技术移民宝典

程序员海外求职锦囊

陈东锋　涂锋　著

中国工信出版集团　电子工业出版社

技术移民宝典

程序员海外求职锦囊

陈东锋　涂峰　著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书介绍了如何在硅谷求职，以及剖析了具有代表性的 43 道热门硅谷公司的面试题，从面试技巧、基础知识、解题思路和效率优化等方面总结面试和解题规律。全书分为四部分共 15 章，包含出国工作途径、IT 求职准备、实战访谈，以及常见数据结构、算法、大数据、系统设计等方面的题目和解题思路，并提炼出解题的 5 个步骤：复述/提问、举例、观察、编码和测试。本书精选出的面试题是硅谷热门公司的高频题，可以用来作为面试前的练习。对于每道题，本书尽可能给出多种解法，对于解决日常工作中遇到的问题也有一定启发性。

本书适合正在应聘程序员相关职位的就业人员阅读和参考，特别是打算寻求美国 IT 公司职位并想通过技术移民实现美国梦的程序员。

图书在版编目（CIP）数据

技术移民宝典：程序员海外求职锦囊 / 陈东锋，涂峰著. —北京：电子工业出版社，2016.4

ISBN 978-7-121-28195-2

Ⅰ. ①技… Ⅱ. ①陈… ②涂… Ⅲ. ①电子计算机－工程技术人员－职业选择－基本知识－美国 Ⅳ. ①TP3

中国版本图书馆 CIP 数据核字(2016)第 033614 号

策划编辑：符隆美
责任编辑：徐津平
印　　刷：三河市华成印务有限公司
装　　订：三河市华成印务有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：720×1000　1/16　印张：13　字数：250 千字
版　　次：2014 年 3 月第 1 版
　　2016 年 4 月第 2 版
印　　次：2016 年 4 月第 1 次印刷
印　　数：4000 册　定价：45.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

随着越来越多 IT 工程师寻找国外工作机会，介绍和总结国外热门 IT 公司面试过程及面试内容的需求变得越来越迫切。美国最新移民改革 CIR 方案更倾向于技术移民，这将使得今后会有更多国内程序员去美国工作。笔者亲身参与了国内和美国一些热门 IT 公司的面试，同时也作为面试官面试过不少人，熟知海内外 IT 公司招聘流程和面试方式。通常来说，去美国 IT 公司工作有三种途径。

- 直接申请美国公司职位，拿 H1B 签证工作。不少热门 IT 公司直接在国内招人，比如 Facebook、Amazon、Microsoft、Google 等。越来越多的程序员选择这条路，一方面是因为美国签证的条件放宽了，另一方面是因为硅谷公司面试并没有比国内公司难多少。
- 在国内的跨国公司工作一年后，内部转组到美国的分部，使用 L1 签证。例如，从微软中国转至微软西雅图总部工作。
- 申请攻读美国学校的计算机科学硕士或博士学位，毕业后再找工作，即由 F1 签证转为 H1B 签证。

这三种途径都需要成功通过公司的技术面试。热门 IT 企业的面试方式大

致相同：1 或 2 轮电话面试，通过之后，又有 4 或 5 轮的现场面谈。其中 80% 的面试是技术面试，每轮技术面试大约 45 分钟，扣除双方自我介绍和提问时间，花在技术面试的时间大约为 30 分钟。由于技术面试时间的限制，面试的题目一般不会太难，比大学生编程比赛（ACM）的题目简单很多，但是，面试者需要具备一些编程面试技巧，并对算法、数据结构熟练掌握才能在限定时间内完成。这对要求在白板上写程序和无 Bug（Bug free）的公司来说尤其重要，比如 Facebook。

在编程面试过程中，光有解法却写不出来代码是行不通的，这只会让面试官觉得你夸夸其谈，不会实际编程。在编程面试里，切记“**让代码说话**”这条准则。在本书面试题相关的章节中，笔者贴出了面试题的全部代码，是为了更多时候让代码来说话。针对每道面试题，我们通常会有如下步骤。

- **复述/提问**：用自己的话复述面试官的题目，以免偏题。面试官给出的面试题并非一开始就很明确，需要多次问答来确定题意、边界条件、时间和数据结构限制等。
- **举例**：可以与提问同步进行，主要用来确认输入和输出结果。
- **观察**：通过举例来总结规律，思考可能使用到的结构和算法，然后设计一种你认为最优的算法。
- **编码**：和面试官沟通你的算法之后，开始在白板编码。
- **测试**：使用个别例子，把你的代码测试一遍。

在以上 5 个步骤里，看时间是否充裕，有些步骤可以省略。比如，如果面试官已经把问题说得很清楚了，那么复述可以省略。在本书当中，笔者也会按照这 5 个步骤的解题技巧来阐述面试题的解题方案。

笔者根据自身作为面试官的多年经历，并收集了网上众多的热门 IT 公司面试题目，精选了 43 道题来代表当前热门和高频的面试题。本书内容覆盖了基础的数据结构：数组、链表、树、堆栈、字符串等，以及高频率出现的算法，

如动态规划、俩指针、排列组合、优先遍历等。此外，还覆盖了系统设计解题思路和案例分析。涂峰加入了再版的写作，以自身“肉身翻墙”的经历为本书丰富了求职准备、实战访谈、系统设计等章节。本书的内容分为以下四个部分。

- **硅谷求职**：硅谷公司文化、技术移民、简历、面试、录用谈判和职业发展。
- **实战访谈**：先行者所传授的出国途径、面试技巧、硅谷企业文化、职业规划。
- **算法面试**：动态规划、俩指针、优先遍历、哈希、排列组合。
- **系统设计**：实战技巧、案例分析、阅读推荐。

此外，附录还提供了数据结构和算法总结及海量数据分析，以供读者快速查阅。

本书有以下几个特点。

- 本书是市面上第一本介绍硅谷求职和技术移民美国的书。
- 本书也是第一本对已实现技术移民美国的先行者进行访谈的书。
- 精选出的面试题是硅谷各家热门公司的高频题，极其具有代表性。
- 完整介绍了系统设计题的实战技巧，对应聘高级职位者很有借鉴意义。
- 总结了常见数据结构的对应算法，提炼出一套解题规律。对于类似题目，有很强的借鉴意义。
- 本书提供了完整的可运行的源代码。
- 对于每道题，本书尽可能给出多种解法，对解决我们在日常工作中遇到的问题有一定启发性。

虽然本书大部分的代码是用 Java 编写的，但很容易转化为 C++/.NET 代码，因此，本书也适合 C++/.NET 程序员阅读。

由于本人水平有限，书中的题目并不能完全代表当前热门公司的编程面试

的各个方面，虽然经过多轮审核，不少解法依然可能有漏洞或者错误，希望广大读者能给予指正。

在本书的写作过程中，我得到了很多朋友、同事的帮忙，包括汪纯子、周泽勇、俞明辉、吴盛萱、杨超、尹杭锋和于东东等。感谢他们帮忙校对文字、审核代码。同时，感谢电子工业出版社的工作人员，尤其是符隆美——大到全书的架构，小到文字的推敲，她都给予了我极大的帮助，从而使本书的质量有了极大的提升。

最后，我要衷心地感谢我的妻子 Emily。感谢她在过去几年中对我的理解和支持，为我营造了一个温馨而浪漫的家，让我能够心无旁骛地写书。谨以此书献给她以及我们的女儿 Ella。

陈东锋

2015 年 11 月于上海张江

目　　录

第一部分　硅谷求职

第 1 章　硅谷公司 3

1.1　硅谷简介 3

1.2　传奇旗帜 7

1.2.1　微软 8

1.2.2　谷歌 10

1.2.3　亚马逊 11

1.2.4　Facebook 13

1.2.5　Twitter 14

1.2.6　Epic 14

1.3　技术移民 15

1.3.1　签证和绿卡 16

1.3.2　税率和生活 19

第 2 章　求职准备 21

2.1　职位选择 23

2.2 公司选择 24
2.3 人际关系 27
2.4 求职渠道 30

第 3 章 简历 32

3.1 简历特点 33
3.2 简历结构 36
3.3 简历优化 39

第 4 章 面试 43

4.1 面试准备 43
4.2 面试流程 49
4.3 编程面试 51
4.4 注意事项 52

第 5 章 聘书与职业发展 56

5.1 聘书 57
5.1.1 聘书要素 57
5.1.2 决策因子 58
5.1.3 薪酬谈判 61
5.1.4 接受、延期或婉拒 63
5.2 职业发展 64
5.3 优秀工程师 66
5.4 职业晋升 70

第二部分 实战访谈

第 6 章 对身在美国和即将赴美工作的工程师访谈 77

互联网资深大牛董飞 77

创业者徐淼华 82
留美计算机博士张喆 85
微软软件工程师乔成 88
Broadcom 硬件测试工程师蒋波韡 90
硅谷初创公司大数据处理软件工程师常新宇 93

第三部分　算法面试

第 7 章　俩指针 99

面试题 1：两数之和 I ☆☆ 99
面试题 2：两数之和 II ☆☆☆☆ 101
面试题 3：Top K☆☆☆ 103
面试题 4：两数组第 k 个值☆☆☆☆☆ 107
面试题 5：有序数组去重☆ 109
面试题 6：数组分水岭☆☆☆ 111

第 8 章　动态规划 113

面试题 7：最长递增子序列☆☆☆☆ 114
面试题 8：最小化数组乘积☆☆☆☆ 116
面试题 9：刷房子☆☆☆ 117
面试题 10：编辑距离☆☆☆☆ 118
面试题 11：最长回文子串☆☆☆☆☆ 120
面试题 12：最大公共子串☆☆☆☆ 121

第 9 章　优先遍历 123

面试题 13：填充图像☆☆☆☆ 123
面试题 14：单词替换规则☆☆☆☆ 124
面试题 15：有向图遍历☆☆☆☆ 126

第 10 章　哈希 128

面试题 16：最长不同字符的子串☆☆☆☆ 128
面试题 17：常数时间插入删除查找☆☆☆ 129
面试题 18：对数时间范围查询☆☆☆☆ 130
面试题 19：实现 LRU 缓存☆☆☆☆ 130
面试题 20：经过最多点的直线☆☆☆ 133

第 11 章　堆栈 136

面试题 21：局部最大值☆☆☆ 136
面试题 22：数据流最大值☆☆☆☆ 138
面试题 23：产生逆波兰式☆☆☆ 139
面试题 24：逆波兰式计算☆☆☆ 140
面试题 25：设计 Min 栈☆☆☆☆ 142
面试题 26：最小公共祖先☆☆ 143
扩展问题 1 144
扩展问题 2 147

第 12 章　排列组合 149

面试题 27：翻译手机号码☆☆☆ 149
面试题 28：数组签名☆☆☆☆ 151
面试题 29：组合和☆☆☆ 153
面试题 30：N 皇后☆☆☆☆ 155

第 13 章　杂项 157

面试题 31：实现迭代器 peek() ☆☆☆ 157
面试题 32：实现复杂的迭代器☆☆☆☆ 158
面试题 33：实现 BlockingQueue ☆☆☆ 160
面试题 34：随机数产生器☆☆☆☆☆ 161

面试题 35：找出明星☆☆☆ 163
面试题 36：根据概率分布产生随机数☆☆☆☆ 163
面试题 37：随机采样☆☆☆ 164
面试题 38：统计电话号码个数☆☆☆ 165
面试题 39：海量数据高频词☆☆☆ 166
面试题 40：多台机器的中值☆☆☆☆ 166

第四部分　系统设计

第 14 章　实战技巧及准备 171

14.1　实战技巧 172
技巧 1：不要惊慌 172
技巧 2：与面试官积极交流 173
技巧 3：厘清需求 173
技巧 4：先框架再细节 174
技巧 5：留意错误处理 174
14.2　常见知识点 175
14.3　如何准备 177

第 15 章　系统设计例题 180

面试题 41：大数据存储☆☆☆☆ 180
面试题 42：大并发处理☆☆☆☆ 182
面试题 43：大数据收集☆☆☆☆ 185
系统知识阅读 188

附录 A　数据结构与算法 191

附录 B　海量数据结构 192

第一部分　硅谷求职

第 1 章 硅谷公司

硅谷是世界 IT 中心，汇聚了顶级的电子、芯片、软件、互联网等 IT 公司。本章将介绍硅谷的地理位置和发展历程，简述六家具有代表性的硅谷公司的文化和特点，最后描述国内的软件工程师寻找美国工作的三种途径及技术移民美国的方法。

1.1 硅谷简介

硅谷（Silicon Valley）指美国加利福尼亚州的旧金山以南，沿着 101 公路从门罗公园、帕拉托、山景城、太阳谷到硅谷的中心圣克拉拉，再经坎贝尔直达圣何赛的这条狭长地带。硅谷这个词最早见于 1971 年 1 月 11 日《每周商业》报纸关于电子新闻的标题。之所以名字当中有一个“硅”字，是因为当地企业多数是从事加工制造高浓度硅的半导体行业和电脑工业，而“谷”字则是从圣克拉拉谷中得到的。

硅谷气候宜人，四季如春。庞大的信息产业结合完美的气候和优美的环境聚集了世界顶级的高科技人才，包括数以万计的中国优秀科技人才。有人说，北京和硅谷的区别并不在于天空的颜色，而在于一边的年轻人大多谈论的是买房、买车，被催着结婚生子，可另一边的年轻人谈论的则是改变世界的梦想以及如何让自己一辈子没白活。你能想象十几年前睡在硅谷大学大道 165 号车库地板上那几个 20 岁出头的小伙子吗？他们后来分别创办了 Youtube、LinkedIn 和 Yelp。

上千家高科技公司的总部设在硅谷（见图 1），近 40 家被福布斯收录在世界 500 强排名中。硅谷暴富的故事吸引一批又一批年轻人来硅谷创业。成功的创业公司总是具有这样的传奇色彩：不用考虑你的社会阶层、教育、种族、国籍或其他的各种因素，只要你是人才，便能出人头地。这就是硅谷的核心精神：只要有智慧、野心和伟大的想法，任何人都可以筹集资金开公司。

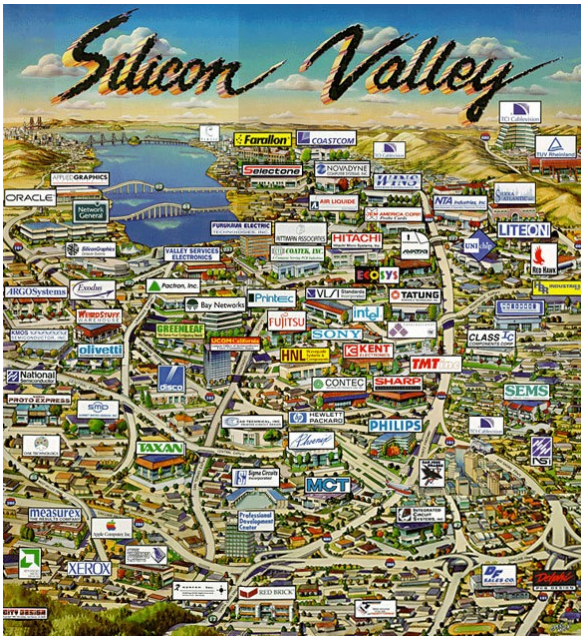


图 1　硅谷公司地图

从 20 世纪 60 年代起，硅谷这块狩猎的沼泽地飞速发展，在短短几十年后成为高科技工业中心，创造了巨大的物质财富，取得了引以为傲的非凡成就。当硅谷取得巨大成功之后，不少人试图在美国其他地区复制硅谷，但均以失败告终。硅谷之所以成为世界 IT 中心，是因为其特定的时代背景。硅谷取得成功的原因很多，而以下几个因素对硅谷的发展至关重要。

一、以技术推动行业发展的时代背景。这种时代发展主要表现在三波技术飞跃上，而这三波均起源于硅谷。第一波，晶体管的发明带动了半导体行业快速发展。1956 年，晶体管的发明人威廉 · 肖克利（William Shockley）在斯坦福大学南边的山景城创立了“肖克利半导体实验室”。来自该半导体实验室的八位工程师创办了仙童（Fairchild）半导体公司，他们被称为“八叛逆”。“八叛逆”里的诺伊斯和摩尔后来创办了英特尔（Intel）公司。在仙童工作过的人中，斯波克后来成为国民半导体公司的 CEO，另一位桑德斯则创办了 AMD 公司。第二波，软件产业兴起。除了半导体工业，硅谷同样以软件产业著称。施乐公司在 Palo Alto 的研究中心在 OOP（面向对象的编程）、GUI（图形界面）、以太网和激光打印机等领域都有开创性的贡献。现今的许多著名企业都得益于施乐公司的研究，例如苹果和微软先后将 GUI 用于各自的操作系统。第三波，20 世纪 90 年代末互联网服务兴起以及新世纪的移动互联网跳跃式的发展，带动了全世界互联网发展，加快信息流通，方便了世界各地的人们交流。

二、斯坦福大学及斯坦福工业园。二战结束后，美国返回大学的学生骤增。为满足财务需求，同时给毕业生提供就业机会，斯坦福大学开辟了工业园，允许高新技术公司租用此地作为办公区。最早入驻的公司是 1930 年由斯坦福毕业生创办的瓦里安公司（Varian Associates）。工业园同时为民用技术的初创企业提供风险资本。惠普公司是最成功的例子之一。在 20 世纪 90 年代中期，斯坦福工业园逐渐成为技术中心。作为世界十大名校之一的斯坦福大学，源源不断地给 IT 公司输送人才，同时也培养了大量的创业家。

三、活跃和宽容的社会环境。硅谷的许多人来自美国东部和西欧，他们为

摆脱墨守成规的文化和官僚主义的束缚，被加州的特殊机会吸引而来。正如一位风险投资家所说：“东部乃是大公司的地盘，壁垒森严，个人很难立足其间。而加州则是前线，从经济、社会和组织形式来看都没有固定的模式，而且最重要的是它真正重视个人的价值”。近年来，来自世界各地的不少年轻人在硅谷圆了自己的创业梦。尽管失败者占多数，但整个社会对创业失败者有着比其他地方更多的包容。

四、良好的自然条件及完善的基础设施。正如我们前面谈到的，整个旧金山湾有优越的地理条件，阳光明媚、气候舒适，有“天然空调”之称。北加州原有的高质量生活，吸引人们来到此地生活工作。不仅有便利的交通、快捷的通信，更有世界一流的大学，例如斯坦福大学、加州大学伯克利分校等，为发展提供了有力的科技后援——优良、丰富又具流动性的高科技人才。这一切营造出了良好的科技和商业环境。

五、发达的资本市场。美国高科技企业的创业与风险投资的关系很大。苹果电脑公司 1976 年创办时，投资企业家马克库拉投资 9 万美元，借贷 25 万美元，占 30%股份，从而推动了苹果电脑的发展，进而使革命性的个人电脑成为新兴产业。1982 年 Adobe 公司创建时得到了著名风险投资公司 H&Q 的支持，后者获得了百倍的利润回报；Adobe 公司在成长壮大后，又与 H&Q 合资成立了新的风险投资公司，支持高科技企业的建立，得到了丰厚的回报。上述两家公司的建立都起源于独创性的科研成果，而风险投资使成果能及时转化、占领市场，并分别形成个人电脑和电子化出版业这两个新兴产业。没有风险投资的参与，仅靠一人或数人的有限财力，很难使公司快速发展。在美国有许多专业的、高素质的投资家，他们有着丰富的投资经验，一项独创性的技术很容易吸引到大量的资金投入，所以在美国经常可以听到高科技人员一夜之间成为亿万富翁的消息。

1.2 传奇旗帜

Myth flag，译为“传奇旗帜”，分别是八家公司的首字母组合：Microsoft（微软）、Yahoo（雅虎）、Twitter（推特）、Hortonworks（一家新兴的云计算公司）、Facebook（脸书）、LinkedIn（最大职业社交网站）、Amazon（亚马逊）和 Google（谷歌）。这八家公司是硅谷的软件和互联网公司的典型（从严格意义上来说，亚马逊和微软不属于硅谷公司，虽然它们在硅谷有分部，但总部在西雅图）。其中，微软代表了传统软件公司，此外还有甲骨文、Teradata、Adobe 和苹果公司等；雅虎和谷歌是老牌互联网公司的代表，而 Facebook 和 Twitter 则属于新兴互联网公司，发迹于 Web 2.0 的兴起。而最具有活力和爆发力的公司是新一代云计算和移动互联网公司，它们中大部分还在创业阶段，还未上市，Hortonworks 就是其代表之一（再版时 Hortonworks 已经成功上市）。

不同的公司有不同的企业文化。评判一家公司企业文化的标准有很多，其中之一就是组织结构图。图 2 是一张来自网络的有趣的组织结构图，里面包括了亚马逊、谷歌、Facebook、微软、苹果、甲骨文这六家公司的组织结构图。从图中我们可以看出亚马逊有着严格的等级制度；谷歌也有清晰的等级，但是部门之间相互交错；Facebook 就像是一张分布式对等网络；微软则是各自“占山为王”，拥有浓厚的办公室政治文化，类似国企；苹果则是一个人说了算，属于个人独裁，围绕着乔布斯或新 CEO 库克转；最具讽刺意义的是最后的甲骨文，法务部门远远大于工程部门，随时准备向竞争对手提起诉讼，或者应诉。

接下来，我们从上千家软件和互联网公司中挑出六家（这六家拥有大量来自中国的员工），重点从工程师角度阐述公司及公司的文化。

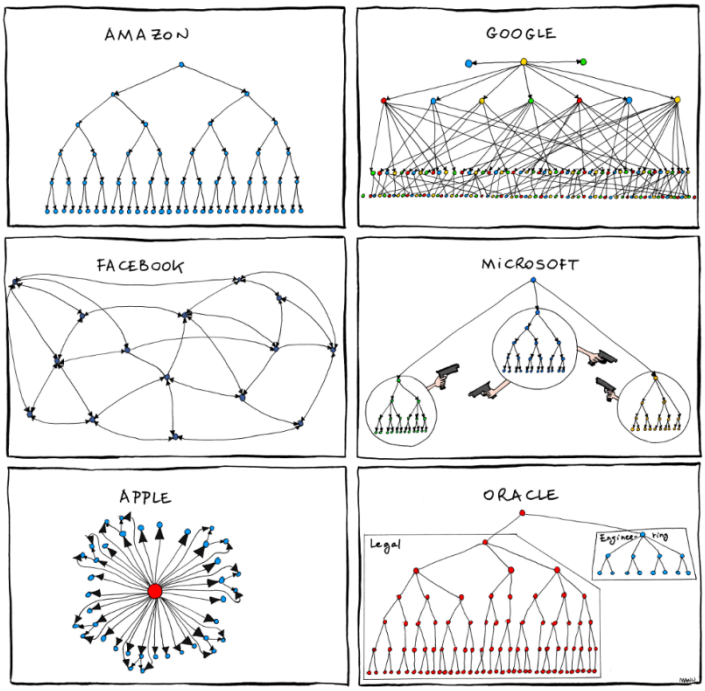


图 2　六家公司架构与文化

1.2.1　微软

微软在 PC 行业一直是霸主，特别是在新旧世纪交替时，Windows 操作系统正处于顶峰，它在 PC 操作系统市场的份额超过 90%。然而，在互联网和移动互联网快速发展的十几年里，微软未能成功转型，或者说未能成功在互联网和移动互联网开拓市场，被互联网公司谷歌等抢去了风头，还被华尔街看衰。这一点从股票上也能体现出来：在 2000 年 1 月 1 日鲍尔默接替比尔·盖茨成为微软 CEO 时，微软股价为 58.719 美元，而市值则为 6163 亿美元；然而，笔者在写本书第一版时，微软股价为 33.27 美元，市值为 2771.4 亿美元，较当时下降 55%。迫于多方压力，微软新任 CEO 纳德拉于 2014 年 2 月上任以来，积极带领微软进行战略转型。一个显著的变化就是微软放低身段，一改过往以行

业标准制定者自居的形象，不再坚持微软那些与主流格格不入的标准。例如就在 2015 年 7 月，微软宣布鼓励正在使用其自身 Silverlight 媒体格式的公司转投 HTML5 的怀抱，并且在微软全新的 Edge 浏览器上将不再支持 Silverlight 技术。同时微软也在加速剥离边缘业务，以期削减成本，集中精力来发展那些更有前景的产业。如 2015 年 6 月宣布将由 AOL（美国在线）负责美国与另外八个国家出售微软产品上的数字广告，意味着微软结束了同谷歌在该领域进行的长达十年的不太成功的竞争。另外一方面微软也正在对自己的组织架构进行调整，以达到提高竞争力的目的。同样是 2015 年 6 月宣布将当前的操作系统事业部（Operating System Group）和微软设备事业部（Microsoft Devices Group）整合为新的 Windows 及设备事业部（Windows and Devices Group）。这次重组可以看作微软对打破部门之间壁垒，形成完整产品生态环境的再一次尝试。

微软正在将自己的业务重心从传统软件扩展到个人计算、云平台等新兴领域。虽然微软的转型之路依旧充满变数，但是微软仍然是一家非常赚钱的公司。2008 年微软很好地应对了全球金融危机，强劲的营收和多样化的产品使微软能够在金融危机期间，在信贷极度紧缩的情况下获得足够多的现金。微软仍然保留着强大的技术和产品研发能力。另外强大的现金流可以保证微软能够通过收购那些前景不错的小公司来实现快速业务扩展的目的。微软的种种努力已经部分得到投资者的认可。在笔者写本书第二版时，微软的股价已经回升到 44.3 美元，市值为 3598 亿美元，相对前期低点，反弹的幅度也是相当令人瞩目的。至少到目前为止，微软依旧是一家在行业内颇具影响力的公司。

微软为员工提供着相当不错的薪水及福利，总部拥有近 2000 名来自中国的员工。笔者有好几位同学和朋友都是从微软上海分公司转到了微软西雅图总部工作，在华众多外企中也算是去美国相对比较容易的一家公司。微软职位最高的华裔是毕业于复旦大学的陆奇，他负责 Office 软件和必应在线搜索引擎，是微软重组后三大事业部门负责人之一，并直接向 CEO 萨蒂亚 · 纳德拉汇报工作。在他的帮助下，必应培养了一大批中国籍的管理层。相比谷歌和

Facebook，微软录用门槛较低，是值得一试的公司。但有一点需要注意的是：微软每年采用末位淘汰的制度，在工作表现（performance）上得低分的员工将被迫辞职。因此，除了做好自己的工作以外，还需要和上司保持良好的关系。

1.2.2 谷歌

谷歌是一家很酷并且富有创造力的公司，除了称霸搜索领域，它还拥有一个神秘实验室——Google X。该实验室可以脱离其核心业务而进行自由的产品研发。从无人驾驶汽车到谷歌气球，再到谷歌眼镜，一项项酷味十足的创造都是 Google X 实验室的杰作。谷歌技术驱动型的文化的确有助于员工尽展所能。谷歌形成了著名的“20%自由时间”规则，即允许员工把工作日 20%的时间用于与本职工作无关的项目。谷歌本身就像一个创新引擎，通过创始人奖金、同事评优及 20%的自由时间等，驱动员工不断创新。App Engine 论坛、Google X 实验室及其他的开源项目等扮演着创新的练兵场。正是从这架创新机器里诞生了诸如 Gmail、Chrome 等具有战略意义的重要产品，而它们也都来自底层员工的创新。自 2014 年年底以来，谷歌财报多次低于华尔街的预期，谷歌管理层也承受着投资人的诸多指责，其中一个争论的焦点就是公司是否应该将如此之大的财力投入到那些虚幻的项目中。不过谷歌 X 实验室负责人 Astro Teller 指出仅仅 Google Brain 创造出的价值几乎就可以冲抵目前 Google X 已产生的所有花费了。

据最新报道，“20%自由时间”规则已名存实亡。一个从谷歌离职的高官抱怨现在的谷歌是一家专注于广告的公司，而不是原来的不断驱动员工创新的技术公司。他认为昔日的谷歌靠内容赚钱，就像电视台一样，通过制作最好的电视剧，吸引最大的广告收入，但是今天的谷歌看起来似乎把更多的精力放在了广告本身上。这也许算是谷歌对投资人的一种妥协吧。从目前来看，谷歌仍然保持良好的上升势头，公司在诸多技术领域依旧保持着领先。Google X 实验室不断地扩展着公司未来的想象空间。2015 年 8 月谷歌为了更好的发展而成

立一家母公司 Alphabet，把优势业务如搜索和广告、操作系统 Android、视频 YouTube、地图和必要的支撑部门集中在新的“Google”公司，以支撑新公司的盈利和基础估值，同时把不同领域的项目和业务分散到其他子公司内，比如 Google X、Google Capital、智能家居 Nest 等，各自发展。

谷歌连续多年被评为年度最佳雇主，其奢华的工作环境举世公认：舒适宽敞的员工工位、人性化的休息室、完备的游乐场、常年运转的健身房、瑜伽室、按摩室、沐浴室，等等。大楼每一层一般都会设置一个巨大的休息室，里面提供按摩椅、桌上足球、台球、游戏机。另外，用心排列的零食陈列架囊括了所有主流零食，上面摆满了可乐、雪碧、苹果汁、橙汁等时下最火的饮料，冰箱里还有各式各样的冰激凌、酸奶等。员工按摩可以随时预约，而餐饮常常是五星级酒店大厨的手艺。最关键的是所有这一切都是免费的。最近，谷歌又提供了一项新福利：如果员工不幸去世，其配偶还能在未来的 10 年里享受到去世员工的半数薪酬；他们的未成年子女每月还能收到 1000 美元的生活费直至成年。除了半数薪酬，配偶还能获得去世员工的股权授予。另外一方面在谷歌的工作经历也能增加个人在职场的竞争力，很多硅谷新兴公司的骨干成员都有着谷歌的工作背景。

谷歌每年都会在全球招聘数量众多的工程技术人员。但是被谷歌录用除了需要拥有过硬的技术实力以外，还需要运气，因为这里的每个岗位都竞争太激烈了。

1.2.3 亚马逊

亚马逊看重的是创意人才，它并不只是一家电子商务公司。亚马逊员工背景多样，有些是自由派艺术家、大学学者、摇滚音乐家，还有些是职业滑冰选手、赛车选手等。亚马逊的“未来利润分享制”，把所有员工包括仓库员工、公司职员及行政经理、最高主管，全部纳入到公司的认股专项计划中。

亚马逊上市十几年以来，一直处于不盈利或微盈利状态，其根本原因是零售的利润太低。这也导致了亚马逊工程师的福利在顶级 IT 公司中较为一般。我们先谈一谈亚马逊对于工程师来说的优势。

