在《设计模式》这本小册中我们提到过，即使是在瞬息万变的前端领域，也存在一些具备“一次学习，终生受用”特性的知识。从工程的角度看，我推荐大家着重学习的是设计模式；从面试的角度看，性价比最高的知识体系则无疑是**算法与数据结构**。

很多前端同学在平日的学习里，只要看到“算法”或者“数据结构”这样的关键字，基本是拔腿就跑；进取心稍微比较强的同学，也难免陷入学了懵、懵了忘、忘了就投降的恶性循环。在实际的面试中，大部分同学面对算法和数据结构相关的面试题，知识储备几乎为0——这个短板，即便是有志于大厂面试的候选人，也少有人能逃脱。

（我希望这种学不会、学不好的宿命能够被这本小册终结掉）

换个角度看，作为一个前端，如果你愿意拾起这块知识，那么其实已经超越了不少知识储备为0的对手。不过站在面试的角度来看，仅仅是“拾起”、或者说只是“意思意思”、临阵磨枪做了那么几十道题，这样建立起来的知识结构，距离面试官对你的预期可能还是差些火候——注意，这里说的是“差些火候”，而非“不可抵达”。

直接来说，前端算法面试不需要你天资过人，不需要你拿过 ACM 的金银铜牌。它确实需要你投入一些时间来建立知识体系、精做好题。但是这不意味着你必须花上一年半载去杠一本《算法导论》，更不意味着无边无际的题海战术——面试是个技术活，不是体力活；理解面试本身，比理解算法更重要。

明确学习动机，不要购买焦虑——算法能力和工程能力，不应混淆

各位决定要学一样东西之前，首先要清楚自己是为什么而学——有目的性的学习，才能够带来最高的学习效率、才意味着你对时间的尊重。

相识即是缘，这里和大家分享一个学习观念：**前端工程师如果不是为了面试，那么不建议花大力气折腾算法（尤其是在业余时间本身非常有限的情况下），你应该考虑把更多的时间用来做工程**。

很多前端同学跟我说他刷算法题不是为了面试，而是为了“变强”。我问他为啥觉得刷题就能变得更强，他说公众号里说算法才是编程的灵魂，学了算法写项目就能超神。哎，其实哪有那么神奇，软件工程世界没有银弹，就算有，这个银弹也不会是算法。

（注：银弹这个词，很多同学可能在涉及前端工程化的各种文章中都见过或者听过、却不知道是啥意思，这算是行业“黑话”，指**极端有效的解决方案**。出自《没有银弹：软件工程的本质性与附属性工作》这篇论文，论文本身也非常nice，推荐感兴趣的同学有空读读）

把算法当银弹，这真的是一个很严重的误区——对前端工程师来说，最重要的是啥？在笔者看来，最最关键的是**工程能力**。
所谓工程能力，本质是“解决问题的能力”，无论是硬编码实力、还是架构思想，其本质都是为了解决问题这个终极目标而服务。

算法训练固然会从一定程度上辅助到你的工程能力（比如提高你代码的严谨度、开拓解决问题的思路等等），但肯定没有直接做工程来得快。
（btw说到工程能力，为什么不去学学《设计模式》和《前端性能优化》呢orz...）

我有时候看到一些培训机构说学了算法就晋升了、学了算法就加薪了，这样的说法实在不太现实——对前端来说，硬编码能力完全可以从工程中积累，稍微复杂的算法实现也大多有现成的库可以调用，业务开发中、甚至说技术攻关时，算法都很难构成瓶颈。因此目前来看，学算法最大的意义仍然是帮你搞定面试、提高你 offer 的 base。

在笔者从业的数年中，见过太多工程能力极强、却连二叉树都翻转不动、甚至说不知道啥是斐波那契数列、不会写快排的前端，其中不少还是相当资深的前辈老哥。这些人不会写算法，是真的；这些人强到头秃，也是真的。所以说算法能力和工程能力不能混为一谈，具体问题就应该具体解决。

克服能力短板，不做算法逃兵

话说回来，既然不会写快排也不影响做工程，为啥现在前端面试却总爱考算法?

