目所有到知识的一些基本原理，所以大家如何写了某种技术，那么最好是要对他负责哈，比如用了 Redis，那么就应该好好总结 Redis，用了消息队列，那么就要好好准备下消息队列。

2、基于项目做一些拓展或者故障处理问题：很多项目其实本身不难，难的是数据量大了应该怎么办，所以面试官往往也会问一些，比如知乎某个问题突然访问量非常大，你要怎么设计；还有就是线上故障处理，这个一般就是考察实践了，如果你好好做过自己的项目且在线上运行过，那么就很有可能遇到这种问题，比如 MySQL 某条语句执行的好慢，你怎么排查？MySQL 突然挂了，你应该怎么找到挂的原因?

应该怎么做

我说个实话，如果你觉得自己不会觉得心虚，你项目就算不跟着敲代码，其实也没事，就是只看视频，然后直接把源码导进来，直接研究代码执行逻辑，这样子的话，做起来快很多。

但是呢，容易心虚 + 存在一些风险，比如被问到某个细节，你没印象，那就凉了，所以是，风险与收益共存。

实不相瞒，我之前做项目，是第一个项目开始跟着做，第二个项目觉得很多东西类似，就直接导入代码来运行了。

那我的建议是，大家尽量跟着敲下代码，如果一直出错或者很多重复性自己不想干的，那么复制部分源码。

另外做项目的话，需要注重的就是，某些高频考点知识的运用，比如redis，消息队列，mysql 索引设计理由。因为面试官，对你的项目也不大了解，而一个项目很多东西是共同的，比如 mysql 索引设计，慢查询，这些面试官就算不懂你的项目，那么也可以问。

所以呢，在做项目的时候，你可以多关注下这些很有可能成为考点的知识的使用。

在简历中如何写项目

很多人在写项目的时候，写了和没写一样，比如：

1、项目有订单模块，前端调用对应借口后会把数据返回给前端，然后进行展示

这个其实说和没说一样，这哪个项目不是这样?

我上面说的，很多项目的考察，本质是对项目背后的考察，并且也和大家说了做项目，就是体现你确实掌握的一些技术。

所以我们在描述的时候，就要具体一点，就是要说明你用了啥技术，并且这个技术解决了什么问题，比如你利用 Redis 中有序集合实现了一个排行榜系统。这种就很具体，面试官也方便问，问你有序集合的底层结构，问你跳表。

所以你在写项目的时候，完全可以引导面试官问你哪些问题，预判面试官会问的问题，然后提前去准备好对应的面试题。

比如很多项目都会有登录功能，那么可以说用 cookie 和 session 解决了登陆态问题，用 MD5 + 盐值 解决了用户密码加密问题，你说了这些，面试官肯定就会问你相关情况了，比如问你，分布式 session 又该如何解决呢？浏览器突然关闭了，你又如何让用户自动退出呢?

总之就是，你需要写的是：你用 xxx 技术解决了 yyy 问题，并且这个 xxx 技术是你希望面试官会问的。

最后，大家希望做项目，写项目的时候，能够好好思考一下！

笔试注意事项

不过本次文章主要来大家讲一讲**笔试**的一些注意事项，帅地是根据自己的校招笔试经历总结出来的几个注意事项，希望大家能够少走一些坑。

一、平时训练时不要在 IDE 打代码

很多人在 LeetCode 或者牛客网做算法题的时候，可能都喜欢直接在 IDE 写好代码，然后再粘贴到网页上运行。

但我的建议是，你要适应在网页上直接打代码，不要依赖 IDE 这些工具。

为什么?

这个我就有过教训了，之前春招面试的时候，面试官是和我共享浏览器网页的，所以需要我在网页上看我打代码，不能在 IDE 上打，我用的是 Java，当我没有了 IDE 这些工具，在网页上写的代码，错误一大堆。

要嘛错别字，要嘛忘记了某个**方法**的名字，甚至输入输出函数具体类名都忘了，，，总之就是，没有了 IDE 的代码自动提示，不仅写得慢，还错误一大堆。

所以，我建议大家平时训练时不要在 IDE 上打代码。

而且也有部分读者跟我反馈，也有部分公司在笔试的时候，也是不准从浏览器切屏到 IDE 的。

不过这里需要说明的时候，网页端的编辑器，好多还是有**代码高亮**功能的，就像牛客网和 LeetCode 都是有代码高亮功能，直接在里面写代码也是很快的。

所以，如果你还在依赖 IDE，那么帅地建议你，在这段时间，把习惯改过来，免得秋招吃了大亏。

二、笔试的代码不要发给其他人

笔试有两种题型，一种是选择/填空题，一种是算法编程题，前者是不允许你切换界面的，担心你作弊，后面大部分公司是支持你切换界面到 IDE 中写代码和调试的，当然，也有部分不允许。

居然可以切换界面，意味着可以切换到微信之类的，有时候你做好了一道算法题，你的朋友或者同学会让你把代码给他，你有时就直接发给他了。

但我的建议是，代码**尽量**不要给其他人，主要担心你们代码的相似度太高，会被判作弊。

你可能会说，我把变量名改一下不就好了？

改变量名用处不大的，如果你学过编译原理，那么你应该知道，代码会被编译成一棵**语法树**，相似度可以通过语法树来判断，所以改变量名还是有风险的，如果实在要给，我建议是改一下一些逻辑的顺序。

但是还是尽量不要给，我秋招那会，就有一个朋友，腾讯笔试时，做好之后，把代码给他同学了，然后被判为作弊了，被腾讯拉黑了。

然后我替他咨询了下公司的 HR，说是不会永久拉黑的，一般这种的话，1-3年就会释放还是啥。

当然，被判为作弊，还是小概率事件，因为有些题，解法很相似，写起来代码还是很像的，所以不会轻易判断你作弊的，就怕万一。

所以我的建议就是，对你心仪的公司，如果自己算法题能过做出来，那么代码尽量不要给其他人，除非你是笔试很菜，一群人团队作案，那就另当别论。

三、总结

关于笔试，有一说一，还是挺难，大家这段时间，一定要多练练手感，大家加油，等你们的好消息！

关于春招的几个常见问题

不少小伙伴都咨询了我校招相关的问题，其中不乏迷茫，没准备好，要不要参加春招，该如何复习，面试都问什么...等等。所以这篇文章，帅地采用自问自答的形式来回答几个共性问题，如果你还是在校生，或者刚毕业不久，或者想要参加面试，那么可以看一看。

这些问题我先汇总一下，方便你们根据问题查找对应的回答。

1、春招面试官都会问什么？

2、春招开始了，我好多还没有复习，要不要先不参加春招，全力准备秋招？

3、面试问项目，我没有实际的项目经验怎么办？

4、春招暑假实习，留言转正的概率大吗?

5、之前算法题刷的很少，眼下算法该如何准备?

6、目前在一家外包公司实习，不过学到的东西不多，是要继续实习还是辞职全力备战春招?

7、笔试好难啊，有什么技巧吗，有人作弊吗?

8、之前有过实习经验，我不参与春招实习了，全力备战秋招可以吗?

1、春招面试官都会问什么?

无论是春招还是秋招，基本都是问算法+基础知识+项目+语言相关，下面我详细回答下各种知识点，方便你准备

（1）算法题：面试算法题其实问的不是很多，可能总共就问个两三道（字节跳动除外），并且问的难度都不难，感觉就 leetcode 中的 easy 和 medium 难度吧，并且很多都是原题。

（2）基础知识：一般就问计算机网络会多一些，然后就是操作系统和 Mysql，其实计算机网络问来问去就那几个问题，大家可以多看看，操作系统就更加死板了，就那几个，至于是哪几个？牛课网去看看面经总结下就好了，至于 mysql 的话，索引是重点。

（3）项目：我觉得春招实习，项目问的还是挺多的，甚至比秋招还要多，所以大家一定要亲手做一个项目，一般面试官会根据你项目涉及到的技术栈，问你一些相关的问题，例如你用了 redis，那么会问你 redis 啥的，还会问你如何优化之类的。

（4）语言相关：一般就Java，C++，Go，Python 这几门，你学哪一种，和面试官说下就好了。如果你是学 Java 的，那么多线程，哈希表，垃圾回收，这些就要注意了。

2、春招开始了，我好多还没有复习，要不要先不参加春招，全力准备秋招?

别问，问就是**一定要参加春招实习，哪怕你啥也没准备**，一定要把春招作为练手，拿不拿的到 offer 倒没关系，至少可以涨一波经验，这会在秋招，给你带来非常非常多的帮助，因为你会发现，面试问来问去就哪些问题，面多了，在一次又一次的总结之后，回答起来会变的很有逻辑。

有人可能会说，我不参加春招，我准备好之后再参加日常实习中，也可以涨面试经验啊?

个人认为，日常实习和春招还是有区别，春招会比日常实习难，而且春招非常非常贴近秋招，所以我还是建议你去参加春招，如果还没有准备，如今最重要的就是，先写好一份简历，然后去各大官网网站投递。

3、面试问项目，我没有实际的项目经验怎么办?

不需要公司的真实项目，你的对手也大部分是没有真实的线上项目的，自己随便找个项目做就可以了，现在不是有很多培训班的免费视频吗？你可以去找一个来做，也可以去慕课网，牛客网这些，付费几百块钱，找一个来做，也就几百块，投资下自己，还是非常值得的。

如果你对应的技术栈学习了，其实全力以赴，一个星期就可以跟着视频，做完一个项目，剩下的就是面试过程不断被虐，然后不断涨经验，不断变强。

4、春招暑假实习，留言转正的概率大吗?

首先，我想说的是，不管概率大不大，你都要一边准备秋招，是的，一边实习，一边准备秋招，到时候还得一边实习一边偷偷面试，手里多拿几个 offer，总归比较放心。

至于转正概率？我认为还是挺大的，每个公司的概率不一样，具体还得看公司招了多少实习生，秋招有多少 HC 分给了实习生，像今年腾讯招了 8000 实习生，那转正概率肯定比去年低。

总的来说，按照往年的一些数据，我觉得转正的比例有 1/2～1/4。

5、之前算法题刷的很少，眼下算法该如何准备?

如果春招来不及准备了，大家可以在面试之前，把一些简单且高频的算法面试题撸几遍，例如这几道：快速排序，归并排序，希尔排序，链表反转，指针指向链表的中间节点/倒数第二个节点，两数之和，还有几道简单的 DP 问题：例如 不同路径（leetcode 62题），最小路径和（leetcode 64题）。

并且一遍把剑指offer 刷个两三遍，如果你没刷过剑指 offer，可能是来不及刷了，但是可以为秋招准备。

6、目前在一家外包公司实习，不过学到的东西不多，是要继续实习还是辞职全力备战春招?

我先说一件事，就是无论你是否继续实习，都要先去投递简历，参加春招，先看看自己被发起面试的概率大不大。

假如你已经实习了几个月，也算有点实习经验了，并且觉得自己能够静下心来复习，我觉得离职出来全力准备春招/秋招也是可以的，但是如果你刚入职，并且实力/学历都很一般，那么还是可以一边实习一边先工作一两个月再离职出来全力准备秋招。

7、笔试好难啊，有什么技巧吗，有人作弊吗?

我跟大家说的实话，在春秋招的笔试中，确实挺难，有很多人都是组队一起做的，在牛客网做编程题的实话，是可以切换屏幕的，也就是说，可以从浏览器切换到别的软件应用，这个时侯，是可以做一些手脚的。虽然有摄像头，但是摄像头死角太多，也就是说，笔试作弊的人还是挺多的，而且好多还是同一个实验室的人。

关于笔试作弊这个事情，如果被发现，后果 还是挺严重，之前我们那一届，就有好几组人，腾讯笔试时，被发现了，直接被腾讯拉近黑名单了，因为**诚实**这玩意，太重要了。

但是，帅地想说的是，大家好好把握，帅地其实也作弊过，但是我不是团队的，我是自己一个人，团队最容易的就是，代码相似度太高，容易被检测出来，有人可能会说，自己一个人也能作弊？我只能说，自己好好研究吧，作为一个博主，在这里倡导大家作弊显然是不对的，我还是建议大家别作弊，你们自己好好体会，好好研究属于自己的方法论吧，这个方法论，也可以为了秋招笔试而研究。

8、之前有过实习经验，我不参与春招实习了，全力备战秋招可以吗?

假如你之前有去过不错的公司实习过，并且春招没有拿到大厂实习 offer，我觉得是可以全力备战秋招的，好好补充自己薄弱的环节，直奔秋招提前批（七月份就开始了），但是，还是那句话，一样可以去投递下简历，以放松的心情去参加下面试/笔试。

上面这些问题，是很多读者在微信问了问题之后，我总结提炼出来的共性问题，希望可以给还处于迷茫的你的一个参考。

学到什么程度可以找日常实习?

之后就有好些人问帅地：学到啥程度才能拿到面试日常实习的岗位啊?

学到什么程度才能xxx，这种问题我被问的太多了，例如算法学到什么程度才能通过笔试……

事实上，这是一个很难衡量的问题，面试能否拿到 offer，还和运气，公司剩下的名额，你是否符合面试官胃口等等相关。

不过今天咱们不去具体衡量这些，今天就和大家**聊聊企业对日常实习生要要求**。

大公司

和春秋招大致一样，大公司招聘日常实习生时，主要考察**基础**以及你的学习能力，而对项目，框架这些，要求较少。

为什么呢?

我说个实话，你们平时做项目用到的技术，去了公司，基本用不到，因为很多大公司都有自己的一套框架，所以你去了公司，大概率需要重新去学习。

既然要重新学习，那么**你的能力**就显的非常重要，能力越好，那么可以上手做任务的速度越快，而**能力**是由多方面构成的，例如你手里已经掌握的知识，你严谨的思维等等。

所以在大公司的日常实习中，你需要向面试官证明，你的能力是可以的。

那么如何证明呢?