- 与时俱进的技术。在软件开发环境、开发框架、网络监控、软件部署等技术上，强于传统的软件公司。因为全公司的软件开发都基于网络，并采用大量的开源技术，所以对于新入职的员工来说，容易上手；对于老员工来说，容易跳槽。
- 良好的业内名声。业内都认为亚马逊的工程师技术过硬，当他们去面试谷歌、Facebook 和 Twitter 等福利更好的公司时，均有面试的机会。
- 开放的公司文化。对于工程师来说，在亚马逊没有其他大公司的办公室政治，工程师拥有自主权，有活做，但活不重；内部转组相对容易，而且没有微软的末位淘汰机制，公司基本上不裁员。

当然也有不好的地方。

- “臭名昭著”的 on-call 制度。on-call 意味着你需要一天 24 小时随时待命，准备解决可能出现的问题。因为亚马逊的零售及云服务都是 7×24 运转，而且在中国、日本、欧洲的零售分部也由美国支持，所以几乎所有的开发组都有 on-call 制度。让研发工程师处于随时待命的状态，部分原因是亚马逊没有一支完善的运维团队。仓库一旦出现问题，那边工人会直接通过 on-call 跟研发工程师联系，这样效率非常高，而且节省了运维的人力成本，但是给研发工程师们带来了巨大压力，工程师们任何时候都有被呼叫的可能。假设你在半夜 2 点被叫醒，一直处理到 6 点才修复问题，并且第二天你还得正常上班，无论怎么想都是一件可怕的事。
- 不合理的绿卡制度。对于新入职两年内的低级别的工程师，公司不给办绿卡（接下来一章会解释绿卡的重要性）。其他大公司，比如微软、谷歌和 Facebook 等，入职第一天公司即可协助员工办理绿卡。

- 工程师晋升缓慢。特别是在传统电子商务部门，从 SDE I 到 SDE II 要等待 4 年。升职 SDE III 更是机会渺茫。部分原因是管理混乱，管理人员变动过快，导致底层工程师的职称评定被拖后。
- 一般水平的员工福利。除了基本工资，对比其他公司亚马逊有些福利没有，有些福利则较差，比如医疗保险比同城的微软就相差很多。

尽管亚马逊拥有让人诟病的 on-call 制度，但它还是很适合刚毕业的学生，或者是单身工程师。况且，如果你来自国内的互联网公司，比如百度、淘宝或腾讯，对于这种 on-call 制度你将表示毫无压力。另外一方面，亚马逊面试难度相较于谷歌、Facebook 还是略低一些。

1.2.4 Facebook

Facebook 是一家世界顶级 SNS（社交网络网站）公司，于 2004 年上线。Facebook 原意是“花名册”，美国大学通常把这种印有学校社区所有成员的花名册发放给新来的学生或者员工，用来帮助大家认识学校的其他成员。Facebook 以草根自媒体为主要的传播形式，顺应了 Web 2.0 时代以个人为中心的潮流。短短几年时间内，就赶超了雅虎，成为全球前三大网站之一。

与谷歌相比，Facebook 更像国内的互联网公司，对技术本身并没有多高的追求，而是更重视产品和用户体验。当有事情需要做的时候，Facebook 不会在请教所谓专家正确的解决办法上浪费太多时间，而只是坐下来写代码并确保这个代码能够正常执行。Facebook 的员工普遍都比较年轻但也相应的缺乏经验。不过由于 Facebook 本身面试技术门槛和喜欢炒员工鱿鱼的文化，Facebook 的工程师的平均水平保持在相当高的水准上。

与大部分硅谷新贵公司类似，Facebook 在给员工有挑战的工作要求的同时，也付给了丰厚的待遇。根据 Glassdoor 的调查统计，Facebook 的软件工程师的 2014 年平均基本年薪（base pay）为 12 万美元，略低于谷歌的 12.8 万，

但是高于微软的 11.1 万和亚马逊的 10.5 万。除此之外，还有丰厚的签字费和股权激励，以及免费的三餐、健身费用报销等。2013 年 Facebook 甚至投资 1.2 亿美元在其总部附近打造员工住宅园区，用于帮助员工解决日益高涨的硅谷地区房价问题。

1.2.5 Twitter

Twitter 是一家媒体公司，同时还是一家科技公司。Twitter 的 CEO Dick 表示 Twitter 是一家媒体领域的技术公司，但它卖的不是技术，它的核心业务是广告。在最近的一项调查中，Twitter 给出的薪水在行业内排名最高，达到人均 11.5 万美元，同时它在赠与股票方面也很大方，高级工程师通常每年拿到的股票价格与薪水持平。

Twitter 的公司文化和谷歌类似，有各种标准的小福利，如免费午餐、晚餐，每人一台 MacBook Pro，工作时间弹性且换团队和项目比较容易，还有令人满意的工作环境。公司用心对待老员工，不用虚空的荣誉头衔和奖励，而是选择把老员工们提升到有实权的岗位。Twitter 正在实现成为一个由工程技术人才驱动的企业。有媒体报道称，在 2011 年的人才竞争中，Twitter 每离开 1 人后，公司从其他公司雇佣近 11 人，新进人数是离开人数的 11 倍，这表明 Twitter 具有独特的吸引力。

1.2.6 Epic

Epic 在 IT 行业并不出名，但它一直是医疗软件排行第一的软件公司。Epic 的名声不大，其中一个原因是它还没上市，另一个原因是它的总部在美国中部威斯康辛州的麦迪逊郊区。本书挑选 Epic 作为美国普通软件公司的典型，原因是它的地理位置较偏，薪水较其他行业高，但福利一般，老板抠门。

Epic 在 Verona，麦迪逊的郊区，位置较为偏僻。附近餐馆少，员工很难找

到外出吃饭的地方。据离职员工称，程序员的年终奖也少，股票基本上没有。公司代缴的医疗保险部分常常是从员工工资里扣除。

Epic 不是传统软件开发公司，而是一家提供软件服务的公司，这使得 Epic 不是很重视技术，而是把客户关系放在第一位。从而影响了 Epic 在技术上的创新。另外，由于 Epic 不是一家上市公司，老板有绝对的权威，他的喜好决定了 Epic 发展的走向。很多人把 Epic 作为职业发展中的一个跳板，锻炼几年后就跳槽到西部的纯软件开发公司了。当然，如果你喜欢中西部悠闲自在的生活，也不愿承受硅谷高房价所带来的压力，Epic 这类公司绝对是不错的选择。

1.3 技术移民

对于国内程序员来说，通常有三种途径去美国工作。

1．进入跨国公司工作一年以上，然后内部转组到美国的部门。例如，进入微软中国工作一年后，申请转组到美国总部工作，若能批准，则申请 L-1 签证去美国工作。

2．直接应聘美国公司。这其实没有想象的那么难。当前不少美国 IT 公司直接从国内招人，比如谷歌、Facebook 等。这些企业看中的是国内程序员扎实的计算机基础及丰富的项目经验。拿到美国公司的聘书（offer）之后，申请的是 H1B 签证去美国工作。不过，按照目前 H1B 申请与批准情况来看，H1B 名额很快就用完了；每年 4 月 1 日开始提交申请，而要到 10 月 1 日才能合法工作，这要看公司能否等你半年。当然，很多大企业愿意等。如果你拿不到名额，有些公司可以安排你先去别的国家工作，等到下一年申请到名额了或者通过 L1 签证，再派你去美国。通常大公司在全球各地都有分公司，可以提供这种便利。不过在美国以外期间的福利待遇会同当地水平保持一致，相对于美国岗位的待遇往往有所缩水。下面列出几家常见互联网公司的例子。

- 亚马逊：加拿大温哥华，英国伦敦，中国北京
- Facebook：加拿大温哥华，英国伦敦
- 谷歌：澳大利亚悉尼，爱尔兰都柏林，中国上海
- 微软：加拿大温哥华，爱尔兰都柏林，中国上海和北京

3．去美国留学，然后找工作。先持 F1 学生签证去美国读书，毕业后，找到了工作，转为 H1B 签证。笔者就是走这条路径：在中国科技大学读计算机硕士期间，考 GRE 和托福并申请美国的学校，毕业后拿到全额奖学金去北卡罗来纳州立大学攻读计算机科学博士；四年后，毕业去微软工作。这也是很多在美工作的中国程序员走的路。其实，如果你对研究不感兴趣的话，完全可以申请硕士，而不是博士，也不用全额奖学金。留学已经不难了，只要你有一定的经济基础。但这种途径的坏处是花费时间长，代价也很大。

在美国工作的中国籍程序员经历的过程大致相同：找到一份工作，申请工作签证；在工作过程中，申请绿卡；等绿卡批准之后，再过 5 年才能申请美国国籍。在这个章节，我们将介绍美国的工作签证、绿卡和生活情况。

1.3.1 签证和绿卡

如果你不是美国人，也不是美国绿卡持有者（即永久居民），你进入美国工作之前得申请签证。如果申请人希望以非移民身份在美国短期工作，根据美国移民法的规定，应根据工作类型申请相应签证。大部分短期工作签证都要求准备雇佣申请人的美国雇主提交申请批件，并获得美国公民及移民事务处（USCIS）的批准，然后才能申请工作签证。

工作签证主要有三种：H1B、L-1 和 O1。O1 签证颁发给各行业的杰出人才，笔者认识的人当中只有一位通过 O1 签证赴美成功，其发表过多篇重量级的论文。所以一般工程师拿不到。在这里，我们主要介绍 H1B 和 L-1。

H1B（专业人士）：属于非移民签证的一种。如果申请人希望去美国从事

预先安排的专业技术工作，则应申请 H1B 签证。此类签证要求申请人在准备就职的专业领域拥有学士以上学位（或同等学力）。USCIS 会审查决定申请人被雇用后将从事的工作是否构成专业技能工作，申请人是否符合该专业技能的要求。申请人的雇主应向美国劳工部提交劳工情况申请表，包括雇佣双方之间签订的合同条款。持有 H1B 签证的人可以在美国工作 3 年，之后可以再延长 3 年。如果 6 年之后，签证持有人的身份没有发生变化（如获得绿卡），那么签证期满之后，持有人必须回国一年，才能再次申请同类签证。H1B 申请人的家属可以申请 H-4 签证。如果你是持有 H1B 签证的主申请人，则你的配偶或未婚子女（21 岁以下）将获发 H-4 签证以便陪同赴美。根据 2015 年 2 月美国移民局公布的新政策，法律允许符合一定条件的 H-4 签证持有者（H1-B 家属）在美国工作。H1B 每年的配额是 65000 人，其中 5000 人是专门留给新加坡和智利公民。全球其他国家共享剩余的 60000 个名额。另外对于在美国获得美国学校硕士及以上学历的人，每年还有额外 20000 个配额。随着 2010 年之后美国经济的逐步复苏，H1B 申请人数每年都在增加。如 2015 年 H1B 申请人数超过了 17 万，而 2016 年 H1B 的申请人数更是破纪录地达到了 23.3 万。由于申请人数远远超过每年的配额，美国移民局一方面会在接收到一定数目的申请表之后停止接收新的申请（最近几年，基本上一两周的时间内移民局就收到了足够多的申请表），另外移民局也会对已收到的申请表进行抽签。抽签是对普通 60000 个名额和额外 20000 个名额分别进行。其中拥有美国硕士及以上学历的申请人，如果没有在额外的 20000 个名额抽签中被抽中，可以再参与普通的 60000 名额的抽签。根据最近几年的经验，额外 20000 个名额的抽签中签率为 50%左右，而普通 60000 个名额抽签的中签率不到 30%。H1B 签证申请过程再次验证了一句话：“运气也是实力的一部分”。

L-1（公司内部调职者）：如果你是一家跨国公司的雇员，被临时派往美国的上级机构、子公司或附属单位工作，则需申请 L-1 签证。此处所述跨国公司可以是一家美国或外国企业。要获得 L-1 签证，申请人必须属于管理层、高管层或是拥有专业知识的人员，并且被公司派往美国担任此类职务，但赴美前后

的职位头衔不必完全相同。此外，在申请赴美工作的前三年中，申请人须在该国际公司的美国境外机构连续工作满一年。只有在申请人准备就职的美国公司或附属机构获得 USCIS 的申请批件后，才能申请 L-1 签证。如果你是持有有效的 L 类签证的主申请人，则你的配偶或未婚子女（21 岁以下）将获发 L-2 签证。根据最近出台的法律规定，你的配偶也有机会申请就职许可。

如前所述由于工作签证有种种限制，比如 H1B 签证年限为 6 年，6 年后你必须离境一年以上才能再进入美国，L-1 只能为一家公司工作，换了公司就得重新申请，等等，一般中国程序员都会在工作过程中申请绿卡。绿卡持有者除了出入境方便以外，还容易换工作，可选雇主范围也会更广，因为有海量的小公司和与政府项目相关的外包公司不愿意或者不方便为外国人申请工作签证。

美国移民法将职业绿卡申请分为五大类，分别为第一优先（EB1，杰出研究人员和学者移民），第二优先（EB2，高等技术移民），第三优先（EB3，普通技术移民），第四优先（EB4，特殊类移民），以及第五优先（EB5，投资移民）。劳工移民的申请类别不同，拿到绿卡的时间也会不同。劳工移民第三优先（EB3）是最慢的一种。一般来说，程序员会申请 EB2 或 EB3，特别突出的人才可以尝试一下 EB1。目前，EB2 从申请到批准的时间大概是 4～6 年，EB3 需要 5～8 年，而 EB1 等待时间最短，基本上一年之内获批。由于这几年 IT 行业好转，大量中国和印度的程序员涌进美国，绿卡排期也越来越长。有人说，申请美国绿卡是一条看不见尽头的漫漫长路，犹如坐监，俗称“移民监”。

美国总统奥巴马对媒体称 2014 年他的工作重心是移民改革，他将积极推动新移民法案 CIR 顺利通过国会两院，并最终使之成为法律。新移民改革方案主要倾向于给滞留美国的非法移民绿卡，同时开放高科技移民，以保持美国的全球竞争力。对于高科技职业移民来说，绿卡申请的等待时间大大缩短，同时每年工作签证 H1B 配额将翻倍，甚至达三倍以上。最直接的影响是大量高科技人员将移民美国，包含中国的顶级程序员。

除了职业移民，还有亲属移民。美国移民法允许与美国公民或美国永久居民有亲属关系的外国公民移民到美国，这就是美国亲属移民。美国亲属移民分为两种：无配额限制亲属移民和配额限制亲属移民。亲属移民视各类别的不同，从申请到签证获批，等候的时间从六个月到十二年不等。无配额限制亲属移民等待时间较短，只有美国公民的直系亲属才能申请。他们包括美国公民的配偶、美国公民的 21 岁以下未婚子女、美国公民的配偶同其前夫或前妻所生的 18 岁以下未婚子女，以及美国公民合法领养的 16 岁以下未婚子女和 21 岁以上美国公民的父母。

1.3.2 税率和生活

在美国有句谚语：“世上只有两件事你逃不过：一件是死亡，另一件就是缴税。”美国税种繁多，税率复杂，也有类似国内“起征点”的宽免额。但和中国不同的是，这个“起征点”是不固定的，一般来说，大概是年收入在 7500 美元以下的免征。不过在美国有正常收入的人，其年收入都会超过这个数字，所以才会说“逃不过缴税”——几乎每个有收入的人都必须缴税。为保证中低收入家庭和个人的生活水平不会因纳税过多而下降，美国税法还规定了各种扣除、免税收入和退税制度。

这里简单列举一下几个主要的税种。

- Federal Income Tax（联邦税）：由联邦政府征收，同时也是美国联邦政府收入的最大来源。联邦税有两个特点。第一，计算方式是阶梯式的，收入越高边际税率越高，从 15%到 38%不等；第二，家庭结构（比如有无配偶、有无子女）极大影响最终税率，单身的赋税最重。
- State Income Tax（州税）：由所在地的州政府征收。有些州没有州税，比如西雅图（微软、亚马逊总部所在地）所在的华盛顿州。大部分州均有州税，税率从 7%到 11%不等。
- Sale Tax（消费税）：在你购买商品时，商家代收的税，不会从你的薪水

中扣除。商店展示的商品价格都是税前的价格。绝大部分州都有消费税，税率和州税类似。个别州没有消费税，比如俄勒冈州。因为华盛顿州有9.5%的消费税，所以生活在西雅图的微软员工在购买大件的时候，常常驱车三个小时去没有州税的波特兰购买。

- Social Security Tax（社保基金）：美国政府用于养活贫困线以下的家庭、残疾人和无退休金的老人等。社保基金按照固定税率征收，税率为12.4%，个人与雇主各付一半，即 6.2%，征税基数是薪资扣除商业医保剩下的部分。它的征收额有上限，每年都在调整。2015 年最高征收税基是年收入 11.85 万美元。
- Medicare Tax（社会医疗保险）：主要用于医院支付无医保人员的费用。这个税也按固定税率征收，税率为 1.5%，但没有上限。

此外，还有商业保险税和房产税。商业保险税一般由雇主代付，不会从你的薪水中扣除。房产税税率每个州都不一样，大概在 1%左右。报税的方式和国内不同，在美国是以家庭为单位报税，而在国内是以个人为单位报税，而且国内没有退税制度。在美国，你会看到很多已婚妇女做家庭主妇，一方面是因为以家庭为报税单位，如果夫妻双方均工作，则赋税更重，她们税后的薪水常常只够请保姆的钱，还不如自己带小孩；另一方面，她们认为自己带小孩对小孩的教育更有利，而且她们的生活更自由。

总体来说，对同等收入人群，美国税率比中国税率高。假设同是年薪 60 万人民币，在国内税率大概是 28%，而在美国税率为 38%。虽然美国个税比中国高，但是同等水平的程序员在美国的生活质量却远超中国。这涉及收入与物价对比。大件商品，比如汽车、化妆品、名牌衣服等，在美国的售价即便换成人民币也比国内便宜。硅谷房价一直很高，在美国也能排上前几名，我们以北京和硅谷的房价为例，北京的百度程序员，年薪 20 万人民币，不吃不喝买一套普通住房需要 15 年以上。而在硅谷，刚毕业的硕士生能拿到 10 万美元年薪，不吃不喝买一个普通独栋别墅需要 10 年左右。是的，你没看错——是独栋别墅！

第 2 章 求职准备

通常来说，寻求美国程序员的职位必须做好前期准备，比如，你打算做什么工作，开发、产品设计还是测试？打算应聘哪些类型的公司，大公司还是创业型公司？需要花多长时间做好技术面试的准备？从哪里获取更多的信息？如何准备简历？每家公司的面试风格是什么样子？等等。

首先，我们看看 2013 年拿到 Facebook 聘书的国内程序员贴在网络上的求职历程。

1 月 12 日：通过朋友推荐，收到 Facebook 猎头的邮件

1 月 26 日：预约了第一次电话面试的时间

1 月 29 日：一个美国人问了一个简单的问题和一个 DFS 算法题，很快完成

1 月 30 日：收到邮件说通过了，约了第二次电话面试的时间

1 月 31 日：第二个电话面试的面试官是一个中国人，也问了两个编程题目

2 月 5 日：收到反馈说让我去硅谷现场面试，可是在春节假期无法办签证

2 月 25 日：Facebook 发了邀请函，我赶紧去预约美国大使馆面签

2 月 27 日：顺利通过美国签证面试，面试官居然没看我的邀请函

3 月 3 日：说去美国现场面试后可能赶不上今年 H1B 签证的申请，所以改成 Skype 面试

3 月 5 日：半夜，我进行了 4 轮编程面试

3 月 6 日：一大早 Facebook 就发信来说收到两个面试官反馈，说对我评价很好

3 月 8 日：我拿到 offer

3 月 11 日：律师开始跟我联系，让我准备 H1B 材料

3 月 15 日：我把所有的材料发给律师

3 月 23 日：律师效率很高，LCA 被批准

3 月 28 日：H1B petition 提交，据说要抽签

4 月 11 日：律师给我发邮件，说 H1B 被批准

5 月 20 日：收到 Fedex 快递，律师给我寄的申请材料和 I797B

5 月 23 日：收到邮件说要对我进行背景调查

6 月 14 日：背景调查通过

7 月 8 日：故意挑女儿一岁生日时，一个 H1B 带两个 H4，去上海领事馆面签

9 月 4 日：收到邮件让我去中信银行取护照，终于一切都定了。就等着公司提供搬家服务，32 岁的我即将带着老婆孩子去美国开始新生活

从上面的例子，我们可以看出：（1）确定求职目标，即 Facebook 的大公司及软件开发职位；（2）通过已有的人际关系，找到内部推荐；（3）准备好技术面试；（4）有了聘书之后，赶在 4 月 1 日之前申请 H1B 名额；（5）有了 H1B 名额之后，即可去美国大使馆或领事馆面试签证。在这个章节，我们先介绍常见的硅谷职位、如何建立求职人际关系及获取求职信息的渠道。在接下来的几个章节，我们将陆续介绍如何撰写简历、如何准备面试及如何与招聘你的公司谈判，等等。

2.1 职位选择

谈起软件行业的职位，大家自然就会想到软件开发工程师，其实硅谷的职位远不止这个。只要我们在招聘网站搜索公司名字，就可以看到大量的不同性质的职位。与开发相关的有研发工程师、测试工程师、运维工程师、产品经理、前端工程师、数据科学家、架构师、基础框架工程师、用户体验工程师，等等。本书会重点介绍大量招聘外国人的职位，如研发工程师、测试工程师、产品经理和数据科学家。

研发工程师（SDE）

这是最普通的职位，也是需求量最大的职位。这需要你精通一门面向对象语言，比如 Java 或 C++，最好还能掌握一门脚本语言，比如 Python、php 或 Shell，加上若干个项目经验，此时你就能胜任初级的软件开发工程师的工作。如果你想应聘 SDE II 或者以上的岗位，则往往要求你在复杂的问题排查、系统设计和性能调优等方面表现出与工作年限相符的能力。对于资深研发工程师岗位，技术细节的理解和大局观的把握往往是考察的重点。一般来说，研发工程师更具有自由度和自主性，比如能够设计新产品特性、提交并修复 BUG 等，但晋升竞争激烈，因为每个团队都有大量的研发工程师，想脱颖而出并非易事。

测试工程师（QA）

在有些公司，比如微软，测试工程师又被称为 SDET，即测试开发工程师。这是很多人会忽视的职位，也有很多人会误解，认为这种职位没什么技术含量。其实在一些公司里，测试工程师编写代码的量不会小于研发工程师。测试工程师与研发工程师相比，缺少自由度，通常他们的工作与项目捆绑在一起，开发的产品多是内部使用的测试工具，以及各种报表统计平台。但是，由于测试工程师进入门槛较低，并且晋升快，所以适合做事细腻的女性工程师。

产品经理（PM）

如果你的英文较好，而且在产品方面有敏锐的洞察力，那么产品经理也会是一种很好的选择。产品经理是市场和开发之间的桥梁，通常需要把客户和用户需求转化为产品文档，并兼顾产品的营运。产品经理在上班时不是在开会，就是在去开会的路上，因为他们需要和开发、测试、用户体验等团队协调，甚至和客户见面。产品经理并非都是科班出身，他们来自各行各业。在不少 IT 公司里，公司 CEO 就是最大的产品经理。

数据科学家（Data Scientist）

伴随着大数据的出现，对这个职位的需求量也越来越大，目前成为硅谷最热门的职位之一，薪水相比研发工程师只多不少。数据科学家就是采用科学方法，运用数据挖掘工具寻找新的数据洞察的工程师，通常需要有大数据处理、机器学习和数据分析的经验。它的准入门槛较高，但如果你拥有计算机博士或者统计学博士学位，则可以试一试。

2.2 公司选择

你打算去大公司、中小公司，还是创业公司？这也是每次跳槽时大家都会去考虑的问题。在硅谷虽然有很多名头响亮的大公司，但更多的还是锐意进取的小公司，并且每天都在诞生新的创意公司、新的小公司。当技术、资金、理念和创新有大量机会摩擦碰撞时，当有这么一种机制帮助你不断实践梦想而不必承担失败的后果时，并且大量人才聚集在硅谷时，创业就变得自然而然了。

在大公司工作的好处大家都知道，待遇优厚、稳定性高、绿卡申请便捷，等等。而在创业公司（startup）工作的好处则是，可以更全面地接触系统内各个方面的技术，更直接地观察公司的运作，与同事的关系也更简单融洽。不管

是大公司还是小公司，你必须在面试之前和对方的人力资源部门确认 H1B 政策，即公司是否支持外国人在美国合法工作（可以先把绿卡政策放在一边，等到你有多个聘书之后，再问也不迟）。

小公司更缺人才，他们更会不惜代价地引进一个人才。另外，小公司的面试难度也相对较低，毕竟申请的人也较少，但是有些小公司的面试流程可能会比较长，7 或 8 轮技术面试也是常有的事情。最后，如果小公司成功被收购甚至上市，那么你的股票收入很可能会远大于工资所得，而且你更容易进入公司高层。因此，不妨也试试小公司，至少可以作为一个跳板。一旦进入了这个圈子，工作机会也就慢慢铺开了，用不了多久猎头们就会盯上你，成天鼓动你跳槽。

那么到底该选择大公司还是小公司呢？对于刚毕业踏入职场的程序员而言，创业公司不确定性太大了，同时由于创业公司自身尚处在发展的初始阶段，首要问题是解决生存问题，因此在项目管理、内部流程、员工培养等方面往往比较简单、甚至是欠缺的。对于刚参加工作的人来说，这些因素对其长远的成长有一定的负面影响。而大中型公司往往在这些方面做得很不错。另外一方面，小公司的工作压力和强度一般也都大于大中型企业，一个人往往要身兼数职。结果员工多陷入日复一日的日常工作问题的处理当中，缺少对知识系统地学习和吸收，始终停留在经验的层面。程序员工作能力的提升不能只靠经验的积累，有些时候也需知识来做催化剂使之突破瓶颈得以升华。例如处理 core dump 的经验积累到一定程度时，你就需要操作系统内存管理的相关知识来帮助你将能力提升到更高境界。但是这些知识的获取都是需要时间的，而这又是小公司工作岗位最欠缺的。总的来看，大中型公司因为其系统的培训体系、严谨的产品流程和相对轻松的工作压力使其成为初入职场第一份工作不错的选择。

不过对于有多年工作经验的程序员而言，小公司是一个不错的选择。大公司产品分工细化，很容易陷入重复劳动的状态。十年的工作经验可能蜕变成一年工作经验被重复了十遍。另外在大公司项目众多，相当多的项目处于平稳发

展状态，个人缺乏晋升机会。

所有的创业公司都值得去吗？一位斯坦福大学商学院的教授给刚毕业的学生的建议是，第一份工作应该选在一家势头不错的中型公司，因为这样的公司更有可能取得成功。他认为，一家成功的企业会让你获得更有价值的东西，即行业洞察力。

同时，他不推荐初始的创业公司。他认为，大部分的创业公司都会以失败告终，这意味着加入它们的风险与获得的回报将不成正比。这个理念对于投资来说也很重要，以最小的风险获得最大的回报。他通过与 10 到 15 家的顶级风投交流后，得出了一份将会快速发展并值得加盟的企业名单。他推荐这些科技企业时只考虑公司的成长性，他认为这比你能获得的收入、职位头衔都要重要。这些企业的年收入在 2000 万到 3 亿美元之间，它们增长迅速并且在可预期的未来会保持这种增长势头。

以下是一部分被推荐的中型、快速增长公司的列表。

- Acronis：对于保存在物理媒介、虚拟媒介及云端的数据进行保护和灾难恢复管理的公司。http://www.acronis.com
- Arista Networks：大型数据中心与计算业云端网络解决方案提供商。http://www.artistanetworks.com
- Boku：全球移动支付平台。http://www.boku.com
- Box：云存储服务提供商。https://www.box.com
- Cloudera：基于 Apache-Hadoop 的软件开发商。提供基于 Hadoop 及其他 Apache 平台的服务、培训及资格证明授权。http://www.cloudera.com
- Dropbox：云存储服务提供商。https://www.dropbox.com
- Evernote：笔记记录与整理应用开发商。https://evernote.com
- Palantir Technologies：专注于数据问题的软件提供商，又是一家大数据公司。https://www.palantir.com

- Quantcast Corp：网络评测分析公司。https://www.quantcast.com
- Spotify：音乐搜索，分享与即时播放应用开放商。https://www.spotify.com
- Square：信用卡移动支付服务。https://squareup.com
- Yousendit：企业协作服务提供商。http://yousendit.com

本书再版时，上述公司列表中，Arista Networks 和 Square 已经上市，Box 和 Dropbox 正准备申请上市，而 Evernote 传出经营困难的新闻。这类新兴公司真可说是“其兴也勃焉，其亡也忽焉”。那么，作为应聘者的我们应该怎么评估一家新兴公司的未来呢?