关于这点，我个人理解其实是一个区分度的问题——早年的前端从业人数少、业务逻辑本身也比较轻。现在嘛...你懂的（笑）。无论是为了检验候选人的逻辑思维能力、考察“聪明度”，还是说单纯只为了过滤掉不具备科班基础的候选人，算法题目总是能够又快又好地帮助前端团队达到各种各样的招聘目的。

至于“意义”这个东西，其实是见仁见智。比如笔者个人就很不推崇前端团队用算法来衡量前端工程师的价值，但笔者老板所持的观点却恰恰相反..... （摊手）。

总之，作为一种简单、高效、易操作的人才筛选手段，许多前端团队（尤其是大厂团队）围绕算法这个命题点，都会有它自己的一套考察标准。

与之对应的，是很多同学在准备面试的时候，会以“前端学算法没意义”来作为自己逃避“老大难”的借口。如果你也曾持有、或正在持有这种想法，希望你能认识到两件事情：

1. 逃不掉的，老哥。
2. 算法面试考察的内容非常稳定，是真正的“一次学习、终生受用”，它值得你投入时间。

因此，站在候选人的角度来考虑，“意义”清不清楚其实不要紧，关键是“会考”这一点是明确的。
不管你即将面对的是什么样的团队、什么脾气的面试官，算法能力都不建议裸奔。

不吹不黑聊聊面试—算法能力并非少数人的“专利”

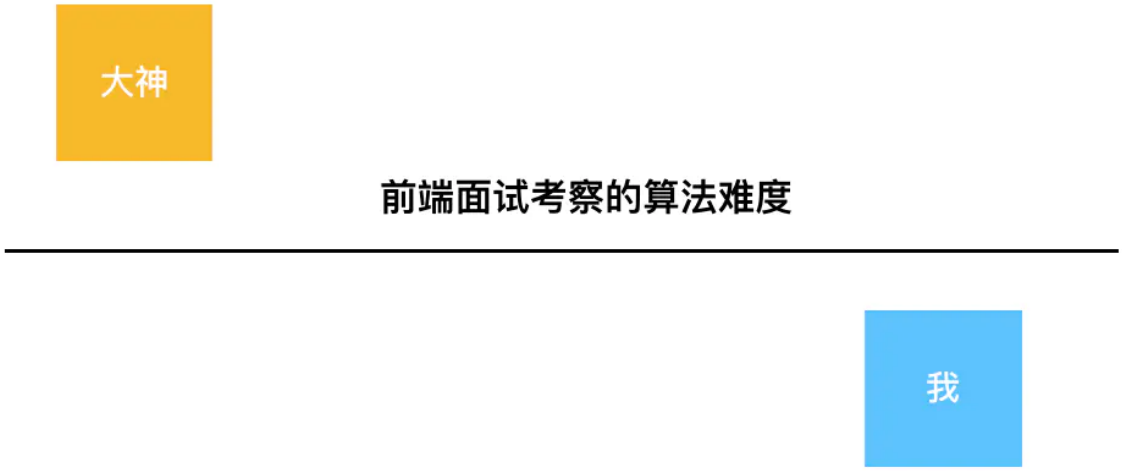
面试场景下所要求的算法能力并非少数人的专利，而是普通人通过合理的训练也能够习得的必要本领。

为了佐证这一点，我来给大家举个例子：
这本小册立项在19年3月份，一年多以来，出于个人发展的需要，笔者也前前后后参加了不少面向前端/全栈工程师的算法面试，在这个过程中，接触到的候选人基本可以分为两类：