证明的方式有很多，例如

1、你拿过一些有挑战性的奖牌，例如 ACM，毫无疑问，这个 ACM 奖牌足以证明你自己，面试官不会拒绝这样的人才。

2、强大的理论基础：你八股文很溜，当然，不是你背的熟悉，而是你真的掌握了，并且能够很好回答面试官的问题。

3、有自己优秀的开源项目：一个项目涉及到方方面面，虽然你其他方面差一些，但这并不影响你的 coding 能力。

等等

你只要掌握的这些其中之一，就算你其他差一些，面试官也会愿意找你，核心就是你向面试官证明了你自己能行，所以呢，在大厂日常实习面试中，**深度很重要，只有深度，才能让面试官更加亲睐你**。

话又说话来，日常实习生无法转正，公司为啥要招日常实习生呢？（当然，部分可以转正）

一般公司在业务比较忙的时候，会招一些**实习生来打杂，干一些简单的事**，对于公司来来说，招日常实习生有如下好处：

1、可以低价招一些人进来干简单的事，这个主要基于：面试官相信在他的培养下，你是可以帮到忙，这也是为啥面试官看重你的能力，以及实习的周长得 3 ～ 6 个月

> 注意：面试日常实习的时候，有时候面试官会直接问你能够实习的时长，这里建议往 6 个月说，尽管你并不会实习 6 个月，先稳住面试，拿到 offer 再说。

2、公司对你的培养，也是对人才的储备，例如你有很大的概率，在毕业后来这家公司，那么公司就可以节省培养的成本。

事实证明，如果实习期间觉得这家公司的文化不错，大部分人也会更加倾向于去实习过的公司，而对于大部分实习生来说，由于关注的因素不多，往往也都会觉得公司环境挺不错。

3、大公司之间抢人才，去了这家公司，日后会有更大的概率，倾向于去这家大公司，那么公司就达到了抢人才的目的。

总之，你不用担心去了啥也没干还拿薪水，资本家自会有他的目的。

小公司

说完了大公司招日常实习生的要求以及目的，我们再来说一说小公司。

小公司没有那么多精力来培养你，而且很多人去小公司，都是当作**跳板**，**混经验**，很大概率不会留下来，所以你去小公司，最好是能一来就干活，不然白白培养你了。

所以你最好是做过项目，掌握了框架，一来培训几天就上手，所以小公司在招聘上，和大公司还是有 很大的区别。

如果你只是会算法这些，估计小公司的不会要你的，尽管你看似挺优秀（因为太优秀的人，他们也留不住）

而且，小公司日常实习生也招的很少，因为价值不大，所以往往是招那些，**愿意转正的实习生**（所以你会看到，很多小公司不会直接给你正式的 offer，而是需要 你先去实习，实习几个月后在转正）

总结

上面总结了大公司与小公司的找人逻辑，如果你想要面试大公司的日常实习，那么你最好有一个可以拿的出来的东西来证明自己，至于八股文用不用掌握的很齐全，这个也是可以不用的，没有项目也没关系，但是得有一些东西来证明你自己。

所以也不要在纠结我应该学到什么程度了，如果你还在纠结这个，不清楚是否已经达到要求了，那么最简单的方式就是，马上写简历，去投递一波，体验下真实的面试。

掌握了这些逻辑，还在大二大三/研一研二的同学，如果学有余力，那么可以考虑去尝试一波日常实习的投递哦。

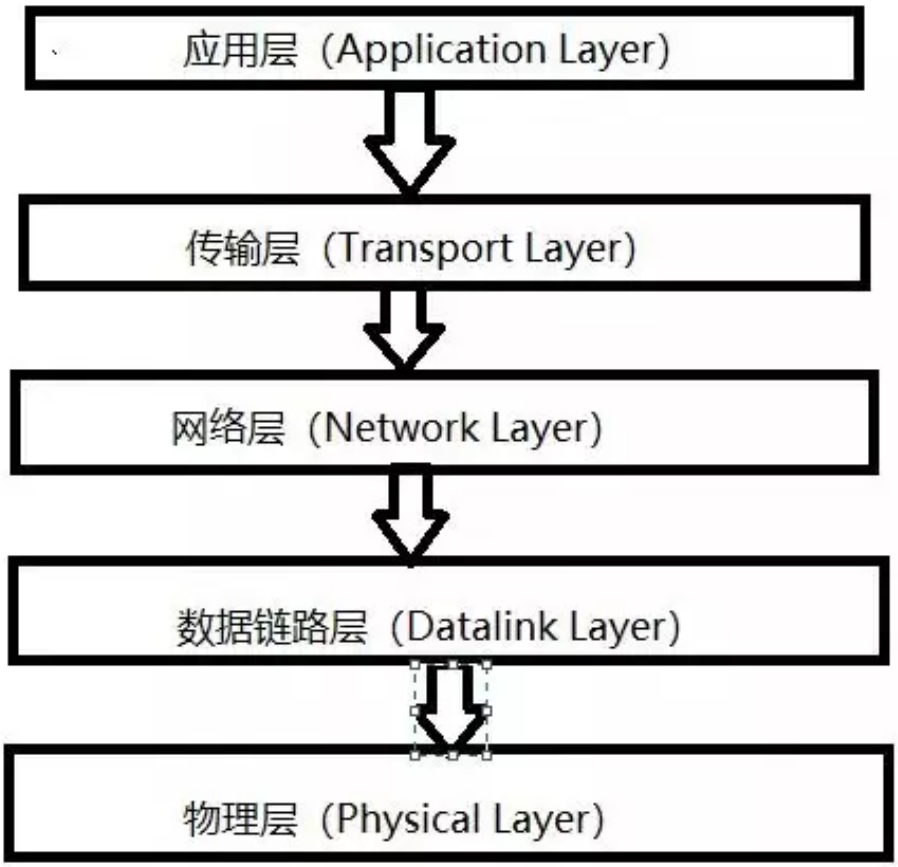
通用计算机技术学习

一文读懂计算机五层网络模型

前言

天各一方的两台计算机是如何通信的呢？在成千上万的计算机中，为什么一台计算机能够准确着寻找到另外一台计算机，并且把数据发送给它呢?

可能很多人都听说过网络通信的 5 层模型，但是可能并不是很清楚为什么需要五层模型，五层模型负责的任务也有可能经常混淆。下面是网络通信的五层模型



说实话，五层模型的具体内容还是极其复杂的，不过今天这篇文章，我将用最简洁的模式，通过网络通信的五层模型来讲解**一台计算机是如何找到另外一台计算机并且把数据发送给另一台计算机的**，就算你没学过计算机网络，也能够听的懂。

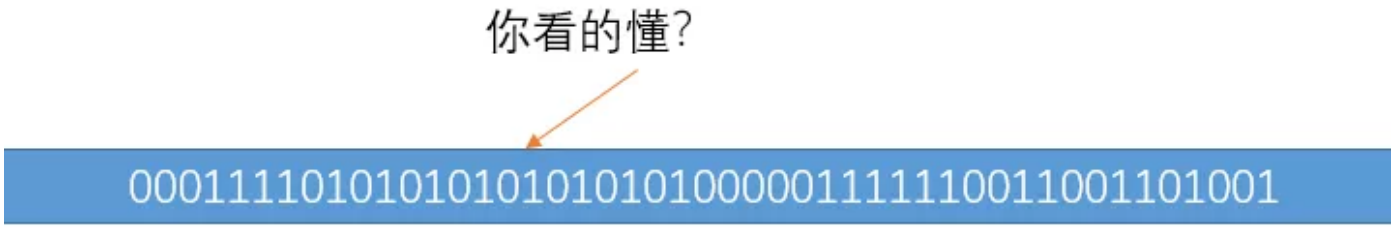
1. 物理层

一台计算机与另一台计算机要进行通信，第一件要做的事是什么？当然是要把这台计算机与另外的其他计算机连起来啊，这样，我们才能把数据传输过去。例如可以通过光纤啊，电缆啊，双绞线啊等介质把他们连接起来，然后才能进行通信。

也就是说，物理层负责把两台计算机连起来，然后在计算机之间通过高低电频来传送0,1这样的电信号。

2. 数据链路层

前面说了，物理层它只是单纯着负责把计算机连接起来，并且在计算机之间传输0，1这样的电信号。如果这些0，1组合的传送毫无规则的话，计算机是解读不了的。一大堆0，1谁知道是什么鬼啊。



因此，我们需要制定一套规则来进行0，1的传送。例如多少个电信号为一组啊，每一组信号应该如何标识才能让计算机读懂啊等等。

于是，有了以太网协议。

1. 以太网协议

以太网协议规定，一组电信号构成一个数据包，我们把这个数据包称之为**帧**。每一个桢由标头(Head)和数据(Data)两部分组成。

帧的大小一般为 64 - 1518 个字节。假如需要传送的数据很大的话，就分成多个桢来进行传送。

对于表头和数据这两个部分，他们存放的都是一些什么数据呢？我猜你眯着眼睛都能想到他们应该放什么数据。　毫无疑问，我们至少得知道这个桢是谁发送，发送给谁的等这些信息吧？所以标头部分主要是一些说明数据，例如发送者，接收者等信息。而数据部分则是这个数据包具体的，想给接守者的内容。

大家想一个问题，一个桢的长度是 64~1518 个字节，也就是说桢的长度不是固定的，那你觉得标头部分的字节长度是固定的吗？它当然是固定的啊，假如不是固定的，每个桢都是单独发的，那计算机怎么知道标头是几个字节，数据是几个字节呢。所以标头部分的字节是固定的，并且固定为18个字节。

把一台计算的的数据通过物理层和链路层发送给另一台计算机，究竟是谁发给谁的，计算机与计算机之间如何区分，，你总得给他们一个唯一的标识吧?

于是，MAC 地址出现了。

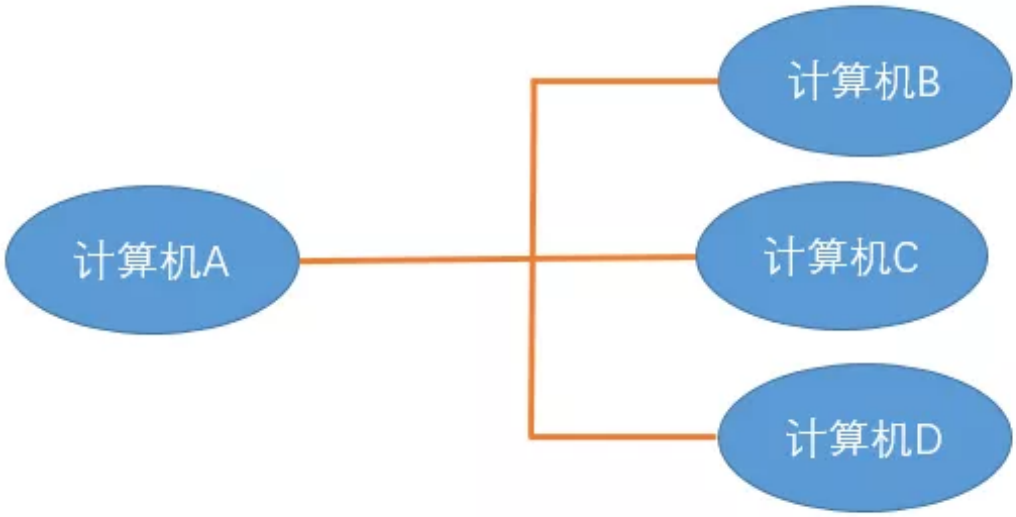
2. MAC 地址

连入网络的每一个计算机都会有网卡接口，每一个网卡都会有一个唯一的地址，这个地址就叫做 MAC 地址。计算机之间的数据传送，就是通过 MAC 地址来唯一寻找、传送的。

MAC地址 由 48 个二进制位所构成，在网卡生产时就被唯一标识了。

3. 广播与ARP协议

(1). 广播



如图，假如计算机 A 知道了计算机 B 的 MAC 地址，然后计算机 A 想要给计算机 B 传送数据，虽然计算机 A 知道了计算机 B 的 MAC 地址，可是它要怎么给它传送数据呢？计算机 A 不仅连着计算机 B，而且计算机 A 也还连着其他的计算机。　虽然计算机 A 知道计算机 B 的 MAC 地址，可是计算机 A 却不知道知道计算机 B 是分布在哪边路线上，为了解决这个问题，于是，有了**广播**的出现。

在同一个**子网**中，计算机 A 要向计算机 B 发送一个数据包，这个数据包会包含接收者的 MAC 地址。当发送时，计算机 A 是通过**广播**的方式发送的，这时同一个子网中的计算机 C, D 也会收到这个数据包的，然后收到这个数据包的计算机，会把数据包的 MAC 地址取出来，与自身的 MAC 地址对比，如果两者相同，则接受这个数据包，否则就丢弃这个数据包。这种发送方式我们称之为广播,就像我们平时在广场上通过广播的形式呼叫某个人一样，如果这个名字是你，你就理会一下，如果不是你，你就当作听不见。

(2). ARP 协议。

那么问题来了，计算机 A 是如何知道计算机 B 的 MAC 地址的呢？这个时候就得由 ARP 协议这个家伙来解决了，不过 ARP 协议会涉及到IP地址，我们下面才会扯到IP地址。因此我们先放着，就当作是有这么一个 ARP 协议，通过它我们可以知道子网中其他计算机的 MAC 地址。

3. 网络层

上面我们有说到子网这个关键词，实际上我们所处的网络，是由无数个子网络构成的。广播的时候，也只有同一个子网里面的计算机能够收到。

假如没有子网这种划分的话，计算机 A 通过广播的方式发一个数据包给计算机 B , 其他所有计算机也都能收到这个数据包，然后进行对比再舍弃。世界上有那么多它计算机，每一台计算机都能收到其他所有计算机的数据包，那就不得了了。那还不得奔溃。 因此产生了**子网**这么一个东西。

那么问题来了，我们如何区分哪些 MAC 地址是属于同一个子网的呢？假如是同一个子网，那我们就用广播的形式把数据传送给对方，如果不是同一个子网的，我们就会把数据发给网关，让网关进行转发。

为了解决这个问题，于是，有了 IP 协议。

1. IP协议

IP协议，它所定义的地址，我们称之为**IP地址**。IP协议有两种版本，一种是 IPv4,另一种是 IPv6。不过我们目前大多数用的还是 IPv4，我们现在也只讨论 IPv4 这个版本的协议。

这个 IP 地址由 32 位的二进制数组成，我们一般把它分成4段的十进制表示，地址范围为0.0.0.0~255.255.255.255。

每一台想要联网的计算机都会有一个IP地址。这个IP地址被分为两部分，前面一部分代表**网络部分**，后面一部分代表**主机部分**。并且网络部分和主机部分所占用的二进制位数是不固定的。

假如两台计算机的网络部分是一模一样的，我们就说这两台计算机是处于同一个子网中。例如 192.168.43.1 和 192.168.43.2, 假如这两个 IP 地址的网络部分为 24 位，主机部分为 8 位。那么他们的网络部分都为 192.168.43，所以他们处于同一个子网中。

可是问题来了，你怎么知道网络部分是占几位，主机部分又是占几位呢？也就是说，单单从两台计算机的IP地址，我们是无法判断他们的是否处于同一个子网中的。

这就引申出了另一个关键词————**子网掩码**。子网掩码和IP地址一样也是 32 位二进制数，不过它的网络部分规定全部为 1，主机部分规定全部为 0.也就是说，假如上面那两个IP地址的网络部分为 24 位，主机部分为 8 位的话，那他们的子网掩码都为11111111.11111111.11111111.00000000，即255.255.255.0。

那有了子网掩码，如何来判端IP地址是否处于同一个子网中呢。显然，知道了子网掩码，相当于我们知道了网络部分是几位，主机部分是几位。我们只需要把 IP 地址与它的子网掩码做与(and)运算，然后把各自的结果进行比较就行了，如果比较的结果相同，则代表是同一个子网，否则不是同一个子网。

例如，192.168.43.1和192.168.43.2的子码掩码都为255.255.255.0，把IP与子码掩码相与，可以得到他们都为192.168.43.0，进而他们处于同一个子网中。

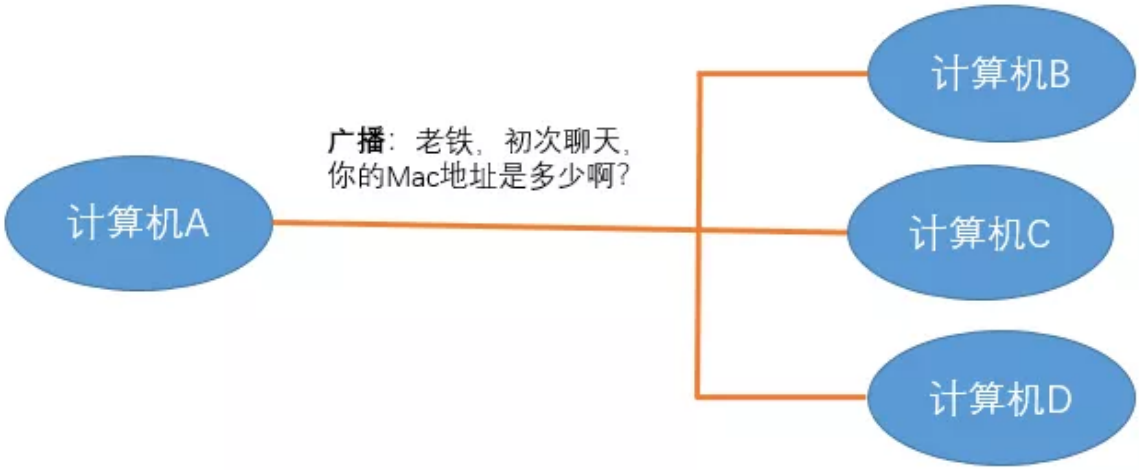
2. ARP协议

有了上面IP协议的知识，我们回来讲一下ARP协议。

有了两台计算机的IP地址与子网掩码，我们就可以判断出它们是否处于同一个子网之中了。

假如他们处于同一个子网之中，计算机A要给计算机B发送数据时。我们可以通过ARP协议来得到计算机B的MAC地址。

ARP协议也是通过广播的形式给同一个子网中的每台电脑发送一个数据包(当然，这个数据包会包含接收方的IP地址)。对方收到这个数据包之后，会取出IP地址与自身的对比，如果相同，则把自己的MAC地址回复给对方，否则就丢弃这个数据包。这样，计算机A就能知道计算机B的MAC地址了。



可能有人会问，知道了MAC地址之后，发送数据是通过广播的形式发送，询问对方的MAC地址也是通过广播的形式来发送，那其他计算机怎么知道你是要传送数据还是要询问MAC地址呢？其实在询问MAC地址的数据包中，在对方的MAC地址这一栏中，填的是一个特殊的MAC地址，其他计算机看到这个特殊的MAC地址之后，就能知道广播想干嘛了。

假如两台计算机的IP不是处于同一个子网之中，这个时候，我们就会把数据包发送给网关，然后让网关让我们进行转发传送

3. DNS服务器

这里再说一个问题，我们是如何知道对方计算机的IP地址的呢？这个问题可能有人会觉得很白痴，心想，当然是计算机的操作者来进行输入了。这没错，当我们想要访问某个网站的时候，我们可以输入IP来进行访问，但是我相信绝大多数人是输入一个网址域名的，例如访问百度是输入 www.baidu.com 这个域名。其实当我们输入这个域名时，会有一个叫做DNS服务器的家伙来帮我们解析这个域名，然后返回这个域名对应的IP给我们的。

因此，网络层的功能就是让我们在茫茫人海中，能够找到另一台计算机在哪里，是否属于同一个子网等。

4. 传输层

通过物理层、数据链路层以及网络层的互相帮助，我们已经把数据成功从计算机A传送到计算机B了，可是，计算机B里面有各种各样的应用程序，计算机该如何知道这些数据是给谁的呢?