在笔者看来，拥有以下特质的以技术为导向的创业公司具备高成长的潜质：(1) 业务属于当前热门题材；(2) 技术门槛高，模式不易被复制；(3) 拥有一支优秀团队，成员背景往往令人印象深刻；(4) 公司之前一直保持着快速成长。

2.3 人际关系

内部推荐和个人引荐是找工作的最好的两种方式。不仅公司喜欢招推荐来的人，自己也会觉得这样可以找到适合自己技能和兴趣的工作，而且推荐人一般也能得到物质奖励。找推荐人需要在平时就注意建立人际关系网。

人际关系网不是到需要的时候才开始建立的。当要为新工作动用关系的时候，才和对方建立关系，则为时已晚。关系的建立是长久之事，敞开心扉并乐于助人可以让你快速地建立起庞大的关系网，而这个网也会随着你展现出的对于别人生活的价值而进一步加深。关系网是在你用不着它的时候建立的，和你建立关系网的人可能是你的校友、同事、朋友，也可能是专业的猎头。

平时在和同事、朋友相处的过程中，要保持真诚，并能够积极帮助他人。

那些一直施惠于人而不考虑回报的人，到最后才会有一大群对其心生感激的人。在关系网的建立上，特别要注意不能过于关注对方的一点，以免显得过于功利。

随着社交网络的广泛使用，在找工作的过程中，也需要充分利用 Twitter、Facebook 和 LinkedIn 这些“社交工具”。但是，我们需要认识到的是它们的功用大不相同。Facebook 和 LinkedIn 可以帮助维护现有关系。一般来说，你不会和陌生人在 Facebook 上聊天。即使聊，你也不会想着以此来影响自己的职业发展。Twitter 则不仅限于朋友圈的交流，同时也能大大促进交友圈的扩大。

下面就介绍一下如何有效利用这些社交渠道。

- LinkedIn，可以用于朋友或职场间的交流。先从你的校友和好朋友开始，把你认识或者不认识的在美国工作的校友都加入到你的关系网中。一位风险资本家就鼓励与自己合作过的企业家们“把每一个你见到的人都加为好友，而且要快，不能等”。用 LinkedIn 交流有一个好处，就是你可以为别人写推荐信，同时也可以请别人为自己写。搜索一下与自己兴趣相关的小组并参与到其中的讨论，那些负责招聘的人肯定喜欢在那里出没。LinkedIn 也是猎头出没的地方，有针对性地把大公司的猎头加入到你的关系网中。
- Facebook，由于在生活社交领域做得极为出色，所以很多人都忽略了它的职场功能。Facebook 是服务于社会的，这一点对于职场社交来说显然很有价值。当你需要咨询别人或向他人求助的时候，上 Facebook。通常你所要做的就是更改一下状态标签而已。
- Twitter，可以成为你与他人互相联系、开展对话的一种极为有效的工具。大多数没有利用好 Twitter 的人是因为自己没有花时间在上面。要是你觉得自己可以投入些精力在账户维护上，那么我建议你开一个 Twitter 账户，然后把自己职业方面的想法或者有趣的文章贴上去。如果可以持续性地发帖，你就可以开始寻求关注了。关注你感兴趣的与行业相关的

人士，他们也会关注你。在你的 E-mail 签名栏、Facebook 和 LinkedIn 里都放上你的 Twitter 链接。找到你很想见面的那些人的 Twitter，针对他/她的 Tweet 发表自己的观点，以期和他们建立联系。在 Twitter、Facebook 和 LinkedIn 必须使用英文，一方面让别人更好地了解你，另一方也可以锻炼你的英文写作能力。

- 校友，引荐你的人也可能是你的校友，因此，你要尽可能从师兄师姐手里获取在美国工作的校友花名册。硅谷的华人每年会按照学校举行校友聚会，所以很容易获取你所在学校的通讯录。你可以根据花名册上面的联系方式，分别加上他们的 LinkedIn、Facebook 和 Twitter 账户。

有了这些关系网之后，接下来的一步是通告天下，告之所有朋友你的未来计划。你将会发现，不少素未谋面的朋友会突然来找你，他们会主动帮你推荐一些公司和职位。如果你有广为人知的才能和可靠的技术，只需要让认识你的人知道你要找什么工作，大家都会乐意帮忙的。如果你有使用像 Facebook 和 Twitter 这样的社交网络，仅仅在上面发条消息没准儿就能帮助你联系到梦想的公司。你也可以更直接一点，问问朋友们有谁在相关公司工作。说不定你在谷歌工作的朋友认识一些微软的员工。各种方法不妨都试一试。

相中了一家公司，却找不到门路怎么办？答案是要学会宣传自己。找到那家公司的员工并和他们交朋友。这一点可以通过 LinkedIn 或者 Twitter 完成。如果他们需要帮助，想办法帮助他们。把自己和他们的圈子联系起来。当然不要做得过火，没人喜欢被盯梢。

与此同时，新开一个技术博客，涉及一些你感兴趣的科技内容，积极参与 GitHub，并尽可能参与开源项目。下载一些你感兴趣的软件和工具，看看是不是可以做进一步的改进。一旦发现有漏洞，马上向原先的开发者报告。上述方法可以让你在招聘者找到你之前就展示出自己的技能。很多招聘者会通过线上的个人资料寻找工作人选，而这些就是让他们来找你的好办法。有了这些，你就可以把 GitHub 账号贴到你的简历中（在下一章，也会谈到 GitHub 在简历里

的分量）。

不过，在与对方公司员工交往之初先不要寄上你的简历，以免显得过于功利。招聘者很清楚收到的是不是群发邮件。所以，重在质量，而非数量。比起泛泛而谈，一封专为某家公司量身定制的邮件会让你走得更远（在第 3 章，将谈到如何量身制定简历）。

要尽可能和最相关的招聘人员取得联系。如果找不到负责你心仪职位的招聘人员，也要说明自己的兴趣所在，再请别人帮忙转交简历。不过更好的是能明确知道那位招聘人员的姓名或是具体的职位名称。

2.4 求职渠道

除了前期建立校友、同事、朋友甚至猎头的关系网，你还得了解每个公司的薪资水平、绿卡政策，等等。

在求职之初，应该定一个 20 到 40 家目标公司的列表，结合你的关系网，找到相关的朋友和猎头，然后与他们进行积极沟通。如果找不到内部人推荐，则需要在公司网站上注册，经常查看、投递公司的新职位。

以下是几个查找薪水与公司信息的网站。

- www.h1bwage.com　H1B 薪水查询网站，可以查看公司给外国人办理 H1B 签证时提供的薪水。这个数字不含奖金和股票。
- www.glassdoor.com　含有针对各个公司的常见面试问题及公司评价。
- www.careercup.com　主要是技术面试题。

此外，还有一些常用的查找职位的网站。

- Indeed.com　职位搜索聚合网站，将来自众多企业网站和招聘网站的信

息综合在一起，信息比较全面。但也没有包含全部信息，有的小公司或者小网站上的信息就没有被收录。

- Linkedin.com　根据笔者的亲身经历，LinkedIn 提供的高端职位较多，容易获取猎头提供的最新职位。如何充分利用 LinkedIn 是一门艺术。网站上有职位搜索功能，搜到的职位可以直接申请，也可以通过点击直达目标公司的网站提交简历。加入各种与职业发展相关的群组，每个群组均有人会发布新的职位信息。完善自己的简历和技能标签，以便猎头找上门来。
- Monster.com　老牌综合性求职网站，类似国内的 51job.com。在 LinkedIn 冲击下，现在是日趋衰落了，来自猎头而不是公司发布的招聘信息比例越来越高，而且职位大部分是合同工。尽管如此，你还得充分利用 Monster，因为在上面投递简历比较方便，而且只要定期更新简历，就有机会被猎头看中。笔者的第一份工作就来自 Monster，更新简历的时候被微软的猎头看中。综合性求职网站还有 careerbuilder.com 和 hotjobs.com。
- Dice.com　IT 行业招聘网站，网站功能类似 Monster.com，会有些中小型公司的招聘职位。此外，还有一些 IT 垂直类的招聘网站，例如 www.computerjobs.com 和 www.computerwork.com。
- Craigslist.com　用来查看本地小公司的招聘信息。

第 3 章 简历

猎头或人事部门通常只会在每份简历上停留 20 秒钟，并在这短短 20 秒钟内，决定你的简历是否继续转交给技术经理，由技术经理或招聘经理决定是否开启面试流程。因此，一份有吸引力的简历是你入职的敲门砖。

我们先看看硅谷公司的人事部门是怎样阅读简历的。以下是一个列表，每个选项之前是加权分值。

（+15 分）如果简历中提及和职位相符的技能超过 5 次以上。
（+8 分）如果简历中提及和职位相符的技能 3 次到 5 次。
（+4 分）如果简历中提及和职位相符的技能 1 次到 2 次。
（+4 分）求职信（Cover Letter）提到了招聘人员或招聘公司。
（+2 分）简历含有求职信。
（-10 分）没有提到和职位描述相关的技能。
（-15 分）没有大学学位。

技术经理又是如何阅读候选者的简历的呢？

（+11 分）为开源软件贡献过代码。
（+8 分）有一个分享技术知识的博客。
（+7 分）编程竞赛参与者。
（+7 分）使用 LaTeX 生成的简历。
（+7 分）在 Google 和 Microsoft 实习过。
（+6 分）掌握脚本语言（Python/Perl/Ruby）。
（+5 分）知道 3 种以上的编程语言。
（+5 分）拥有和职位相似的经验。
（+4 分）曾创业或开过公司。
（+4 分）拥有 GitHub 账户。
（+0 分）有奖学金。
（-1 分）有博士头衔。
（-2 分）没有求职信。
（-2 分）在简历中说自己懂 Word/Excel。
（-2 分）在简历中有拼写和语法错误。
（-3 分）简历的文字字号太小。
（-4 分）所有的编程经验只是在学校中。

从中我们可以看出简历内容及简历结构都很重要。在接下来的几个章节中将阐述哪些要写、哪些不应该写在简历上。

3.1 简历特点

一份强大的简历应该能在短短几秒钟之内抓住阅读者的眼球。简历里的每一行都应该有助于加固雇主想聘用你的决心。既然如此，为什么候选者还要把他那些含糊或无法证明的内容列在简历里呢？每份简历只需要珍贵的几行，简明扼要地把你的长处展现出来。在提交简历前，审视每一行并且问自己，为什

么它能帮助我获得面试机会？如果你给不出一个理由，那这行就不该出现在简历上。一份良好的简历至少包含如下特点。

- **有针对性**

在用打字机的年代，编辑一份简历是一个耗费体力的过程，求职者经常需要影印 200 份，并把相同的简历发到每家公司。优质的、有针对性的简历无疑能帮你加分。随着简历编辑成本的降低，定制高水准的简历也变得越来越易行。在这个竞争日益残酷的时代，这一点额外的工作或许不能让你的简历脱颖而出，但至少能让它登得上台面。千万要记住，申请不同公司要使用不同的简历和求职信。同样是 C++的研发工程师，互联网公司和嵌入式公司的要求和侧重点肯定是不同的。前者往往看重应聘者是否具有高并发问题处理经验，而对于后者则更希望应聘者具有对内核驱动的了解。

必须在简历适当的位置提及应聘的公司的职位所要求的技能和经验，针对性地撰写简历可以帮助你突出自己和岗位的契合点，这可以使 HR 觉得你就是他们要找的人。撰写有针对性的简历并不难，通过公司网站的职位信息，以及公司团队人员的博客，了解一下他们需要有什么样经验的人，应聘者需要掌握哪些技能，或是什么样性格的人。你也可以问问自己，这个职位上的人会遇到哪些问题？如果是你又该如何解决？即使你还没有能力全部解决面临的问题，也要向 HR 表明你有解决问题的技能。

- **量化成果**

有没有见过一个广告说“我们利润中的一部分捐赠给了慈善机构”？这种说法是很讨巧的。因为那一部分可能只是 0.0001%，但这说法又完全没错。作为面试官的我，当看到一份简历提到“提高原有系统性能”或者“降低服务延迟”时，我就有点纳闷：为什么你不告诉我究竟是多大幅度？

量化你的工作就能向雇主展示你能带来多大的影响，使你的陈述更直观，

也更有意义。如果你表示完成这些量化的改进，从而降低了公司的成本或者提高了效率，雇主就会想聘请你。对于程序员来说，用更技术化的术语量化成果则更具冲击力，比如缩短了 30 秒的延迟，修复了 10 个致命的漏洞，把算法复杂度从 O(n2)下降为 O(nlogn)。下面是一个例子，修改前为“fixed several bugs for our CRM system”，我们添加了修改 bug 之后为整个业务系统节省的成本比例，如“fixed three biggest bugs for our CRM system, leading to a 25 percent reduction in customer support calls”。

但是，也不能过多使用数字，因为你的成就可能在一个同行工程师面前令人印象深刻，但技术水平略低的人力资源 HR 可能才是审查你简历的那个人。要确保你的简历令每个人都印象深刻。

- **结果导向**

如果简历读起来太像一份工作描述，很可能就是做错了。简历应该能突出你做了什么，而不是你应该做什么。在量化成果的基础上，加上结果导向，就像强有力的组合拳。每家公司都想要一个“能把事情做完”的员工。尽量避免使用这些词：贡献、参与和协助。因为这些词表明你曾经专注在一些职责上而非完成的事情上。毕竟，很多在微软工作的人都可以说“在微软 Windows 系统上我有贡献”，但这又说明了什么呢?

- **突出差异**

同一岗位，HR 通常会收到很多份技术背景类似的简历。为了保证自己的简历不会在第一轮的筛选中就被过滤掉，我们需要在自己的简历中突出那些能够使你有别于

他人的信息。例如同样是工程方向的软件工程师，如果你有过一些与数据挖掘算法或者推荐算法相关的项目经验或者你对这些方面有一定的涉猎，都可以在自己的简历中介绍一下。但是也不要矫枉过正，不要让这些增加差异化的

信息在简历中占据太多篇幅。否则会让看你简历的人困惑甚至误解。

➢ **干净、专业和简洁**

很多 HR 会因为一个错字扔掉你的简历。他们有这么多的简历要浏览，为什么要浪费时间在沟通能力差的人身上呢？因此，一定要从如下方面仔细检查简历。

- 简练。在简历里，避免大块的文本。通常来说人们讨厌阅读大段文字，一般会跳过该段落。简历应该是一项项集合，一项大约含有一到两行的文字。
- 拼写。避免拼写错误。通过从后往前阅读简历的方式来检查拼写错误。
- 语法。可以使用 Microsoft Word 的语法检查器，但不要完全依赖于这个工具。找个母语为英语的朋友或者有强大的语法和拼写能力的人审查你的简历。
- 普通字体。使用标准的字体，如 Times New Roman 或 Arial 字体，使用的字号不小于 10 磅。Comic Sans 字体是绝对不能接受的。
- 一致性。可以使用逗号或分号分隔列表中的项目，但必须一致。用句号结束每一个项。确保格式一致，比如加粗、下画线、斜体等。你用什么样的格式往往并不重要，重要的是它们应该保持一致。
- 避免使用第一人称。避免使用“我”、“本人”或“我自己”。在整个简历中，除了客观的语句，尽量使用第三人称。

3.2 简历结构

通常投递简历时，会附上一封求职信（cover letter），简明扼要地阐述你满足这个职位的要求和胜任这份工作的理由及对这份工作的渴望。

标准简历按时间倒序阐述工作经历、教育背景等。一份标准简历至少包含工作经验和教育背景，也可以包括求职目标、技术技能、奖项和荣誉等。选择包括哪些部分取决于你的技能、背景和应聘的职位。

- **求职目标**（Objective）

求职目标应该简明扼要，比如“PHP 软件开发工程师”。其实求职目标不是必填项，只有当它增加了重要的信息才使用，比如更换工作角色。例如，如果你以前做产品管理，但现在希望将工作重点转移到搜索营销角色上时，那么求职目标可能对指定的招聘人员有价值。但是，如果申请一个 PHP 职位而且你之前的职位也做 PHP 开发，则并不需要指定。

- **要点**（Highlight）

把一些你认为招聘公司会看中的要点列在简历里。针对不同职位，写不同要点。例如，应聘数据挖掘和数据分析相关的职位，你的要点可能是以下几点。

- Work experience in Microsoft and IBM.
- Extensive programming and system design experience, and solid background in data structure and algorithms.
- Experience with data mining, data analyst, and recommendation systems.

- **工作经历**（Work Experience）

对于大多数应聘者，工作经验应该是简历中最重要的一部分。工作经验的介绍至少包含职位头衔、公司名称、地理位置和就业时间。如果你是在一家有很多产品的大公司工作，如微软或亚马逊，则可以列出你的团队。对于最近的工作经历应该有一到两行，每行大约四项。每项应着眼于你的成就（结果导向）而不是你的责任，应该尽可能用数字支持（量化成果）。对于那些有着多年工作经验的程序员而言，在其职业生涯中可能会经历很多项目，也接触运用过多种技术。应该重点介绍最近 1-2 年的相关工作，而对于之前的经历只需要突出

闪光点就可以了（简洁）。如果你是应届毕业生，则可以把实习经历或者课程项目作为工作经历。如果你参与过开源项目，不管大小，都可以把链接贴到简历中，这会让人眼前一亮。

- **教育**（Education）

工作经验通常比教育更重要。如果你是非应届生，虽然必须包括教育，但是这部分应该短，把更多的空间给工作经验。很多只有两三年工作经验的应聘者也只是简单列出他们的专业和学位，而使用大量篇幅介绍他们的工作项目。如果你是应届生，教育这栏则是你的重点，用大篇幅介绍你的教育背景，至少包含主修/辅修学位、课程作业、论文题目、GPA、参赛记录和获奖情况等。如果你的 GPA 不高，最好还是不要写在简历上了。

- **技能**（Skills）

本栏应列出会应用的软件、编程语言或其他特定的技能。为了避免冗长，使用简洁的列表方式。需要注意的是，不应该列出那些“显而易见”的技能，如会使用微软 Office。同样，熟悉 Windows 和 Mac 也不必写，除非可以列出一些不太明显的项，如 Linux。对于程序员来说，对编程语言需要注明哪些是精通，哪些是熟悉，哪些只是了解而已。根据面试官的亲身经历，同时精通 5 种以上的编程语言是不太可能的。如果你真的精通很多种编程语言，那就请列出应聘职位所需求的 3 种语言吧。另外当我们去应聘一个程序员岗位时，主要的竞争对手是那些与自己有着类似工作背景的人。而这些人的技术背景也往往是类似的。那么如何保证自己能够在这种同质化的竞争中脱颖而出呢，其关键就是要针对性地突出那些能够使你有别于其他具有相似技术背景的应聘者的技能。例如同样是 6 年工作经验的 C++后端研发工程师，简历中如果突出自己拥有较强的 CORE 文件分析能力，往往会为你在之后的竞争带来一定的加分。又例如对于 JAVA 研发工程师而言，对 JAVA 虚拟机内部机制的深刻理解也会为你的简历增色不少。其实这个已经不仅仅是书写简历的问题，更多涉及平时

自我能力的培养。其核心就是“拒绝平庸”（make yourself special）。

- **奖项和荣誉（Awards and Honors）**

如果有奖励或荣誉，可以在工作经验或教育栏里列出。如果想强调奖项，并且简历中还有空间，你也可以专门列出一个奖励专栏。需要明确的是这些奖励能帮助你同其他候选人区分开。无论使用哪种方式，你应该列出日期、获得该奖项的原因以及难度。如果可以量化获奖难度，那是最好不过了。

3.3 简历优化

一旦完成简历初稿，就需要进行一遍又一遍的优化。哪些用词可以改进？简历会不会太长？我的简历有没有特色？

- **简历用词**

如表 1 中的单词可以让你的简历看起来更专业，也更准确。

表 1

项目相关	技术相关	管理相关
Approved	Architected	Assigned
Catalogued	Assembled	Attained
Classified	Built	Chaired
Compiled	Coded	Contracted
Dispatched	Designed	Consolidated
Implemented	Developed	Coordinated
Monitored	Devised	Delegated
Prepared	Engineered	Developed

续表

项目相关	技术相关	管理相关
Processed	Fabricated	Directed
Purchased	Initiated	Enhanced
Recorded	Maintained	Evaluated
Reorganized	Operated	Executed
Retrieved	Overhauled	Forced
Screened	Programmed	Improved
Specified	Redesigned	Increased
Tabulated	Reduced	Led
Validated	Remodeled	Organized
	Repaired	Planned
	Solved	Prioritized
	Trained	Produced
	Upgraded	Scheduled
	Utilized	Strengthened
		Supervised

➢ 不应包含的内容

对于在美国或加拿大的职位，简历不应该包括种族、宗教、性取向、婚姻状况，或其他任何与歧视有关的信息。照片也不应该包含在简历里，因为照片包含歧视有关的信息。招聘者讨厌这些额外信息，因为这些信息可能导致公司被人以种族歧视等方面提起诉讼。

➢ 简历不宜过长

简历不宜过长，正如前面章节所说招聘人员只有 20 秒钟时间读每份简历。招聘人员无法精读每一行简历去掘地三尺寻找最相关的人选。审查简历的过程

更像是跳读而不是精读。那么，简历应该多长？在美国，两页的简历应该是合理的。更短的通常是更好的，比如一页。一页纸的简历迫使你有选择性地填写，只包括最好的东西。简历内容过长，是因为价值一般的内容混合了进来。招聘者就会把你作为一般的候选人，而不是最好的候选人。

- **个性化简历**

大多数的应聘者最大胆的也只是用彩纸打印自己的简历而已，而另一些人则会做一些更创新的尝试。例如，有一位应聘者就将自己的简历贴在一个巨大的弹力球上交给了谷歌，而另一位应聘者则将自己的简历印在了一块蛋糕上交给了 Facebook。虽然这不能帮他们拿到聘书，但可以肯定的是他们的简历都被看过了。这些不走寻常路的求职者不仅展现出了非凡的创意，而且还秀出了热情。在某些例子中，他们还能表现出自己是否懂这家公司。

最近一位美国设计师 Eric Gandhi 制作了一份别具一格的简历，看上去好像是 Google 的搜索结果（见图 3）。关键词是“富有创造力、勤奋、有天赋的优秀设计师”，搜索结果的第一个链接就是他自己，Eric Gandhi。他后来被著名的气象频道（Weather Channel）录用。

但切记：这种剑走偏锋的求职方式很有可能遭到不喜欢这种方式的公司或招聘者厌烦。他们可不希望在“一本正经”的工作环境中见到这种“哗众取宠”的人。

最后国外也有一些专门公司（如 TopResume）会为你的简历提供质量评估报告，并提出相应的修改意见。当然这些服务通常都是需要付费。

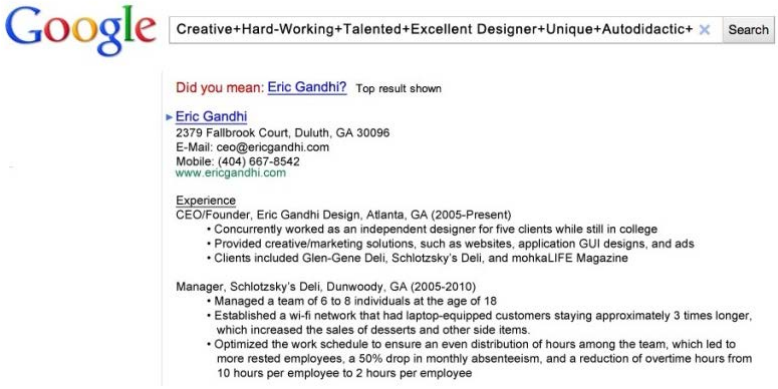

Google
Creative+Hard-Working+Talented+Excellent Designer+Unique+Autodidactic+
Search
Did you mean: Eric Gandhi? Top result shown
Eric Gandhi
2379 Fallbrook Court, Duluth, GA 30096
E-Mail: ceo@ericgandhi.com
Mobile: (404) 667-8542
www.ericgandhi.com
Experience
CEO/Founder, Eric Gandhi Design, Atlanta, GA (2005-Present)
• Concurrently worked as an independent designer for five clients while still in college
• Provided creative/marketing solutions, such as websites, application GUI designs, and ads
• Clients included Glen-Gene Deli, Schlotzsky's Deli, and mohkaLIFE Magazine
Manager, Schlotzsky's Deli, Dunwoody, GA (2005-2010)
• Managed a team of 6 to 8 individuals at the age of 18
• Established a wi-fi network that had laptop-equipped customers staying approximately 3 times longer, which increased the sales of desserts and other side items.
• Optimized the work schedule to ensure an even distribution of hours among the team, which led to more rested employees, a 50% drop in monthly absenteeism, and a reduction of overtime hours from 10 hours per employee to 2 hours per employee

图 3　个性化简历

第 4 章 面试

硅谷公司的程序员面试，也并非高不可攀，绝大部分还是考那些基本功，比如数据结构和算法、对面向对象编程的理解、语言的特性、领域内的专业知识等，跟国内的面试差不太多，甚至有的比国内公司还简单。笔者的一位朋友拿到谷歌总部的程序员 offer 之后，他想转到国内的谷歌，被要求加一轮谷歌中国的面试。很不幸的是额外一轮的面试失败了，他只能留在谷歌总部工作。

4.1 面试准备

让我们在开始讨论如何面试之前，先将注意力放在面试之前的准备工作之上。虽然硅谷公司程序员面试并非是不可逾越的天堑，但是如果应聘者没有经过认真的准备，也很难进入自己心仪的公司。笔者本人也参加过多家美国 IT 公司的面试，大公司小公司都有，最终都通过了这几家公司的技术面试。简单来说每次成功面试的背后都是很多日日夜夜的精心准备工作。

提到面试准备工作，很多人都会想到了一句俗语“临阵磨枪不亮也光”。我们身边有很多人也都是这么做的，接到面试通知之后再翻出书本复习，或者去网络上查找是否有该公司最近泄漏出来的面试题，以期自己也能被问到同样的问题。这样的准备方式基本就是将自己面试成功与否压在了运气上。对于这样的程序员，我们首先要劝他们端正态度，不要把面试看作毕其功于一役的赌博，而更应该将其看作对自己当前水平的一次评估。在此基础上，好的准备工作应该按照长期和短期来分别规划。对于长期准备工作，就是台上几分钟，台下十年功。能力提升的质变需要日常知识积累这一缓慢的量变过程的，没有任何捷径。临近面试突击学习也许能够帮助你应付一些简单的问题，一旦面试官问题深入到细节原理或者对问题稍作变化，你马上会露出马脚。另外一方面而言，面试不是日常工作，不允许你查资料。因此我们需要在面试之前有一段时间来回顾总结我们已经掌握的东西。这就是短期准备。下面我们就来看看该如何做好这两种面试准备。

- **长期准备**

1．制定目标：制定目标就是要帮助自己想清楚到底期望得到什么。目标的重要性不言而喻。只有有了明确的目标，我们才能进一步弄清楚自己应该掌握什么样的技能，加入怎样的公司，才能进行真正的准备工作等。你目标可以很具体，如我要加入谷歌搜索部门，也可以不那么具体，如我要成为全栈工程师。但是目标一定要能落地，空中楼阁般的理想是毫无意义。另外，一旦确定了目标就不要轻易更换。

2．突破语言关：要在美国工作，当然英语听、说、读、写也要足够顺畅流利，但并非要很好才行。一般对于技术人员的语言要求不算太高，能够工作中沟通就可以了。但如果你的英语还有障碍，那还是要努力提高，否则你的职业发展会出现瓶颈，因为没人愿意跟交流困难的人共事。仅仅是口语，是可以在短时间内提升的，比如盲听网络视频、复述名人演讲、模仿美剧中的语音语调、参加口语培训班等。另外除了日常沟通口语的练习之外，额外需要重点准

备技术术语的英文表达方式。例如面向对象编程相关的封装（encapsulation）多态（polymorphism），或者算法数据结构中的最小生成树（Minimium Spanning Tree）等等。平时坚持看英文版的文档和技术书籍能够帮助你培养语感，更好地理解应对面试过程中的技术问题。一轮面试的时间通常不会超过 1 个小时，如果相当比例的时间是花在弄懂彼此想表达的意思上，也就基本意味着通过面试的机会很渺茫。

3．练好基本功：大部分硅谷公司都将聪明（smart）作为招收程序员的重要考量标准，因为经验在公司看来可以后天慢慢培养。那么如何在短短几十分钟的面试中判断应聘者是否聪明呢？面试官一般都会选择考察算法和数据结构这类基本基本功，看应聘者能否灵活应用这些知识来解决问题。当然这些题目的难度一般不会太高，远远达不到 ACM 那种复杂度。在本书的第二部分，会列出一些常见的这类问题供读者参考。如果平时注意花时间练习这类问题，会大大提高你在面试过程中的相应速度，甚至在解答问题的过程可以给出多套解决方案并能比较彼此的优劣。这些都将给面试官留下深刻的印象。Linux 内核的发明者 Linus Torvalds 曾说过“talk is cheap, show me the code”。对于基本功的练习不要只停留在书本知识，要真正动手写成代码，因为面试的时候，通常都会要求你把自己的解决方案写成可执行的代码。现在有很多网站都提供在线答题的功能（如 topcode，leetcode）。网站会罗列出一些难度适中经典问题，并提供编译环境和测试数据集来验证用户提交的代码正确与否。我们可以充分利用这些工具，苦练基本功。基本功的熟练和提升很难一蹴而就，需要长时间的积累，一般为 2-5 个月，视个人基础而变。对于那些离开校园参加工作多年的“老”程序员们，可能在工作中不太需要使用这类技术，早就淡忘了， 但是基本功的准备对于一次成功的面试仍然不可或缺。

4．关注技术细节：面试官面试时往往会针对应聘者的简历内容来发问，考察应聘者过往工作经验的含金量，尤其当你应聘的公司和你之前工作方向接近时。笔者就曾经在谷歌技术面试中被问到各种索引数据压缩的问题，因为笔

者在自己的简历中提到参与并开发过搜索引擎。这就要求应聘者在平日的工作中要关注自己使用技术的发展动向、实现的细节原理。只有这样才能在面试官发问时做到从容不迫。另外也要求应聘者在日常工作时不要只是一味埋头苦干，时时抬起头了解一下行业内同样问题的解决方案有时能够达到事半功倍的效果。笔者也面试过很多程序员，有相当比例的应聘者对于那种需要死记硬背的知识点如数家珍，但是一旦面试官针对这些知识点深入到细节，他们就顾左右言其他了。例如之前面试过一个有多年 C++开发背景的后端开发程序员，其对 C++继承多态的定义作用滔滔不绝，也提及虚函数表在其中的作用，可是当笔者顺着他的思路稍微深入到虚函数表的存在形式以及如何被用来实现动态绑定这些细节时，应聘者就卡壳了。他一开始回答的那些知识点，稍微有 C++编程经验的人都能回答。别人知道你知道，别人不清楚的你也懵懂。这样的应聘者怎么可能在众多竞争对手中脱颖而出呢。细节决定成败，只有通过对细节的把握才可以凸显你和其他应聘者的不同。在工作学习中要“拒绝平庸”（make yourself special）。

5．提升大局观：对于那些竞争资深程序员的岗位的应聘者而言，一定要准备应对系统设计能力的考察。对于资深程序员岗位，面试官一般都会要求应聘者设计一套系统来满足各种业务需求。系统设计问题往往取材于业内经典的技术难题或者是面试官之前遇到的真实问题。这类问题本身并不一定有所谓的标准答案或者最优解，面试官往往是想看看应聘者如何处理第一次遇到的技术挑战，权衡性能和实现复杂度之间额平衡等等。因此系统设计问题能够很好地考察应聘者是否具备成为团队中技术负责人的潜力，也能让面试官更清楚地了解应聘者在之前工作经历中扮演的角色。关于这类问题本身，本书后面的章节将给出一些例子供读者练习熟悉。不过应聘者要想真正在面试中对这类问题做到成竹在胸，在平时工作中，不能只关注自己手上的那一亩三分地，需要花一定的精力与时间去了解周围相关领域甚至是那些与自己工作没有太多交集但是很热门的领域。保持一定广度知识体系对于程序员整个职业生涯而言都是必要的。

6．培养软实力：软实力（soft skill）包含的内容很多，涉及性格特质、为人处事等多个维度。对于程序员而言，面试中最容易被考察也是工作中最重要的软实力是团队合作、风险意识。特别是大公司的面试中，面试官通常会给出一些日常工作中经常遇到的场景，问应聘者会如何应对。常见的问题如，如何和其他团队合作，如何评估项目风险，如何应对项目开发中突发的额外工作量，如何应对工作中的压力等等。要想回答好这类问题，应聘者不仅要有过硬的技术背景，更应该具备高超的工作智慧。也就是我们经常听到的“表现得很职业（professional）”。这些软实力的培训主要来自于课程学习和工作中的观察。很多大中型公司都会定期开展在员工中开发这种培养软实力的课程，如时间管理、团队合作等。另外一方面，工作的团队中往往会有这样拥有高超工作智慧的同事，我们可以在日常工作中观察这些高手们是如何应对工作中的种种挑战，同时问问自己如果遇到这类问题会如何处理。勤于思考是成为一个高价值程序员的关键因素之一。这种思考不应该仅仅局限在技术层面。