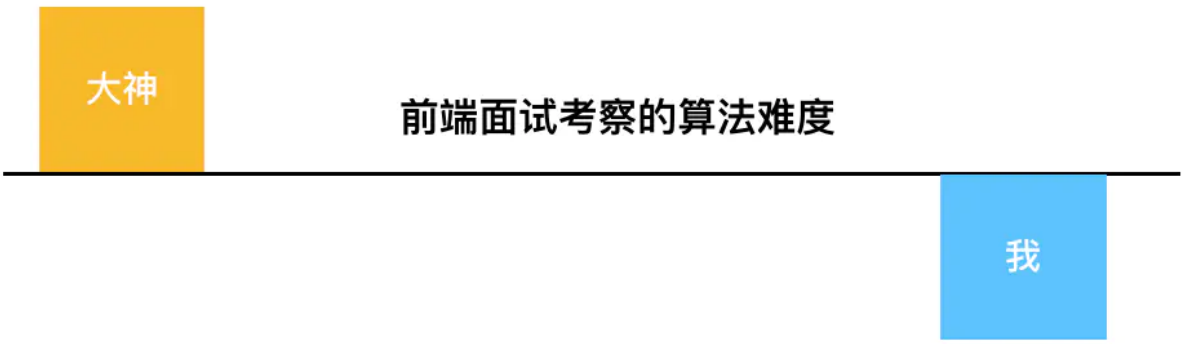
1. 普通人，比如我。非科班出身，0算法竞赛经历（工程竞赛倒是搞了一堆哈哈哈/羞），科班基础靠自学，在大量生产级别的工程中成长、早期进大厂也基本靠性能优化、工程化等面向大前端的研发能力（那时，国内前端圈还不流行考算法）。
2. 竞赛选手/类竞赛选手。竞赛选手一般是科班出身，ACM 金/银/铜获奖者或者至少有个说得上的名次。这波同学对算法的积累普遍会比较“厚”，其中不乏天赋型选手。所谓“类竞赛选手”，指虽然没有参加

过大型算法竞赛，但个人对算法存在执念，业余时间光《算法导论》能刷好几轮的同学（还真有，而且不止一个orz）。

诚然，**对竞赛选手来说，做完竞赛题再去做面试题，可以理解为降维打击。但对于笔者来说，在前期没有特别研究过算法的情况下，直接上手面试题都还有点吃力**。我们三者的水平关系最初大概如下图：



作为高端玩家，竞赛选手通过适度降低自己的身段，自然能够匹配到算法面试要求的水平线；而作为一个求生欲极强的非科班选手，笔者在修福报之余，**通过科学的刷题训练、总结套路，也可以抵达算法面试要求的水平线**。所以最后上考场的时候，我们三者的关系可能就会发生一些变化：



虽然笔者至今也不知道大神是凭借什么样的脑回路成为大神的、更没有机会去接受针对竞赛同学提供的“高端教学”，**但笔者在强烈的求生欲驱使下通过孜孜不倦的摸索和探究、也能够内化出一套属于自己的“打法”，最后达成自己的面试目标**。

同时，由于笔者深深地知道做对一道题不容易，反而在解题时会更加谨慎，甚至会去做很多高端玩家不屑于做的努力——比如尝试去揣摩出题人的意图、研究不同题目类型的分布情况和考察频率、研究如何在不会的题目上拿分等等。**这样只针对面试、只解决面试问题的“专注模式”，使笔者在前端面试这个维度的算法题上所向披靡。再加上本身是做工程出身，两手一起抓，有时在整体面试结果上甚至会超越竞赛选手**。

从另一个角度看，大神或许生来就是大神、硬杠智力取胜；而普通人则需要摸爬滚打才能摸索出普通人的赢法。大神或许永远不懂普通人从0开始有多难，但只要有一个普通人能够从0到1，他的赢法就存在着高度复用的可能性。大神讲题听不懂很正常，毕竟是降维打击；但咸鱼翻身的普通人给仍在挣扎的普通人讲题，那就是普通人之间的惺惺相惜、几乎不存在理解屏障——**这本小册的目的，就是做没有理解屏障的讲解**。

没错，算法确实是博大精深的一门学问，笔者至今仍不懂“算法之美”，但这并不影响笔者解决大多数大厂和外企的算法面试题——毕竟，在这样的场合，理解面试本身，比理解算法更重要。

以题为纲——不讲“算法之美”，只有“解题套路”

学习这本小册，你需要的前置知识很简单——JavaScript 基础，以及一点点的 ES6。

我们的目的也很简单——在有限的时间内，做对关键的已知题目；通过对关键的已知题目进行解构和反思，消化掉其中核心的算法思想和解题套路，从而最大化各位在真正面试时做对任意未知题目的可能性。