这个时候，**端口(Port)**这个家伙就上场了，也就是说，我们在从计算机A传数据给计算表B的时候，还得指定一个端口，以供特定的应用程序来接受处理。

也就是说，传输层的功能就是建立端口到端口的通信。相比网络层的功能是建立主机到主机的通信。

也就是说，只有有了IP和端口，我们才能进行准确着通信。这个时候可能有人会说，我输入IP地址的时候并没有指定一个端口啊。其实呢，对于有些传输协议，已经有设定了一些默认端口了。例如http的传输默认端口是80，这些端口信息也会包含在数据包里的。

传输层最常见的两大协议是 TCP 协议和 UDP 协议，其中 TCP 协议与 UDP 最大的不同就是 TCP 提供可靠的传输，而 UDP 提供的是不可靠传输。

5. 应用层

终于说到应用层了，应用层这一层最接近我们用户了。

虽然我们收到了传输层传来的数据，可是这些传过来的数据五花八门，有html格式的，有mp4格式的，各种各样。你确定你能看的懂?

因此我们需要指定这些数据的格式规则，收到后才好解读渲染。例如我们最常见的 Http 数据包中，就会指定该数据包是 什么格式的文件了。

总结

五层模型至此讲到这里。对于有些层讲的比较简洁，就随便概况了一下。因为如果我说的详细一点的话，篇幅肯定会特别特别长，我着已经是尽最大的努力以最简洁的方式来讲的了。如果你想详细去了解，可以去买计算机网络相应的资料，强烈推荐《计算机网络：自顶向下》这本书。希望我的讲解能让你对计算机之间数据的传输有个大概的了解。

一文读懂进程间通信

有一次面试的时候，被问到进程之间有哪些通信方式，不过由于之前没深入思考且整理过，说的并不好。想必大家也都知道进程有哪些通信方式，可是我猜很多人都是靠着”背“来记忆的，所以今天的这篇文章，讲给大家详细着讲解他们是如何通信的，让大家尽量能够理解他们之间的区别、优缺点等，这样的话，以后面试官让你举例子，你也能够顺手拈来。

1、管道

我们来看一条 Linux 的语句

```
netstat -tulnp | grep 8080
```

学过 Linux 命名的估计都懂这条语句的含义，其中”|“是**管道**的意思，它的作用就是把前一条命令的输出作为后一条命令的输入。在这里就是把 netstat -tulnp 的输出结果作为 grep 8080 这条命令的输入。如果两个进程要进行通信的话，就可以用这种**管道**来进行通信了，并且我们可以知道这条**竖线**是没有名字的，所以我们把这种通信方式称之为**匿名管道**。

并且这种通信方式是**单向**的，只能把第一个命令的输出作为第二个命令的输入，如果进程之间想要互相通信的话，那么需要创建两个管道。

居然有匿名管道，那也意味着有**命名**管道，下面我们来创建一个命名管道。

```
mkfifo  test
```

这条命令创建了一个名字为 test 的命名管道。

接下来我们用一个进程向这个管道里面写数据，然后有另外一个进程把里面的数据读出来。

```
echo "this is a pipe" > test   // 写数据
```

这个时候管道的内容没有被读出的话，那么这个命令就会一直停在这里，只有当另外一个进程把 test 里面的内容读出来的时候这条命令才会结束。接下来我们用另外一个进程来读取

```
cat < test  // 读数据
```

我们可以看到，test 里面的数据被读取出来了。上一条命令也执行结束了。

从上面的例子可以看出，管道的通知机制类似于**缓存**，就像一个进程把数据放在某个缓存区域，然后等着另外一个进程去拿，并且是管道是**单向传输的。**

这种通信方式有什么缺点呢？显然，这种通信方式**效率低下**，你看，a 进程给 b 进程传输数据，只能等待 b 进程取了数据之后 a 进程才能返回。

所以管道不适合频繁通信的进程。当然，他也有它的优点，例如比较简单，能够保证我们的数据已经真的被其他进程拿走了。我们平时用 Linux 的时候，也算是经常用。

2、消息队列

那我们能不能把进程的数据放在某个内存之后就马上让进程返回呢？无需等待其他进程来取就返回呢？

答是可以的，我们可以用**消息队列**的通信模式来解决这个问题，例如 a 进程要给 b 进程发送消息，只需要把消息放在对应的消息队列里就行了，b 进程需要的时候再去对应的消息队列里取出来。同理，b 进程要个 a 进程发送消息也是一样。这种通信方式也类似于**缓存**吧。

这种通信方式有缺点吗？答是有的，如果 a 进程发送的数据占的内存比较大，并且两个进程之间的通信特别频繁的话，消息队列模型就不大适合了。因为 a 发送的数据很大的话，意味**发送消息（拷贝）**这个过程需要花很多时间来读内存。

哪有没有什么解决方案呢？答是有的，请继续往下看。

3、共享内存

共享内存这个通信方式就可以很好着解决**拷贝**所消耗的时间了。

这个可能有人会问了，每个进程不是有自己的独立内存吗？两个进程怎么就可以共享一块内存了？

我们都知道，系统加载一个进程的时候，分配给进程的内存并不是**实际物理内存**，而是**虚拟内存空间**。那么我们可以让两个进程各自拿出一块虚拟地址空间来，然后映射到相同的物理内存中，这样，两个进程虽然有着独立的虚拟内存空间，但有一部分却是映射到相同的物理内存，这就完成了**内存共享**机制了。

4、信号量

共享内存最大的问题是什么？没错，就是多进程竞争内存的问题，就像类似于我们平时说的**线程安全**问题。如何解决这个问题？这个时候我们的**信号量**就上场了。

信号量的本质就是一个计数器，用来实现进程之间的互斥与同步。例如信号量的初始值是 1，然后 a 进程来访问**内存1**的时候，我们就把信号量的值设为 0，然后进程b 也要来访问**内存1**的时候，看到信号量的值为 0 就知道已经有进程在访问**内存1**了，这个时候进程 b 就会访问不了**内存1**。所以说，信号量也是进程之间的一种通信方式。

5、Socket

上面我们说的共享内存、管道、信号量、消息队列，他们都是多个进程在一台主机之间的通信，那两个相隔几千里的进程能够进行通信吗？

答是必须的，这个时候 Socket 这家伙就派上用场了，例如我们平时通过浏览器发起一个 http 请求，然后服务器给你返回对应的数据，这种就是采用 Socket 的通信方式了。

总结

所以，进程之间的通信方式有：

1、管道

2、消息队列

3、共享内存

4、信号量

5、Socket

讲到这里也就完结了，之前我看进程之间的通信方式的时候，也算是死记硬背，并没有去理解他们之间的关系，优缺点，为什么会有这种通信方式。所以最近花点时间去研究了一下，
整理了这篇文章，相信看完这篇文章，你就可以更好着理解各种通信方式的由来的。

腾讯面试：一条SQL语句执行得很慢的原因有哪些?

说实话，这个问题可以涉及到 MySQL 的很多核心知识，可以扯出一大堆，就像要考你计算机网络的知识时，问你“输入URL回车之后，究竟发生了什么”一样，看看你能说出多少了。

之前腾讯面试的实话，也问到这个问题了，不过答的很不好，之前没去想过相关原因，导致一时之间扯不出来。所以今天，我带大家来详细扯一下有哪些原因，相信你看完之后一定会有所收获，不然你打我。

开始装逼：分类讨论

一条 SQL 语句执行的很慢，那是每次执行都很慢呢？还是大多数情况下是正常的，偶尔出现很慢呢？所以我觉得，我们还得分以下两种情况来讨论。

1、大多数情况是正常的，只是偶尔会出现很慢的情况。

2、在数据量不变的情况下，这条SQL语句一直以来都执行的很慢。

针对这两种情况，我们来分析下可能是哪些原因导致的。

针对偶尔很慢的情况

一条 SQL 大多数情况正常，偶尔才能出现很慢的情况，针对这种情况，我觉得这条SQL语句的书写本身是没什么问题的，而是其他原因导致的，那会是什么原因呢?

数据库在刷新脏页我也无奈啊

当我们要往数据库插入一条数据、或者要更新一条数据的时候，我们知道数据库会在**内存**中把对应字段的数据更新了，但是更新之后，这些更新的字段并不会马上同步持久化到**磁盘**中去，而是把这些更新的记录写入到 redo log 日记中去，等到空闲的时候，在通过 redo log 里的日记把最新的数据同步到**磁盘**中去。

不过，redo log 里的容量是有限的，如果数据库一直很忙，更新又很频繁，这个时候 redo log 很快就会被写满了，这个时候就没办法等到空闲的时候再把数据同步到磁盘的，只能暂停其他操作，全身心来把数据同步到磁盘中去的，而这个时候，**就会导致我们平时正常的SQL语句突然执行的很慢**，所以说，数据库在在同步数据到磁盘的时候，就有可能导致我们的SQL语句执行的很慢了。

拿不到锁我能怎么办

这个就比较容易想到了，我们要执行的这条语句，刚好这条语句涉及到的**表**，别人在用，并且加锁了，我们拿不到锁，只能慢慢等待别人释放锁了。或者，表没有加锁，但要使用到的某个一行被加锁了，这个时候，我也没办法啊。

如果要判断是否真的在等待锁，我们可以用 **show processlist**这个命令来查看当前的状态哦，这里我要提醒一下，有些命令最好记录一下，反正，我被问了好几个命令，都不知道怎么写，呵呵。

下来我们来访分析下第二种情况，我觉得第二种情况的分析才是最重要的

针对一直都这么慢的情况

如果在数据量一样大的情况下，这条 SQL 语句每次都执行的这么慢，那就就要好好考虑下你的 SQL 书写了，下面我们来分析下哪些原因会导致我们的 SQL 语句执行的很不理想。

我们先来假设我们有一个表，表里有下面两个字段,分别是主键 id，和两个普通字段 c 和 d。

```
mysql> CREATE TABLE `t` (
  `id` int(11) NOT NULL,
  `c` int(11) DEFAULT NULL,
  `d` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB;
```

扎心了，没用到索引

没有用上索引，我觉得这个原因是很多人都能想到的，例如你要查询这条语句

```
select * from t where 100 <c and c < 100000;
```

字段没有索引

刚好你的 c 字段上没有索引，那么抱歉，只能走全表扫描了，你就体验不会索引带来的乐趣了，所以，这回导致这条查询语句很慢。

字段有索引，但却没有用索引

好吧，这个时候你给 c 这个字段加上了索引，然后又查询了一条语句

```
select * from t where c - 1 = 1000;
```

我想问大家一个问题，这样子在查询的时候会用索引查询吗?

答是不会，如果我们在字段的左边做了运算，那么很抱歉，在查询的时候，就不会用上索引了，所以呢，大家要注意这种**字段上有索引，但由于自己的疏忽，导致系统没有使用索引**的情况了。

正确的查询应该如下

```
select * from t where c = 1000 + 1;
```

有人可能会说，右边有运算就能用上索引？难道数据库就不会自动帮我们优化一下，自动把 c - 1=1000 自动转换为 c = 1000+1。

不好意思，确实不会帮你，所以，你要注意了。

函数操作导致没有用上索引

如果我们在查询的时候，对字段进行了函数操作，也是会导致没有用上索引的，例如

```
select * from t where pow(c,2) = 1000;
```

这里我只是做一个例子，假设函数 pow 是求 c 的 n 次方，实际上可能并没有 pow(c,2) 这个函数。其实这个和上面在左边做运算也是很类似的。

所以呢，一条语句执行都很慢的时候，可能是该语句没有用上索引了，不过具体是啥原因导致没有用上索引的呢，你就要会分析了，我上面列举的三个原因，应该是出现的比较多的吧。

呵呵，数据库自己选错索引了

我们在进行查询操作的时候，例如

```
select * from t where 100 < c and c < 100000;
```

我们知道，主键索引和非主键索引是有区别的，主键索引存放的值是**整行字段的数据**，而非主键索引上存放的值不是整行字段的数据，而且存放**主键字段的值**。不大懂的可以看我这篇文章：面试小知识：MySQL索引相关　里面有说到主键索引和非主键索引的区别

也就是说，我们如果走 c 这个字段的索引的话，最后会查询到对应主键的值，然后，再根据主键的值走主键索引，查询到整行数据返回。

好吧扯了这么多，其实我就是想告诉你，就算你在 c 字段上有索引，系统也并不一定会走 c 这个字段上的索引，而是有可能会直接扫描扫描全表，找出所有符合 100 < c and c < 100000 的数据。

为什么会这样呢?

其实是这样的，系统在执行这条语句的时候，会进行预测：究竟是走 c 索引扫描的行数少，还是直接扫描全表扫描的行数少呢？显然，扫描行数越少当然越好了，因为扫描行数越少，意味着I/O操作的次数越少。

如果是扫描全表的话，那么扫描的次数就是这个表的总行数了，假设为 n；而如果走索引 c 的话，我们通过索引 c 找到主键之后，还得再通过主键索引来找我们整行的数据，也就是说，需要走两次索引。而且，我们也不知道符合 100 c < and c < 10000 这个条件的数据有多少行，万一这个表是全部数据都符合呢？这个时候意味着，走 c 索引不仅扫描的行数是 n，同时还得每行数据走两次索引。

所以呢，系统是有可能走全表扫描而不走索引的。那系统是怎么判断呢?

判断来源于系统的预测，也就是说，如果要走 c 字段索引的话，系统会预测走 c 字段索引大概需要扫描多少行。如果预测到要扫描的行数很多，它可能就不走索引而直接扫描全表了。

那么问题来了，**系统是怎么预测判断的呢?** 这里我给你讲下系统是怎么判断的吧，虽然这个时候我已经写到脖子有点酸了。

系统是通过**索引的区分度**来判断的，一个索引上不同的值越多，意味着出现相同数值的索引越少，意味着索引的区分度越高。我们也把区分度称之为**基数**，即区分度越高，基数越大。所以呢，基数越大，意味着符合 100 < c and c < 10000 这个条件的行数越少。

所以呢，一个索引的基数越大，意味着走索引查询越有优势。

那么问题来了，怎么知道这个索引的基数呢?