- **短期准备**

1．公司背景调查：针对你的目标公司清单，从多方渠道获取每家公司的公司文化、面试流程、招聘渠道、面试方式、题目范围等方面的信息。每个公司特别是那些大型公司都有自己的面试特点，例如亚马逊喜欢问 OOP、OOD 问题。你可以针对自己调查的结果，提高自己面试准备工作的效率。获取这些信息的渠道很多。如果有认识的朋友同学在这家公司那是最好不过了。即使没有，也可以去网上查询，例如可以在 glassdoor，indeed 这类求职网站搜索该公司的面试经历，还可以去很多海外求职论坛求助。

2．知识点复习：这种面试前的知识点复习，时间紧任务重，不能毫无重点。一般来说应聘者应该主要按照以下两个方向安排自己的复习工作。（1）复习自己的简历上提及的内容。时间上越靠近当前，越需要重点复习。否则面试时如果连自己最近做的东西都解释不清楚，就很难让面试官感到满意了。（2）针对之前所做的公司背景调查，有针对性的复习，特别是熟悉行业背景知识。

例如目标公司是从事互联网广告投放业务，那么提前了解一下互联网广告是如何投放的。在面试过程中，你就会给面试官留下一个印象，你有一定行业背景，比其他应聘者更容易适应这份工作。在面试最后决定阶段，这点小小的不同可能会起到意想不到的作用。

3．行为问题（behavior question）：这部分很多人往往会忽略，其实海外公司都喜欢问行为问题。如你最近一年遇到的最大的技术挑战是什么，你是如何克服的；又如你最近一年最自豪的成就是什么？还有你职业生涯中犯的最大错误是什么？如此总总。对于这类问题，应聘者回答时要注意自己的答案给面试官的感觉，不要给人一个很做作的感觉。例如面试官问你自己最大的缺点是什么？如果答案是诸如自己太努力而老是忘记休息之类，面试官是不会感到高兴的。但是应聘者同时也要注意不要让自己诚实的答案的给自己形象带来太大的负面影响。例如同样是问自己最大的缺点，不喜欢与人合作这类回答肯定是等于宣判了自己面试的死刑。即使真要如实回答，最后也要加上自己现在是如何尝试克服这些缺点。

上面列出这几点涵盖了程序员面试的大部分准备工作。但是对于初级程序员和资深程序员，面试中考察的重点会有所不同，相应的准备工作也应该各有侧重。初级程序员通常不会有太引人瞩目的项目经验。对于初级程序员，公司更关注的是其是否具备未来成功的潜质、自我学习的能力与激情等。因此面试官一般也会重点考察这类应聘者是否有扎实的基本功（如算法、数据结构、操作系统知识等）。按照我们之前提到的准备条目，初级程序员应该更关注基本功锤炼和软实力的增强。

而对于资深程序员，扎实的基本功是必不可少（在后面章节，读者可以看到，即使是资深程序员，硅谷公司也会注意考察其算法和数据结构的水准），但是同时公司还会特别关注这些有多年工作经验的应聘者是否具备一定技术深度，对应着我们前面提到的关注技术细节和提升大局观。每一位读者可以参照我们上面的建议并结合自己的特点，系统地开展自己的面试准备工作。

面试还有运气成分，这导致再充分的准备也无法确保你能获得梦寐以求的工作。笔者的建议是把你想要获得的工作按优先级进行排序，当你进行最后一次面试时，结局将如你所愿，因为面试是个积累经验的过程，通常面了几次后表现会更加成熟。换句话说，把你不太想去的公司的面试放在前面作为练习，积累经验后再开始面试你想去的公司。

4.2 面试流程

我们以某人的 Facebook 面试流程为例，透视硅谷公司是如何面试程序员的。

第一轮电话面试

输入一个字符串数组，有些字串是同位词（Anagrams），分组输出这些 Anagrams。

第二轮电话面试

实现两个长的数字串相乘。

现场面试（onsite）

一共四个面试官，每场面试 45 分钟，中途除了 40 分钟午饭时间之外，没有其他任何休息时间。

第一个面试官，编程题目（Coding Question）。给定一个函数，可从文件中读取固定大小；实现函数，根据指定的大小读取文件。

第二个面试官，设计题目（Design Question）。一位经理加上一位旁听者。主要问 switch infrastructure 相关的东西，算是泛泛而谈。

第三个面试官，行为考察（Behavior Question）。主要问：为什么要跳槽？为什么要来 Facebook？在你做过的所有项目中最得意的是什么？等等。

第四个面试官，编程题目（Coding Question）。又有一个旁听者。实现一个固定容量的 pipe。于是悲剧发生了，事后忍不住免冠徒跣以头抢地。忘了这

个是要求考虑循环数组的。写完后，在面试官的冷眼下惶恐修补，但是还是超时了。因为“可恶”的 Facebook 文化，一定要留出时间来问问题。

加上电话面试总共 6 轮，其中 4 轮是编程面试，1 轮是技术设计，剩下 1 轮是行为考察。目前在硅谷程序员面试里，编程面试占绝大比例，并逐步取代基于项目经验的面试。这有很多原因，其中最大的原因之一是之前很多印度人求职时简历作假，项目作假，如果公司光是考核项目，很难甄别候选者，所以不管三七二十一，都要求候选者在白板上写程序。还有个原因是不少硅谷公司认为只要招到足够聪明的人就行，项目经验可以慢慢培养。我们会在下一个章节介绍如何准备编程面试。

当然，项目经验也很重要，几乎每轮有 5 到 10 分钟时间来讨论你的项目经验。如果我们从面试官的角度来考察候选者，我们会在意候选者的经验以及解决问题的能力。

- **经验**

面试中，我们会问这些问题：你遇到最难的问题是什么？是如何解决的？你是怎么设计这个系统的？是怎么调试和测试你的程序的？你是怎么做性能调优的？什么样的代码是好的代码？等等。重要的是你对知识的运用和驾驭，对做过的事情的反思和总结。最好是能让面试官从你的经验中受益。

- **解决问题的能力**

我们会出很难的题目，难到我们自己也没有解决方案，所以我们想从候选者那里得到的不是那个解题的答案，而是解题的思路和方法。在面试官看来，解难题的过程更重要，通过解题过程来考察应聘者的思路、运用知识的广度和深度、应聘者是否有经验、沟通是否顺畅等。当然，最终是要找到题目的答案。通常从面试官的角度，他们会考察应聘者如下几个方面。

- 应聘者在解算法题时会不会分解或简化这个难题？

- 应聘者在解算法题时会不会使用一些基础知识，如数据结构和基础算法？
- 在讨论的过程中应聘者有没有钻研能力？进一步优化解决方案？
- 应聘者是否有畏难情绪？是否能承受一定压力？
- 应聘者是否有和我们交流的能力？如果以后成为同事，能否较快融入现有的团队里？

4.3 编程面试

每轮编程面试有 45 分钟，扣除双方自我介绍和提问的时间，花在编程上的时间大约为 30 分钟。由于受到面试时间的限制，面试的题目不会太难，比大学生编程比赛（ACM）题目简单很多，但是，需要一些编程面试技巧及对算法、数据结构的熟练掌握才能在限定时间内完成。这对要求在白板上写程序和代码无 Bug（Bug Free）的公司来说尤其重要，比如 Facebook。而习惯使用 IDE 来获取函数或校正拼写的应聘者应该特别注意。

在编程的面试中，光有解法却写不出来代码是行不通的，这会让面试官觉得你只会夸夸其谈，却不会编程。在编程面试里，切记“让代码说话”这条准则。针对每道面试题，通常应该进行如下步骤。

- 复述/提问：用自己的话复述面试官的题目，以免偏题。面试官给出的面试题并非一开始就很明确，需要多次提问来确定题意、边界条件、时间和数据结构限制等。
- 举例：可以和提问同步进行，主要用来确认输入和输出结果。
- 观察：通过举例来总结规律，思考可能使用到的结构和算法，然后设计一种最优算法。
- 编码：和面试官沟通算法之后，开始在白板上编码。

- 测试：使用个别例子，把代码测试一遍。

在以上 5 个步骤里，视时间充裕情况，有些步骤可以省略。比如，如果面试官已经把问题说得很清楚了，那么复述可以省略。在把代码交给面试官之前，一定先要自己检查一遍代码。当遇到难题时不要慌张，也许面试官考察的正是你的抗压能力，冷静下来，把常见的数据结构和算法往里套；实在没有头绪时，可以要求面试官给些提示（很多人不知道这一点，这也是硅谷公司和国内公司面试的不同之一），积极和面试官沟通你遇到的问题。当遇到见过的题目时，千万不要沾沾自喜，马上背出答案，这只能适得其反；你得再次确认题目是否和自己记忆中的一模一样，即便一模一样，你也得假装思考几分钟，然后把思路和方案告诉面试官。不过一般技术面试有好几轮，如果面试官问了一个前面几轮的面试官已经问过的问题，你还是应该主动告知，因为面试官通常都会写面试反馈报告，里面会记录面试过程中自己问了那些问题。最后综合评定时，会汇总所有面试官的报告来决定你是否通过了技术面试。很容易就会发现同样的问题向你提出了多次，因此隐瞒这个对你面试结果不会带来任何正面的影响，相反会让面试官们觉得你不够诚实，这个往往是致命的。

编程面试所需的准备时间视个人的基础不同而不同，从 2 个月到 1 年不等。平时做项目时就试着减少对 IDE 的依赖，时刻记录工作过程中遇到的难点及攻克的方案。对于如何准备编程面试，笔者提供的建议是：练习、练习还是练习。在本书第二部分和第三部分，笔者贴出了面试题的全部代码，更多时候是想让代码来“说话”。

4.4 注意事项

技术能力和项目经验需要靠一段时间的学习和积累。但是，面试的现场表现的确可以在短时间内准备和提高。在这里，我们分享一些面试的技巧和注意

事项。

- **准备持久战**

如果是电话面试，请准备好耳塞，这样便可以解放双手；确保网络畅通，方便在线写代码。如果是现场面试，请保证前一天充足睡眠，吃饱早餐（注意：美国的牛奶比国内浓，避免喝过多牛奶导致身体不适），提前 30 分钟到公司。咖啡或茶可以提神，午饭不宜过饱，以免下午犯困；在每轮面试间隔，争取几分钟的休息时间。

- **保持自信**

走进面试房间之前，检查一下自己的衣着打扮。深吸一口气保持镇定，通过自我暗示，默默地告诉自己能行。走进房间，主动给面试官打个招呼，给一个简短有力的握手，留下一个自信的第一印象。交流时，定时保持眼神接触，这是我们中国人所缺乏的，毕竟两国文化不一样。

- **表达顺畅**

不要觉得自己的英文说得不好，怕说错，说话很小声；有时候又因为紧张，越说越快。因为你说话声音很小、有口音，加上语速太快，所以面试官就很难听明白你要说什么。因此，说话的时候声音一定要足够大，让对方能听得很清楚。同时，你可以主动放慢自己的语速，也可以要求对方适当放慢他们的语速。不用担心你的英语不够好，只要能确保对方能理解你的意思就行了。在大部分硅谷的 IT 公司里，本身就拥有大量外籍员工，比如大量来自东欧和亚洲的母语非英语的工程师。他们可以理解你的英语表达并且愿意减慢语速。如果语言交流还有障碍，可以借用文字，在白板或白纸上写下关键的问题和方案。

- **确认双方对问题的理解一致**

在 4.3 节里，我们谈到编程面试的第一步为复述/提问，其实这对所有面试

都适用。硅谷的公司面试往往是使用工作里面碰到的实际问题，在此基础上做一定的简化，改成可以在 30 分钟之内解决的问题。问题本身很有可能是似是而非的。如果你没有跟面试的同事确认你们俩对问题的理解是否一样，那很容易发生的情况是你按照自己的理解花 30 分钟写出了程序，然后面试官说“这不是我要的答案啊”。比较稳妥的方法是你主动针对问题提出一些讨论，比如用几个简单而有代表性的例子把问题的输入和期待的输出都描述清楚，双方都同意这一组输入输出，然后再动手解决，这也自动地给你一组测试例子了。

- **确认对方理解你的方案**

笔者见过不少例子是面试者用很复杂的算法（往往需要一篇论文来证明其正确性）来得到一定的算法复杂度优化。大部分的工程师并没有那么专注于学术，面试官很可能看不明白。而面试者本人也很难在 10 分钟内把一篇论文的正确性说清楚。这样的结果使别人不可能很快地同意你的解决方案，而在 45 分钟的面试里面，面试官就很可能会不赞成录取你。因此，如果一个算法不能在 5 分钟之内解释清楚，就不要在面试时用这个算法。

- **互动交流**

把面试官当成你的同事，而不是监考老师。在整个面试的过程中，技术题目答案的优劣大概只占一半的权重，同样重要的是看你能否从头到尾解决一个实际问题。你得在短时间内做出权衡：理解问题的本质、约束条件、时间和空间的取舍、面试时间限制和工作量的取舍及程序的易读性。而获取这些权衡，必须要做到主动沟通，从面试官那里得到帮助。最优秀的面试者往往把自己沉浸在与面试官共同工作的状态，主动分析问题，甚至对问题提出异议，一起讨论，就解决方案达成一致，然后挽起袖子写出一个漂亮的程序，再根据讨论的例子对程序进行测试，最后提出对自己的解决方案的看法及下一步可改进的地方。这样走下来，即便写出来的程序不是最优的，而你在整个过程里表现出来的经验和解决问题的能力都会让人刮目相看。

➢ 抓住提问环节

在面试最后都有 5 分钟左右的提问时间，千万别浪费提问的机会。在面试过程中，应该表现出就像已经获得工作邀请一样。这非常有趣，因为不仅是企业在面试你，你也在面试企业。果断地提问一方面体现出你对这份工作的渴望，另一方面也方便你以后挑选公司。泛泛的提问可能很难获得你想要的答案，所以除了问一些有关团队和技术的问题以外，还可以提出一些有关公司如何运营的细节问题，特别是针对创业公司。以下就是笔者曾经提出的对笔者后来有用的问题，看看哪一条适合你：

- 公司的决策速度如何?
- 工程师在测试一项产品提议时一般会采取何种措施?
- 团队间是如何协作的?
- 对新人是否有正式的指导计划?
- 新人能从公司学到多少东西?
- 会不会提供管理培训或尝试新事物的机会?

第 5 章

聘书与职业发展

经过一系列紧张激烈的面试，恭喜你拿到一家甚至多家硅谷 IT 公司的聘书（offer）。那么，接下来的事情对你来说将是一个幸福的“烦恼”：我要不要接受这家公司的 offer？我应该选择哪一家公司呢？

当拿到聘书时，不要立即接受或者拒绝，可以礼貌地说：“我非常开心能有这个机会。请给我一段时间让我认真思考一下，再给您答复。非常感谢！”一般来说 HR 会给你一周到一个月的时间考虑，因为他们也知道你会同时面试好几家公司。接不接受 offer，取决于很多因素，比如待遇、职业发展、工作幸福感等。不少人选择去工资最高的公司，又或者是不好意思拒接对方的盛情邀请，而去了一家 HR 比较主动的公司，不幸的是，在入职一段时间后你才发现当前这家公司并不是最适合自己的。要知道频繁跳槽对你的职业发展并不利。

5.1 聘书

如何评价一份聘书（offer）呢？它包括了薪水、奖金、股票、假期、医保等，而且还含有工作内容和职责。当你做出选择时，你必须考虑自己的职业方向、公司文化、未来的工作伙伴，甚至会考虑到自己的家人。然而，更加棘手的是，你不可能知道所有的事情，比如到底要工作多少个小时？每年能有多少加薪？等等。此外，当你阅读聘书时，还可能需要考虑与职业发展相关的问题，比如这份工作对于你的职业发展是否有利？是否会为你的简历增色？是否能帮助自己在职业道路上提升？以及会考虑到与薪资待遇相关的问题，比如他们付你多少钱？其他待遇有哪些（比如医疗、股票等）？与你自身幸福感相关的问题，比如你会喜欢这份工作吗？和同事会相处愉快吗？工作地点是你理想的吗？

在接下来的章节，笔者会帮助你分析聘书并且告诉你做出决策时需要考虑哪些因素，以便你为自己作出正确选择。

5.1.1 聘书要素

除了基本薪水以外，硅谷 IT 公司的聘书常常包括股票（stock）、股权（option）、签字费（sign-on）等。如何在不同的指标间进行比较和衡量呢？给每样都贴一个价格标签，然后与你计划在这家公司的年数相除。比如，假设亚马逊报价 10 万美元年薪加 2 万元签字费，微软为 5 千美元的签字费加上 10.5 万年薪。这两家公司哪家待遇好呢？这取决于你打算在这家公司干多久。如果两年后就离开，那么亚马逊更好。换句话说，在一家公司干的时间越长，一次性支付的津贴就越显得不重要。

要看透聘书的薪资待遇，就要看聘书中列出的所有项目，包括那些没有书

面记录的项目。下面列举的项目，拿到越多越好。和国内 IT 公司不同的是，硅谷 IT 公司也会给普通级别的程序员搬家费、股票（创业公司是股权）、签字费等，而不只是给新入职的高级工程师或管理层。

表 2

核心要素	其他福利
基本薪水	年终奖
签字费	年度加薪
搬家费	员工购股计划
优先认股权	401K 计划
无偿配股	医疗保险
休假	免费午餐、食品、健身房等

表 2 中的一些因素，诸如年终奖和每年的加薪幅度，不是很容易就能明确知晓，因为公司都不太愿意透露这方面的信息。但是如果你能找到一位内部员工打探一下的话，也许能了解一般标准是怎样的，好一些的情况又是怎样的。

在比较聘书时，一定要把工作地考虑进去。正如我们在前面章节提到的税率问题，各州个人所得税、消费税不一样，各地消费水平也不一样，你需要比较工作地点的生活成本。

5.1.2 决策因子

在考虑薪酬的同时，还需要考虑以下的决策因素。

- **绿卡政策**

作为外国人，你没法忽视雇主的绿卡政策，除非只打算出国工作几年，攒些经历，然后回国。如果希望以后在美国长久生活，接收 offer 前先问问 HR：公司对绿卡的申请是不是很支持？一般要多长时间才能开始？有些公司需要

员工工作两年之后才开始协助员工申请绿卡，而有些公司在员工入职之时就开始办理。

➢ **个人目标**

你 5 年以后想做什么？是开一家自己的公司？还是加入一家创业公司？又或者是想在某一个领域做世界一流的技术专家？还是想在一家大公司里面做管理或是做一个深耕技术的工程师？回答这个问题很重要，因为无论薪水多少、加薪速度多快，我们都会觉得太少。最终的成就感和幸福感往往来自于个人成长和个人目标的实现。找到一家能给自己的成长提供相应机会的公司，如果这家公司还能实现自己的个人目标，那就更好了。

如果你以后打算创业，那么加入有发展前景的创业公司更合适。如果你打算老老实实地在公司里爬梯子，那么可以向你未来上司或 HR 询问一下公司为员工提供的职业路线，不管是技术路线，还是管理路线。如果做得好，三年之后公司会提供什么样的职业道路？公司会为你的职业规划提供什么样的培训和支持？是否支持员工到大学里面在职攻读 MBA？是否可以在公司内改变职业路线，比如从技术换管理？跨部门调动是否有很高的门槛？等等。

如果你是行业新人，那么可能会更看重行业培训。通常，大公司比中小公司，有着更为严格的培训体系。比如谷歌，它会给每一位新员工进行一个为期两周的入职培训。通过这些课程，新员工可以了解谷歌作为一家公司的运作情况，并且深入了解自己本职工作的需求。除了新员工培训以外，有些公司还会提供一些进修课程，这些课程会安排在公司内部或是当地的大学进行。

➢ **晋升空间**

几乎每家公司都会对工程师分等级，但是在聘书上不会写出你的等级。只有主动询问 HR，才能知道自己的等级。此外，你还得了解，整个公司的工程师等级是怎样定义和划分的？如果有多份聘书，不同公司的等级在数字上不一

样，那么如何换算不同公司的等级？技术团队如何做业绩评估？晋升的过程是怎么样的？是谁决定我的业绩和晋升？等等这些问题。

如果你想得到快速提升，那么加入成长型公司（或团队）是一个非常好的选择。成长型公司意味着要招入新人，当然这些新人要有人领导，这个领导者或许就是你。

此外，你还得了解公司内部的提拔机制。一些公司习惯从公司内部提拔优秀员工，而另一些则喜欢从外部吸纳高管人员。比如 Intel 就是一个常常从内部员工中选拔管理人员的公司。而在早期的谷歌，很多管理者都是外聘的。

- **公司文化和前景**

快速增长的公司会有更多的学习和晋升机会，但淘汰率也更高；已经成型的大公司不太会因为业绩问题而淘汰个人，但更有可能解散整个部门。在专注产品的公司工作，更容易学到产品开发的知识，这样的公司更重视能把握整个开发流程的人才，而那些只希望在一个领域钻研的人则不一定做得开心；专注技术的公司则相反。工程师文化很强的公司会把工程师看成创造机会的原动力，所以聚集了很多具有很强自主性的顶尖人才，这些的公司对个人创造力和主动性要求也高；而销售文化很强的公司则往往把工程师看作成本，工资能压则压，工程师往往只需要跟进销售团队的反馈意见，不要求很强的主动性。

作为外国人，还有个因素不可忽视，那就是工作的稳定性。如果你所在的团队不稳定，或者整个公司在裁员中，那么这种情况对你申请绿卡会很不利，同时也会对你的生活构成重大威胁。

- **幸福感**

没有幸福感的员工会减少工作时间，工作效率降低并且不久就会辞职。在接受一份觉得可能会让自己不开心的工作之前，好好想想自己是否能够应付。接受工作之前，需要想清楚的是，什么能让你快乐？是和你一起工作的人吗？

还是因为这份工作能激发你的才智，获得成就感？又或是你认为这份工作很有意义可以影响人们的生活？等等。

其实通过一个细节也能看出在工作中你是否有幸福感，通过一个周末的休息，看看周一的你是精神百倍，还是无精打采、厌倦上班。充足时间的休息可以缓解身体疲劳，但没法解决厌烦的工作情绪。

你的顶头上司和你的同事也会影响你的幸福感。你也希望看到公司能有一个氛围能帮助刚从中国到来的新员工适应语言、工作、生活和文化的改变。你和上司的关系会在最大程度上决定你是否喜欢这份工作。请务必和自己的未来老板进行一次谈话，问问他这样的问题：在公司如何做算得上成功？很多朋友入职是因为公司和项目的吸引力，而离职是因为他们和上司关系不和。

当你做决策时，HR 并不能完全帮得上忙，毕竟两者对公司的看法角度不同，因此，你还得找公司里跟自己背景相似的员工聊一下。如果有校友或朋友在那个公司工作，一定要找他问问情况。如果没有，可以跟 HR 回信说：在做决定之前，能不能跟公司里的一个中国工程师聊一下？我想了解他们来美国的过程和工作生活的情况。一般 HR 是很乐意为你找这样的工程师。只有跟背景相似的员工聊天，才能看到公司更真实的一面。

5.1.3 薪酬谈判

在美国，很多交易是明码标价的，没有讨价还价的余地，但是薪酬谈判是个例外。假如拿到了 offer，在谈具体工作事项前，我们首先要解决一个大问题——薪酬谈判。很多人不愿意谈，或表现出这并不重要的样子。这很好，但是你必须准备好这种谈话，因为这是你在开始正式工作前唯一一次讨论薪酬的机会，一旦签订了合同，你的所得就会基于你的表现，除了晋升的时候，你不会有太多重新谈判的机会。

在面试的过程中，如果他们提问到你想要的薪酬或股权，你需要知道的是，

这不是一个随便问问的问题，不要在这样的交谈中随意抛出你想要的数字。你可以告诉他们，自己尚未想好确切的数字，但可以告诉他们你希望获得的薪酬水准是多少。

- **永远不要接受第一次报价**

薪酬谈判是多次来回的过程。HR 都是谈判高手，特别是大公司的 HR，他们每周都要负责好多个新人入职。HR 第一次给出的数字通常是留有余地的，而第二次的数字可能是 HR 跟管理团队讨论之后的结果。可以讨价还价的 offer 要素是基本薪水、签字费、搬家费、股票或股权，其他的要素，比如假期、401K 等几乎没有谈判余地。其中基本薪水和搬家费余地很小，因为这些数字和你的级别有关，而且公司不想改变现有的制度，以免激起老员工的不满情绪。不要小瞧签字费和股票，每年你从股票获得的钱可能比你的薪水还高。笔者还见过 Facebook 给新毕业的博士发十万美元的签字费的情况。

- **最好有备选方案**

如果能拿多个 offer，你也就能大概知道自己值多少钱了。不少财大气粗的公司，比如谷歌和 Facebook，能提供其他竞争对手相同的薪水给你。有了多个 offer，在谈判过程中你就站到了主导地位。如果只拿到了一个 offer，还有其他公司没有答复，可以跟其他公司说你有一个待决定（pending）的 offer，让他们加快速度；而跟给你 offer 的公司说你需要一些时间决定，希望对方能够理解和体谅。在这个过程中，HR 很可能会敦促你快速做决定，你需要诚实告之当前进展。

- **做好调研**

如果你了解整个行业的薪资水平和公司给予相似员工的待遇情况，你便可以对提出多少要求做到心中有数。浏览一下我们在第 2 章求职渠道中提到的薪资网站，比如 www.glassdoor.com，对薪资待遇做一番调查。

➢ **适可而止**

公平性是大家都愿意接受的一个讨论基础。从公司角度来说，如果你的待遇低于市场待遇，那你会觉得没有被公平对待，很可能会很快离开公司，这对公司也不是一件好事情。从个人角度来说，一般抱着“不吃亏就好”的心态，不要想着从中得到比别的类似背景的工程师更好的待遇。待遇总是跟期望值挂钩的，如果你真的要到了更好的待遇，公司对你的期望值也会更高。

讨价还价的过程本身很容易让人过于专注在薪酬数字上从而忽视了对其他因素的考虑。在讨价还价的过程中要保持适可而止的态度，以免公司收回 offer。笔者的一个朋友拿到了一家创业公司的 offer 之后，讨价还价了许久，激怒了该公司的 CEO，最后公司把 offer 收回了。

5.1.4 接受、延期或婉拒

在硅谷绝大部分 IT 公司的 offer 是 at-will employment，即雇佣双方的任何一方都可以随时随地无条件地解除雇佣的协议。类似国内有工作期限的合同，这种合同并不是给全职的正式员工，而是给在美国称之为“合同工”（contract）的人。当雇员决定离开公司时，一般应在离开前的两周要礼貌性地通知公司。公司如果需要解除雇佣协议，一般应提供两周或更多的补偿，这取决于双方谈判的结果。如果你接受了 offer，要时刻与对方 HR 和你未来上司保持联系，并表示如果公司有什么变动，请及时告诉你，因为公司可以随时解除雇佣关系。

当你还在面试其他公司而当前这家公司提供的这个职位又不是你最想要的，或者其他什么原因，这时你需要向对方申请延期做决定。在给你 offer 的时候，公司一般都会设个最后期限。其实公司有充分的理由设定最后期限，因为他们不能把位子留给你的同时还去面试其他人选，也不可能把下决定的时间拖得太长。如果你需要延期，就和招聘者实话实说，向对方解释你需要延期的原因，你和其他公司的接触进行到了哪个阶段以及你什么时候可以做出决

定等。

婉拒并不意味着断绝关系。其实更应该看做是“改期再约”。你是喜欢这家公司的，所以千辛万苦完成了所有的招聘程序；他们也是喜欢你的，所以才给你发出了聘书。你们之间的这层关系以后可能用得上。因此，婉拒对方的时候，你应该尽可能通过你们一直以来采用的沟通方式同对方接触。也就是说，如果那个招聘负责人总是给你打电话，那你就要和他致电沟通。换而言之，如果你一直是和经理电子邮件往来的，那你就要第一时间给那位经理写邮件说明。不论是写简短的邮件还是打电话，你都应该找那个与你经常联系的人沟通。

5.2 职业发展

除了深耕技术和做好自己的工作以外，笔者还有如下与职业发展相关的建议。

- **表现专业**

作为职场中人，我们的第一要务是推动项目向前发展，就是常说的 make thing done，这也是公司雇佣我们的根本原因，然后才是通过自己的表现争取涨薪和升职。如果本末倒置，只想着表现自我而忽视项目本身，也许能够在短期取得利益，但是长期下去只会让自己的职场之路越走越窄。那么我们应该如何表现专业呢？讨论问题时对事不对人，积极汇报工作进展，多给上司和组里成员发送 E-mail，留下书面证据。一方面让上司知道你的工作，另一方面也让其他人知道你在干活。如果口语是我们的劣势，那 E-mail 就是最好的交流方式。积极回复群里邮件，不是说让你多揽活干，而是让大家知道你能干这个活，前提是你了解同事所做的工作。

➢ **摸清评价体系**

你必须了解是谁评价你的工作表现（performance）及评价体系的要素是什么，做到有的放矢。和他搞好关系，随时让他知道你的进度。比如，有些公司是经理评价你的工作表现，但是你主要跟组长干活。在这种情况下，你要想尽办法让经理知道你的进展，千万不要把所有的宝押在组长上面，以为他会把你的表现如实上报给经理。

➢ **迎合上司**

做技术的程序员多少有些傲气，很多时候你觉得一件事情应该这么做，但是老板却不这么认为。迎合上司比固执己见更重要。上司都是孤独的，他需要有人迎合、认同他，他也会提拔和他意见一致的人。

➢ **寻找导师**

找到一位职业导师，比如直接上司或者资深工程师，他可以让你少走弯路，更快融入团队。如果有机会，你也可以做别人的导师，扩展你的人脉。

➢ **抓住机会**

通常来说，如果你为一个快速发展的项目干活，那么你得到提升的机会就越大。因此，必要时候，看准了就换组、换项目。如果关键岗位突然出现空缺，而你又有八成把握胜任这个职位，在这个时候你应该毫不犹豫挺身而出，积极竞聘，而不是等到你有了百分之百的能力时再竞聘。你要知道很多高官都是在岗位上把能力锻炼出来的。如果你一直在走技术路线可能很难被提升，因为做策略的工作通常比做工程的工作更容易得到提升，由于策略的工作直接影响公司收入，从而能引起公司上上下下所有人的关注。