其实不难

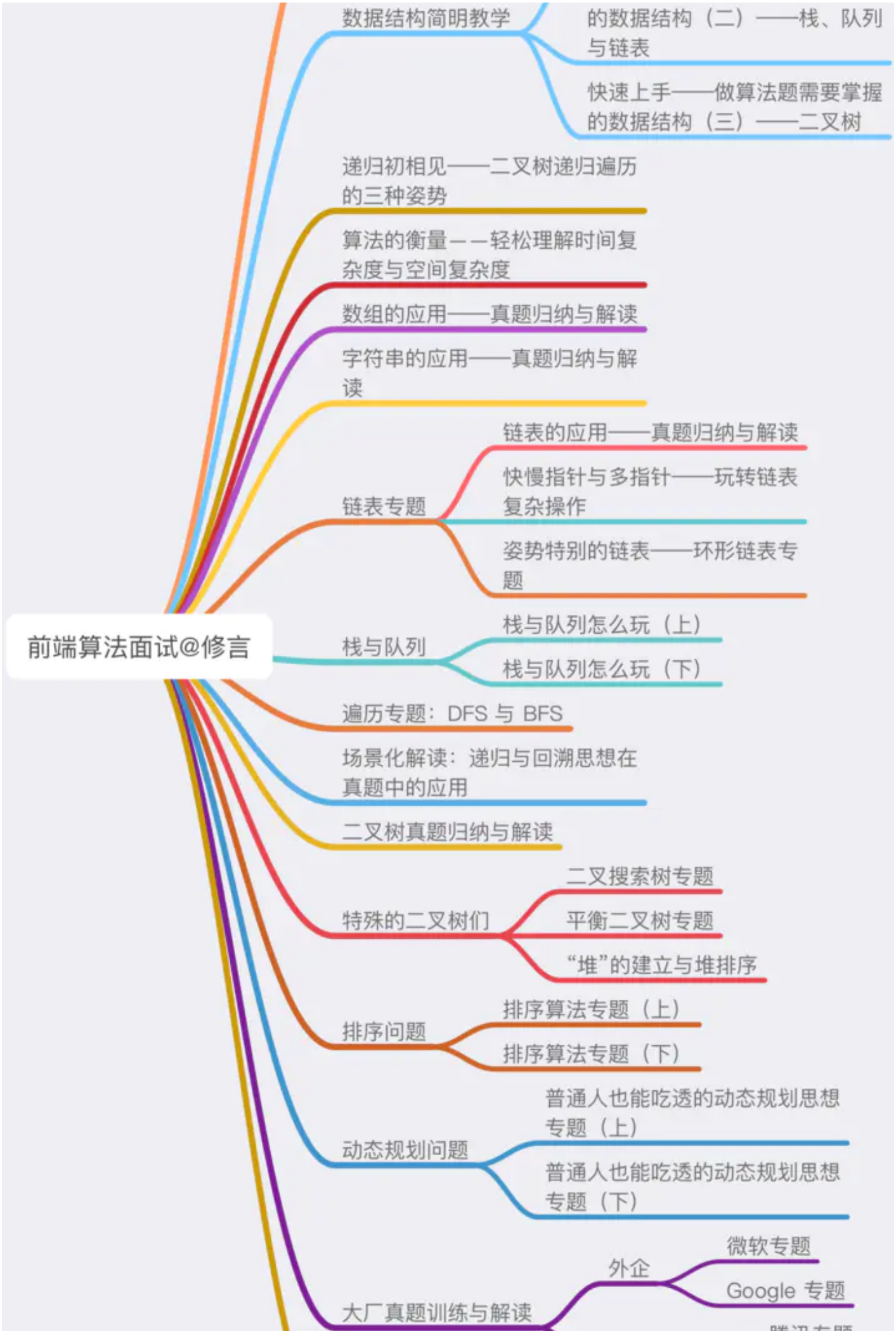
许多同学对面试的恐惧，其实本质上是对“未知”的恐惧——他不知道自己面试的时候到底会不会做到跨越自己知识边界的题。事实上，大可不必为此焦虑，就像再完美的学霸也未必做到每场考试都拿150分一样——面试题是具有随机性的，没有哪一种备考手段，可以确保我们在解决任何一道随机面试题时都能做到百分百不出错。
与此同时，实际上任何一场算法面试，都不会单单因为你没有做对所有的题目而给你亮红灯。`拿 offer !== 拿满分`。我们的目标是拿 offer，而不是拿满分。拿到满分需要一些运气，或许没有固定的规律可言；但拿到 offer 的规律就明确多了——比如需要扫除哪些知识盲点、需要建立怎样的分析问题的脑回路、遇到不会的题目如何尽可能在解题过程中做出正确的表现、赢得面试官的好评等等（关于面试官定义和分析候选人的脑回路，我们等知识体系搭起来之后会和大家仔细掰扯，前期我们最重要的还是要好好做题、做好题），这些我们都可以通过面试前做充分的准备来把握主动权。