系统当然是不会遍历全部来获得一个索引的基数的，代价太大了，索引系统是通过遍历部分数据，也就是通过**采样**的方式，来预测索引的基数的。

扯了这么多，重点的来了，居然是采样，那就有可能出现**失误**的情况，也就是说，c 这个索引的基数实际上是很大的，但是采样的时候，却很不幸，把这个索引的基数预测成很小。例如你采样的那一部分数据刚好基数很小，然后就误以为索引的基数很小。**然后就呵呵，系统就不走 c 索引了，直接走全部扫描了**。

所以呢，说了这么多，得出结论：**由于统计的失误，导致系统没有走索引，而是走了全表扫描**，而这，也是导致我们 SQL 语句执行的很慢的原因。

> 这里我声明一下，系统判断是否走索引，扫描行数的预测其实只是原因之一，这条查询语句是否需要使用使用临时表、是否需要排序等也是会影响系统的选择的。

不过呢，我们有时候也可以通过强制走索引的方式来查询，例如

```
select * from t force index(a) where c < 100 and c < 100000;
```

我们也可以通过

```
show index from t;
```

来查询索引的基数和实际是否符合，如果和实际很不符合的话，我们可以重新来统计索引的基数，可以用这条命令

```
analyze table t;
```

来重新统计分析。

既然会预测错索引的基数，这也意味着，当我们的查询语句有多个索引的时候，系统有可能也会选错索引哦，这也可能是 SQL 执行的很慢的一个原因。

好吧，就先扯这么多了，你到时候能扯出这么多，我觉得已经很棒了，下面做一个总结。

总结

以上是我的总结与理解，最后一个部分，我怕很多人不大懂**数据库居然会选错索引**，所以我详细解释了一下，下面我对以上做一个总结。

一个 SQL 执行的很慢，我们要分两种情况讨论：

1、大多数情况下很正常，偶尔很慢，则有如下原因

(1)、数据库在刷新脏页，例如 redo log 写满了需要同步到磁盘。

(2)、执行的时候，遇到锁，如表锁、行锁。

2、这条 SQL 语句一直执行的很慢，则有如下原因。

(1)、没有用上索引：例如该字段没有索引；由于对字段进行运算、函数操作导致无法用索引。

(2)、数据库选错了索引。

常用算法技巧总结

对于算法技巧，之前的文章也写过一些算法技巧，不过相对零散一些，今天我把之前的很多文章总结了下，并且通过**增删查改**，给大家总结一些常用的**算法解题技巧**，当然，这些也不是多牛逼的技巧，不过可以让你的代码看起来更加短小精悍，如果你能够充分掌握这些技巧，能够混合运用起来，那么写出来的代码，必然可以让别人**拍案叫绝**。

1、多思考能否使用位运算

如果你去看一些大佬的解题代码，你会发现大部分代码里都会出现**位运算**相关的代码，而且不瞒你说，如果我看到一个人的代码里，如果出现了位运算，我就会感觉这个人还是**有点东西**。

最简单地位运算使用场景就是当我们在进行除法和乘法运算的时候了，例如每次遇到 n / 2，n / 4，n / 8这些运算地时候，完全可以使用位运算，也可以使你地代码运行效率更高，例如

n / 2 等价于 n >> 1
n / 4 等价于 n >> 2
n / 8 等价于 n >> 3。

当然，如果你现在去找个 IDE 写个代码测试下 n / 2 和 n >> 1 的运行效率，可能会发现没啥差别，其实并非没有差别，而是大部分编译器会自动帮你把 n / 2 **优化**成 n >> 1，不过我还是建议你写成 n >> 1，这可以让你的代码显的更加牛逼一些，给面试官的印象可能也会好一些。当然，我说的说**可能**。

还有一个非常常用的就是奇偶的判断，判断一个数是否说奇数，常规操作长这样

```
if( n % 2 == 1){
    dosomething();
}
```

不过你可以采用**与**运算来代替 n % 2，改成这样

```
if( (n & 2) == 1){
    dosomething();
}
```

你去看源码的话，基本都是采用这些位运算的，如果你用惯，以后遇到这些代码，看起来也会比较容易懂。

上面列举的这个说最常用的，也说不上什么技巧，不过建议可以多使用熟悉，对于位运算的技巧，我推荐你熟悉如下几个。

1、利用 n & (n - 1)消去 n 最后的一位 1

在 n 的二进制表示中，如果我们对 n 执行

n = n & (n - 1)

那么可以把 n 最右边的 1 消除掉，例如

n = 1001
n - 1 = 1000
n = n & (n - 1) = (1001) & (1000) = 1000

这个公式有哪些用处呢?

其实还是有挺多用处的，在做题的时候也是会经常碰到，下面我列举几道经典、常考的例题。

（1）、判断一个正整数 n 是否为 2 的幂次方

如果一个数是 2 的幂次方，意味着 n 的二进制表示中，只有一个位 是1，其他都是0。我举个例子，例如

2^0 = 0.....0001

2^1 = 0.....0010

2^2 = 0....0100

那么我们完全可以对 n 执行 n = n & (n - 1)，执行之后结果如果不为 0，则代表 n 不是 2 的幂次方，代码如下

```
boolean judege(int n){
     return (n & (n - 1)) == 0;//
}
```

如果你使用常规手段对话，得把 n 不停着除以 2，最后判断得出结果，用这个位运算技巧，一行代码搞定。

（2）、判断 正整数 n 的二进制表示中有多少个 1

例如 n = 13，那么二进制表示为 n = 1101,那么就表示有 3 个1，这道题常规做法还是把 n 不停着除以 2，然后统计除后的结果是否为奇数，是则 1 的个数加 1，否则不需要加 1，继续除以 2。

不过对于这种题，我们可以用不断着执行 n & (n - 1)，每执行一次就可以消去一个 1，当 n 为 0 时，计算总共执行了多少次即可，代码如下：

```
    public int NumberOf12(int n) {
        int count = 0;
        int k = 1;
        while (n != 0) {
            count++;
            n = (n - 1) & n;
        }
        return count;
```

代码不仅更加短小精悍，而且效率更高，关于 n & (n - 1)，我就暂时举例这两个，主要是后面还有非常多的技巧要写。

2、异或(^)运算的妙用

关于异或运算符，我们先来看下他的特性

特性一：两个相同的数相互异或，运算结果为 0，例如 n ^ n = 0;

特性二：任何数和 0 异或，运算结果不变，例如 n ^ 0 = n;

特性三：支持**交换律和结合律**，例如 x ^ (y ^ x) = (x ^ y) ^ x;

案例1：只出现一次是数

> 问题：数组中，只有一个数出现一次，剩下都出现两次，找出出现一次的数

常规操作就是一边遍历数组一边用哈希表统计元素出现的次数数，最后再遍历哈希表，看看哪个数只出现了一次。这种方法的时间复杂度为 O(n)，空间复杂度也为 O(n)了。

我们刚才说过，两个相同的数异或的结果是 0，一个数和 0 异或的结果是它本身，所以我们把这一组整型全部异或一下，例如这组数据是：1， 2， 3， 4， 5， 1， 2， 3， 4。其中 5 只出现了一次，其他都出现了两次，把他们全部异或一下，结果如下：

由于异或支持交换律和结合律，所以:

1^2^3^4^5^1^2^3^4 = （1^1)^(2^2)^(3^3)^(4^4)^5= 0^0^0^0^5 = 5。

通过这种方法，可以把空间复杂度降低到 O(1)，而时间复杂度不变，相应的代码如下

```
int find(int[] arr){
    int tmp = arr[0];
    for(int i = 1;i < arr.length; i++){
        tmp = tmp ^ arr[i];
    }
    return tmp;
}
```

关于位运算的技巧真的挺多，不过由于篇幅原因，我就暂时先举例这么多，重点的要告诉你，平时在刷题的时候，多留意下这些技巧，然后可以总结下来，之后自己遇到的时候可以应用上去。

2、考虑是否可以使用数组下标

数组的下标是一个隐含的很有用的数组，特别是在统计一些数字，或者判断一些整型数是否出现过的时候。例如，给你一串字母，让你判断这些字母出现的次数时，我们就可以把这些字母作为下标，在遍历的时候，如果字母a遍历到，则arr[a]就可以加1了，即arr[a]++;

通过这种巧用下标的方法，我们不需要逐个字母去判断。

我再举个例子：

> 问题：给你n个无序的int整型数组arr，并且这些整数的取值范围都在0-20之间，要你在 O(n) 的时间复杂度中把这 n 个数按照从小到大的顺序打印出来。

对于这道题，如果你是先把这 n 个数先排序，再打印，是不可能O(n)的时间打印出来的。但是数值范围在 0-20。我们就可以巧用数组下标了。把对应的数值作为数组下标，如果这个数出现过，则对应的数组加1。

```java
public void f(int arr[]) {

        int[] temp = new int[21];
        for (int i = 0; i < arr.length; i++) {
            temp[arr[i]]++;
        }
        //顺序打印
        for (int i = 0; i < 21; i++) {
            for (int j = 0; j < temp[i]; j++) {
                System.out.println(i);
            }
        }
    }
```

我在举一个例子

> 假如给你20亿个非负数的int型整数，然后再给你一个非负数的int型整数 t ，让你判断t是否存在于这20亿数中，你会怎么做呢?

有人可能会用一个int数组，然后把20亿个数给存进去，然后再循环遍历一下就可以了。

想一下，这样的话，时间复杂度是O(n)，所需要的内存空间

4byte * 20亿，一共需要80亿个字节

如果采用**下标法**，我们可以把时间复杂度降低位 O(1)，例如我们可以这样来存数据，把一个 int 非负整数 n 作为数组下标，如果 n 存在，则对应的值为1，如果不存在，对应的值为0。例如数组arr[n] = 1，表示n存在，arr[n] = 0表示n不存在。

那么，我们就可以把20亿个数作为下标来存，之后直接判断arr[t]的值，如果arr[t] = 1，则代表存在，如果arr[t] = 0，则代表不存在。这样，我们就可以把时间复杂度降低到O(1)。

那么大家想一下，**空间上可以继续优化吗?**

答是可以的，因为如果不需要统计**个数**，我们我们不需要 int 数组，用**boolean**类型的数组他不香吗？boolean类型占用的空间更少。

那么大家想一下，还能继续优化吗?

答是可以的，可以用 bitmap 算法，具体我这里不展开，感兴趣看这篇文章：【面试现场】如何判断一个数是否在40亿个整数中?

关于下标法的，在做题的时候，真的用到提多，这里推荐大家以后做题的时候可以关注一下，我就暂时先讲这么多。

3、考虑能否使用双指针

双指针这个技巧，那就更加常用的，特别是在**链表**和**有序数组**中，例如

> 给定一个整数**有序**数组和一个目标值，找出数组中和为目标值的两个数，并且打印出来

一种简单的做法就是弄个两层的 for 循环，然而对于这种有序的数组，如果是要寻找某个数之类的，大概率可以考虑双指针，也就是设置一个头指针和尾指针，直接看代码吧，代码如下：

```
int find(int arr[], int target){
    int left = 0;//头指针
    int right = arr.length - 1;// 尾指针
    while(left < right){
        if(left + right == target){
            // 找到目标数，进行打印，这里我就不执行打印操作两
        }else if(left + right < target){
            left ++;
        }else{
            right --;
        }
    }
}
```

在 leetcode 中的三数之和和四数只和都可以采用这个类型的双指针来处理。

当然，双指针在链表中也是非常给力的，例如

在做关于单链表的题是特别有用，比如“判断单链表是否有环”、“如何一次遍历就找到链表中间位置节点”、“单链表中倒数第 k 个节点”等问题。对于这种问题，我们就可以使用双指针了，会方便很多。我顺便说下这三个问题怎么用双指针解决吧。

例如对于第一个问题

我们就可以设置一个慢指针和一个快指针来遍历这个链表。慢指针一次移动一个节点，而快指针一次移动两个节点，如果该链表没有环，则快指针会先遍历完这个表，如果有环，则快指针会在第二次遍历时和慢指针相遇。

对于第二个问题

一样是设置一个快指针和慢指针。慢的一次移动一个节点，而快的两个。在遍历链表的时候，当快指针遍历完成时，慢指针刚好达到中点。

对于第三个问题

设置两个指针，其中一个指针先移动k个节点。之后两个指针以相同速度移动。当那个先移动的指针遍历完成的时候，第二个指针正好处于倒数第k个节点。

你看，采用双指针方便多了吧。所以以后在处理与链表相关的一些问题的时候，可以考虑双指针哦。

关于双指针，在这里也是给大家提个醒，重要的还是要大家多考虑，以后才能顺手拈来。

4、从递归到备忘录到递推或者动态规划

递归真的太好用了，好多问题都可以使用递归来解决，不过 80% 的递归提都可以进行**剪枝**，并且还有还多带有备忘录的递归都可以转化为**动态规划**，我本来是要举例一个二维DP的动态规划题，较大家从递归 =》递归+备忘录 =》动态规划 =》动态规划优化的。

不过写起来有点多，并且有一定的难度，感觉有点偏离来这篇文章所有的**技巧总结**，所以我还来列举一个简单的例子吧，这个例子重在告诉大家**遇到递归的题，一定要考虑是否可以剪枝，是否可以把递归转化成递推**。

例如这个被我举烂的例子

（1）.对于可以递归的问题务必考虑是否有重复计算的

当我们使用递归来解决一个问题的时候，容易产生重复去算同一个子问题，这个时候我们要考虑状态保存以防止重复计算。例如我随便举一个之前举过的问题

问题：一只青蛙一次可以跳上1级台阶，也可以跳上2级。求该青蛙跳上一个n级的台阶总共有多少种跳法?

这个问题用递归很好解决。假设 f(n) 表示n级台阶的总跳数法，则有

f(n) = f(n-1) + f(n - 2)。

递归的结束条件是当0 <= n <= 2时, f(n) = n。因此我们可以很容易写出递归的代码

```
    public int f(int n) {
        if (n <= 2) {
            return n;
        } else {
            return f(n - 1) + f(n - 2);
        }
    }
```

不过对于可以使用递归解决的问题，我们一定要考虑是否有很多重复计算，一种简单的方法就是大家可以画一个图来看下。如这道题

状态	题号	题目	通过率	难度
已通过	#5	最长回文子串	27.6%	中等
已通过	#10	正则表达式匹配	25.3%	困难
已通过	#32	最长有效括号	28.5%	困难
已通过	#44	通配符匹配	25.2%	困难
已通过	#53	最大子序和	47.9%	简单
已通过	#62	不同路径	57.0%	中等
已通过	#63	不同路径 II	31.8%	中等
已通过	#64	最小路径和	62.9%	中等
已通过	#70	爬楼梯	47.1%	简单
已通过	#72	编辑距离	55.0%	困难

显然对于 f(n) = f(n-1) + f(n-2) 的递归，是有很多重复计算的。这个时候我们要考虑状态保存。例如用hashMap来进行保存，当然用一个数组也是可以的，这个时候就像我们上面说的**巧用数组下标**了。可以当arr[n] = 0时，表示n还没计算过，当arr[n] != 0时，表示f(n)已经计算过，这时就可以把计算过的值直接返回回去了。因此我们考虑用状态保存的做法代码如下：

```
//数组的大小根据具体情况来，由于int数组元素的的默认值是0
```

```
    int[] arr = new int[1000];
    public int f(int n) {
        if (n <= 2) {
            return n;
        } else {
            if (arr[n] != 0) {
                return arr[n];//已经计算过，直接返回
            } else {
                arr[n] = f(n-1) + f(n-2);
                return arr[n];
            }
        }
    }
```

这样，可以极大着提高算法的效率。也有人把这种状态保存称之为**备忘录法**。

(2).考虑自底向上

对于递归的问题，我们一般都是从上往下递归的，直到递归到最底，再一层一层着把值返回。

不过，有时候当n比较大的时候，例如当 n = 10000时，那么必须要往下递归10000层直到 n <=2 才将结果慢慢返回，如果n太大的话，可能栈空间会不够用。

对于这种情况，其实我们是可以考虑自底向上的做法的。例如我知道

f(1) = 1;
f(2) = 2;

那么我们就可以推出 f(3) = f(2) + f(1) = 3。从而可以推出f(4),f(5)等直到f(n)。因此，我们可以考虑使用自底向上的方法来做。

代码如下：

```
public int f(int n) {
        if(n <= 2)
            return n;

        int f1 = 1;
        int f2 = 2;
```

```
        int sum = 0;

        for (int i = 3; i <= n; i++) {
            sum = f1 + f2;
            f1 = f2;
            f2 = sum;
        }
        return sum;
    }
```