5.3 优秀工程师

作为在异国他乡打拼的工程师，你面临不仅仅是文化差异的压力，还有身份问题、从零开始的经济压力以及几乎可以忽略的社会关系资源等等。因此，你必须成为优秀员工，比优秀的人更优秀。一般来说，加盟一家公司，是因为公司产品、公司文化、公司发展前景等公司的因素吸引了你；而你被迫离开那家公司，更多因素是你的上司不喜欢你。为了确保工作的稳定性，我们必须要和上司搞好关系。

- **优秀工程师和普通工程师**

优秀工程师充分了解项目、技术方案、技术前沿情况，凭借自己丰富的经验确保项目按期完成；并且勇于承担责任，以项目的成功与否来衡量自己。普通工程师却不愿意承担责任，项目的失败归咎于项目经理无能、排期不合理、产品经理没有产品意识、公司不够重视、人力有限、没人指导等一大堆借口。

优秀工程师及时发现问题、及时解决，自己没法解决的时候，积极寻求上级、同事甚至外部的支援。普通工程师发现问题的时候，藏着窝着，等着别人发现后被动去解决。

优秀工程师会采用书面形式和口头形式与产品经理进行清晰的交流，共同决定产品的走向。他们还会制作项目 wiki、常见问题解答、开发简报以及附加材料，供产品经理、营销人员和上司参考或使用。普通工程师会抱怨自己整天都在为产品经理、营销人员等上下游同事解答问题，忙得不可开交。

优秀工程师会把一些重要问题（技术选型、架构选择、核心算法优化、重要功能实现等）以书面方式记录下来。普通工程师只以口头方式表达自己的意见，抱怨上司不允许他的方式来做。一旦失败，往往会说自己早就料到会失败。

优秀工程师愿意分享技术、帮助同事、扩大自己的技术影响力。普通工程师不愿意分享自己的技术，不愿意帮助他人，采用多一事不如少一事的处事原则。

优秀工程师偏重清晰明了，普通工程师对显而易见的事情从不解释。优秀工程师对自己的职责和技术方案有明确的认识，普通工程师总想让别人告诉他该做什么。

优秀工程师讨论问题的时候，就事论事，对事不对人。普通工程师会意气用事，上升到人身攻击，指责他人不配合。

优秀工程师总是对自己的工作充满激情，实时响应上司号召。普通工程师通常袖手旁观，在头脑风暴会议通常保持沉默，一旦项目遇到挫折，暗地里冷嘲热讽。

优秀工程师总表现出超强的执行力，也能对正确地划分任务优先级。普通工程师总是让经理觉得不能胜任，哪怕是很小的模块。

优秀工程师抛出项目不足之处的同时，也能给出改善性建议。普通工程师只是抱怨，甚至全盘否定，平时工作里也是充满负能量。

优秀工程师每周会按时提交自己的工作报告，因为他们遵守纪律。普通工程师往往会忘记按时提交工作报告，因为他们不重视纪律。

- **与上司沟通**

笔者认识的很多程序员都有这样的误区：平时尽可能离经理远远的，要沟通也是等经理来找，而自己不会主动找经理沟通。根据一个著名互联网公司的人力资源报告，平均下来每个月经理和每个下属的沟通时间不超过 30 分钟，而对于优秀工程师和比较差的工程师沟通的时间更少。经理平时工作都很忙，如果你不找他，他基本上没时间找你沟通。为了和经理保持良好关系，你必须

定期花点时间和经理沟通。

沟通分为不定期沟通和例行会议。不定期沟通是为了解决某些特定的问题，通常也会伴随着一些决策的产生。当你工作中遇到瓶颈和困惑时，你需要和上司沟通一下，哪怕是短短的几分钟。当你觉得英语口语交流没法清晰表达自己想法，你可以通过 E-mail 方式来代替面谈。文字方式还有个好处是让你思维更精密，情绪更受控制，表达也更加清晰，同时也让经理对你要讲的内容有初步的认识，方便后续面对面的交流。E-mail 在硅谷公司里使用比即时通讯软件更普遍。每当你开发某些重要功能之后，通过 e-mail 给经理发送上线报告，同时抄送给全组。一方面，你可以从中记录和积累平时工作；另一方面，施加你的技术影响力，让经理和同事们知道你的付出和才能。上线报告应该包含所解决问题的背景、解决方案以及产生的效果。这些效果最好能以数据来说话，可以是性能指标、收益指标，同时配有图表，更加直观展示你的成果。平时也积极回复群里邮件，让大家知道你能干这个活，或者谁能干这个活。

例行会议有三类：一对一会议、部门会议以及项目总结会议。在多人会议（比如部门会议）之前，如果你觉得你的想法会和上司的相左，或者你有让上司吃惊的事情宣布，那么你最好提前和上司沟通一下，以免让他觉得意外，从而导致严重的后果。

一对一会议主要目的在于互通信息以及彼此学习，它也是维系双方从属关系最主要的方法。作为下属，你要准备一份一对一会议纲要，事先仔细考虑一下要提出来讨论的议题。除了让上司了解你的工作状况以外，你还得向上司寻求资源和帮助。上司本来就是提供资源的人。为了让一对一会议更有效率，你在会议中做笔记。这主要是为了让你在会议中集中精神，并消化你所吸收的信息；同时也让你的上司觉得你很重视。会后，你还需要发份会议纪要，让上司确认，以免双方理解有偏差。

➢ 绩效管理

绩效决定你的饭碗、奖金、加薪和升职。绩效如此重要，笔者强烈建议你在入职第一个星期内，必须和上司沟通你的绩效目标、评估方法以及反馈频率。

上司会根据你的工作、职称设定你的绩效目标。此外，你还得了解清楚上司对你的期望，方便上司后续以此来判断你的绩效是否符合期望。绩效评估对于任何一个管理者来说都不是简单的事情，大致包含两部分：评估下属的绩效，以及把结果告诉下属。评估绩效时会受很多因素干扰，比如第一印象、刻板的偏见、偶然事件、绩效目标不明确等，很难做到完全客观。只有通过平时积累，不断让上司知道你的产出，这样才能让上司排斥主观因素，做出合理的评估。除了产出，还有时间因素、长期和短期绩效、团队和项目因素，也应该纳入绩效评估的参考因子。

不要等到出了绩效评估结果之后，才找上司沟通绩效反馈，因为这时候结果已经是铁板钉钉了。平时的一对一会议是让上司反馈你绩效的最好时机，它可以让你实时调整工作方式。对于我们在硅谷工作的外国人来说，不可忽视的是身份维持问题：一旦你失去了工作，你的 H1B 身份很快会失效，到时候你得被迫打道回国。很不幸的是，不少硅谷 IT 公司，比如 Facebook 和微软，采取末尾淘汰制度，就像国内的百度和腾讯一样，定期淘汰低于预期的员工。绩效的结果一般采用打分制，有 5 分制和 3 分制。这两种打分制大同小异，对应有三档：优秀、符合预期、低于预期。正如橄榄球的两端，优秀和低于预期这两个高低档比例相对小，其中优秀档比例稍微比低于预期的大一些，而处于中间的符合预期档最多。如果你很不幸落在低于预期这档，你要么赶紧和上司商量补救方案，要么赶紧找到下家。

不管我们表现得多好，总还是有改进的余地。在绩效沟通的末尾，和上司一起寻找你成长的方案是一种明智的做法。

5.4 职业晋升

你的新职称很容易在面试过程或者 offer 上获得，但一般情况下，硅谷 IT 公司不会在 offer 里写明你的技术级别，你需要主动咨询 HR。一个职称拥有不止一个技术级别，比如微软高级工程师拥有 63 和 64 两个级别，这两个级别待遇和要求均不一样。入职以后，你将面临技术晋升（俗称“爬梯子”）各个方面的挑战。

- **职称和晋升**

软件工程师职称在硅谷 IT 公司里相通，彼此有对应关系。工程师级别有软件工程师、高级软件工程师、资深软件工程师、架构师和科学家。

软件工程师（Software Engineer）是一个入门级别职称，它的基本要求是在一定范围技术指导下可以完成一般难度的模块开发和调研工作，而且工程师能够对自己负责的业务有短期规划，思路合理、明确，并且有一定的产出。

高级软件工程师（Senior/Lead Software Engineer）的技术范围由一个中型方向扩展到一个乃至多个技术方向，在做技术规划和项目推进时能综合考虑技术方向内各因素，合理安排中长期工作，技术影响力也由中型方向扩展到技术方向。

资深软件工程师（Staff Software Engineer）能够完全把握一个技术方向。随着级别提升，对技术把握能力、技术影响力和迁移能力要求也会越来越高。技术迁移能力体现在已掌控的技术方向内证实了技术迁移能力，更快速地迁移到相关方向，并且掌握这个方向的重点和细节。

架构师（Architect/Principal Software Engineer）具有系统级分析和设计能力，重点在于完全把握若干个大的技术领域、技术深度充分，在某个大的技术

领域是专家，做出影响整个公司的成果和产出，具有不可替代的作用。技术把握能力更强，在大领域内能够完全把握，能够综合该领域所有技术方向给出技术决策以及长期规划，并且能够在该领域内进行前瞻性的研究、开拓、规划，对于非自己产品线的相关领域的问题，能够根据数据、介绍做出符合逻辑、科学的判断。

科学家（Scientist/Chef Architect）属于金字塔顶端，需要具备在业界世界范围内领先的成果和产出，并对公司有巨大帮助作为支撑。

大公司的职称评定经过三个阶段：申请，答辩和发布。申请时，你不但需要准备你的项目和产出，而且提供代码；提交申请后，你的上级审核并邀请更高级别同事对你评论。申请高级别的职称，需要参与述职答辩；评委由技术委员会指派，通常是你所在大部门的更高级别的工程师。正式发布之前，会有预发布流程，方便管理者和技术委员会沟通最终结果。对于小公司而言，相对随意一些，通常是管理者而不是技术委员会来决定你的晋升。

- **述职要素**

技术晋升一般有工作时间限制，一年以上不等，而且每年晋升的申请次数也有限制，大部分公司只有一次机会。因此，我们要把握好来之不易的晋升机会。晋升成功与否，不仅仅考察你在述职时的临时表现，还要求你注重平时积累，“滴水石穿非一日之功”。

“低头编码，抬头思考”，是程序员工作的常态。辅助记录积累的工具除了 Email，还有 Evernote、微软 One Note 等云笔记。每周记录你的产出，定期抽象和提炼，定期更新。如果你有机会挑选项目的话，项目的技术深度以及是否公司核心业务，这是两大首选因素。在公司重点推动的新业务里，员工晋升也会很快。如果你在发展相对平稳的项目上，技术重构是帮助你晋升的最佳利器，即“同样事情要用不一样方法来做”。平时注重演讲能力培养，多练习，多做报告。如果你有机会挑选上司的话，请选择愿意对员工职业规划提供资源的上

司，而不是把全部精力放在项目的交付以及对下属的安排主要是工作的分配，以及项目监督是否按时完成的上司。

述职答辩如同论文答辩一样，只是台下评委从老师换成不太熟悉你工作的同事。笔者根据自身当评委的经历，以及硅谷朋友的经验，提供以下临场发挥的建议。

- 表现自信：表述的声音洪亮，而且拥有语音语调、抑扬顿挫，抓住评委的注意力。我们知道在硅谷上班，着装都很随意，但是，在述职时候，笔者建议穿戴整洁和正式（比如休闲商务装），这表示你的重视和对评委的尊重。
- 熟悉内容：有意记忆述职内容，增加与评委的眼神交流次数。美国人看中交谈过程中的眼神交流，因此，你不能看着投影、照着 PPT 内容来读，应该面向评委，流利阐述你的工作，辅助肢体语言展现你个人的魅力。
- 抓住重点：在短短几十分钟时间内，你需要优先表述与你申请级别相匹配的重点项目。重点项目应该包含如下特征。
  a．项目与申请的级别相匹配。技术委员会对不同的技术级别期望候选人所表现出的能力及产出是不同的（详见“职称与晋升”章节）。虽然你参与过不少项目，对公司也有贡献，但是如果这些项目不足以支撑你所申请的级别，你的付出对于晋升来说就是白费。
  b．拥有业务支撑的从无到有的突破，或者由数据支撑的技术优化点。
- 差异化述职：技术述职也会区分不同方向，比如业务线路、算法、工程等。每个方向的侧重点有所不同，根据你所做的项目特点做差异化述职，就像田忌赛马一样胜出。
- 细说过程：每当我们挑选技术方案的时候，会花费很多时间去调研、评估、折中等。你需要把这个过程阐述出来，让评委知道你是如何思考、如何决策的。一个好的技术方案可能需要从多个方面综合考虑，比如开

发效率、阶段性收益、风险与规避、性能折中、内外部的工具选择等。

- 熟悉评委：在答辩之前，摸清你的评委背景，准备他们可能提问的问题答案。

在同事们述职过程中，笔者也看到了些述职的误区。

- 罗列项目，即重点不突出。大篇幅罗列你所做过的众多项目，而这些项目不足于支撑你申请的技术级别。此外，过多介绍非技术领域的能力，比如项目管理、团队管理、跨团队协调、培养新人成果等。如果非技术领域贡献比例过大，技术委员会无法判断你在技术方面上是否达到了申请的级别。
- 没法正确区分“我”与“我们”，即没有提及个人贡献，或者混淆个人和团队的贡献。虽然你参与了有挑战的重大项目，但是如果其中的个人贡献不好辨别，评委则很难判断你的能力。把团队的贡献说成自己的贡献，则是另一个极端，也是不可取。
- 无法面对评委质疑。笔者遇到一个案例，当评委质疑申请人的技术方案是否考虑完备时，举了一个 badcase，而申请人辩称这是小概率事件。在实际工作过程中，系统崩溃通常是由一个小概率事件触发。
- 可度量及可比性不强。当前技术分工越来越细，新技术层出不穷，因此评委无法真正了解候选者的工作和领域。这时候，如果你能提供更详细的可度量的数据和产出，同时通过对比你的项目和公司内部的参考项目，就能帮助评委来理解你的工作。

➢ 角色转换

继续走技术路线还是转行走管理路线？这是一个经久不衰的话题。介于担当硅谷公司高管的华人凤毛麟角的窘迫形势，笔者鼓励大家勇敢转型管理，多多提携同胞。

管理者和工程师最大区别就是职责不同，管理者的工作是达成业绩，提升整个团队的产出，而不再是单打独斗，自扫门前雪。此外，管理者还负责建设团队和培养下属。从技术岗位到管理者的转变过程中，我们需要快速完成的几项能力：发展人际关系、自我认知、分工与协调和处理压力等。

第二部分　实战访谈

在第二部分里，我们采访了一些已经在美国工作和即将去美国的工程师。这些先行者详细介绍了他们的出国途径、面试技巧、硅谷的企业文化、职业规划以及当前热点等经历。

第 6 章 对身在美国和即将赴美工作的工程师访谈

互联网资深大牛董飞

董飞，全球最大的在线教育平台 Coursera 的工程师；知乎达人，开辟有“董老师在硅谷”专栏；本科就读于南开大学，硕士就读于美国杜克大学，曾先后就职于酷讯、百度、亚马逊和 Linkedin 等企业。他是 InfoQ 中文站的作者，还是 QCon（全球软件开发大会）的讲师，目前关注创业趋势、大数据和在线教育等领域。以下是 2015 年 7 月 InfoQ 中文站对董老师的采访内容。经董老师授权，我们做了些删减。

问：您先后就职于众多知名公司，在此期间一直从事大数据相关的工作吗？能简单介绍下么？

一直看好董老师：自从 2010 年开始，大数据就一直被看好，我的工作主线和数据处理、大数据架构相关。几年前在国内做一些数据的抽取挖掘工作，

当时从事的百度云计算项目，BaiduApp Engine，后来演化成百度的移动云，包括百度云盘。再后来去美国读书就针对性地对 Hadoop 这个技术浪潮做了深入关注和分析。之后就职于亚马逊的云计算部门，在 LinkedIn 做广告分析，以及在 Coursera 做数据的架构。

问：你如何评价大数据在过去几年的发展？

董老师：大数据的飞速发展得益于开源技术的普及。在十多年前，Oracle，IBM 等大公司也希望搭建高可靠、分布式系统，但当时技术集中在少数几个公司，他们只提供一些很高端的服务。

IBM 是一个特别有意思的公司，他们在技术推广和社区建设方面贡献很大，特别是开源技术。我从他们的社区里学习了很多。开源软件帮助大数据技术的传播。随着软、硬件的升级，互联网公司在很多应用的领域，比如社交、广告，对数据和实时性的需求量都在一个特别大的升级过程中，所以技术就必须赶紧跟上。这当中又体现出一些技术的进步，根据摩尔定律，内存、硬件的成本都在不断降低，SSD 硬盘的速度在不断提升，这都给之前看似不可能的任务提供了机会。所谓的时势造英雄，硬件的升级和成本的降低，给大家创造了很好的机会。

问：很多大公司在做大数据，很多创业公司也在为大数据服务，你觉得未来大数据会变成一种服务吗？有必要自己建立一套还是用第三方的？

董老师：让我来做大胆预测的话，数据架构的门槛可能会越来越低，低到就像现在的云计算，已经成为一个非常普及的渠道，像水电一样方便获取。

对创业公司来说，它若需要后端存储、计算、消息中间件等服务，就可用现成的服务。我已经看到一些公司，比如说 LeanCloud，他们降低和解决了其他一些创业公司的技术门槛。

如果自己建立一套服务，在美国，像 Google 出来的人，他们就喜欢搭建

自己的平台。Google 本身就有一套非常完备的从部署到开发到云的服务，他们看到这些好的服务就有一种依赖性，希望把这服务做一个翻版和简化。这个过程还需要几年的操作时间。一些大公司因为他们自身大的流量和历史的负担很难接受第三方，更愿意自己开发维护。还有一种可能是做一个技术平台，比如百度、阿里他们也有很多开源的技术方案。

问：您在国内工作过，后来又出国，现在在硅谷，如今国内的创业氛围很浓，跟硅谷对比有哪些区别？

董老师：硅谷这三十年的发展是一代胜一代的递进发展。我能感受到他们非常重视人才培养，从小注重动手能力，包括软件工具、硬件工具。学校教育风格就是如此。现在硅谷的趋势也是软硬件相结合。硅谷的另一个特点是注重原生的创新能力，不屑于抄袭。如果有已经很好的产品，就不会再做相同或类似的，他们要在某些领域上有创新。所以硅谷在生态上更友好、更强调服务化。西方国家很愿意为体验、为服务买单，他们认为因付出服务而收费是理所当然的。

由于国内在早期缺乏保护知识产权意识，对盗版的问题没有处理好，开发者在早期没有地位，也没有商业变现途径，导致后来很难做大软件公司。但反过来，也有一些公司提供免费入口的模式获得不错的商业机会。

问：对于创业公司来说，如何找到并留住合适的人才？

董老师：优秀的技术人才在美国有很多选择机会。如果一个公司走失了几个核心人才，公司会不会垮了呢？不会。一方面，公司在制度上留得住人，有期权，也就是金手套。如果公司发展得好，员工的股票收益会大于薪水。另一方面，公司安排员工每隔一段时间轮换着承担一个服务，服务出了问题都要你负责，团队轮岗一遍后，一旦团队有人出走其他人都可以顶上。这不但对人的技能提升有好处，而且避免了因人员流动导致的工作中断。

公司还提供良好的氛围，比如说技术分享，开展读书会，让员工感到公司

不是要压榨员工的价值，而是让他们更好地提升发展。《Alignment》一书中提到，有些公司如 Google，人才走了可以保留一到两年职位，欢迎随时回来。当然，创业公司的未知风险很大，要给人才看到公司成长的空间，还有企业领导人的魅力。

问：你对国内在线教育的市场现状怎么看待？跟国外在线教育相比，差距在哪？

董老师：国内做法和关注点与国外很不一样，投资人更喜欢家教、O2O，因为这些更市场化，能够更快地看到变现。在美国，教育也是很大的市场，但教育阻力同样也很大。私立公立学校各有各的利益集团。很多大学也做了在线平台，学校不会把所有资源放上去，而是先放一些浅显课程。教育的前景，不是由投资人简单决定的。教育应该慢慢培养，慢慢成长，不应该短期爆发，教育有巨大社会贡献。

中美两国的市场都很大，中国更侧重短期效应，希望我们在摸索过程中能够探索出一些新的东西。我看到的趋势是个性化、定制化、自适应和场景化教学。

问：大数据在在线教育中具体是如何应用的？价值是什么？可否提供用户画像（又称用户角色，作为一种勾画目标用户、联系用户诉求与设计方向的有效工具）或其他服务？

董老师：用户画像希望把用户分析全面，维度很高且稀疏。在线教育的用户，针对不同的课程，要有一个精准化的推荐，可以根据他的年龄、地理信息和教育水平，这些都跟年龄层相关。比如一个人工作了三四年，想要升职，可能学习管理类课程，培养管理能力，我们应该怎么做推荐？一个 IT 专业学生刚毕业，我们又应该怎么做推荐？我们针对具体人群对其进行精准化的定位，也要考虑变现能力，针对有一定经济收入的群体，结合强烈动机，将其转化为收入的来源。

问：很多专注于技术的年轻人想学东西但是不知道从何入手，不知道从哪里找到合适的学习资源、学习方法和路径，对于自己职业生涯的规划没有概念，你能否根据自身的经历给年轻人一些建议？

董老师：拿我自己举例，为什么会加入 LinkedIn，首先自己是它的忠实用户，其次在这里可以看到很多人的简历，当你找到一个坐标，了解自己想做的那一类人，就可以看到他们的学习模式，以及他们是怎么成长的。除了少数天才，很多人都是从底层做起的，3~10 年之后，才到达一个较高的境界。

当然，没有哪个人的路是被别人所定义和指导的，路是自己走出来的，我们要找到自己合适的路。我们相信突破性、颠覆性的东西是存在的。另外，现代人强调专业化，你要找到自己的方向，找到自己擅长又感兴趣的地方做下去，你要相信自己能够做好，还要坚持。一万小时定律表明，你一直在这个垂直领域做下去，很可能就成了这个领域的顶尖人物。

问：非计算机专业想入行大数据领域，门槛会不会很高？

董老师：个人觉得在全民创新创业的年代，很多岗位肯定缺人，尤其是技术研发人员，还有一部分是产品经理，相对来说技术研发人员的需求远远大于其他工种的需求。你只要愿意去钻研一些热门的技术，就会有非常大的成长空间。有些人不是学技术出身，也没关系。如果你能踏踏实实地落实到一行行的代码，比高谈阔论一些趋势反而更有意义。

未来的技术人才有一个非常大的短缺，中国未来 5 年可能会有比硅谷更大的技术人才需求，这些大多都是技术开发者，对素质的要求较高。认真做事积累非常重要，大多数人都不是天才，需要投入时间精力去学专业技能。

问：大数据入门者想提高水平，请董老师推荐一下学习路径。

董老师：这跟中国教育有关。我受过中美两方面教育。中国教育太基础，很多课程非常呆板，完全是教科书式的。美国的教育方式经常是讨论式、实验

式的，每堂课都有关于课程的项目。比如 NBA 球探，每一场球赛，数据化建模完全反应出来。每行每业都有很好的实际结合点，初学者如果从一开始就学建设模型和数字化实验，就可以更好地训练理性逻辑思维。

问：从个人梦想来说，将来最想做或者现在已经开始做的一件事情是？

董老师：在美国，很多人过了温饱阶段，他们都过着很简朴、平常的生活，追求改变世界实现梦想。一直强调不忘初心，这是他们内心当中最认可的事情。

确实，从短期来看这样做可能需要取舍甚至牺牲，但要看到更多美好的东西，我们先确保证身心健康，路还长着，不要有暴发户的心态，定长期一点的目标。举个例子，我曾接到一个面试是让我画一幅画，描绘未来十年理想的生活。通过这个任务来激发你对未来的一个画卷设想，想象力非常重要，不能局限于现在。也许你现在还会迷茫和不满，但为了长远的美好目标，要相信自己的努力，幸运是偏好勇者的。

创业者徐淼华

徐淼华，在中国科学技术大学少年班本科和计算机系硕士毕业后，前往佛罗里达国际大学攻读博士学位，中途退学参加工作。曾经任职于微软总部，目前在西雅图创业。

问：我们知道您是赴美留学后留下工作，对于这条赴美工作的路径有什么看法么？

淼华：我认为工程师要是走学术路线的话，留学就比较有必要。如果想进入工业圈，留学作用相对就没有那么大，国内工作直接出国挺好的。

问：我们知道以留学生身份出来的华人在美国创业很不容易，您能详细给

我们介绍您的创业历程么？

淼华：我是一个不安分的人。做过很多创业的尝试，写过共享软件，搭建过把国内产品放在 eBay 销售的网上自动平台，做过 eBay 用户信用评估，还曾经直接淘商家的热门货或者打折货在各处销售，现在做面向美国拥有优秀信用的个人提供合理利用信用度的咨询和服务的网站（http://rewards2cash.com），另外还有一些想法在筹划中。开始创业的时候只能利用全职工作的业余时间，非常辛苦，但是有一种与为大公司工作（或者说为别人工作）完全不一样的心情和感觉。我认为创业就是要多尝试、多了解，找到客户真正的问题（也就是 pain point）和解决的办法，那样创业就成功一半了。

问：对于我们这些外国人来说，是应聘大公司，还是创业公司？各有什么利弊？

淼华：大公司稳定，这对刚出国的人很重要，尤其是身份问题。

问：当前国内有大量程序员打算出国工作，您有什么建议可以给他们吗？

淼华：他们需要适应美国的语言、文化和面试方式。就面试而言，推荐《进军硅谷——程序员面试揭秘》（编者注：该书为本书第一版）。这本书可以帮助面试者熟悉美国公司的招聘方式，提高面试技巧。

问：对富有经验的国内程序员，您有什么建议可以给他们？

淼华：鉴于国外对国内工作的了解未必很清楚，所以得好好准备自身明确又有说服力的工作经验，同时做好技术面试的准备。

问：国内的软件工程师可以通过哪些渠道去获取硅谷公司招聘信息？他们该如何准备面试？

淼华：国外的招聘网站有 indeed、glassdoor、monster 等。准备面试得总结自己的工作经验，熟悉常见的面试题目。比如刚才提到的《进军硅谷——程序

员面试揭秘》就很有帮助。

问：听说研发工程师的岗位竞争非常激烈，您觉得硅谷哪些职位比较容易应聘成功？另外，哪些技能或者特质能够使应聘者更容易得到美国公司招聘人员的青睐？

淼华：现在硅谷的求职市场还是很火热的。学习速度快、适应能力强的应聘者在哪里都受欢迎。

问：您觉得微软更看重面试候选者的哪些能力？

淼华：我在参加面试的时候感觉微软团队最看重技术能力，即用技术解决问题的能力。美国公司的确很注重团队合作和个人行为方面的问题，但是工作归根结底是帮公司完成任务，如果申请工程师一职，技术永远是第一位（当然申请管理岗位就不一样了）。

问：求职者应该如何选择硅谷的 IT 公司？在决策的过程中需要考虑哪些因素？薪酬、企业文化还是晋升通道？

淼华：薪酬、企业文化和晋升通道都很重要。第一次出国工作的朋友们还需要考虑身份问题，最好是稳定的大公司。其次，要考虑企业的成长预期，企业成长快，机会多，工作获得的回报也更丰厚。

问：您觉得应该如何跟国外公司的 HR 谈 Offer？跟国内的公司有什么不同吗？

淼华：我没有在国内工作过。个人觉得谈 Offer 最容易的情况就是有别家的 Offer 做标杆。

问：基于您多年的美国生活经历，您对那些刚踏入美国或者准备踏入美国的新移民有什么建议？

淼华：开车要小心。特别是转弯让直行，美国执行得非常严格。

问：对于那些已经结婚生子的国内程序员而言，尝试出国工作，有哪些需要注意的事项？国外的生活，对家庭有怎样的影响？

淼华：出国要尽早。因为小孩越小越容易适应新环境。

问：您觉得硅谷下一代的互联网技术浪潮会是什么？对世界带来什么样的改变和影响？

淼华：互联网在各种传统行业的运用，使闲置资源得到有效的利用。Uber 减少了空车，Airbnb 利用了空房，类似的应用在各行各业都会出现，资源的使用会更高效和更优化。

留美计算机博士张喆

张喆，留美博士，毕业后在美国的一个国家实验室做研究员。之后在 IBM 研究院工作了 3 年多，现在是 Cloudera 的工程师。

问：我们知道您是先到美国留学，然后留下工作，能具体介绍一下么？

张喆：我们那个年代，美国公司直接到中国招人的机会还不是很多。所以大部分来美国工作的工程师都是先留学再工作。现在有不少人直接从国内来美国工作。先留学的话（一般是读研究生），会多花几年时间，但是校园里的经验对求职者日后的美国生活还是有一些帮助的。特别是要从事和学术界联系相对紧密的方向，比如分布式系统，就要求工程师对行业新方向感觉敏锐。

问：作为互联网教育行业的领头羊，Cloudera 公司更看重面试候选者的哪些能力？

张喆：Cloudera 的面试风格和谷歌、脸书不太一样。我们没有常规招聘（general hire），每个组都有针对性地招人。大部分的组做企业软件的后台，所

以很看重操作系统、分布式计算的基本功。当然也看重编程的基本功，但是在算法上不会太难。

问：您加入 Cloudera 这些年来，对于 Cloudera 的企业文化、管理风格，最大的感受是什么？与之前在 IBM 研究院工作的氛围有什么不同吗？

张喆：Cloudera 是一家 Pre-IPO 公司，比 IBM 的规模小很多。最大的一个感受就是会议少了很多，80%的时间都在干有用的事情。在 IBM 的时候大概有一半以上的时间都在计划、开会。Cloudera 的管理更直接，比如申请经费、请人来做报告都很容易。

问：当前国内有大量程序员打算出国工作，您有什么建议可以给他们吗？

张喆：有认识的人推荐还是挺重要的。尤其是小公司，推荐的比例很高。另外，如果能参与开源社区的开发，那就更好。

问：国内的软件工程师可以通过哪些渠道去获取硅谷公司招聘信息？他们应该如何准备面试？

张喆：如果你没有认识的人推荐，就去 Linkedin 和推特上关注一些感兴趣的公司，或者直接去公司的 jobs（招聘）网站。

问：求职者应该如何选择硅谷的 IT 公司？在决策的过程中需要考虑哪些因素？薪酬、企业文化还是晋升通道？

张喆：如果你比较年轻，我建议找技术上有挑战的项目。等技术成长到了一定程度，可以供你选择的职业范围就更大了。比如可以去初创期的公司独立负责一块业务。我说的这些比较适用于做后端系统的程序员。现在很多社交网站，如 Uber、Airbnb 这些应用招很多前端，或者全栈工程师，这方面我不是很了解。