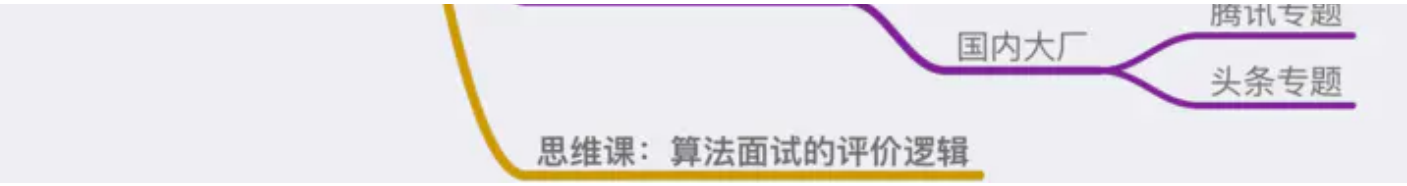
方法论与知识结构

这里同时也要给大家抛出我们小册提倡的学习方法——**以题为纲**。从优质的题目中学习、从有关联的题目中提取知识点和解题套路，以此来建立自己的算法面试知识体系。

另外，关于许多同学早期反馈过的“算法理论太抽象”这个问题，在本书中我也会给出我的解法——所有的理论性知识都会**以图解为主、文字为辅**的形式来作讲解，把抽象的理论转换为具体的**画面和动图**。一图胜千言，就算你之前对相关理论没有任何认知，也不妨碍你从0开始理解。

我们整个小册的知识结构规划如下：


前端算法面试@修言
数据结构简明教学
的数据结构（二）——栈、队列与链表
快速上手——做算法题需要掌握的数据结构（三）——二叉树
递归初相见——二叉树递归遍历的三种姿势
算法的衡量——轻松理解时间复杂度与空间复杂度
数组的应用——真题归纳与解读
字符串的应用——真题归纳与解读
链表专题
链表的应用——真题归纳与解读
快慢指针与多指针——玩转链表复杂操作
姿势特别的链表——环形链表专题
栈与队列
栈与队列怎么玩（上）
栈与队列怎么玩（下）
遍历专题：DFS 与 BFS
场景化解读：递归与回溯思想在真题中的应用
二叉树真题归纳与解读
特殊的二叉树们
二叉搜索树专题
平衡二叉树专题
“堆”的建立与堆排序
排序问题
排序算法专题（上）
排序算法专题（下）
动态规划问题
普通人也能吃透的动态规划思想专题（上）
普通人也能吃透的动态规划思想专题（下）
大厂真题训练与解读
外企
微软专题
Google 专题



前20+个小节均为（基于大量真题的）讲解性质小节，这部分内容主要的目的是帮助大家吸收基础知识、吸收做题的方法和思路。在题目分布上，小册会尝试对齐真实前端面试场景下的分布情况：以 medium 难度为主，辅以一部分的 easy 题目用作引导、再辅以少量的 hard 题目开拓思路。

小册所涉及的题目难度，会随着你学习的逐渐深入一点一点地拔高。等到讲解部分完全结束后，大家会对算法知识体系有一个整体的把握，在这个基础上，“大厂真题训练”环节，我会带大家一起做一些综合性更强、区分度也更高的题目。

笔者注：

> 和设计模式小册一样，在本书中，笔者也做了一些内容设计上的探索。 相信大家都会有这样的记忆：在学习一些优质的前端框架时，其作者为了能够迅速抓住开发者的感性认知，会在文档的开头设计“快速上手”环节，目的是为了帮助开发者以最快的速度建立起驾驭框架的自信。 本书尝试学习这些优质文档的目录结构、将本身理论性较强的数据结构知识，转化为编码占主导地位的“快速上手”环节。希望能够借此引导各位从具体出发去认知抽象，以最快的速度找到写代码的感觉、切入迎战算法面试题的状态。

现在，明确了学习目的，明确了学习路径，也明确了“万里长征的第一步”。接下来，我们就从数据结构开始、先帮各位砸实硬编码能力的基本功。

冲！