我们也把这种自底向上的做法称之为**递推**。

根据这种带备忘录的递归，往往可以演变成**动态规划**，大家可以拿 leetcode 这两道题试试水

leetcode 的 62 号题：https://leetcode-cn.com/problems/unique-paths/

leetcode 的第64题：https://leetcode-cn.com/problems/minimum-path-sum/

总结一下

当你在使用递归解决问题的时候，要考虑以下两个问题

(1). 是否有状态重复计算的，可不可以使用备忘录法来优化。

(2). 是否可以采取递推的方法来自底向上做，减少一味递归的开销。

5、考虑是否可以设置哨兵位来处理临届问题

在链表的相关问题中，我们经常会设置一个头指针，而且这个头指针是不存任何有效数据的，只是为了操作方便，这个头指针我们就可以称之为**哨兵位了**。

例如我们要删除头第一个节点是时候，如果没有设置一个哨兵位，那么在操作上，它会与删除第二个节点的操作有所不同。但是我们设置了哨兵，那么删除第一个节点和删除第二个节点那么在操作上就一样了，不用做额外的判断。当然，插入节点的时候也一样。

有时候我们在操作数组的时候，也是可以设置一个哨兵的，把arr[0]作为哨兵。例如，要判断两个相邻的元素是否相等时，设置了哨兵就不怕越界等问题了，可以直接arr[i] == arr[i-1]?了。不用怕i = 0时出现越界。

当然我这只是举一个例子，具体的应用还有很多，例如插入排序，环形链表等。

总结

关于上面说的技巧，我只能说**熟能生巧**，居然要熟，首先你得要有机会接触到这样一算思想，而我上面的这些总结，便是给你找来机会接触这些思想，并且还都给出了例子，大家可以好好消化下，这篇文章的内容有些虽然是之前总结过的，不过这一次增加了一些新的东西和例子，还是花了不少时间，希望能够给大家带来一些帮助勒。

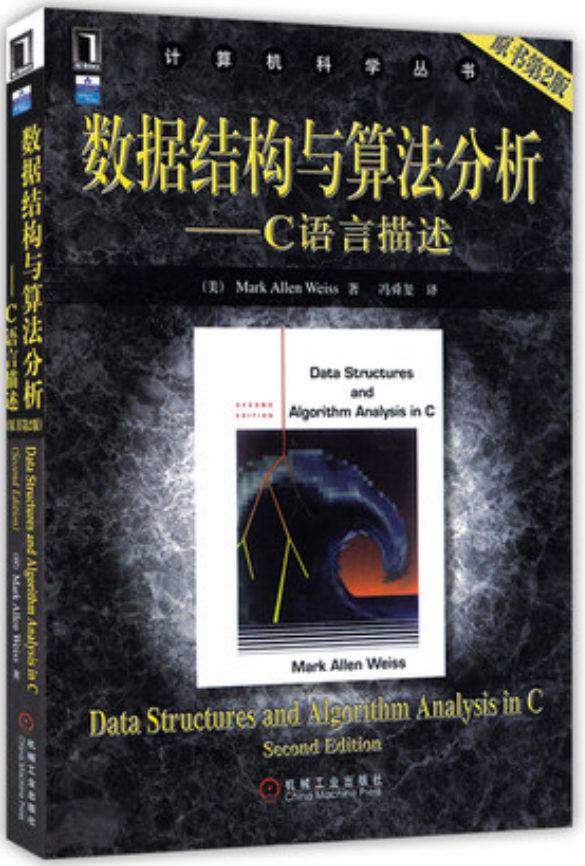
小白成长，大学看过的一些优质书籍

有时有些读者问我，数据结构与算法该怎么学？有书籍推荐的吗？Java 初学者该怎么学等等。今天我就给大家介绍一些我这几年看过的一些**自认为优秀的书籍**，由于我看的大部分书籍可以说都是通用的，所以如果你有时间的话，还是挺建议看看的，特别是学生。

数据结构与算法

数据结构与算法相关的书籍应该是我看的最多的一种数据吧，从大一到现在，从未间断过，下面就介绍下从大一到现在都看过哪些**自认为优秀的书籍**，注意，我不知道适不适合你，但我觉得看的过程中很舒服。

1、数据结构与算法分析（c 语言描述版）



我相信大部分人大学看的教程都是清华大学出版社严蔚敏写的那本书，说实话，作为初学者，那本书我没能坚持看下去，可能比较适合大佬看吧。我自己买了一本《数据结构与算法分析（c 语言描述版）》，挺薄的，不过感觉很棒，这本书让我学到了很多，个人感觉也挺容易懂的，代码实现是采用 C 语言来实现的，不是伪代码，如果你想学习数据结构，我觉得这本书是个不错的选择。班级里有挺多人看了《大话数据结构》，挺他们说也挺不错，不过我没看过。

2、挑战程序设计竞赛

这边书也是大一时看的，如果你想刷题，我挺推荐这本书，里面分初级、中级到高级。虽然每道题没有讲的特别详细，但当时都看懂了，真心不错。不过高级那部分我是没看，初级和中级看着挺舒服。也是学到挺多的，推荐给大家。

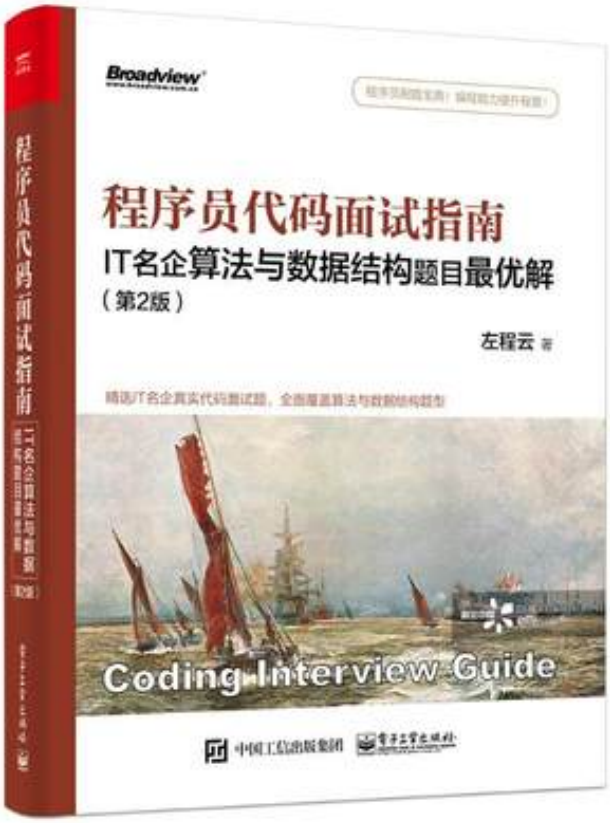
3、编程之美

不用说，很美，这本书是我今年刚入手看的，只能用**强烈推荐**来形容，在这本书里，学到了挺多技巧，里面列举的题也不是特别难，目前看了 80%，真香。刚开始我听别人说如果要准备面试谷歌什么的建议看，我以为很难，迟迟没买来看，不过，我看的过程中，感觉还好，相信你也能看的懂，想学习算法、刷题的，强烈推荐。

4、编程珠玑

这本老早就听别人说过了，去年看的，不过也是看了80%左右，和编程之美一样，强烈推荐，这本书里的题，说实话，感觉比编程之美有意思，

5、程序员代码面试指南：IT 名企算法与数据结构题目最优解



这本书是牛客网的左程云写的，这本书重在带你刷题，每道题的解法也是讲的挺详细的，而且，这本书是一个专题一个专题带你刷题的，从栈和队列、链表、二叉树、递归与动态规划、字符串等等。我之前的链表打卡就是从这里找的。大家可以按照自己的弱点挑着刷。对了，代码是采用 Java 实现的，不过你会 C 语言的话，一样能看懂。真心不过，递归和动态规划里面好几道题都命中这次春招笔试了，当然，类似而已。然而，那时我还没有去看这本书动态相关的专题。推荐给大家。

当然，数据结构与算法的还有很多优秀的书籍，我自己也看过不少，不过以上这些，我觉得很不错。自己也买过算法导论、算法第四版等，不过，没看的下去，就先介绍这么多吧，如果你有看过什么优秀的书籍，欢迎留言。

计算机基础

计算机基础这方面，我看的大多数都是学校的教材，这些就不介绍了，不过自己也买一些课外的，感觉很不错，介绍给大家。

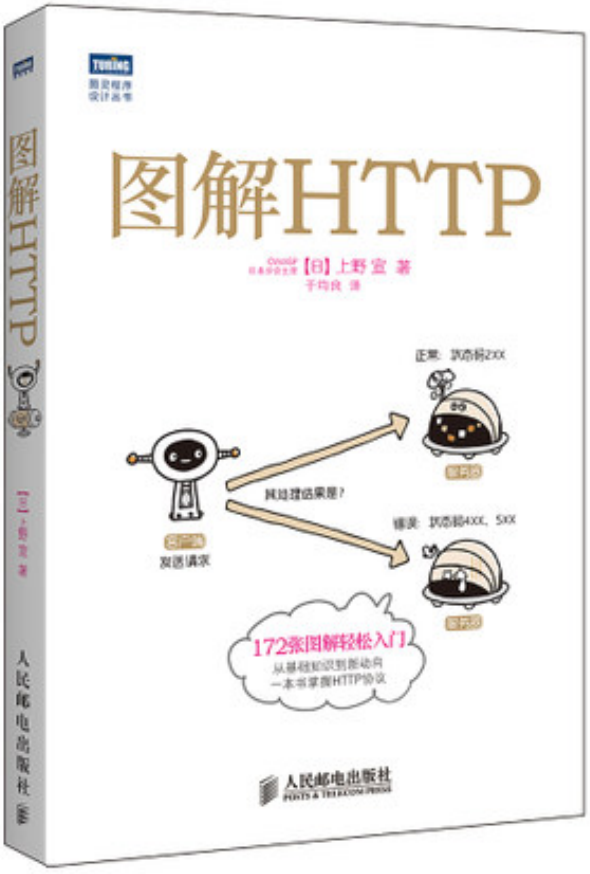
1、程序是怎么跑起来的

我觉得，了解程序是如何跑起来是每个程序员都必须掌握的，我看过相关的书籍是学校发的操作系统，操作系统有多么枯燥我就不说了。不过这本《程序是怎么跑起来的》的书，我觉得就算你是完全不懂的小白也能看懂，如果你对学习操作系统感兴趣，或许可以买这本书当作入门，像讲故事一样，讲的挺有趣的，两天就能看完了。不过，讲的不深，想要再深入的话，还得看操作系统相关书籍。

2、计算机网络：自顶向下


计算机科学丛书
PEARSON
原书第6版
计算机网络
自顶向下方法
(美) James F. Kurose Keith W. Ross 著
Computer Networking
A Top-Down Approach Sixth Edition
Computer Networking
A Top-Down Approach
KUROSE ROSS
机械工业出版社
China Machine Press

图解 HTTP


TURING
图灵程序设计丛书
图解HTTP
【日】上野宣 著
正常 状态码2XX
网址理论果是?
错误 状态码4XX、5XX
发送请求
172张图解轻松入门
从基础知识到原动力
一本书掌握HTTP协议
人民邮电出版社

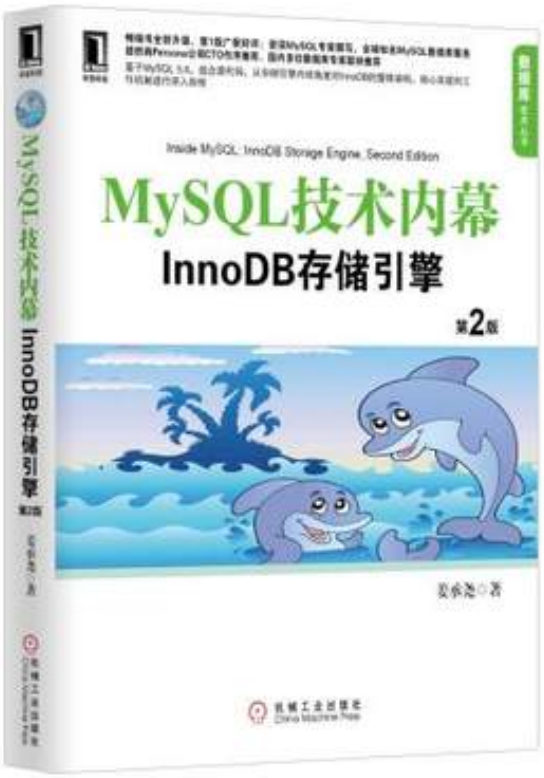
计算机网络那本书对于 http 并没有讲很多，如果你想继续了解 http 的话，就可以考虑看《图解http》这本书了，居然是**图解**，那么将会有大量的图片，让你轻松读懂晦涩的知识点，相信你两天就能看完了，不过我建议你最好做一下笔记，不过，有些东西你很快就忘光光了，笔记面试的时候，还得要你把一些东西说出来。

MySQL

1、MySQL必知必会

MySQL 的书看的真心少，不过感觉《MySQL必知必会》讲的好不错，想学习 sql 语句的可以看看。

2、MySQL技术内幕：InnoDB存储引擎



学习MySQL，InnoDB 引擎是必须得学的，这本书讲的真心不错，讲了很多原理，例如索引、锁等相关的原理，如果说《MySQL必知必会》是入门，那么这本就是进阶了，这本书我很早就买了，不过也是最近刚看，不过我是看了极客时间的 MySQL 相关专栏再来看这本书的，感觉收获不少，推荐。

MySQL 业界最火的可能就是《高性能MySQL》，这本书我也买了，看了一些一直没去看，好厚啊，好像80-90 买的，亏大。如果你想折腾，或许高性能这本书可以看看，不过，建议挑着看，别从头到尾看，除非你时间很多。

Java 相关

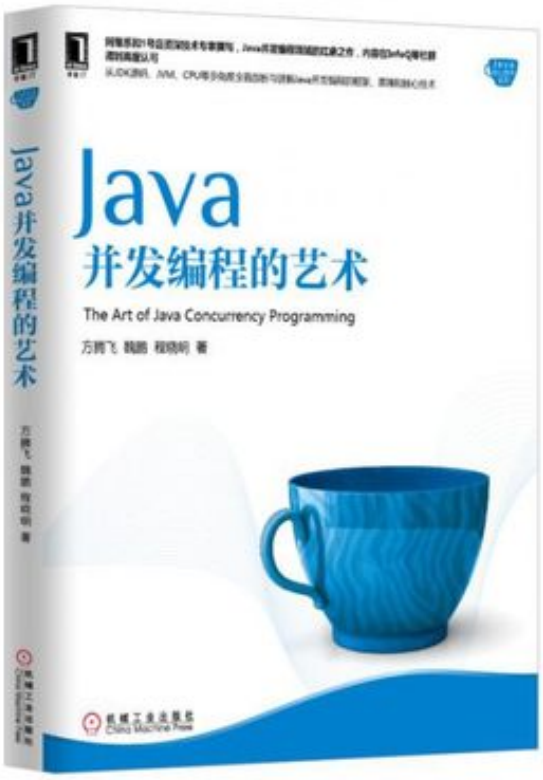
由于我的主要使用语言是 Java，所以 Java 这方面也是看过不少书籍，下面就介绍一些我觉得学 Java 不可错过的书籍吧。

Java编程思想

编程思想这本书个人感觉不错，不过不适合入门，我是看尚学堂高淇 300 集视频入门的，哔哩哔哩直接搜索就行了，说时候，讲的是非常非常不错，初学者强烈推荐这个视频。

我说了，单单视频是不够的，之后入手了《Java编程思想》这本书，不过我是挑着章节看的，看了之后，解决了非常非常多的疑惑，感觉自己对 Java 的理解更上一层楼了，不过，这本书看的时候，你可能会觉得有点啰嗦，不过没办法，国外的书籍大部分都这样，喜欢**扯**，不过我也是挺喜欢这种扯的，总之，强烈推荐（不建议从头看，可以挑着看）