问：您觉得应该如何与国外公司的 HR 谈 Offer？跟国内的公司有什么不

同吗？

张喆：我觉得要求可以直接提，大不了对方说不能满足。选择与 HR 谈，就不会对将来的同事或经理造成尴尬。尽量多拿几个 Offer，这样谈判就有分量了。

问：在 Cloudera 这类型创业公司工作，您觉得具备哪些特质更容易获得晋升的机会呢？

张喆：最重要的还是加入要早、要主动、要做高影响力的项目。

问：基于您多年的美国生活经历，您对那些刚踏入美国或者准备踏入的新移民有什么建议？

张喆：生活上没什么需要特别注意的，现在硅谷的生活跟国内都挺接轨。住房比较贵，建议个人做好财务规划。

问：对于那些已经结婚生子的国内程序员而言，尝试出国工作，有哪些需要注意的事项？国外的生活，对家庭有怎样的影响？

张喆：虽然这几年有些改变，但美国的主流还是夫妻一人工作（尤其是有孩子的时候）。所以要做好这个心理准备。如果另一半想工作，现在 H4（配偶签证）很方便。

问：您觉得硅谷下一代的互联网技术浪潮是什么？会对世界带来什么样的改变和影响？

张喆：根据 Gartner 的“技术周期”报告（http://www.gartner.com/newsroom/id/ 2819918），大数据技术将要步入成熟期，之后的浪潮我觉得在物联网（IoT）方向。

微软软件工程师乔成

乔成，本科毕业生于复旦大学通信工程专业。在微软上海部门工作多年之后，转岗到微软西雅图总部。

问：我们知道你是通过 L1 来美国工作的，能够给我们简单介绍一下吗？对于这条来美路径的利弊你有什么看法？

乔成：L1 签证是在美国公司海外分支工作超过一年以上的人才可以申请的签证，和 H1B 一样是工作签证。L1 签证分 L1A 和 L1B 两种。前者针对公司高管或管理层，后者针对普通的工程师。它的优点：不需要抽签，一般只要公司支持申请都可以拿到。L1 签证的配偶拿的是 L2 签证，可以在美国工作。它的缺点：签证只有在支持你申请 L1 签证的公司工作期间有效，不能跳槽。L1 最长期限是 5 年，比 H1B 少一年。

问：听说研发工程师的岗位竞争非常激烈，你觉得美国 IT 行业哪些职位比较容易应聘？另外哪些技能或者特质能够使我们更容易得到公司招聘人员的青睐？

乔成：IT 行业的领域很宽泛，具体到每个细分的领域情况都不一样。就拿时下最热门的互联网领域而言，开发工程师无疑是需求最多的职位，相应的机会也很多。有和招聘职位相关的工作经验，熟悉这个领域内的工具、术语，能和面试官就不同方案的优劣进行讨论的工程师无疑最受青睐。

问：对于我们这些外国人来说，是应聘大公司，还是创业公司？各有什么利弊？

乔成：这由个人的喜好决定。大公司工作稳定，福利好，另外在申请绿卡上面也比较有优势。创业公司发展空间大，如果上市，有一夜暴富的可能。

问：对于那些已经结婚生子的国内程序员而言，打算出国工作，有哪些需要注意的事项？国外的生活，对家庭有怎样的影响？

乔成：这需要慎重考虑。如果有可能，最好先花几个月时间带家人来体验一下国外生活。同时要对第一年可能遇到的困难和不适应做好心理准备。

问：当前国内有大量 IT 从业人员打算出国工作，您有什么建议可以给他们吗？

乔成：美国的月亮未必圆，现在国内的机会更多，而且不会面临在美国需要面对的语言劣势，文化差异等情况。如果确实有这方面打算，可以在国内的外企先尝试一下，多一些机会到国外出差体验，同时建立一些关系。如果技术不是问题，那么需要提高的就是英语和沟通技巧，不管你将来是不是想走管理路线。

问：国内的软件工程师可以通过哪些渠道去获取硅谷公司招聘信息？他们应该如何准备面试？

乔成：刷 LeetCode 的题是目前必不可少的准备，还可以看 TechCrunch 及类似的网站。或者在 LinkedIn 上向猎头或你心仪公司的招聘专员毛遂自荐。

问：求职者应该如何选择硅谷的 IT 公司？在决策的过程中需要考虑哪些因素？薪酬、企业文化还是晋升通道？

乔成：如果是大公司，就要看具体的部门。有三个条件，排序如下：好老板 > 好团队 > 好项目。

问：对于美国公司的企业文化、管理风格，您最大的感受是什么？与之前在国内工作的氛围有什么不同？

乔成：不要以为美国没有办公室政治，官僚主义是大公司的通病，不管是微软还是谷歌。现在在美国总部，感受比在外企的中国分部更加明显。

问：您觉得在美国公司工作，具备哪些特质更容易获得晋升？

乔成：善于思考，能抓住重点，对日常工作能提出改进的意见，善于沟通，与人合作，能说服别人，带新人等。能推动项目前进，未必需要自己写代码。

Broadcom 硬件测试工程师蒋波韡

蒋波韡，复旦微电子硕士毕业，之后在 Broadcom 上海分公司从事芯片测试。工作一年后从公司内部转岗来到 Broadcom 加州总部。

问：对于我们这些外国人来说，是应聘大公司，还是创业公司？各有什么利弊？

蒋波韡：大公司相对稳定，如果应聘创业公司，在经济上升周期也许可以获得巨大的杠杆，抓住一个能 IPO 或者被买下的机会，不说一辈子衣食无忧，也抵得上在大公司工作数十年。

但是创业公司的机会与经济周期息息相关。美国是一个很成熟的市场，经济周期始于美联储在经济萧条时候的大规模刺激或者降息，制造的流动性依次进入债券、股市、实体经济，直至影响到整个社会商品，明显通货膨胀，经济过热。然后加息，回收流动性，最后经济回落。在这个周期里面，通过大量的创业公司，流动性从股市到实体经济完成转移。在美国硅谷，资本和劳动力这两大生产要素迅速向未来新技术、新生产力的创业公司集中，高效地把这些新技术从无到有、从小到大的做起来。不好的地方就是会不可避免产生泡沫。当人人都觉得创业可以赚大钱，所有在 top 领域的人都很兴奋的时候，往往正处在曲线的至高点。从过去 100 年的数据看，IPO 数量最多的时候往往是股市的最高点，圈钱最容易，各大 VC 也视风险于无物。但是乐极生悲，接下来 6 个月到 1 年的时间，股票往往开始熊市，经济随后陷入衰退，创业则到了寒冬期

(杠杆效益显著，收益大也可能损失很大)。

所以选择创业的最好时机是经济开始复苏的时候，这个时候公司对股票也相对慷慨，每年的成长率惊人。几年后，等大量公司争相 IPO 的时候就是创业者享受胜利果实之时。

问：当前国内有大量 IT 从业人员打算出国工作，您有什么建议可以给他们?

蒋波韡：国内近年来发展相当迅速，就薪水和技术水平的方面和美国相比差距已经不太大了。如果你还年经，从体会不同生活、增加自己视野的角度来看，可以来美国尝试。如果你已经在国内事业有成，朋友人脉都很广，到另外一个地方从头开始也是一件值得考虑的事情。目前技术的领导者可能还是在硅谷，但是我要提醒大家一句话，不要把美国想得太美好。

问：国内的软件工程师可以通过哪些渠道去获取硅谷公司招聘信息？他们应该如何准备面试?

蒋波韡：Mitbbs（笔者注：mitbbs.com 是一个海外留学生论坛）的求职版块有很多面经。求职者把过去一年的面经大概看一遍，包括从各大公司到热启动类型公司。基本上就都有数了。还可以找人参考或者找招聘的信息。网上直接投简历的效率比较低，通过 Linkedin 或者论坛上找联系人的途径，命中率要高很多。

问：求职者应该如何选择硅谷的 IT 公司？在决策的过程中需要考虑哪些因素？薪酬、企业文化还是晋升通道?

蒋波韡：我觉得薪水和企业文化是比较重要的因素。不同公司之间的企业文化差别很大。比如 Facebook 喜欢开放、自由交流和比较扁平化的管理，他们偏向于先把东西做出来，再慢慢优化。Apple 的组与组之间就比较封闭，很多产品信息那使在公司内部也不能相互分享。如果企业文化与个人的性格不匹

配，工作肯定不开心。企业文化观点可能比较宽泛，直接关系的是你所在的组和直接上司。上司是喜欢放手让员工去做，还是更喜欢微管理，当你和其他组有争论的时候，上司是为员工挺身而出还是让你承担责任。这些情况可能和国内差不多。一般初创公司氛围比较好，大家工作有热情(因为有共同奋斗的前景)。大公司的政治因素很多，人多的地方就是江湖。

至于晋升通道，对华人来讲普遍存在玻璃天花板的问题。个人觉得主要是语言和文化的原因。当然还是有不少华人，通过付出更多的努力和斗争经验，冲破天花板做到高管。但就目前的情况来看，高管主要是白人和印度裔。

问：基于您多年的美国生活经历，您对那些刚踏入美国或者准备踏入的新移民有什么建议？

蒋波韡：刚来美国觉得中美差别很大，待久了，就觉得人跟人都差不多，共性是大部分的，差异是小部分的。适应新生活没有想象中的那么难。

问：您觉得应该如何跟国外公司的 HR 谈 Offer？跟国内的公司有什么不同吗？

蒋波韡：Offer 是一定要谈的，HR 在发 Offer 的时候都会在薪资上留有讨价还价的余地，这时候你可以多争取一些薪水，不过你最大的筹码是同时拿多个 Offer，这样通过的概率会比原来高很多。毕竟公司给了 Offer 都希望你来，此时权利反转，主动权已经在你的手里路。

问：对于美国公司的企业文化、管理风格，您最大的感受是什么？与之前在国内工作的氛围有什么不同？

蒋波韡：老板跟下属的关系更加平等（当然这只是表面上的，内心不要真的这么认为）。

美国文化更加重视家庭。有时候我觉得，我们中国人的文化过于强调奋斗

和拼搏。人一生的精力和时间就是你的资源，资源是稀缺的，怎么分配自己的时间和精力来达到最优解值得我们深思熟虑。我们把大量的资源投入在职业奋斗上，因为它见效快，很容易获得正面反馈。但是经济学基本原理是边际效益递减，当你的生活从温饱、小康到富裕，再继续投入，你能获得的快乐，可能更多的只是一个数字上的满足感，并未给你的人生带来实质改变。花时间跟家人在一起，游玩，经营夫妻感情，陪小孩子玩耍，这些感情投资也许要付出几十年时间，但老了之后家庭和睦，子女孝顺比什么都强。如果一直以来的投入都很少，对这方面的倾斜，边际产出则更多，也许“抽”时间干的事情，才是真正值得你做的。

硅谷初创公司大数据处理软件工程师常新宇

常新宇，美国 Kent 州立大学博士，现在硅谷一家初创大数据处理公司从事软件工程师的工作。

问：听说研发工程师的岗位竞争非常激烈，你觉得在美国 IT 行业哪些职位比较容易应聘？另外哪些技能或者特质能够使我们更容易得到公司招聘人员的青睐？

常新宇：目前大多数公司对 Web 应用和手机 APP 的开发需求很大，在招聘的职位描述中，经常可以看到 PHP、.NET、安卓等需求。对于大公司的应聘，应聘者的背景和算法的水平则比较被看重。

问：对于我们这些外国人来说，是应聘大公司，还是创业公司？各有什么利弊？

常新宇：确实，这两种选择各有利弊。大公司不怎么需要加班，有更高的薪水和更多的假期，并且在大公司的工作经验也会让你的简历看上去更有吸引

力。当然，缺点就是如果分到不重要的组，得不到个人成长，工作也没有乐趣，另外就是升职困难。

创业公司则正好相反。员工往往需要独当一面，有更多个人成长的机会。而且公司对初创员工也更加重视并给予更多表现的机会。当然最重要的就是对于公司上市的渴望和干劲。美中不足是经常需要加班，比较辛苦。另外公司也存在倒闭的风险，工作稳定性不如大公司。

问：对于那些已经结婚生子的国内程序员而言，尝试出国工作，有哪些需要注意的事项？ 国外的生活，对家庭有怎样的影响？

常新宇：出国在外，可以为下一代创造更好的条件。孩子在英语环境下接触美国的文化和教育，从而可以更好的留在美国生活，避开国内万人过独木桥的竞争。在这点上对于喜欢美国生活的家庭是非常值得尝试的。因为在美国付出同样的辛苦和所能得到的生活的性价比更加诱人。让下一代也生活在这样环境中的做法是国内很多人求之不得的。即使几年后回国，孩子们的英语能力已经不需要担心了，这是在国内花多少钱也买不来的。

然而整个家庭搬过来，假设这几年就职者的配偶并没有工作的身份，不能工作，再加上文化和语言上的不适应，难免会面对很大的压力和孤独。再者，异国的环境对于老人而言更是不容易适应，如果老人要子女照应，也是一个问题。所以长期工作的话，去留是个不得不慎重考虑的问题。

问：当前国内有大量 IT 从业人员打算出国工作，您有什么建议可以给他们？

常新宇：在美国一定要有医疗保险，美国没有保险看病动辄几千美元，拔一次牙比往返机票都贵。

问：国内的软件工程师可以通过哪些渠道去获取硅谷公司招聘信息？他们应该如何准备面试？

常新宇：硅谷大公司的面经在网上都可以查到，算法题在很多练习算法的网站上面也都可以找到，有些公司甚至有自己固定的题库，更甚至有些公司还会建议你去算法的网站上练习练习再来面试。然而我觉得在众多有备而来的应试者中，个人的情商也很重要，让面试官觉得愿意和你一起工作的技能也是非常重要的。我也听说有些论坛和 QQ 群中，会有很多相关的讨论，甚至会有大公司的人出来做内推。

问：基于您多年的美国生活经历，您对那些刚踏入美国或者准备踏入的新移民有什么建议？

常新宇：在来到美国之前，大家可以通过当地的华人论坛或者 QQ 群来认识一些人，一般公司都会安排住处，但是刚到美国的时候一定会有很多的难处，有很多需要人帮助的地方。所以最好在来美国之前先通过互联网认识一些朋友。刚来的时候如果能够得到当地华人的帮助，可以省去非常多的麻烦。

美国基本什么都有卖，硅谷地区更是如此。所以不必按照有些网站上说的什么都带，我个人认为值得带的东西是在国内常用的药品。因为刚到美国如果需要买药未必能描述清楚。

第三部分　算法面试

在第三部分章节里，我们重点阐述编程面试题所涉及的数据结构，以及常用的算法。

在本书中，每道面试题的标题后面均附上了星号标示的难易度，即一颗星代表容易，三颗星代表中等，五颗星代表非常难。

第 7 章 俩指针

在比较数组或链表元素时，通常需要两个指针遍历整个数组或链表。这两个指针或者在数组的同一端做同向移动，或者在数组的两端做相向移动，而且移动速度可能不一致。通过两个指针解决比较、移动、挑选数组或者链表元素的方法，我们称之为“俩指针”。

面试题 1：两数之和 I ☆☆

给定一个整型数组，是否能找出其中的两个数使其和为某个指定的值？

提问

应聘者：输入数组是有序的吗？

面试官：你可以假定是无序的。

举例

输入数组为{1，5，7，3}以及指定的目标值为 10，我们可以从中找出两个数 3 和 7，和为 10。

一种非常直观的办法就是使用两个循环，从数组里提取一个数，然后在该数之后的部分找出另外一个数，计算这两个数之和，看看是否等于指定的值。这种暴力破解的方法显然不是面试官想要的。那么，能否降低暴力破解 O(n2)的时间复杂度呢？可以尝试先把该数组排序，排序之后，从首尾两端移动，一次移动一端的指针，直至相遇或者找出两个数的和为指定的值为止。假设当前首尾指针分别为 i 和 j，其中 i<j，如果 A[i]与 A[j]之和大于指定的值，那么要找的两个数一定在 j 的左侧，因此，尾指针 j 往前移动一步。如果 A[i]与 A[j]之和小于指定的值，那么要找的两个数一定在 i 的右侧，所以首指针 i 往后移动一步。

如何证明如上的解法是正确的？反证法。假设 A[i]与 A[j]之和大于指定的值，并且有个 k，k 大于 j，使得 A[i]+A[k]为指定的值。由于 A[j]<A[k]，并且 A[i]与 A[j]之和大于指定的值，那么 A[i]+A[k]必定大于指定的值，所以不可能有 k，而且 k 在 j 的后面，使得 A[i]+A[k]为指定的值。

```
boolean hasSum(int[] A, int target){
  boolean res=false;
  if(A==null || A.length<2) return res;
Arrays.sort(A);
  int i=0, j=A.length-1;
  while(i<j){
      if(A[i]+A[j] == target){
         res=true;
         break;
      }else if(A[i]+A[j] > target){
         //目标值过小，则向前移动尾部指针，减小两数和
         j--;
      }else{
```

```
//目标值过大，则向后移动首部指针，增加两数和
i++;
}
}
return res;
}
```

通过排序，使得时间复杂度降至 O(nlogn)。在 while 循环里，至多扫描一遍数组就可以得出结果，所以最终程序的时间复杂度为 O(nlogn)，n 为数组的长度。

扩展问题

如果允许使用额外的存储空间，能否继续降低时间复杂度？

面试题 2：两数之和 II ☆☆☆☆

设计一个类，包含如下两个成员函数。

Save(int input)

插入一个整数到一个整数集合里。

Test(int target)

检验是否存在两个数和为输入值。如果存在这两个数，则返回 true；否则返回 false。

提问

问题：在这个整数集合中是否允许有相同值的元素？

面试官：允许。

对于这道面试题，我们必须注意整数集合里的数可能有重复。如果 Test 函数的输入值 target 为集合里某个数的两倍，我们该如何判断？光在哈希表里

存放<值，下标>还不能满足需求，而需要改变为记录每个数出现的次数，在哈希表里存放<值，个数>。如果某数两倍为 target 值，那么只有该数出现两次或两次以上，才能返回 true。

```
    public class TwoSum {
       HashMap<Integer, Integer> hm =new HashMap<Integer,
Integer>();
       public void save(int input){
           int originalCount = 0;
           if(hm.containsKey(input)){
            //判断是否已经存在，如果存在，则读取个数
    originalCount = hm.get(input);
           }
           hm.put(input, originalCount + 1);
       }

       public boolean test(int target){
         Iterator<Integer> it = hm.keySet().iterator();
         while(it.hasNext()){
              int val = it.next();
              if(hm.containsKey(target - val)){
                 //需要判断一下是否有这种特殊情况
                 boolean isDouble = target == val * 2;
                 if(!(isDouble && hm.get(val) == 1))
                    return true;
              }
         }
         return false;
       }
    }
```

数组是一种最常见的数据结构，并且在众多面试题中占很大的比例。有关数组的面试题解法繁多，但通常归纳为如下几种：排序、俩指针、动态规划、排列组合等。对于二维数组的遍历，还可以考虑深度优先遍历或者广度优先遍历的方法。

面试题 3：Top K☆☆☆

求一维数组中最小的 k 个数。

提问

在给出答案之前，需要和面试官沟通，明确如下几点。

- 机器内存是否能容纳整个数组？
- 最小的 k 个数是否要求有序？
- 能否修改输入数组？
- 时间复杂度的要求：是 O(nlogn)，还是 O(n)？

方案一：排序

先把数组从小到大进行排序，取前 k 个数。时间复杂度为 O(nlogn)。如果数组过大，机器内存无法同时容纳整个数组，则需要使用外部排序。

方案二：使用堆

（1）创建一个最小堆，初始化大小为 k；堆顶为堆的最大元素。

（2）扫描一遍数组，往最小堆插入数据，如果堆的元素个数已经达到 k，那么新元素需要和堆顶比较，如果小于堆顶，则移除堆顶，插入新元素。

（3）最终我们得到 k 个最小的元素。时间复杂度为 O(nlogk)。

方案三：快排分区函数

快排的分区函数：选择一个数，把数组的数分为两部分，把比选中的数小或者相等的数移到数组的左边，把比选中的数大的数移到数组的右边，返回分

区后的选中数所在的下标。

```
int partition(int[] data, int start, int end){
    if(start>end) return -1;
    int index = start; //可以随机选择 pivot，不一定是第一个元素
    //第一次交换
    int tmp = data[index];
    data[index] = data[end];
    data[end] = tmp;
    for(int i=start; i<end; i++){
        if(data[i] <= data[end]){
            if(i!=index){
                //第二次交换
                tmp = data[index];
                data[index] = data[i];
                data[i] = tmp;
            }
            index++;
        }
    }
    //第三次交换
    tmp = data[end];
    data[end] = data[index];
    data[index] = tmp;
    return index;
}
```

快排的分区函数广泛运用在排序和海量数据挑选当中，笔者希望面试者能做到 5 分钟之内默写出来。

对数组调用分区函数之后，如果返回的小标是 k-1，那么数组左边的 k 个数（数组小标从 0 到 k-1 时，有 k 个元素）就是最小的 k 个数（这 k 个数不一定是排了序的）。问题转化为不停调用 partition 函数，直至返回的下标为 k-1。这种方法的平均时间复杂度为 O(n)，因为它并未实现真正的快速排序算法。

```
void getTopK(int[] data, int k){
    int start =0, end=data.length-1;
    int index = partition(data, start, end);
    while(index !=k-1){
        if(index > k-1) {
              //从 index 前面重新找
              end = index -1;
              index = partition(data, start, end);
        }else{
              //从 index 后面重新找
              start = index +1;
              index = partition(data, start, end);
        }
    }
    for(int i=0; i<k; i++){
        System.out.println(data[i]+"\t");
    }
}
```

扩展问题

一个直角坐标系上有 N 个点，找出离原点最近的 k 个点，请设计数据结构并计算时间复杂度。

如何把上述扩展问题转化为“Top K”问题，使它的时间复杂度为 O(n)？这取决于我们如何设计数据结构。为这些点设计一个数据结构，并含有到原点的距离。

```
class Point{
    double x;
    double y;
    double distance; //到原点的距离
    Point(double x, double y){
        this.x = x;
        this.y = y;
        distance = Math.sqrt(x*x+y*y);
```

```
        }
    }
```

对于这些点的集合，调用分区函数（排序依据是点的 distance 大小），如果返回的小标是 k-1，那么前 k 个点就是离原点最近的 k 个点，把问题转化为“Top K”，因此，时间复杂度平均为 O(n)。

扩展问题

给定一维整型数组和一个整数 k，找出和不小于 k 的数目最少的子数组。

我们依然可以利用 partition 函数，把数组一分为二，并把此问题转化为“Top K”的问题。不过此时，我们需要对 partition 函数进行修改：1）对数组进行降序排列，即大的数放在数组左边，小的数放在数组右边；2）分区的同时计算左边数的和。

有了新的 partition 函数之后，我们可以循环调用该函数。把左半部分的和同 k 比较，如果大于等于 k，那么找到符合条件的分区，但不确定左半部分的子数组的元素个数是否是最少的，所以我们还需要继续调用 partition 函数；如果左半部分的和比 k 小，则相应更新 k 的值，k = k –左半部分的和，因为下次调用 partition 函数时，忽略左半部分的元素了。

```
void getTopKII(int[] data, int k){
   int start=0, end=data.length-1;
   int last=-1; //记录上次的子数组和不小于 k 的尾部元素下标
   //记录分区后的左半部分的和，java 无法传参，我们使用 Arraylist 来代替
   ArrayList<Integer> currSum = new ArrayList<Integer>();
   int index = partition(data, start, end, currSum);
   while(index >= 0){
       if(currSum.get(0) >= k) {
            //记录当前下标，因为我们找到了从 0 到 index 子数组
//并且满足和不小于 k 的条件。
            last=index;
```

```
                //从 index 前面重新找
                end=index-1;
                index = partition(data, start, end, currSum);
            }else{
                //从 index 后面重新找
                start = index+1;
                //因为更新了 start，忽略了新 start 之前的元素，
                //所以我们要相应调整 k
                k = k –currSum.get(0);
                index = partition(data, start, end, currSum);
            }
        }
        for(int i=0; i<= last; i++){
            System.out.println(data[i]+"\t");
        }
    }
```

面试题 4：两数组第 k 个值☆☆☆☆☆

> 两个有序数组 A 和 B，分别拥有 m 和 n 的长度，求其合并后的第 k 个值。

考虑这个问题开始时会觉得很简单，但随着我们思考的深入，会发觉难以达到面试官的要求。让我们从简单的解法开始。

方案一：归并排序

把数组 A 和 B 分别看作排序数组的左半部分和右半部分，然后进行归并操作，从两个数组的头部开始取出第 k 个值。只移动元素较小的数组，总共移动了 k 步，因此，时间复杂度为 O(k)。

方案二：归并和二分查找

与其按顺序挑出下一个排序的元素，我们不如考虑通过二分查找，同时在 A 和 B 里找，时间复杂度为 O(log(m+n))。感兴趣的读者可以试写一下实现代码。

方案三：二分操作

能否只在一个数组里使用二分查找呢？通过观察我们知道：1）不需要得到前 k 个数，只要第 k 个数；2）使用二分查找 A 时，不需要在 B 使用二分查找，而是通过各自的下标来计算该元素在合并之后的位置是否等于 k-1（因为数组下标从 0 开始，所以第 k 个元素的下标是 k-1）。

对于 A[i]和 B[j]，我们得到了在他们之前的 i+j 个元素，如果包含这两个元素本身，那么我们有 i+j+2 个元素。假设 max(A[i-1]， B[j-1]) < B[j] < A[i]，我们无法判断 A[i]是否为合并后的第 i+j+2 个元素，因为当前无法确定 B[j+1]和 A[i]的关系；但是，可以肯定 B[j]是第 i+j+1 个元素，为什么？因为在他们之前有 i+j 个元素比 A[i]和 B[j]小，那么第 i+j+1 个小的元素必定在 A[i]和 B[j]之间产生，而 A[i]大于 B[j]，B[j]就是第 i+j+1 个小的元素。

回到问题本身，我们需要保证两个等式：

（1）分别在数组 A 和 B 找出 i 和 j，使得 i+j+1 = k。

（2）满足 max(A[i-1]， B[j-1]) < B[j] < A[i])。我们不能保证能找到第二个不等式，因为不能保证 A 和 B 的元素是犬牙交错的。如果找不到相应的 A[i]，则说明 A 数组的元素过小，这时候在满足第一个等式的情况下，如果 j>=0，则取 B[j]的值；否则取 A[i-1]的值。

```
int findKthSortedArrays(int[] A, int[] B, int k) {
        int m = A.length, n=B.length;
        if(m>n) return findKthSortedArrays(B, A, k);
```

```
            int left = 0, right = m;
            while (left < right) {
                //二分查找
                int mid = left + (right - left) / 2, j = k-1 - mid;
                if (j >= n ||  A[mid] < B[j]) {
                    //小标 i 尽可能往前移动，找到比 B[j]大的第一个
A[i]
                    left = mid + 1;
                } else {
                    right = mid;
                }
            }
            int Aiminus1 = left - 1 >= 0? A[left - 1] :
Integer.MIN_VALUE;
            int Bj = k -1- left >= 0? B[k -1- left] :
Integer.MIN_VALUE;
            //取 B[j]和 A[i-1]中的一个
            return Math.max(Aiminus1, Bj);
    }
```

由于我们只对较短的数组进行二分查找，所以时间复杂度为 O(log(min(m,n)))。

面试题 5：有序数组去重☆

给出一个有序数组，就地移除重复元素，保持每个元素只出现一次并返回新的数组长度。

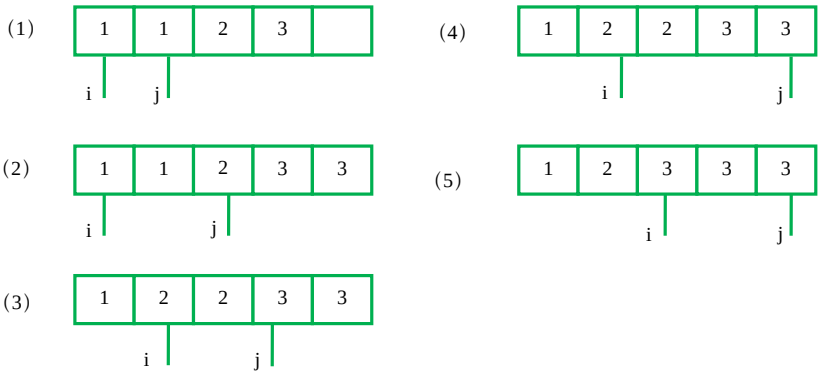
举例

输入数组 A=[1, 1, 2, 3, 3]，输出长度 3，A 为[1, 2, 3]。

使用两个下标 i 和 j，其中 i 记录新数组最后一个元素的位置，j 遍历整个

数组，通过 j 移动来移动 i。当 j 和 i 指向不同值的元素时，把 j 指向的值拷贝至 i 指向的元素，然后将 i 往后移动一位。当 j 和 i 指向元素值相同时，不对 i 做操作，只移动 j，以保证 i 遍历过的元素值各不相同。

以数组 A 为例，




```
int removeDuplicates(int A[]) {
      int i=0; //第一个指针
      int j;  //第二个指针
      int n = A.length;
      if (n<=1) return n;
      for (j=1;j<n;j++) {
            //j 在 i 前面走
            if (A[j] != A[i]) { //如果两者值不同，则在 i+1 位置保存
A[j]的值
                  A[++i]=A[j];
            }
      }
      return i+1;  //返回下标+1 的长度。
}
```

不需要进行删除元素的操作，进而避免每次删除带来移动后续元素的开销。因此，算法复杂度为 O(n)，而且不需要额外分配线性空间。

扩展问题

如果允许重复元素最多出现两次呢?

放宽把 j 指向的值拷贝至 i 指向的元素的条件：除了当 j 和 i 指向不同值的情况，我们还加上 j 指向的值不等于 i-1 指向的值的情况。

```
int removeDuplicatesII(int A[]) {
        int n = A.length;
        if (n <= 2) return n;
        int i = 1; // 第一个指针
        int j;   // 第二个指针
        for (j = 2; j < n; ++j) {
            //复制俩指针的值不等，或者第二个指针值不等于第一个指针的前一个值
            if (!(A[j] == A[i] && A[j] == A[i - 1]))
                A[++i] = A[j];
        }
        return i + 1;
}
```

面试题 6：数组分水岭☆☆☆

在一维数组中，找出一个点，使得其所有左边的数字均小于等于它，所有右边的数字都大于等于它。要求在线性时间内返回这个点所在的下标。

提问

应聘者：是不是一定能找出这个点?

面试官：不一定。

应聘者：如果找不到这个点，如何处理?

面试官：返回-1。

举例

输入 A={1, 0, 1, 0, 1, 2, 3}，返回下标 4 或 5。

首先，从左到右扫描一遍数组，通过一个辅助布尔数组记录哪些大于等于其之前所有元素的元素；其次，从右到左第二遍扫描数组，如果其后所有元素大于等于当前元素，而且在第一次遍历时当前元素大于等于之前的所有元素，那么程序返回其下标。

```
int getMagnitutePole(int[] A ){
    if(A == null || A.length == 0)    return -1;
    boolean[] isCurrMax = new boolean[A.length];
    int first= A[0]; //第一个指针记录当前最大值
    for(int i = 0; i < A.length; i++) {
        if (A[i] >= first) {
            first = A[i];
//如果比当前最大值还大，那么认为它大于之前的所有元素
            isCurrMax[i] = true;
        }
    }
       int second = A[A.length-1]; //第二个指数记录当前最小值
    for(int i = A.length - 1; i >= 0; i--) {
        if(A[i] <= second) {
            second = A[i];
//不小于左边元素，又不大于右边元素，返回该元素小标
            if(isCurrMax[i])  return i;
           }
        }
        return -1;
    }
```

第 8 章 动态规划

动态规划的方法是将一个问题化解为多个子问题，当某个给定子问题的解已经算出时，将其记忆化存储，以便下次解决同一个子问题时直接给出答案，避免重复计算。动态规划适用于有重叠子问题和最优子结构性质的问题，通常能够把指数级的算法复杂度下降为多项式的算法复杂度。

动态规划面试题的注意事项：

（1）需要一个辅助空间，记录子问题的解决结果。这种辅助空间通常是三维以下的数组。

（2）明确各个子问题之间的关系，以避免重复计算。

（3）通常对数组和字符串的高难度面试题非常有效。当遇到此类的面试题而又毫无头绪时，动态规划是首要选择。

面试题 7：最长递增子序列☆☆☆☆

给出一个数组，找出最少元素，使得将其删除之后，剩下的元素是递增有序的。

观察

删除最少的元素，保证剩下的元素是递增有序的。换一句话说，找出最长的递增有序序列。

找出最长递增有序序列有多种方法。在这里，我们除了需要额外一维数组 dp 记录以及每个元素为结尾的最长子序列的长度以外，还需要一个哈希表，用来保存在最长子序列的元素。dp[i]代表以输入数组 A[i]为结尾的最长子序列长度，dp[i]通过如下方式计算：

$$dp[i] = \begin{cases} 1，初始化时每个元素构成一个序列 \\ \max(dp[i], dp[j]+1)，其中\ j < i，并且\ A[i] >= A[j] \end{cases}$$

最后，通过回溯哈希表的方法，把需要删除的元素剔除。

```
ArrayList<Integer> minDelete(int[] A){
   ArrayList<Integer> res = new ArrayList<Integer>();
   //通过倒序追踪记录最长的递增子序列
   HashMap<Integer,Integer> bt= new
HashMap<Integer,Integer>();
   int[] dp= new int[A.length]; //记录每个元素为结尾的最长子序列长度
   int maxCount=0;
```

```
    int end=0; //最长递增子序列的最后一个元素
    for(int i=0; i<A.length; i++){
       dp[i] = 1;
       for(int j=0; j<i; j++){
          if(A[i]>=A[j]){
             dp[i]=Math.max(dp[i], dp[j]+1);
             if(maxCount<dp[i]){
                maxCount=dp[i];
                bt.put(i,j);
                end=i;
             }
           }
        }
    }
    int k=A.length-1;
    while(k>=0){
        //加入需要被删除的元素
        while(k>end){ res.add(A[k]); k--; }
        k--;
        if(bt.containsKey(end)){
             end=bt.get(end); //求出 LIS 的上一个元素
        }else end=-1; //LIS 到头了
    }
    return res;
}
```

面试题 8：最小化数组乘积☆☆☆☆

给出两个数组 A 和 B，两个数组大小分别为 m 和 n，其中 m<n。现要求将(n-m)个 0 插入 A，使得 A*B 的值最小，求其最小乘积。

举例

两个数组 A={1, -1}，B={1，2，3，4}，如果把两个零插入数组 A 中间，得到 A'={1, 0, 0, 1}，使得数组乘积最小，即为 -3。

一个二维数组 dp 记录 A 和 B 不同位置组合的最小乘积，dp[i][j]代表 A[0…i]与 B[0…j]组合的最小乘积。当 i=0，j=0 时，只有一种组合，无法插入 0，即 A[0]*B[0]；当 i=0 时，j 可以从 0 到 n-m，也就是 n-m 个 0，加上 B[j]本身与 A[0]进行组合；当 i>0 时，因为我们只能往较短数组 A 插入 0，所以我们要么累加 A[i]*B[j]+dp[i-1][j-1]，要么累加 0*B[j]+dp[i][j-1]，取两者较小值。

$$dp[i] = \begin{cases} A[i]*B[j]，当\ i=0，j=0\ 时 \\ min(A[i]*B[j], dp[i][j-1])，当\ i=0，0<j<=n-m\ 时 \\ min(A[i]*B[j]+dp[i-1][j-1], dp[i][j-1])，当\ 0<i<=j<=i+n-m\ 时 \end{cases}$$

```
int minProduct(int A[], int B[]){
      int m=A.length, n=B.length;
      int[][] dp = new int[m][n]; //记录每种组合的最小乘积
      for(int i = 0; i < m; i++) {
          for(int j = i; j < i + (n - m) + 1; j++) {
                if(j > i) {
                     //要么使用 A[i]*B[j]，要么使用 0*B[j]
                     if(i > 0) dp[i][j] = Math.min(A[i] * B[j]
+
dp[i - 1][j - 1], dp[i][j - 1]);
```

```
                        else dp[i][j] = Math.min(A[i] * B[j],
dp[i][j - 1]);
                    } else {
                        if(i==0){
                           dp[i][j] = A[i] * B[j];
                        }else{
                           dp[i][j] = A[i] * B[j] + dp[i-1][j-1];
                        }
                    }
                }
            }
            return dp[m-1][n-1];
    }
```

面试题 9：刷房子☆☆☆

在一条街上，给 H 个房子刷墙，要求每个房子刷一种颜色，相邻房子不能刷同一种颜色。每种颜色成本不同，求最小刷墙成本。

一个二维数组 dp 记录每个房子刷每一种颜色的最小成本，比如，dp[i][j] 代表第 i+1 个房子刷第 j+1 种颜色的最小刷墙成本：

`d[i][j] = min(d[i-1][k])+color[j]`，其中 `0<=k<color.length, k!=j`。

最后，找出最后一栋房子刷各种颜色之后的最小值，即为整条街的最小刷墙成本。

```
 int minCost(int h, int[] color){
   if(h==0 || color.length==0) return 0;
```

```
    int min=Integer.MAX_VALUE;
    int[][] d= new int[h][color.length];
    for(int i=0; i<h; i++){
        for(int j=0; j<color.length; j++){
            if(i==0) d[i][j] = color[j]; //初始化第一个房子
            else{
                 min=Integer.MAX_VALUE;
                 //找出在刷第 j 种颜色时，最小代价
                 for(int k=0; k<color.length;k++){
                      if(k==j) continue;
                      min = Math.min(min, d[i-1][k]);
                 }
                 d[i][j] = min+color[j];
            }
        }
     }
     min=Integer.MAX_VALUE;
     //找出最后一栋房子刷各种颜色之后的最小值
     for(int j=0; j<color.length; j++){
           min = Math.min(min, d[h-1][j]);
     }
     return min;
  }
```

面试题 10：编辑距离☆☆☆☆

> 输入两个单词 word1 和 word2，求出从 word1 转换为 word2 的最少步骤。每个转换操作算一步。转换操作限定于：删除一个字符，插入一个字符和替换一个字符。

只是求出最少的步骤，而不是需要具体的转换过程，因此，我们考虑动态规划的方法。二维辅助空间的元素 dp[i][j]代表从 word1[0…i-1]转换为 word2[0…j-1]的最少步骤。

$$dp[i][j] = \begin{cases} dp[i-1][j-1]，当\ word1[i-1]字符与\ word2[j-1]字符相等 \\ 1+\min(dp[i-1][j-1], dp[i-1][j], dp[i][j-1])，当上述条件不成立时，取修改，删除\ word1\ 字符，删除\ word2\ 字符的最小值 \end{cases}$$

在初始化边界时，特别需要注意空串的时候，当一方是空串时，最少步骤即为另一方的字符个数。

```
    int editDistance(String word1, String word2) {
              int len1=word1.length(), len2=word2.length();
              if(len1==0) return len2;
              if(len2==0) return len1;
              int dp[][] = new int[len1+1][len2+1];
              //初始化边界，只有 word1 的字符
              for(int i=0;i<=len1;i++)  dp[i][0]=i;
              //初始化边界，只有 word2 的字符
              for(int j=0;j<=len2;j++)  dp[0][j]=j;
              for(int i=1;i<=len1;i++) {
                   for(int j=1;j<=len2;j++) {
                        if(word1.charAt(i-1) ==
word2.charAt(j-1))
                             dp[i][j]=dp[i-1][j-1];
                        else {
                             dp[i][j]=1+Math.min(dp[i-1][j-1],
//修改
    Math.min(dp[i-1][j],dp[i][j-1])); //删除
                        }
                   }
              }
              return dp[len1][len2];
    }
```

面试题 11：最长回文子串☆☆☆☆☆

输入一字符串 S，找出其最长回文子串(plindromic substring)。

二维布尔数组的元素 dp[i][j]记录从 s[i]到 s[j]组成的子串是否为回文。dp[i][j]计算如下：

$$dp[i][j]=\begin{cases}\text{dp[i+1][j-1]，当 s[i]与 s[j]相等时 false}\\ \text{当 s[i]与 s[j]不相等时}\end{cases}$$

与以前初始化边界不同，这次初始化对角线：我们认为单个字符是回文，如果是紧挨着的两个相同字符也是回文。

```
String longestPalindrome(String s) {
      int n = s.length(), longestBegin = 0, maxLen = 1;
      boolean dp[][] = new boolean[n][n];
      //单个字符是回文
      for (int i = 0; i < n; i++)  dp[i][i] = true;
      //紧挨着的两个相同字符也是回文
      for (int i = 0; i < n-1; i++) {
            if (s.charAt(i) == s.charAt(i+1)) {
                  dp[i][i+1] = true;
                  longestBegin = i;
                  maxLen = 2;
            }
      }
      for (int i=n-3; i >= 0; i--) {
            for (int j =i+2; j < n; j++) {
                  if (s.charAt(i) == s.charAt(j) &&
dp[i+1][j-1]) {
                        //如果两端字符相同，那取决于内部子串的情况
                        dp[i][j] = true;
```

```
                        longestBegin = i;
                        maxLen = j-i+1;
                    }
                }
            }
            //截取回文
            return s.substring(longestBegin, longestBegin+maxLen);
    }
```

动态规划的算法复杂度为 O(n2)，还有一种是将每个位置作为中心点遍历的 Manacher 方法（http://en.wikipedia.org/wiki/Longest_palindromic_substring），可以达到 O(n)。面试的时候可以提一下 Manacher 方法，但很多面试官并不知道有这种算法。

面试题 12：最大公共子串☆☆☆☆

给出两个字符串 s1 和 s2，返回其最大的公共子串。

二维整型数组的元素 dp[i][j]记录的是从 s1 的开始字符到 s1 的第 i 个字符组成的字符串，和从 s2 的开始字符到第 j 个字符组成的字符串，这两个字符串的最大公共子串长度。dp[i][j]计算如下：

$$dp[i][j] = \begin{cases} \text{dp[i-1][j-1]+1，当 s[i]与 s[j]相等时} \\ \text{0，当 s[i]与 s[j]不相等时} \end{cases}$$

在计算 dp[i][j]的过程中，记录了公共字串的最大长度，同时也记录了在当前最大长度下的公共字串。

```
        public String LCS(String s1, String s2){
             String res="";
             if(s1==null || s1.length()==0 || s2==null ||
s2.length()==0)
    return res;
             int max=0, m=s1.length(),n=s2.length();
             int[][] dp = new int[m][n];
             //计算到 s1 的第 i 个字符和 s2 的第 j 个字符为止的最大公共子串长度
             for(int i=0; i<m; i++){
                  for(int j=0; j<n;j++){
                       if(s1.charAt(i)==s2.charAt(j)){
                            if(i==0 || j==0) dp[i][j]=1; //边界
情况
                            else{
                           dp[i][j] = dp[i-1][j-1]+1; //加上当前
长度
                            }
                            //记录最大长度和子串
                            if(dp[i][j]>max){
                          max=dp[i][j];
                          res = s1.substring(i-dp[i][j]+1, i+1);

                            }
                       }
                  }
             }
             return res;
        }
```

第 9 章 优先遍历

深度优先遍历（Depth-First-Search）是沿着树的深度遍历树的节点，尽可能深的搜索树的分支。当节点 v 的所有边都被探寻过，搜索将回溯到发现节点 v 的那条边的起始节点。整个进程反复进行直到所有节点都被访问为止。广度优先遍历（Breadth-First-Search）是从根节点开始，沿着树的宽度遍历树的节点。如果所有节点均被访问，则算法中止。优先遍历经常用在树和图遍历、二维数组保存的图像处理、字符串替换等。

面试题 13：填充图像☆☆☆☆

假设输入二维数组代表一张位图，元素的不同整数值代表不同颜色，先输入位置(x,y)，同时用一个新的颜色替换原有颜色，并且向周边（上下前后四个方向）替换颜色，直到遇上新的颜色为止，以形成一个单色的区域。

对于每个位置做同样的操作，我们可以采用深度优先遍历或者广度优先遍

历。对于每个位置，当遇上当前颜色和新颜色不一致时，替换为新颜色，然后遍历其四个方向上的邻居。当遇上当前颜色和新颜色一致时，不做任何操作，直接返回。

```
    void bucketfill(ArrayList<ArrayList<Integer>> image, int
newcolor, int x, int y){
        int m=image.size(), n=image.get(0).size();
        int oldcolor=image.get(x).get(y);
        if(oldcolor==newcolor)  return; //新旧一样
        image.get(x).set(y,newcolor);  //设置新颜色
        //向左遍历
        if(x>0 && image.get(x-1).get(y)==oldcolor){
            bucketfill(image,newcolor,x-1,y);
        }
        //向上遍历
        if(y>0 && image.get(x).get(y-1)==oldcolor){
            bucketfill(image,newcolor,x,y-1);
        }
        //向右遍历
        if(x<m-1 && image.get(x+1).get(y)==oldcolor){
            bucketfill(image,newcolor,x+1,y);
        }
        //向下遍历
        if(y<n-1 && image.get(x).get(y+1)==oldcolor){
            bucketfill(image,newcolor,x,y+1);
        }
    }
```

面试题 14：单词替换规则☆☆☆☆

输入字符串和一组变换规则，输出所有通过变换规则之后的字符串。

举例

输入字符串“face”，变换规则如：‘a’→‘@’，‘e’→‘3’，‘e’→‘E’等。输出：“f@c3”，“fac3”，“face”等。

与前一道题“单词变换”类似，从字符串的第一个字母开始，获取每个字母的变体，加上字母本身（即不做变换的情况）做深度优先遍历，直至最后一个字母。为了避免重复输出结果，需要通过一个不含重复元素的集合做去重。

```
    void genStringByRule(String s,
    HashMap<Character, ArrayList <Character>> hm){
      HashSet<String> set = new HashSet<String>();
      set.add(s); //不输出自身
      genStringByRule(s, hm, 0, set, “”);
    }

    void genStringByRule(String s,
    HashMap<Character, ArrayList<Character>> hm,
    int start, HashSet<String> set, String op){
      if(start==s.length()){
          //找到一个结果
          if(!set.contains(op)){
               //如果不重复，则输出
               System.out.println(op);
               set.add(op);
          }
          return;
      }
      ArrayList<Character> mutations = hm.get(s.charAt(start));
      if(mutations!=null){
          for(Character c: mutations){
              //递归调用每种字符变换
              if(c!=null) genStringByRule(s, hm, start+1, set,
op+c);
          }
      }
      //还要加上保留当前字符的情况
```

```
        genStringByRule(s, hm, start+1, set, op+s.charAt(start));
    }
```

面试题 15：有向图遍历☆☆☆☆

给出一个有向图，从 A 节点走到 B 节点，正好走 N 步，有多少种走法？走过的节点可以重复走。

假设有向图的节点定义如下：

```
class GraphNode{
   int val;
   ArrayList<GraphNode> nexts;
}
```

在这里我们使用广度优先遍历：使用两个队列，分别记录当前遍历的节点，和下一轮需要遍历的节点（即本层节点的下一层子节点）。从 A 节点开始遍历，每遍历完一层时，步数加一；如果遍历时遇到了 B 而且步数恰好为 N，那么走法数相应加一。遍历时并未记录节点的访问情况，这样可以满足走过的节点可以重复走。

```
    int getGraphPaths(GraphNode A, GraphNode B, int N){
    Queue<GraphNode> currQue = new LinkedList<GraphNode>();
          currQue.add(A);
          int count=0, steps = 0;
          while(currQue.size() > 0 &&steps <= N){
               Queue<GraphNode> nextQue = new
LinkedList<GraphNode>();
               steps++;
               while(currQue.size() > 0){
                     GraphNode node = currQue.poll();
                     //获取下一个节点
```

```
                    for( GraphNode next: node.nexts){
                        if(next == B && steps == N){
                            //找到一条路径
                            count++;
                        }
                        //同一轮可能含有重复节点
                        nextQue.add(next);
                    }
                }
                //复制下一轮的结果
                currQue = nextQue;
            }
            return count;
    }
```

第 10 章 哈希

哈希表通常用在要求常数级时间查找，以及字符或数字去重的解题方法中。

面试题 16：最长不同字符的子串☆☆☆☆

输入一个字符串，求不含有重复字母的最长子串的长度。

举例

输入字符串“aaaa”，其不含有重复字母的最长子串为“a”，输出长度为 1。

一般来说，哈希表的键值记录字母和该字母出现的次数。但这次光是记录出现的次数还不够，需要记录字母出现的位置。此外，为了记录长度，我们还需要最长子串的开始位置。每当扫描字符串的字符时，如果该字符已经出现过，而且出现的位置在开始位置之后，更新开始位置。然后，更新哈希表的值，并

且保存最长子串的长度。

```
    int lengthOfLongestSubstring(String s) {
        int n=s.length(), start=0, maxLen =0;
        //使用哈希表记录字符最新出现的位置
    HashMap<Character, Integer> hm = new HashMap<Character,
Integer> ();
        for(int i=0; i<n; i++){
            char c = s.charAt(i);
            if(hm.containsKey(c) && hm.get(c) >= start){
                  //更新出现的位置
                  start = hm.get(c ) +1;
            }
            hm.put(c, i);
            maxLen = Math.max(maxLen, i-start+1);
        }
        return maxLen;
    }
```

面试题 17：常数时间插入删除查找☆☆☆

> 设计一数据结构，使插入、删除、搜索、随机访问都是常数时间。

单是哈希表或者数组的数据结构，均无法满足题目要求。哈希表做不到随机访问是常数级，而数组的数据结构无法做到在常数时间搜索数组元素。因此，我们可以考虑结合两者，使得数组 A 保存元素，而哈希表 HM 的键存储数组元素，其值为元素在数组的下标。

插入：把元素 value 放在数组末端。假设 i 为新元素的下标，那么 HM[value] = i。

删除：使用数组的最后一个元素来替换待删除的元素，通过哈希表获取待

删除的位置，替换成最后一个元素，同时更新替换后元素的下标。最后，将数组大小减少一，并移除待删除元素在哈希表的键值。

搜索：搜索哈希表，判断是否含有此键。

随机访问：获取随机值，返回数组的下标为随机值的元素。

面试题 18：对数时间范围查询☆☆☆☆

> 设计一个键与值数据结构，至少实现如下两个查找功能：查找单个 key，以及支持在给出 key 范围内查找。

数组在存储和管理有序元素时不是很有效，因此，我们可以考虑使用二叉搜索树（BST）。哈希表记录键-值对，而且二叉搜索树只保存键。当查找单个键时，我们只要在哈希表查找该键，如果存在，则返回其对应的值，这样可以在 O(1)时间内完成。当查找给定 key 范围内的值时，先在二叉搜索树找出范围内的 key 值列表，再从哈希表里找出那些 key 对应的值的列表。

面试题 19：实现 LRU 缓存☆☆☆☆

> 实现 LRU 缓存类，至少包含 get 和 put 成员函数。get 和 put 算是对元素的一次访问。

设计思路：使用一个双向队列（含头尾空元素）保存元素，并记录元素的访问顺序；再利用一个哈希表，使得查找元素更便捷。

首先，设计元素类，包含了键与值以及双向指针。

```
class Item<K,V>{
    K key;
    V value;
    Item prev, next;
    Item(K key, V value){ this.key = key;
this.value = value;prev=null; next=null;}
    Item(){key=null; value=null; prev=null; next=null;}
}
```

其次，设计 LRUCache 类，包含了队列和哈希表。

```
class LRUCache{
    Item m_start, m_end;
    int m_size, curr_size;
    HashMap<K,Item> m_hm;
    LRUCache(int size){
      //初始化
      m_size = size;
      curr_size =0;
      m_start=new Item();
      m_end= new Item();
      //首尾相接的空队列
      m_start.next=m_end;
      m_end.prev = m_start;
      m_hm = new HashMap<K,Item>();
    }
}
```

然后，实现 get()方法：先判断哈希表是否存在该 key，如果存在，我们还不能直接返回对应的 value，因为 get 也算是对元素的一次访问，我们需要把该元素移动到双向队列的头部。

```
  V get(K key){
     if(m_hm.contains(key)){
         //移动元素至队列头部
         moveToHead(m_hm.get(key));
```

```
            return m_hm.get(key).value;
        }else{
            return null;
        }
    }
    void moveToHead(Item item){
        Item cross = m_start;
        //找出元素的前一个节点
        while(cross.next!=null && cross.next!=m_end &&
cross.next.key!=item.key)
            cross = cross.next;
        if(cross.next!=null && cross.next.key == item.key){
            //连接该元素的前后节点
            Item next= item.next;
            cross.next = next;
            next.prev = cross;
            //insert to head
            insertHead(item);
        }
    }
    void insertHead(Item item){
        //插入到队列头部
        Item next = m_start.next;
        m_start.next = item;
        item.next = next;
        item.prev = m_start;
        next.prev = item;
    }
```

最后，实现 put()方法。

```
    void put(K key, V value){
        Item item = new Item(key, value);
        if(m_hm.contains(key)){
            //如果已经存在了，则直接移动到队列头部
            moveToHead(item);
        }else{
            if(curr_size < m_size){
```

```
                //插入新元素
                insertHead(item);
                curr_size++;
            }else{
                //队列满了，删除队尾，然后再插入新元素
                removeItem();
                insertHead(item);
            }
        }
        m_hm.put(key, item);
    }
    void removeItem(){
        //删除队尾的元素
        Item item = m_end.prev;
        if(item!=m_start){
            Item prev = item.prev;
            prev.next = m_end;
            m_end.prev = prev;
            m_hm.remove(item.key);
        }
    }
```

如果面试官能接受的话，我们还可以继承现成的实现 LRU 的 Java 类 LinkedHashMap。

面试题 20：经过最多点的直线☆☆☆

> 给出一个二维的图，图上有很多点，找出一条穿过最多点的直线。

解题思路：两点可组成一条直线，把图上的每两个点组成的直线计算出来，然后聚合，把相同直线聚合在一起，返回穿过最多点的直线。

首先，设计直线。直线包含了斜率，以及与 x 轴相交的点。在 Java 类中，

判断两个对象相等，必须重写 hashCode 和 equals 两个函数。如果斜率相等，以及与 x 轴相交的点相等，我们认为两个直线是相同的。

```
    public class Line {
        double epsilon = .0001;
        double slope;
        double intercept;
        boolean infiniteSlope = false;
        public Line(Point p, Point q) {
           if (Math.abs(p.x - q.x) > epsilon) {
                //对于不和 Y 轴平行的直线，计算斜率和 x 轴的交点
                slope = (p.y - q.y) / (p.x - q.x);
                intercept = p.y - slope * p.x;
           } else {
                //与 Y 轴平行的直线
                infiniteSlope = true;
                intercept = p.x;
           }
        }
        public boolean isEqual(double a, double b) {
           return (Math.abs(a - b) < epsilon);
        }
    @Override
        public int hashCode() {
           //根据斜率和交点计算哈希值
           int sl = (int)(slope * 1000);
           int in = (int)(intercept * 1000);
           return sl | in;
        }
        @Override
        public boolean equals(Object o) {
           Line l = (Line) o;
           if (isEqual(l.slope, slope) && isEqual(l.intercept,
intercept)
    && (infiniteSlope == l.infiniteSlope)) {
                //如果斜率相同，与 x 轴交点相同，那么我们就认为这两条直线相
同
                   return true;
```

```
            }
            return false;
        }
    }
```

其次，使用哈希表记录两点组成的直线的出现次数。出现次数最多的线，即为穿过最多点的直线。

```
    Line findBestLine(Point[] points) {
    Line bestLine = null;
        //记录每条直线的出现次数
    HashMap<Line, Integer> lineCount = new HashMap<Line,
Integer>();
    for(int i = 0; i < points.length; i++) {
    for(int j = i + 1; j < points.length; j++) {
    Line line = new Line(points[i], points[j]);
                //更新或插入每条线出现的次数
    if(!lineCount.containsKey(line)) {
    lineCount.put(line, 1);
    }
    lineCount.put(line, lineCount.get(line) + 1);
    if(bestLine == null ||
    lineCount.get(line) > lineCount.get(bestLine)) {
                    //保存出现次数最多的直线
    bestLine = line;
    }
    }
    }
    return bestLine;
    }
```

第 11 章 堆栈

通常我们也会使用堆、栈以及队列来解决字符串变换问题，数据流的中值和最大值，以及二叉树的遍历相关的面试题。

面试题 21：局部最大值☆☆☆

输入一个长数组 A，以及窗口大小 w，输出一个数组 B，B[i]代表移动窗口 A[i]…A[i+w-1]的最大值。

举例

输入 A = {1, 3, -1, -3, 5, 3, 6, 7}，窗口大小 w=3，输出如下：

移动窗口	最大值
[1 3 -1] -3 5 3 6 7	**3**
1 [3 -1 -3] 5 3 6 7	**3**
1 3 [-1 -3 5] 3 6 7	**5**
1 3 -1 [-3 5 3] 6 7	**5**
1 3 -1 -3 [5 3 6] 7	**6**
1 3 -1 -3 5 [3 6 7]	**7**

解题思路：使用一个队列，队列的头部保存当前最大值。Java 的 PriorityQueue 既可以当 Queue 使用，也可以当堆来使用，但是默认情况是小的元素排在前面，大的元素排在后面。我们需要设置自己的排列顺序。在移动窗口时，如果队列的头元素不在窗口内，我们将移除队列的头元素，直至它在移动窗口内。

```
    class Pair{
         int val;
         int index;
         Pair(int v, int i){
             val = v;
             index = i;
         }
    }
    void maxSlidingWindow(int A[], int w, int B[]) {
         int n = A.length;
         Comparator< Pair > comparator = new Comparator< Pair >()
{
             @Override
             public int compare(Pair n1, Pair n2) {
                  //设定排序规则，大的在前，小的在后
                  if (n1.val < n2.val)
                       return 1;
                  else if (n1.val > n2.val)
```

```
                    return -1;
                else if( n1.index > n2.index)
                    return -1;
                else if(n1.index < n2.index)
                    return 1;
                else
                    return 0;
            }
        };
        //队列头部保存最大值
        PriorityQueue<Pair> Q = new PriorityQueue<Pair>(w,
comparator);
        for (int i = 0; i < w; i++)
            Q.offer(new Pair(A[i], i)); //初始化前 w 个
        for (int i = w; i < n; i++) {
            Pair p = Q.peek();
            B[i-w] = p.val;
            while (p.index <= i-w) {
                //移除不在窗口内的队头
                Q.poll();
                p = Q.peek();
            }
            Q.push(new Pair(A[i], i));
        }
        B[n-w] = Q.peek().val;
    }
```

面试题 22：数据流最大值☆☆☆☆

> 某一刻开始从无限的整数流(infinite integer streaming)取最大值。

起初看起来觉得无从下手。其实我们可以问面试官：如何获取这些无限的整数流？从某一刻开始保存这些数据流的时候，使用一个队列来记录最大值：

每当加入一个整数时，从队列尾部开始扫描，移除比当前值小或相等的元素（因为用不上），这样队列头保存当前最大值。因此，某一刻开始的最大值，可能在常数时间内获得。

```
ArrayList<Integer> list= new ArrayList<Integer>();
void add(int num){
   for(int i=list.size()-1; i>=0; i--){
        //列表尾部开始扫描，移除比当前值小或相等的元素
        if(num>=list.get(i)){
             list.remove(i);
        }
   }
   list.add(num);
}
int getMax(){
   //取队列头部元素
   return list.size()>0 ? list.get(0): Integer.MIN_VALUE;
}
```

面试题 23：产生逆波兰式☆☆☆

将逆波兰式转化为中序表达式。

举例

输入{"5", "8", "4", "/", "+"}，输出“(5 + (8 / 4))”。

扫描输入字符数组，如果遇到数字，直接加入栈；如果遇到运算符，第一个操作数为栈顶元素，第二个操作数为中间结果，将三者连接起来形成新的字符串作为新的中间结果。如果第一次遇到运算符时，无中间结果，可考虑出栈两次，第一次出栈的数作为中间结果。

```
String genRPNotation(char[] input){
     int n = input.length;
     if(n==0) return “”;
     String result = “”;
     Stack<Character> stack = new Stack<Character>();
     for(int i=0; i<n; i++){
           //区分符号和数字
           if(input[i] == ‘/’ || input[i] == ‘+’
|| input[i] == ‘*’ || input[i] == ‘-’){
                 if(result=="")
                       //第一次遇到运算符时两次出栈
                       result= result+stack.pop();
                 char first = stack.pop();
                 result = "("+first+input[i]+result+")";
           }else{
           //如果是数字，直接压入栈
                 stack.push(input[i]);
           }
     }
     return result;
}
```

面试题 24：逆波兰式计算☆☆☆

给出逆波兰表达式，输出其计算结果。

举例

{"4", "1", "+", "2", "*"} → ((4 + 1) * 2) → 10

{"5", "8", "4", "/", "+"} → (5 + (8 / 4)) → 7

扫描输入字符数组，如果遇到数字字符，将其转化为数字之后直接压入栈；如果遇到运算符，第一个操作数为栈顶元素，第二个操作数为中间结果，计算

此运算式的结果作为新的中间结果。如果第一次遇到运算符时，无中间结果，可考虑出栈两次，第一次出栈的数作为中间结果。

```
double calculateRPNotation(char[] input){
   int n = input.length;
   if(n==0) return 0.0;
   double result=0.0;
   boolean firstSign=true;
   Stack<Integer> stack = new Stack<Integer>();
   for(int i=0; i<n; i++){
         //区分符号和数字
         if(input[i] == '/' || input[i] == '+'
|| input[i] == '*' || input[i] == '-'){
               if(firstSign){
                     result=(double) stack.pop();   //保留精度

                     firstSign=false;
               }
               int first = stack.pop();
               switch(input[i]){
                  //分别处理各种符号
                  case '/': result= first/result; break;
                  case '+': result = first+result; break;
                  case '-': result = first-result; break;
                  case '*': result = first*result; break;
                  default:
                      break;
               }
         }else{
               //压入数字
               stack.push(input[i]-'0');
         }
   }
   return result;
}
```