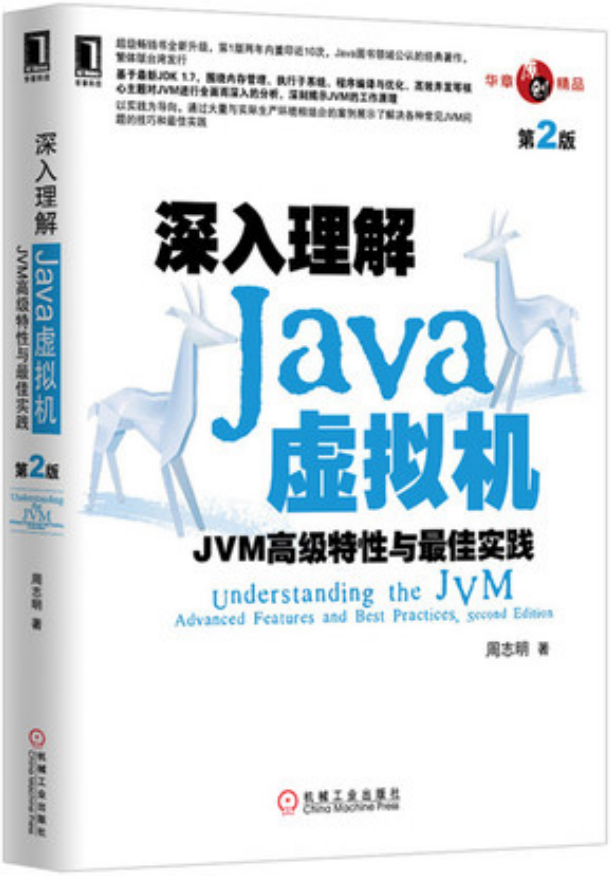
Java 并发编程艺术



学编程，并发是避免不了的，我觉得《Java 并发编程艺术》这本书讲的非常不错，不过说实话，也是挺难的，我一下子就把这本书看完了，然后，看完之后感觉啥也忘了，然后第二次看的时候，感觉比第一次好挺多。总之这本书，我觉得要多看几次，你会有意外的收获。特别是后面，可能刚开始看有点懵，那是因为你菜。不过，多看几次就好了，学这本，听说应付面试也是非常不错的，推荐看。

对了，还要 《Java 并发编程实战》，也是挺不错，不过我只看了一些，感觉自己很多都懂（感觉要被打），就有点看不下去了，所以没看，你们自行选择。

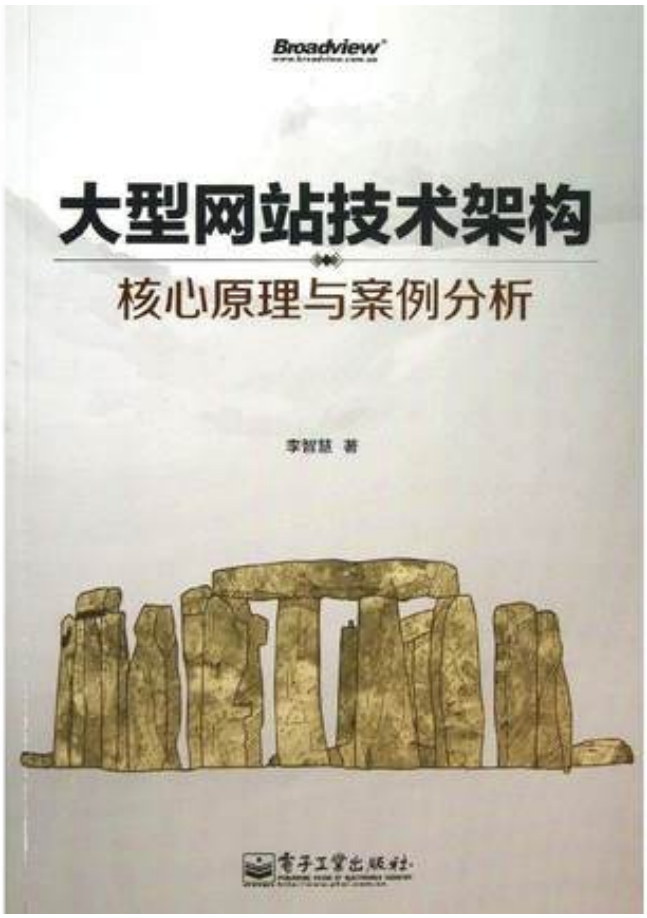
深入理解 Java 虚拟机



呵呵，这本书，不用说，每个学 Java 的我觉得都应该看，不过我可以告诉你的是，第一遍你会看的很难受，确实挺难，比较底层。不过，想要进阶，就得要**死磕**，大家看的时候，有些章节可以先跳过，例如第一章。我是从第三部分的第六章开始看起的，看完再回头看前面的章节，至于为啥这样，我是在某某知识星球听大佬这样说的，所以就这样干了。总之，强烈推荐，以及多看几遍。

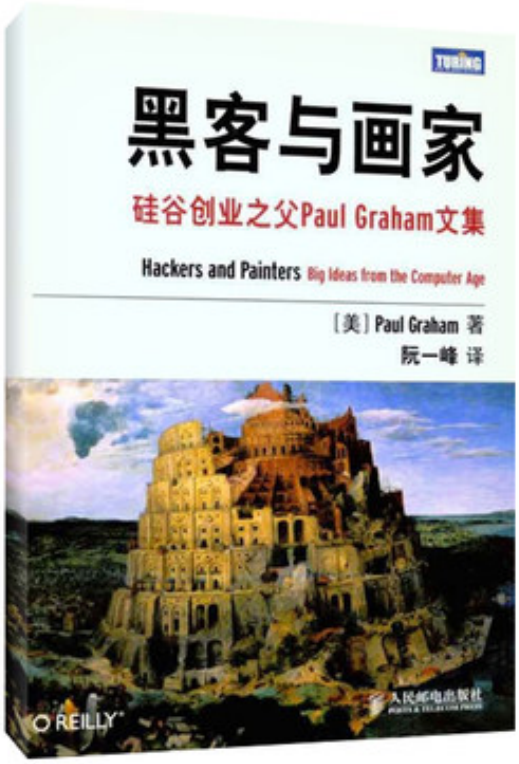
其他

1、大型网站技术结构：核心原理与案例分析



这本书讲的是，一个网站从简单到类似与淘宝这种大型的网站，都经过那些迭代。不过讲的不深，看名字很高级，不过并不难，我觉得挺不错，感兴趣的可以看看，两天就能看完了。

1、黑客与画家



这本书并不是讲黑客或画家的哈，这本书更多的是讲一种思维。我挺喜欢的，觉得挺不错，不是技术书籍。我觉得作为一个程序员，也不能一味看技术书籍，偶尔还是得看看其他方面的书籍，或许，可以拓展我们的思维，这本书就很不错了，推荐给大家。

最后

一不小心已经介绍了十几本了，介绍的都是属于比较基础的书籍，我觉得打好这些基础还是挺重要的。上面的这些书，基本都是我全部看完的了，觉得真心不错，所以推荐给大家，以后再给大家推荐点高级的书哈。

一文读懂动态规划算法

动态规划难吗？说实话，我觉得很难，特别是对于**初学者**来说，我当时入门动态规划的时候，是看 0-1 背包问题，当时真的是一脸懵逼。后来，我遇到动态规划的题，**看的懂答案，但就是自己不会做，不知道怎么下手**。就像做递归的题，看的懂答案，但下不了手，关于递归的，我之前也写过一篇**套路**的文章，如果对递归不大懂的，强烈建议看一看：为什么你学不会递归，告别递归，谈谈我的经验

对于动态规划，春招秋招时好多题都会用到动态规划，一气之下，再 leetcode 连续刷了几十道题

状态	题号	题目	通过率	难度
已通过	#5	最长回文子串	27.6%	中等
已通过	#10	正则表达式匹配	25.3%	困难
已通过	#32	最长有效括号	28.5%	困难
已通过	#44	通配符匹配	25.2%	困难
已通过	#53	最大子序和	47.9%	简单
已通过	#62	不同路径	57.0%	中等
已通过	#63	不同路径 II	31.8%	中等
已通过	#64	最小路径和	62.9%	中等
已通过	#70	爬楼梯	47.1%	简单
已通过	#72	编辑距离	55.0%	困难

之后，豁然开朗，感觉动态规划也不是很难，今天，我就来跟大家讲一讲，我是怎么做动态规划的题的，以及从中学到的一些**套路**。相信你看完一定有所收获

如果你对动态规划感兴趣，或者你看的懂动态规划，但却不知道怎么下手，那么我建议你好好看以下，这篇文章的写法，和之前那篇讲递归的写法，是差不多一样的，将会举大量的例子。如果一次性看不完，建议收藏，同时别忘了**素质三连**。

> 为了兼顾初学者，我会从最简单的题讲起，后面会越来越难，最后面还会讲解，该如何优化。因为 80% 的动规都是可以进行优化的。不过我得说，如果你连动态规划是什么都没听过，可能这篇文章你也会压力山大。

一、动态规划的三大步骤

动态规划，无非就是利用**历史记录**，来避免我们的重复计算。而这些**历史记录**，我们得需要一些**变量**来保存，一般是用**一维数组**或者**二维数组**来保存。下面我们先来讲下做动态规划题很重要的三个步骤，

> 如果你听不懂，也没关系，下面会有很多例题讲解，估计你就懂了。之所以不配合例题来讲这些步骤，也是为了怕你们脑袋乱了

第一步骤：定义**数组元素的含义**，上面说了，我们会用一个数组，来保存历史数组，假设用一维数组 dp[] 吧。这个时候有一个非常非常重要的点，就是规定你这个数组元素的含义，例如你的 dp[i] 是代表什么意思?

第二步骤：找出**数组元素之间的关系式**，我觉得动态规划，还是有一点类似于我们高中学习时的**归纳法**的，当我们要计算 dp[n] 时，是可以利用 dp[n-1]，dp[n-2].....dp[1]，来推出 dp[n] 的，也就是可以利用**历史数据**来推出新的元素值，所以我们要找出数组元素之间的关系式，例如 dp[n] = dp[n-1] + dp[n-2]，这个就是他们的关系式了。而这一步，也是最难的一步，后面我会讲几种类型的题来说。

> 学过动态规划的可能都经常听到**最优子结构**，把大的问题拆分成小的问题，说时候，最开始的时候，我是对**最优子结构**一梦懵逼的。估计你们也听多了，所以这一次，我将**换一种形式来讲，不再是各种子问题，各种最优子结构**。所以大佬可别喷我再乱讲，因为我说了，这是我自己平时做题的套路。

第三步骤：找出**初始值**。学过**数学归纳法**的都知道，虽然我们知道了数组元素之间的关系式，例如 dp[n] = dp[n-1] + dp[n-2]，我们可以通过 dp[n-1] 和 dp[n-2] 来计算 dp[n]，但是，我们得知道初始值啊，例如一直推下去的话，会由 dp[3] = dp[2] + dp[1]。而 dp[2] 和 dp[1] 是不能再分解的了，所以我们必须要能够直接获得 dp[2] 和 dp[1] 的值，而这，就是**所谓的初始值**。

由了**初始值**，并且有了**数组元素之间的关系式**，那么我们就可以得到 dp[n] 的值了，而 dp[n] 的含义是由你来定义的，你想**求什么，就定义它是什么**，这样，这道题也就解出来了。

不懂？没事，我们来看三四道例题，我讲严格按这个步骤来给大家讲解。

二、案例详解

案例一、简单的一维 DP

> 问题描述：一只青蛙一次可以跳上1级台阶，也可以跳上2级。求该青蛙跳上一个n级的台阶总共有多少种跳法。

(1)、定义数组元素的含义

按我上面的步骤说的，首先我们来定义 dp[i] 的含义，我们的问题是要求青蛙跳上 n 级的台阶总共由多少种跳法，那我们就定义 dp[i] 的含义为：**跳上一个 i 级的台阶总共有 dp[i] 种跳法**。这样，如果我们能够算出 dp[n]，不就是我们要求的答案吗？所以第一步定义完成。

（2）、找出数组元素间的关系式

我们的目的是要求 dp[n]，动态规划的题，如你们经常听说的那样，就是把一个**规模**比较大的问题分成几个**规模**比较小的问题，然后由小的问题推导出大的问题。也就是说，dp[n] 的规模为 n，比它规模小的是 n-1, n-2, n-3.... 也就是说，dp[n] 一定会和 dp[n-1], dp[n-2]....存在某种关系的。我们要找出他们的关系。

那么问题来了，怎么找?

这个怎么找，**是最核心最难的一个**，我们必须回到问题本身来了，来寻找他们的关系式，dp[n] 究竟会等于什么呢?

对于这道题，由于情况可以选择跳一级，也可以选择跳两级，所以青蛙到达第 n 级的台阶有两种方式

一种是从第 n-1 级跳上来

一种是从第 n-2 级跳上来

由于我们是要算**所有可能的跳法的**，所以有 dp[n] = dp[n-1] + dp[n-2]。

（3）、找出初始条件

当 n = 1 时，dp[1] = dp[0] + dp[-1]，而我们是数组是不允许下标为负数的，所以对于 dp[1]，我们必须要**直接给出它的数值**，相当于初始值，显然，dp[1] = 1。一样，dp[0] = 0.（因为 0 个台阶，那肯定是 0 种跳法了）。于是得出初始值：

dp[0] = 0.
dp[1] = 1.
即 n <= 1 时，dp[n] = n.

三个步骤都做出来了，那么我们就来写代码吧，代码会详细注释滴。

```
int f( int n ){
  if(n <= 1)
  return n;
  // 先创建一个数组来保存历史数据
  int[] dp = new int[n+1];
  // 给出初始值
  dp[0] = 0;
  dp[1] = 1;
  // 通过关系式来计算出 dp[n]
  for(int i = 2; i <= n; i++){
    dp[i] = dp[i-1] + dp[i-2];
  }
  // 把最终结果返回
  return dp[n];
}
```

（4）、再说初始化

大家先想以下，你觉得，上面的代码有没有问题?

答是有问题的，还是错的，错在**对初始值的寻找不够严谨**，这也是我故意这样弄的，意在告诉你们，关于**初始值的严谨性**。例如对于上面的题，当 n = 2 时，dp[2] = dp[1] + dp[0] = 1。这显然是错误的，你可以模拟一下，应该是 dp[2] = 2。

也就是说，在寻找初始值的时候，一定要注意不要找漏了，dp[2] 也算是一个初始值，不能通过公式计算得出。有人可能会说，我想不到怎么办？这个很好办，多做几道题就可以了。

> 下面我再列举三道不同的例题，并且，再在未来的文章中，我也会持续按照这个步骤，给大家找几道有难度且类型不同的题。下面这几道例题，不会讲的特性详细哈。实际上，上面的一维数组是可以把空间优化成更小的，不过我们现在先不讲优化的事，下面的题也是，不讲优化版本。

案例二：二维数组的 DP

我做了几十道 DP 的算法题，可以说，80% 的题，都是要用二维数组的，所以下面的题主要以二维数组为主，当然有人可能会说，要用一维还是二维，我怎么知道？这个问题不大，接着往下看。

问题描述

一个机器人位于一个 m x n 网格的左上角 （起始点在下图中标记为“Start” ）。

机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为“Finish”）。

问总共有多少条不同的路径?

例如，上图是一个7 x 3 的网格。有多少可能的路径?

说明： *m* 和 *n* 的值均不超过 100。

> 这是 leetcode 的 62 号题：https://leetcode-cn.com/problems/unique-paths/

还是老样子，三个步骤来解决。

步骤一、定义数组元素的含义

由于我们的目的是从左上角到右下角一共有多少种路径，那我们就定义 dp[i] [j]的含义为：**当机器人从左上角走到(i, j) 这个位置时，一共有 dp[i] [j] 种路径**。那么，dp[m-1] [n-1] 就是我们要的答案了。

> 注意，这个网格相当于一个二维数组，数组是从下标为 0 开始算起的，所以 右下角的位置是 (m-1, n - 1)，所以 dp[m-1] [n-1] 就是我们要找的答案。

步骤二：找出关系数组元素间的关系式

想象以下，机器人要怎么样才能到达 (i, j) 这个位置？由于机器人可以向下走或者向右走，所以有两种方式到达

一种是从 (i-1, j) 这个位置走一步到达

一种是从(i, j - 1) 这个位置走一步到达

因为是计算所有可能的步骤，所以是把所有可能走的路径都加起来，所以关系式是 dp[i] [j] = dp[i-1] [j] + dp[i] [j-1]。

步骤三、找出初始值

显然，当 dp[i] [j] 中，如果 i 或者 j 有一个为 0，那么还能使用关系式吗？答是不能的，因为这个时候把 i - 1 或者 j - 1，就变成负数了，数组就会出问题了，所以我们的初始值是计算出所有的 dp[0] [0....n-1] 和所有的 dp[0....m-1] [0]。这个还是非常容易计算的，相当于计算机图中的最上面一行和左边一列。因此初始值如下：

dp[0] [0....n-1] = 1; // 相当于最上面一行，机器人只能一直往右走

dp[0...m-1] [0] = 1; // 相当于最左面一列，机器人只能一直往下走

撸代码

三个步骤都写出来了，直接看代码

```
public static int uniquePaths(int m, int n) {
    if (m <= 0 || n <= 0) {
        return 0;
    }

    int[][] dp = new int[m][n]; //
```

```
    // 初始化
    for(int i = 0; i < m; i++){
      dp[i][0] = 1;
    }
    for(int i = 0; i < n; i++){
      dp[0][i] = 1;
    }
    // 推导出 dp[m-1][n-1]
    for (int i = 1; i < m; i++) {
        for (int j = 1; j < n; j++) {
            dp[i][j] = dp[i-1][j] + dp[i][j-1];
        }
    }
    return dp[m-1][n-1];
}
```

> O(n*m) 的空间复杂度可以优化成 O(min(n, m)) 的空间复杂度的，不过这里先不讲

案例三、二维数组 DP

写到这里，有点累了，，但还是得写下去，所以看的小伙伴，你们可得继续看呀。下面这道题也不难，比上面的难一丢丢，不过也是非常类似

问题描述

给定一个包含非负整数的 *m* x *n* 网格，请找出一条从左上角到右下角的路径，使得路径上的数字总和为最小。

说明：每次只能向下或者向右移动一步。

```
举例:
输入:
arr = [
  [1,3,1],
  [1,5,1],
  [4,2,1]
]
输出: 7
解释: 因为路径 1→3→1→1→1 的总和最小。
```

和上面的差不多，不过是算最优路径和，这是 leetcode 的第64题：https://leetcode-cn.com/problems/minimum-path-sum/

> 还是老样子，可能有些人都看烦了，哈哈，但我还是要按照步骤来写，让那些不大懂的加深理解。有人可能觉得，这些题太简单了吧，别慌，小白先入门，这些属于 medium 级别的，后面在给几道 hard 级别的。

步骤一、定义数组元素的含义

由于我们的目的是从左上角到右下角，最小路径和是多少，那我们就定义 dp[i] [j]的含义为：**当机器人从左上角走到(i, j) 这个位置时，最下的路径和是 dp[i] [j]**。那么，dp[m-1] [n-1] 就是我们要的答案了。

> 注意，这个网格相当于一个二维数组，数组是从下标为 0 开始算起的，所以 由下角的位置是 (m-1, n - 1)，所以 dp[m-1] [n-1] 就是我们要走的答案。

步骤二：找出关系数组元素间的关系式

想象以下，机器人要怎么样才能到达 (i, j) 这个位置？由于机器人可以向下走或者向右走，所以有两种方式到达

一种是从 (i-1, j) 这个位置走一步到达

一种是从(i, j - 1) 这个位置走一步到达

不过这次不是计算所有可能路径，而是**计算哪一个路径和是最小的**，那么我们要从这两种方式中，选择一种，使得dp[i] [j] 的值是最小的，显然有

```
dp[i] [j] = min(dp[i-1][j], dp[i][j-1]) + arr[i][j];// arr[i][j] 表示网格种的值
```

步骤三、找出初始值

显然，当 dp[i] [j] 中，如果 i 或者 j 有一个为 0，那么还能使用关系式吗？答是不能的，因为这个时候把 i - 1 或者 j - 1，就变成负数了，数组就会出问题了，所以我们的初始值是计算出所有的 dp[0] [0....n-1] 和所有的 dp[0....m-1] [0]。这个还是非常容易计算的，相当于计算机图中的最上面一行和左边一列。因此初始值如下：

dp[0] [j] = arr[0] [j] + dp[0] [j-1]; // 相当于最上面一行，机器人只能一直往右走

dp[i] [0] = arr[i] [0] + dp[i] [0]; // 相当于最左面一列，机器人只能一直往下走

代码如下

```
public static int uniquePaths(int[][] arr) {
    int m = arr.length;
    int n = arr[0].length;
    if (m <= 0 || n <= 0) {
        return 0;
    }

    int[][] dp = new int[m][n]; //
    // 初始化
    dp[0][0] = arr[0][0];
    // 初始化最左边的列
    for(int i = 1; i < m; i++){
      dp[i][0] = dp[i-1][0] + arr[i][0];
    }
    // 初始化最上边的行
    for(int i = 1; i < n; i++){
      dp[0][i] = dp[0][i-1] + arr[0][i];
    }
    // 推导出 dp[m-1][n-1]
    for (int i = 1; i < m; i++) {
        for (int j = 1; j < n; j++) {
            dp[i][j] = Math.min(dp[i-1][j], dp[i][j-1]) + arr[i][j];
        }
    }
    return dp[m-1][n-1];
}
```

> O(n*m) 的空间复杂度可以优化成 O(min(n, m)) 的空间复杂度的，不过这里先不讲

案例 4：编辑距离

这次给的这道题比上面的难一些，在 leetcdoe 的定位是 hard 级别。好像是 leetcode 的第 72 号题。

问题描述

给定两个单词 word1 和 word2，计算出将 word1 转换成 word2 所使用的最少操作数。

你可以对一个单词进行如下三种操作：

插入一个字符
删除一个字符
替换一个字符

```
示例：
输入：word1 = "horse", word2 = "ros"
输出：3
解释：
horse -> rorse (将 'h' 替换为 'r')
rorse -> rose (删除 'r')
rose -> ros (删除 'e')
```

解答

还是老样子，按照上面三个步骤来，并且我这里可以告诉你，90% 的字符串问题都可以用动态规划解决，并且90%是采用二维数组。

步骤一、定义数组元素的含义

由于我们的目的求将 word1 转换成 word2 所使用的最少操作数 。那我们就定义 dp[i] [j]的含义为：**当字符串 word1 的长度为 i，字符串 word2 的长度为 j 时，将 word1 转化为 word2 所使用的最少操作次数为 dp[i] [j]**。

> 有时候，数组的含义并不容易找，所以还是那句话，我给你们一个套路，剩下的还得看你们去领悟。

步骤二：找出关系数组元素间的关系式

接下来我们就要找 dp[i] [j] 元素之间的关系了，比起其他题，这道题相对比较难找一点，但是，不管多难找，大部分情况下，dp[i] [j] 和 dp[i-1] [j]、dp[i] [j-1]、dp[i-1] [j-1] 肯定存在某种关系。因为我们的目标就是，**从规模小的，通过一些操作，推导出规模大的。对于这道题，我们可以对 word1 进行三种操作

插入一个字符
删除一个字符
替换一个字符

由于我们是要让操作的次数最小，所以我们要寻找最佳操作。那么有如下关系式：

一、如果我们 word1[i] 与 word2 [j] 相等，这个时候不需要进行任何操作，显然有 dp[i] [j] = dp[i-1] [j-1]。（别忘了 dp[i] [j] 的含义哈）。

二、如果我们 word1[i] 与 word2 [j] 不相等，这个时候我们就必须进行调整，而调整的操作有 3 种，我们要选择一种。三种操作对应的关系试如下（注意字符串与字符的区别）：

（1）、如果把字符 word1[i] 替换成与 word2[j] 相等，则有 dp[i] [j] = dp[i-1] [j-1] + 1;

（2）、如果在字符串 word1末尾插入一个与 word2[j] 相等的字符，则有 dp[i] [j] = dp[i] [j-1] + 1;

（3）、如果把字符 word1[i] 删除，则有 dp[i] [j] = dp[i-1] [j] + 1;

那么我们应该选择一种操作，使得 dp[i] [j] 的值最小，显然有

dp[i] [j] = min(dp[i-1] [j-1]，dp[i] [j-1]，dp[[i-1] [j]]) + 1;

于是，我们的关系式就推出来了，

步骤三、找出初始值

显然，当 dp[i] [j] 中，如果 i 或者 j 有一个为 0，那么还能使用关系式吗？答是不能的，因为这个时候把 i - 1 或者 j - 1，就变成负数了，数组就会出问题了，所以我们的初始值是计算出所有的 dp[0] [0....n] 和所有的 dp[0....m] [0]。这个还是非常容易计算的，因为当有一个字符串的长度为 0 时，转化为另外一个字符串，那就只能一直进行插入或者删除操作了。

代码如下

```
public int minDistance(String word1, String word2) {
    int n1 = word1.length();
    int n2 = word2.length();
    int[][] dp = new int[n1 + 1][n2 + 1];
    // dp[0][0...n2]的初始值
```

```
    for (int j = 1; j <= n2; j++)
      dp[0][j] = dp[0][j - 1] + 1;
    // dp[0...n1][0] 的初始值
    for (int i = 1; i <= n1; i++) dp[i][0] = dp[i - 1][0] + 1;
    // 通过公式推出 dp[n1][n2]
    for (int i = 1; i <= n1; i++) {
        for (int j = 1; j <= n2; j++) {
            // 如果 word1[i] 与 word2[j] 相等。第 i 个字符对应下标是 i-1
            if (word1.charAt(i - 1) == word2.charAt(j - 1)){
              dp[i][j] = dp[i - 1][j - 1];
            }else {
               dp[i][j] = Math.min(Math.min(dp[i - 1][j - 1], dp[i][j
- 1]), dp[i - 1][j]) + 1;
            }
        }
    }
    return dp[n1][n2];
}
```

最后说下，如果你要练习，可以去 leetcode，选择动态规划专题，然后连续刷几十道，保证你以后再也不怕动态规划了。当然，遇到很难的，咱还是得挂。

Leetcode 动态规划直达：https://leetcode-cn.com/tag/dynamic-programming/

三、总结

上面的这些题，基本都是不怎么难的入门题，除了最后一道相对难一点，本来是要在写几道难一点，并且讲**如何优化**的，不过看了下字数，文章有点长了，关于如何优化的，后面再讲吧，在之后的文章中，我也会按照这个步骤，在给大家讲四五道动态规划的题.

一文读懂递归算法

可能很多人在大一的时候，就已经接触了递归了，不过，我敢保证很多人初学者刚开始接触递归的时候，是一脸懵逼的，我当初也是，给我的感觉就是，递归太神奇了！

可能也有一大部分人知道递归，也能看的懂递归，但在实际做题过程中，却不知道怎么使用，有时候还容易被递归给搞晕。也有好几个人来问我有没有快速掌握递归的捷径啊。说实话，哪来那么多捷径啊，不过，我还是想写一篇文章，谈谈我的一些经验，或许，能够给你带来一些帮助。

为了兼顾初学者，我会从最简单的题讲起！

递归的三大要素

第一要素：明确你这个函数想要干什么

对于递归，我觉得很重要的一个事就是，**这个函数的功能是什么**，他要完成什么样的一件事，而这个，是完全由你自己来定义的。也就是说，我们先不管函数里面的代码什么，而是要先明白，你这个函数是要用来干什么。

例如，我定义了一个函数

```
// 算 n 的阶乘(假设n不为0)
int f(int n){

}
```

这个函数的功能是算 n 的阶乘。好了，我们已经定义了一个函数，并且定义了它的功能是什么，接下来我们看第二要素。

第二要素：寻找递归结束条件

所谓递归，就是会在函数内部代码中，调用这个函数本身，所以，我们必须要找出**递归的结束条件**，不然的话，会一直调用自己，进入无底洞。也就是说，我们需要找出**当参数为啥时，递归结束，之后直接把结果返回**，请注意，这个时候我们必须能根据这个参数的值，能够**直接**知道函数的结果是什么。

例如，上面那个例子，当 n = 1 时，那你应该能够直接知道 f(n) 是啥吧？此时，f(1) = 1。完善我们函数内部的代码，把第二要素加进代码里面，如下

```
// 算 n 的阶乘(假设n不为0)
int f(int n){
    if(n == 1){
        return 1;
    }
}
```

有人可能会说，当 n = 2 时，那我们可以直接知道 f(n) 等于多少啊，那我可以把 n = 2 作为递归的结束条件吗?

当然可以，只要你觉得参数是什么时，你能够直接知道函数的结果，那么你就可以把这个参数作为结束的条件，所以下面这段代码也是可以的。

```
// 算 n 的阶乘(假设n>=2)
int f(int n){
    if(n == 2){
        return 2;
    }
}
```

注意我代码里面写的注释，假设 n >= 2，因为如果 n = 1时，会被漏掉，当 n <= 2时，f(n) = n，所以为了更加严谨，我们可以写成这样：

```
// 算 n 的阶乘(假设n不为0)
int f(int n){
    if(n <= 2){
        return n;
    }
}
```

第三要素：找出函数的等价关系式

第三要素就是，我们要**不断缩小参数的范围**，缩小之后，我们可以通过一些辅助的变量或者操作，使原函数的结果不变。

例如，f(n) 这个范围比较大，我们可以让 f(n) = n * f(n-1)。这样，范围就由 n 变成了 n-1 了，范围变小了，并且为了原函数f(n) 不变，我们需要让 f(n-1) 乘以 n。

说白了，就是要找到原函数的一个等价关系式，f(n) 的等价关系式为 n * f(n-1)，即

f(n) = n * f(n-1)。

> 这个等价关系式的寻找，可以说是最难的一步了，如果你不大懂也没关系，因为你不是天才，你还需要多接触几道题，**我会在接下来的文章中，找一些递归题，让你慢慢熟悉起来**。

找出了这个等价，继续完善我们的代码，我们把这个等价式写进函数里。如下：

```
// 算 n 的阶乘(假设n不为0)
int f(int n){
    if(n <= 2){
        return n;
    }
    // 把 f(n) 的等价操作写进去
    return f(n-1) * n;
}
```

至此，递归三要素已经都写进代码里了，所以这个 f(n) 功能的内部代码我们已经写好了。

这就是递归最重要的三要素，每次做递归的时候，你就强迫自己试着去寻找这三个要素。

还是不懂？没关系，我再按照这个模式讲一些题。

> 有些有点小基础的可能觉得我写的太简单了，没耐心看？少侠，请继续看，我下面还会讲**如何优化递归**。当然，大佬请随意，可以直接拉动最下面留言给我一些建议，万分感谢！

案例1：斐波那契数列

> 斐波那契数列的是这样一个数列：1、1、2、3、5、8、13、21、34....，即第一项 f(1) = 1,第二项 f(2) = 1.....,第 n 项目为 f(n) = f(n-1) + f(n-2)。求第 n 项的值是多少。

1、第一递归函数功能

假设 f(n) 的功能是求第 n 项的值，代码如下：

```
int f(int n){

}
```

2、找出递归结束的条件

显然，当 n = 1 或者 n = 2 ,我们可以轻易着知道结果 f(1) = f(2) = 1。所以递归结束条件可以为 n <= 2。代码如下：

```
int f(int n){
    if(n <= 2){
        return 1;
    }
}
```

第三要素：找出函数的等价关系式

题目已经把等价关系式给我们了，所以我们很容易就能够知道 f(n) = f(n-1) + f(n-2)。我说过，等价关系式是最难找的一个，而这个题目却把关系式给我们了，这也太容易，好吧，我这是为了兼顾几乎零基础的读者。

所以最终代码如下：

```
int f(int n){
    // 1.先写递归结束条件
    if(n <= 2){
        return 1;
    }
    // 2.接着写等价关系式
    return f(n-1) + f(n - 2);
}
```

搞定，是不是很简单?

> 零基础的可能还是不大懂，没关系，之后慢慢按照这个模式练习！好吧，有大佬可能在吐槽太简单了。

案例2：小青蛙跳台阶

> 一只青蛙一次可以跳上1级台阶，也可以跳上2级。求该青蛙跳上一个n级的台阶总共有多少种跳法。

1、第一递归函数功能

假设 f(n) 的功能是求青蛙跳上一个n级的台阶总共有多少种跳法，代码如下：

```
int f(int n){

}
```

2、找出递归结束的条件

我说了，求递归结束的条件，你直接把 n 压缩到很小很小就行了，因为 n 越小，我们就越容易直观着算出 f(n) 的多少，所以当 n = 1时，你知道 f(1) 为多少吧？够直观吧？即 f(1) = 1。代码如下：

```
int f(int n){
    if(n == 1){
        return 1;
    }
}
```

第三要素：找出函数的等价关系式

每次跳的时候，小青蛙可以跳一个台阶，也可以跳两个台阶，也就是说，每次跳的时候，小青蛙有两种跳法。

第一种跳法：第一次我跳了一个台阶，那么还剩下n-1个台阶还没跳，剩下的n-1个台阶的跳法有f(n-1)种。

第二种跳法：第一次跳了两个台阶，那么还剩下n-2个台阶还没，剩下的n-2个台阶的跳法有f(n-2)种。

所以，小青蛙的全部跳法就是这两种跳法之和了，即 f(n) = f(n-1) + f(n-2)。至此，等价关系式就求出来了。于是写出代码：

```
int f(int n){
    if(n == 1){
        return 1;
    }
    return f(n-1) + f(n-2);
}
```

大家觉得上面的代码对不对?

答是不大对，当 n = 2 时，显然会有 f(2) = f(1) + f(0)。我们知道，f(0) = 0，按道理是递归结束，不用继续往下调用的，但我们上面的代码逻辑中，会继续调用 f(0) = f(-1) + f(-2)。这会导致无限调用，进入**死循环**。

> 这里有人觉得 f(0) 应该是等于 1，等于 0 还是等于 1 不是很重要，此次我们就当作是 0 处理哈。

这也是我要和你们说的，关于**递归结束条件是否够严谨问题**，有很多人在使用递归的时候，由于结束条件不够严谨，导致出现死循环。也就是说，当我们在第二步找出了一个递归结束条件的时候，可以把结束条件写进代码，然后进行第三步，但是**请注意**，当我们第三步找出等价函数之后，还得再返回去第二步，根据第三步函数的调用关系，会不会出现一些漏掉的结束条件。就像上面，f(n-2)这个函数的调用，有可能出现 f(0) 的情况，导致死循环，所以我们把它补上。代码如下：

```
int f(int n){
    //f(0) = 0,f(1) = 1, 等价于 n<=2时, f(n) = n。
    if(n <= 2){
        return n;
    }
    return f(n-1) + f(n-2);
}
```

有人可能会说，我不知道我的结束条件有没有漏掉怎么办？别怕，多练几道就知道怎么办了。

看到这里有人可能要吐槽了，这两道题也太容易了吧？？能不能被这么敷衍。少侠，别走啊，下面出道难一点的。

> 下面其实也不难了，就比上面的题目难一点点而已，特别是第三步等价的寻找。

案例3：反转单链表。

> 反转单链表。例如链表为：1->2->3->4。反转后为 4->3->2->1

链表的节点定义如下：

```
class Node{
    int date;
    Node next;
}
```

虽然是 Java语言，但就算你没学过 Java，我觉得也是影响不大，能看懂。

还是老套路，三要素一步一步来。

1、定义递归函数功能

假设函数 reverseList(head) 的功能是反转但链表，其中 head 表示链表的头节点。代码如下：

```
Node reverseList(Node head){

}
```

2. 寻找结束条件

当链表只有一个节点，或者如果是空表的话，你应该知道结果吧？直接啥也不用干，直接把 head 返回呗。代码如下：

```
Node reverseList(Node head){
    if(head == null || head.next == null){
        return head;
    }
}
```

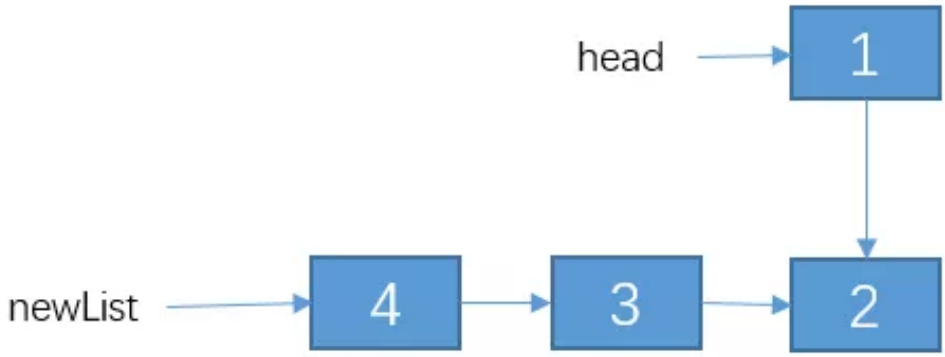
3. 寻找等价关系

这个的等价关系不像 n 是个数值那样，比较容易寻找。但是我告诉你，它的等价条件中，一定是范围不断在缩小，对于链表来说，就是链表的节点个数不断在变小，所以，如果你实在找不出，你就先对 reverseList(head.next) 递归走一遍，看看结果是咋样的。例如链表节点如下

我们就缩小范围，先对 2->3->4递归下试试，即代码如下

```
Node reverseList(Node head){
    if(head == null || head.next == null){
        return head;
    }
    // 我们先把递归的结果保存起来，先不返回，因为我们还不清楚这样递归是对还是错。，
    Node newList = reverseList(head.next);
}
```

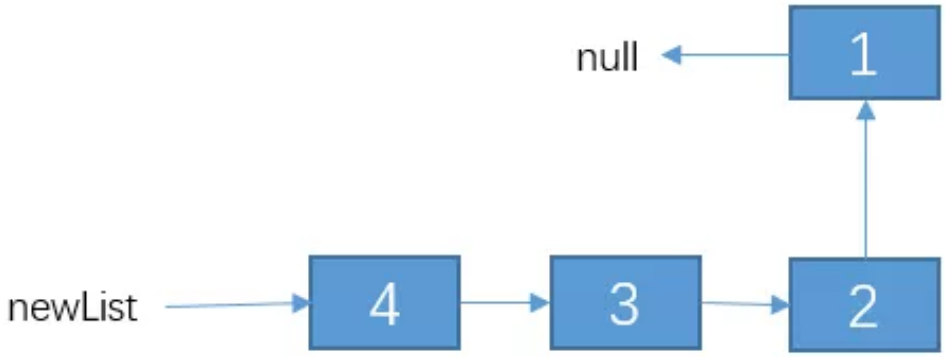
我们在第一步的时候，就已经定义了 reverseLis t函数的功能可以把一个单链表反转，所以，我们对 2->3->4反转之后的结果应该是这样：



我们把 2->3->4 递归成 4->3->2。不过，1 这个节点我们并没有去碰它，所以 1 的 next 节点仍然是连接这 2。

接下来呢？该怎么办？

其实，接下来就简单了，我们接下来只需要**把节点 2 的 next 指向 1，然后把 1 的 next 指向 null,不就行了？**，即通过改变 newList 链表之后的结果如下：



也就是说，reverseList(head) 等价于 **reverseList(head.next) + 改变一下1，2两个节点的指向**。好了，等价关系找出来了，代码如下(有详细的解释)：

```
//用递归的方法反转链表
public static Node reverseList2(Node head){
    // 1.递归结束条件
    if (head == null || head.next == null) {
            return head;
        }
        // 递归反转 子链表
        Node newList = reverseList2(head.next);
        // 改变 1，2节点的指向。
        // 通过 head.next获取节点2
        Node t1  = head.next;
        // 让 2 的 next 指向 1
        t1.next = head;
        // 1 的 next 指向 null.
       head.next = null;
       // 把调整之后的链表返回。
       return newList;
    }
```

这道题的第三步看的很懵？正常，因为你做的太少了，可能没有想到还可以这样，多练几道就可以了。但是，我希望通过这三道题，给了你以后用递归做题时的一些思路，你以后做题可以按照我这个模式去想。通过一篇文章是不可能掌握递归的，还得多练，我相信，只要你认真看我的这篇文章，多看几次，一定能找到一些思路！！

> 我已经强调了好多次，多练几道了，所以呢，后面我也会找大概一些递归的练习题供大家学习，不过，我找的可能会有一定的难度。不会像今天这样，比较简单，所以呢，初学者还得自己多去找题练练，相信我，掌握了递归，你的思维抽象能力会更强！

接下来我讲讲有关递归的一些优化。

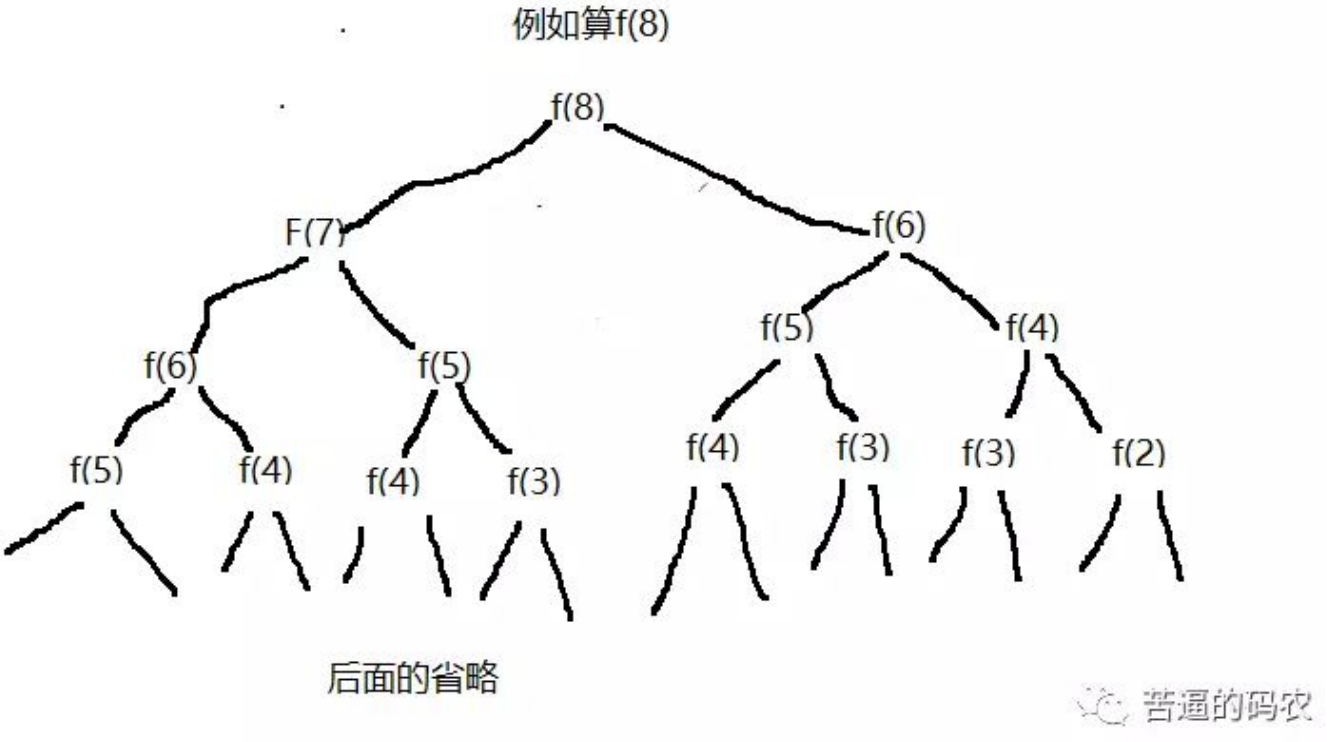
有关递归的一些优化思路

1. 考虑是否重复计算

告诉你吧，如果你使用递归的时候不进行优化，是有非常非常非常多的**子问题**被重复计算的。

> 啥是子问题？ f(n-1),f(n-2)....就是 f(n) 的子问题了。

例如对于案例2那道题，f(n) = f(n-1) + f(n-2)。递归调用的状态图如下：



看到没有，递归计算的时候，重复计算了两次 f(5)，五次 f(4)。。。。这是非常恐怖的，n 越大，重复计算的就越多，所以我们必须进行优化。

如何优化？一般我们可以把我们计算的结果保证起来，例如把 f(4) 的计算结果保证起来，当再次要计算 f(4) 的时候，我们先判断一下，之前是否计算过，如果计算过，直接把 f(4) 的结果取出来就可以了，没有计算过的话，再递归计算。

用什么保存呢？可以用数组或者 HashMap 保存，我们用数组来保存把，把 n 作为我们的数组下标，f(n) 作为值，例如 arr[n] = f(n)。f(n) 还没有计算过的时候，我们让 arr[n] 等于一个特殊值，例如 arr[n] = -1。

当我们要判断的时候，如果 arr[n] = -1，则证明 f(n) 没有计算过，否则， f(n) 就已经计算过了，且 f(n) = arr[n]。直接把值取出来就行了。代码如下：

```
// 我们实现假定 arr 数组已经初始化好的了。
int f(int n){
    if(n <= 2){
        return n;
    }
    //先判断有没计算过
    if(arr[n] != -1){
        //计算过，直接返回
        return arr[n];
    }else{
        // 没有计算过，递归计算,并且把结果保存到 arr数组里
        arr[n] = f(n-1) + f(n-2);
        reutrn arr[n];
    }
}
```

也就是说，使用递归的时候，必要
须要考虑有没有重复计算，如果重复计算了，一定要把计算过的状态保存起来。

2. 考虑是否可以自底向上

对于递归的问题，我们一般都是**从上往下递归**的，直到递归到最底，再一层一层着把值返回。

不过，有时候当 n 比较大的时候，例如当 n = 10000 时，那么必须要往下递归10000层直到 n <=1 才将结果慢慢返回，如果n太大的话，可能栈空间会不够用。

对于这种情况，其实我们是可以考虑自底向上的做法的。例如我知道

f(1) = 1;

f(2) = 2;

那么我们就可以推出 f(3) = f(2) + f(1) = 3。从而可以推出f(4),f(5)等直到f(n)。因此，我们可以考虑使用自底向上的方法来取代递归，代码如下：

```
public int f(int n) {
        if(n <= 2)
            return n;
        int f1 = 1;
        int f2 = 2;
        int sum = 0;

        for (int i = 3; i <= n; i++) {
            sum = f1 + f2;
            f1 = f2;
            f2 = sum;
        }
        return sum;
    }
```

这种方法，其实也被称之为**递推**。

最后总结

其实，递归不一定总是从上往下，也是有很多是从下往上的，例如 n = 1 开始，一直递归到 n = 1000，例如一些排序组合。对于这种从下往上的，也是有对应的优化技巧，不过，我就先不写了，后面再慢慢写。

说实话，对于递归这种比较抽象的思想，要把他讲明白，特别是讲给初学者听，还是挺难的，这也是我这篇文章用了很长时间的原因，不过，只要能让你们看完，有所收获，我觉得值得！

校招训练营

如果你信的过帅地，担心走弯路，也可以来帅地的训练营学习，具体情况可以看这个链接了解：训练营

如何找到帅地?

帅地有两个搜索公众号，你们可以关注，微信直接搜索对应名字就可以了：
每日问帅地：帅地每天会发布读者提问的一些问题

帅地玩编程：帅地会在这个公众号发布大学学习规划等文章

网站：https://www.playoffer.cn，所有的内容会沉淀在这个网站上