面试题 25：设计 Min 栈☆☆☆☆

设计一个栈的数据结构，使得 min、push 以及 pop 的时间复杂度都是 O(1)。

观察

对于普通的栈，push 和 pop 都是 O(1)的时间，但是求最小值，则需要扫描栈内的所有元素，时间复杂度为 O(n)。这就说明需要一个辅助的空间来记录最小值，也许是一个变量，也许是另一个栈，也许是数组、哈希表，等等。

我们注意到最小值将伴随进出栈的操作而改变，常数空间满足不了这种情况，而需要一个线性的空间，比如说新栈来存储最小值，使得记录的最小值跟随着原有栈顶元素的变化而改变。当新元素入栈时，同时将当前最小值压入辅助栈。如何求出当前最小值呢？辅助栈的栈顶一直是最小值，只要把新元素同它比较，取较小值者作为当前新的最小值。

元素出栈时，辅助栈的栈顶也出栈，这样辅助栈在每次出入站时均保留当前的最小值。如果求栈元素的最小值，只要查看辅助栈的栈顶元素即可。

```
class MinStack{
    Stack<Integer> s;
    Stack<Integer> s2;  //辅助栈
    MinStack(){
        s = new Stack<Integer>();
        s2 = new Stack<Integer>();
    }
    public void push(int val){
        s.push(val);
        //同时往 s2 压入最小值
        if(s2.empty() || val < s2.peek()) s2.push(val);
```

```
            else s2.push(s2.peek());
        }
        public int min() throws EmptyStackException{
            //预防 s2 为空，抛出异常
            if(s2.empty()) throw new EmptyStackException("");
            return s2.peek();
        }
        public int pop() throws EmptyStackException {
            //预防 s 或 s2 为空，抛出异常
            if(s2.empty() || s.empty()) throw new
EmptyStackException("");
            s2.pop();
            return s.pop();
        }
    }
```

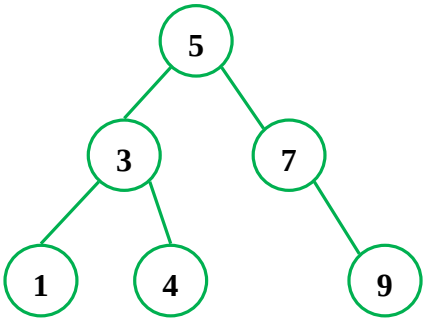
面试题 26：最小公共祖先☆☆

给定两个节点，求它们在一棵二叉搜索树中的最小公共祖先。

举例

一棵二叉搜索树：

节点 1 和节点 9 的最小公共祖先是树根节点 5；而节点 1 和节点 4 的最小公共祖先是节点 3。



观察

二叉搜索树的特点是节点的值大于其左子树的值，小于其右子树的值。因此两个节点的最小公共祖先的值一定介于这两个节点的值之间。

利用二叉搜索树的特点，通过前序遍历二叉树来判断最小公共祖先出现在左子树、右子树，还是节点本身。如何证明通过前序遍历找出最小公共祖先是正确的？我们利用反证法：r 是通过前序遍历算出的 p 和 q 的最小公共祖先（p 和 q 分别在 r 的左右子树里，即 p<r<q），假设存在 r’是 p 和 q 的最小公共祖先，而且 r’比 r 更靠近 p 和 q，则意味着 r’在 r 的子树中， p 和 q 分别在 r’的左右子树里。假设 r’在 r 的左子树里，那么 p<r’<q<r，这与前面的 q>r 矛盾，所以不可能出现有比 r 更靠近 p 和 q 的最小公共祖先。

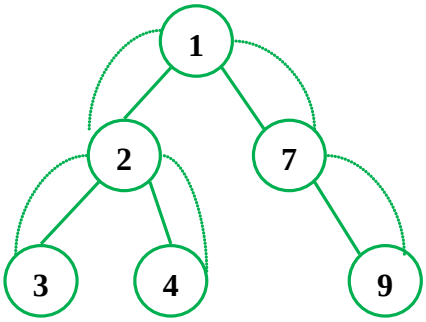
```
TreeNode LCA(TreeNode root, TreeNode p, TreeNode q){
if(root==null || p==null || q==null) return null;
if(root.val > p.val && root.val > q.val){
          //最小公共祖先在左子树
          return LCA(root.left, p, q);
}else if(root.val < p.val && root.val < q.val){
          //最小公共祖先在右子树
          return LCA(root.right, p, q);
}else{
          //返回节点本身
          return root;
}
}
```

扩展问题 1

给定两个节点，求它们在一棵二叉树中的最小公共祖先。每个节点除了有左右节点以外，还有一个指向其父节点的指针。

举例

一棵二叉搜索树：



节点 3 和节点 9 的最小公共祖先是树根节点 1；而节点 2 和节点 4 的最小公共祖先是节点 2。

观察

根据每个节点由指向其父节点的指针获取从节点到树根的路径。有了两条路径之后，求出这两条路径的第一个相同的节点，即为最小公共祖先，解法类似于求两链表的相交点。

首先，判断在二叉树里是否存在这两个节点，如果有则返回这两个节点在树里对应的节点。有了这两个节点之后，根据其指向父节点的指针，求出到根节点的路径。类似于求两链表的相交点。算出两条路径的长度，如果长度不一致，砍掉长度较长的路径的前面部分，以保证两条路径长度相同，然后同步走，同时比较两边的值，值相同的即为最小公共祖先。还有一种办法是使用哈希表，把一条路径的所有节点放入哈希表，然后扫描第二条路径，如果能在哈希表中找到当前节点，则该节点为最小公共祖先。

```
TreeNode LCA2(TreeNode root, TreeNode p, TreeNode q){
    if(root==null || p==null || q==null) return null;
    // 用 pq.left pq.right 分别存储 p 与 q 在二叉树中对应的节点
    TreeNode pq = new TreeNode(0);
```

```
        //找出 p 和 q 在二叉树对应的节点
        help(root, p, q, pq);
        TreeNode pp = pq.left, qq = pq.right;
        //如果二叉树没有 p 或 q，则返回 null
        if( pp==null || qq==null) return null
        int lenp = 0, lenq=0;
        TreeNode up1=pp, up2=qq;
        //计算各种到顶的长度
        while(up1 != root){ up1=up1.parent; lenp++;}
        while(up2 != root){ up2=up2.parent; lenq++;}
        up1=pp;
        up2=qq;
        //借鉴求两链表相交节点的做法，先同步到长度相同的情况
        while(lenp>lenq){
            up1=up1.parent;
            lenp--;
        }
        while(lenq>lenp){
            up2=up2.parent;
            lenq--;
        }
        //往上爬，直到它们相遇，相遇的点就是最小公共祖先
        while(up1!=up2){
            up1=up1.parent;
            up2=up2.parent;
        }
        return up1;
    }
    void help(TreeNode root, TreeNode p, TreeNode q, TreeNode pq){
        if(root==null) return;
        //遍历二叉树，找出 p 和 q
        if(root == p)   pq.left=root;
        if(root == q)  pq.right=root;
        if(pq.left==null || pq.right==null){
            help(root.left, p, q, pq);
            help(root.right, p, q, pq);
        }
    }
```

扩展问题 2

给定两个节点，求它们在一个二叉树中的最小公共祖先。

提问

应聘者：输入的二叉树是二叉搜索树吗？

面试官：不是。

应聘者：给定的两个节点在输入的二叉树里？

面试官：是的。

应聘者：在给定的两个节点 A 和 B 中，其中节点 A 在另一个节点 B 的子树里，那么我可以理解为它们的最小公共祖先为 B。

面试官：对的。

第一次被问到这个问题时，可能没什么头绪。遍历二叉树通常有三种方式：前序、中序和后序。其实，还有两种：前后序和中后序，分别结合两种遍历的方式。比如，前后序，在递归函数体内，先处理当前节点，然后对其左节点和右节点分别递归调用函数，最后结合其左右子树的结果以及当前节点的处理结果，返回最终结果。

在本题中，我们先判断当前节点是否为两个给定节点 p 和 q 之一，如果是，直接返回当前节点，即为最小公共祖先。否则，递归对其左右节点调用函数，然后根据左右子树的结果，返回恰当的值。如果左右子树返回的最小公共祖先均不为空，说明 p 和 q 分别在当前节点的两边，则返回当前节点；如果一个不为空，则返回不为空的结果；如果左右子树返回均为空的结果，则函数最后也返回空。

```
TreeNode LCA3(TreeNode root, TreeNode p, TreeNode q) {
 if (root == null || p == null || q == null) return null;
 if (root == p || root == q) return root; //返回当前节点
 TreeNode left = LCA3(root.left, p, q);
 TreeNode right = LCA3(root.right, p, q);
  //p 和 q 分别在两边，则返回当前节点
 if (left!=null && right!=null) return root;
  //如果 p 或 q 在单边，则返回单边节点
 return left==null ? right : left;
}
```

第 12 章 排列组合

排列组合通常用在字符串或序列的排列和组合当中，其特点是固定的解法和统一的代码风格。通常有两种方法：第一种是类似动态规划的分期摊还（amortized）的方式，即保存中间结果，依次附上新元素，产生新的中间结果；第二种是递归法，通常是在递归函数里，使用 for 语句，遍历所有排列或组合的可能，然后在 for 语句内调用递归函数。

面试题 27：翻译手机号码☆☆☆

输入由数字组成的字符串，输出数字代表字母的所有组合。数字代表的字母参见手机 9 键英文输入键盘。

举例

输入“23”，输出：“ad”，“ae”，“af”，“bd”，“be”，“bf”，“cd”，“ce”，“cf”。

手机 9 键英文输入键盘如下图所示。通常面试官会给出数字和字符的映射关系，或者直接掏出手机展示 9 键英文输入键盘给应聘者看，所以不需要特意记忆这些数字对应的字母。一般来说，数字 0 和 1 所代表的字母为空串，而从 2 到 9 的数字，分别代表字母“abc”，“def”，“ghi”，“jkl”，“mno”，“qprs”，“tuv”，“wxyz”。

在这里，我们使用排列组合的第一种方法：分期摊还(amortized)的方法。从数字字符串第一个数字开始扫描，记录之前组合结果集，求出当前数字对应的多个字母，将每个映射的字母依附至结果集中的每个组合中，产生新的结果集。

扫描至数字结尾时，结果集即为所求的所有组合。

```
        public ArrayList<String> letterCombinations(String digits)
{
            ArrayList<String> res = new ArrayList<String>();
            res.add("");
            if(digits==null || digits.length()==0) return res;
            //数字对应的字母
            String maps[]= {"","","abc","def","ghi","jkl", "mno",
    "qprs", "tuv","wxyz"};
            for(int i=0; i<digits.length(); i++){
                  String s= maps[digits.charAt(i)-'0']; //获取映射
字符串
```

```
                ArrayList<String> tmp = new ArrayList<String>();
                for(int j=0; j<s.length();j++){
                        //把每个对应的字母加入到每个结果集合里
                        for(int k=0; k<res.size(); k++){
                                tmp.add(res.get(k)+s.charAt(j));
                        }
                }
                //保存最新结果集
                res = tmp;
        }
        return res;
    }
```

面试题 28：数组签名☆☆☆☆

给出一个含有 1 到 N 的整型数组的签名，求出其所有可能的字典最小的排列（lexicographically smallest permutation），其中 N<=9。签名是这样计算的：比较相邻元素的大小，如果后者比前者大，输出 I，反之，输出 D。

举例

输入数组的签名“DDIIDI”，排列{3，2，1，6，7，4，5}就是其中一个输出。

含有 1 到 N 的数组长度为 N，其签名长度一定为 N-1。由于数字仅用一次，我们需要记录每个数字的使用情况。输出签名里的字符要么是 D，要么是 I。D 代表下降趋向，遇到 D 时，如果前一个数字为 i，那么只能从 1 到 i-1 里挑一个数字；I 代表上升趋向，遇到 I 时，当前数字只能从 i+1 到 N 里挑出。

使用递归的方法，程序的结构是典型的 for 语句内调用递归函数。

```
    boolean computePerm(String signature, int n, int level,
    StringBuffer sb, boolean[] used){
             if(level==n-1){
                   //遍历完了signature
                   System.out.println(sb.toString());
                   return true;
             }
             //获取前一个数字
             int i= sb.charAt(sb.length()-1)-'0';
             if(signature.charAt(level)=='I'){
                   //找个最接近前一个数字，但又比它大的数字
                   for(int j=i+1; j<=n;j++){
                        if(used[j]) continue; //不能重复使用数字
                        sb.append(j);
                        used[j] = true;
                        perm(signature, n, level+1, sb, used);
                        sb.setLength(sb.length()-1); //恢复使用前
的状态
                        used[j] = false;
                   }
             }else{
                     //找个尽可能比前一个数字小的数字
                   for(int j=1; j<i; j++){
                        if(used[j]) continue;
                        sb.append(j);
                        used[j] = true;
                        perm(signature, n, level+1, sb, used);
                        sb.setLength(sb.length()-1); //恢复使用前
的状态
                        used[j] = false;
                   }
             }
             return false;
        }
```

面试题 29：组合和☆☆☆

给出一组正整数序列和一个目标值，求出所有可能的组合，使得组合里所有元素和为目标值，其中组合里的整数从输入序列里获取。要求：1）每个组合里的元素按照升序排列；2）输出组合里不含有重复的组合；3）从输入序列挑选出的整数，可以多次在组合里出现。

举例

输入整数序列{2, 3, 4, 7}，目标值为 7，那么输出的组合有：{7}, {2, 2, 3}, {3, 4}。

为了让输出元素按照升序排列，我们先对输入的数列进行排序。同样，我们使用递归的方法，程序的结构也是典型的 for 语句内调用递归函数。特别地，我们需要注意以下几点：

（1）记录输出组合的中间结果，每加入或移除一个数，则改变中间结果。

（2）记录剩余数之和，只有该值为 0 时，组合才是有效的。

（3）记录当前位置。因为序列里的数可以重复使用，所以下一次递归时，还可以从当前位置开始，这将体现在递归函数的参数里。

```
    public ArrayList<ArrayList<Integer>> combinationSum(int[]
candidates,
     int target) {
        if (candidates.length==0 || target <=0) return null;
        ArrayList<ArrayList<Integer>> resSet =
    new ArrayList <ArrayList <Integer>>();
        Arrays.sort(candidates);  //排序保证结果是有序的
```

```
        helper(candidates, 0, target, new ArrayList<Integer>(),
resSet);
        return resSet;
    }
    private void helper(int[] candidates, int start, int target,
    ArrayList<Integer> path,
    ArrayList<ArrayList<Integer>> resSet) {
        if(target < 0) return;
        if (target == 0) {
             //找到一个结果
             ArrayList<Integer> result = new
ArrayList<Integer>(path);
             resSet.add(result);
        } else {
             for (int i= start; i<candidates.length; ++i) {
     if (candidates[i]>target) break;  //后面更大的数不可能满足条件
    path.add(candidates[i]);
     helper(candidates, i, target-candidates[i], path, resSet);
    path.remove(path.size()-1); //重置
             }
        }
    }
```

扩展问题

如果序列可能有相同的元素，并且每个元素最多只能使用一次，如何处理？

比如，输入整数序列{2, 3, 4, 7}，目标值为 7，那么输出的组合没有了{2,2,3}，只有{7}和{3, 4}。这里有两个变化：1）每个元素最多只能使用一次，下次递归是不能从当前位置开始的，而是从当前位置的下一个开始；2）由于序列含有相等的元素，哪怕每个元素最多使用一次，也可能出现重复的组合，所以，为了避免输出重复组合，只取第一个相同元素加入组合里。这两个变化在程序里使用黑体标明。

```
    private void helper(int[] candidates, int start, int target,
    ArrayList<Integer> path,
    ArrayList<ArrayList<Integer>> resSet) {
        if(target < 0) return;
        if (target == 0) {
             //找到一个结果
             ArrayList<Integer> result = new
ArrayList<Integer>(path);
             resSet.add(result);
        } else {
             for (int i= start; i<candidates.length; ++i) {
     if (candidates[i]>target) break;  //后面更大的数不可能满足条件
    //避免结果集重复，只取第一个相同值加入结果中
    if(i != start && candidates[i] == candidates[i-1])
    continue;
    path.add(candidates[i]);
    helper(candidates, i+1,
    target-candidates[i], path, resSet);
    path.remove(path.size()-1); //重置
             }
        }
    }
```

面试题 30：N 皇后☆☆☆☆

> 有 N×N 大小的棋盘，放置 N 个皇后，使得任何一个皇后都无法直接吃掉其他的皇后，即任意两个皇后都不能处于同一条横行、纵行或斜线上。

这是一道经典面试题，有很多种解法。最直观的莫过于排列组合的方法。N 个皇后放置在 N×N 棋盘上，必定每行一个皇后，我们将每行中皇后的位置用 1 到 N 的数字来表示，总共有 N 行，这样就是 1 到 N 的数列排列。这种排列只满足了任意两个皇后不能同处在一行或一列上，并不能保证它们不会在一

条斜线上。

在加入数字至排列前，判断该数字是否和排列里的所有数字在“斜线”上：如果两个数字在斜线上，那么两数之差的绝对值等于其下标差的绝对值。

```
        public void NQueen(int n){
             ArrayList<Integer> sol = new ArrayList<Integer>();
             boolean used[] = new boolean[n+1]; //下标从 1 开始算
             //1 到 n 的全排列
             NQueen(n, 0, sol, used);
        }
        public void NQueen(int n, int start, ArrayList<Integer> sol,
    boolean used[]){
             if(start == n){
                   //找到一个方案
                   System.out.println(sol.toString() );
                   return;
             }
             for(int i=1; i<=n; i++){
                   if(used[i]) continue; //不能重复使用
                  if(!checkBoard(sol, i)) continue; //满足对角线限制
                   sol.add(i);
                   used[i] = true;
                   NQueen(n, start+1, sol, used); //找一下个皇后
                   sol.remove(sol.size()-1);
                   used[i] = false;
             }
        }
        public boolean checkBoard(ArrayList<Integer> sol, int q){
             for(int i=0; i< sol.size(); i++){
                   //判断当前的 q 是否和之前所有皇后在对角线上
                   if(Math.abs(i-sol.size()) ==
Math.abs(sol.get(i)-q)){
                        return false;
                   }
             }
             return true;
        }
```

第 13 章 杂项

面试题 31：实现迭代器 peek() ☆☆☆

实现含有 peek()函数的 Iterator 类。

初始化的时候，先把第一个元素提取出来保存在私有变量里，这样 peek()函数直接返回改私有变量的值即可。重写 hasNext()和 next()。next()函数返回当前保存在私有变量的第一个元素，同时求出下一个元素来替换私有变量的值。

```
public class Peekable<T> {
        public Peekable(Iterator <T> iterator) {
            _iterator = iterator;
            getTop(); //提前当前元素
        }

        public boolean hasNext() {
            return _top != null;
```

```
                }

                public T next() {
                     //空就抛出异常
                     if (_top == null) throw new
NoSuchElementException();
                     T currentTop = _top;
                     //找出下一个元素
                     getTop();
                     return currentTop;
                }

                public T peek() {
                     return _top;
                }

                private void getTop() {
                     _top = null;
                     //先取出第一个元素
                     if (_iterator.hasNext()) {
                            _top = _iterator.next();
                     }
                }

                private Iterator <T> _iterator;
                private T _top;
        }
```

面试题 32：实现复杂的迭代器☆☆☆☆

实现一种迭代器处理复合型的数据结构，至少包含 hasNext()和 next()两个函数。

复合型的数据结构定义如下：

```
public interface Data<T> {
     public boolean isCollection(); //是否为一个集合
     // 如果是集合返回集合，如果是单个元素，则返回 null
     public Collection<Data<T>> getCollection();
     // 如果是单个元素则返回该值，如果是集合，则返回 null
     public T getElement();
}
```

初始化时，将输入集合扁平展开之后放入一个列表里，即列表由单个元素组成。去嵌套化之后，next()和 hasNext()函数与普通 Iterator 类类似，需要记录当前位置，并与列表长度比较。

```
public class ComplexIterator<T> implements Iterator<T> {
     private int currIndex; //记录当前位置
     private ArrayList<T> flatColl; //扁平展开之后的集合
     public ComplexIterator(Collection<Data<T>> c) {
          currIndex=0;
          flatColl=new ArrayList<T>();
          flatElements(c);
     }
     //去除嵌套
     private void flatElements(Collection<Data<T>> c) {
          for(Data<T> item : c) {
               if(item.isCollection()) {
                    //若是集合，递归调用
                    flatElements(item.getCollection());
               } else{
                    flatColl.add(item.getElement());
               }
          }
     }
     public T next() {
          if(null==flatColl || currIndex >= flatColl.size())
               throw new NoSuchElementException();
          T t=flatColl.get(currIndex);
          currIndex++;
          return t;
```

```
        }

        public boolean hasNext() {
            return null!=flatColl&&flatColl.size()> 0&&
    currIndex< flatColl. size();
        }
    }
```

面试题 33：实现 BlockingQueue ☆☆☆

实现可支持多线程的 BlockingQueue 类。

对于存、取操作，均需加上锁。使用双端队列来存放元素。当存放元素时，如果队列满了，则需要等待直至队列有空余；取出元素时，如果队列是空的，那么等待直至队列含有新元素为止。

```
    public class BlockingQueue<T> {
        private LinkedList<T> list; //双端队列
    private int limit;//队列长度的限制
        public BlockingQueue(int limit) {
             this.limit=limit;
             list=new LinkedList<T>();
        }

        public synchronized void put(T t) throws
InterruptedException
        {
             while(list.size()==limit) {
                 try{
                     list.wait();  //队列满了，等待其他线程唤醒
                 }catch(InterruptedException e){
                      //
                 }
             }
```

```
            list.add(t);
            if(list.size() >0)    list.notifyAll(); //唤醒读取线
程
        }

        public synchronized T get() throws InterruptedException
        {
            while(list.size()==0) {
                try{
                    list.wait(); //队列为空，等待
                }catch(InterruptedException e){
                    //
                }
            }
            T t=list.pop();
            list.notifyAll(); //通知由于队列满了而等待的线程
            return t;
        }
    }
```

面试题 34：随机数产生器☆☆☆☆☆

给出一个随机数产生器 Rand7()，随机产生 1 到 7 的整数，根据该随机数产生器产生 1 到 10 的随机数。

调用一次 Rand7()还不足以产生 1 到 10 的随机数,我们可以考虑调用两次，将第一次和第二次可能的结果进行组合，如下表所示：

两次调用产生的组合有 49 种情况，如果我们抛弃 41 到 49 的组合，那么剩下有 40 种组合。这 40 种组合，我们可以理解为 1 到 10 的 10 个随机数中每个随机数出现了 4 次，即采用除以 10 取余数的做法，以至于能在同等概率下产生随机数。根据以上二维坐标定位取余数的做法，我们可以扩展至产生任意

的随机数上。

	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14
3	15	16	17	18	19	20	21
4	22	23	24	25	26	27	28
5	29	30	31	32	33	34	35
6	36	37	38	39	40	41	42
7	43	44	45	46	47	48	49

```
int rand10() {
   int row, col, idx;
   do {
     row = rand7();
     col = rand7();  //调用两次
     idx = col + (row-1)*7; //计算组合
   } while (idx > 40); //只取能在同等概率下产生随机数的组合
   return 1 + (idx-1)%10;
}
```

面试题 35：找出明星☆☆☆

给出一个函数 knows(A,B)，如果 A 认识 B，则返回 true，否则返回 false。明星定义为他不认识所有的人，但所有的人认识他。现在给出 N 个人，在线性时间内找出所有明星。

这 N 个人里面最多只有一个明星。为什么呢？如果有两个明星，那么这两个人应该彼此认识，这和明星的定义冲突。借用俩指针的做法，有一头一尾的两个指针指向第 1 到 N 个人。假设这两个指针为 A 和 B，调用 knows 函数，如果 A 认识 B，返回 true，这说明 A 不可能是明星，把 A 向后移动一位。如果函数返回 false，那么 B 也不可能是明星，B 往前移动一位。对于最后剩下的人，我们并不确定其是否为明星，那么我们从头到尾再调用 knows 函数，如果所有人都认识他，那么他就是场上的明星。如果你是球迷，而且了解淘汰赛机制，那么你可以轻而易举地解决这道题。

面试题 36：根据概率分布产生随机数☆☆☆☆

根据输入概率密度（即一维整型数组），产生随机数。例如，输入一维整型数组 w，长度为 L，则返回一个随机数，假设为 i，那么 i 在 0 与 L-1 之间，而且出现 i 的概率是 w[i]/sum(w)。

首先，根据概率密度，需要计算每个输出随机数的概率分布。假设输入 w={1，2，3，2，4}，累加 w 元素得到，P={1，3，6，8，12}。然后，调用随机函数产生 0 到 12-1 的随机数，看看该随机数落在哪个区间。假设产生的随机数为 10，那么它落在[8, 12)区间，则返回下标 4。如果是 6，它落在[6, 8)区间，则返回 3。

```
int randWithWeights(int[] w){
     int n = w.length;
     int dist[] = new int[n];
     dist[0] = w[0];
     for(int i=1; i< n; i++){
           //计算概率分布
           dist[i] = w[i] + dist[i-1];
     }
     Random random = new Random();
     //随机产生 0 到 dist[n-1]-1 的数字
     int rand = random.nextInt(dist[n-1]);
     for(int i=0; i< n; i++){
          //看看落在哪个区间
          if(rand < dist[i]) return i;
     }
     return 0;
}
```

面试题 37：随机采样☆☆☆

随机从 N 个数中取出 K 个数。假设 N 非常大。

其实这道题就是在一个数据流里随机采样，取出 K 个数。首先，把 K 个数填满，填满后，我们再考虑替换结果中的某个数。何时替换？如果需要替换，替换谁？

其次，根据数据流当前的个数，假设为 i，那么在[0, i)区间产生一个随机数 index。如果 index 比 K 小，则把结果的第 index+1 个数替换为数据流的当前元素。

```
int[] pickK(int[] A, int k){
    int buf= new int[k];
    for(int i=0; i< A.length; i++){
      if(i<k) buf[i]=A[i];
      else{
            Random rand= new Random(i);
            //随机产生 0 到 i-1 数字
            int index = rand.nextInt();
            //如果在 k 以内，替换原有的值
            if(index<k) buf[index]=A[i];
      }
    }
    return buf;
}
```

面试题 38：统计电话号码个数☆☆☆

给出 10 亿个手机号码，统计不重复的号码的个数。

如果使用字符串存储号码，大概需要 110GB 内存，才能容纳这 10 亿个手机号码，所以使用计数数组不可行。我们可以考虑使用 BitMap 来标记相应的号码是否出现（不需要统计每个号码出现的次数）。即便如此，单单使用一个 BitMap，内存还是容纳不了上亿的号码。这时候，我们考虑使用二次寻址，即双层桶概念，分区统计，把统计的中间结果存入硬盘。例如，分成一万个分区，标记时，只要导入对应的分区，这样大概需要 60MB 内存。最后，遍历整个 bitmap，统计位为 1 的个数。

更多与海量数据处理相关的数据结构参加附件 B“海量数据结构”。

面试题 39：海量数据高频词☆☆☆

给出 1GB 大小的文件，每行是一个词，内存限制在 1MB 以内，要求返回频率最高的 100 个词。

先统计每个词出现的次数，然后使用一个堆，维护频率最高的 100 个词。

（1）统计每个词出现的次数。如果内存无法容纳这么多单词，利用分布式哈希表，进行分区统计。例如，分开放置到 n 个文件，使用映射 hash(word) % n，使用哈希函数记录每个单词出现的次数，即 key 为单词，value 为次数。

（2）有了这些次数之后，使用一个堆，维护频率最高的 100 个词，把 n 个统计结果文件从头扫描一遍。最后，在堆里的单词就是高频词。

面试题 40：多台机器的中值☆☆☆☆

求存放在 N 台机器上海量数字的中值。

借用快速排序的 partition 思想，把问题转化为对这 N 台机器的数字进行分区，找出第 k （k=total/2）个数（假设总的数字个数为 total 个）。

（1）挑选某个机器作为协调员，向其他 N-1 台机器发送统计数字个数的请求。加上自身的数字个数，计算出总个数，假设为 total。

（2）协调员从自身抽取某个数字作为标杆（pivot），或者从其他机器挑选（如果本机没有进入下一轮计算的数字）。把这个数字发送给所有机器，让所有机器返回比这个数字小或相等的数字个数。累加各台机器返回的结果，假设为 m。如果 m 等于 k，则返回 pivot；如果 m 小于 k，让各台机器丢弃小于等于

pivot 的数（即这些数不进入下一轮计算），更新 k 的值，k=k-m，并重复第二步；如果 m 大于 k，让各台机器丢弃大于 pivot 的数（即不进入下一轮计算），重复以上过程。

（3）重复第二步，直至找出 m 等于 k 为止。

算法分析：假设这些海量数据较为均匀地分布在每台机器上，即每台有 O(total/N)个数。平均而言，如果是第 i 轮计算，每台机器会进行 O(total/(N*2i))比较，共有 O(log(total/N))轮计算。此外，每轮通信和统计每台机器中间结果需要 O(N)时间。总的时间复杂度为 O(total/N+N*log(total/N))，一般来说 N 远小于 log(total)，所以最终为 O(total/N)，即每台机器平均存储量的线性级别时间。

第四部分　系统设计

第 14 章 实战技巧及准备

系统设计是一类很宽泛的问题。相对于具体的算法和数据结构，系统设计问题更难准备。即使那些有多年工作经验的软件工程师在面对这类问题时，有时也会显得力不从心。下面列举了几个常见系统设计问题，都来源于真实的面试问题。

1．设计一个输入法系统，能够支持智能联想功能

2．设计一个文本搜索引擎

3．设计 Facebook news seed 功能

4．设计一个键－值 NoSQL 数据库

不是所有的公司都会在面试中问系统设计问题，但是一般针对有一定工作经验的应聘者，像 Google、Facebook、Amazon 这类公司都会提出一些系统设计的问题来考察应聘者在解决方案（solution）这个层级的设计能力。某种意义上，可以认为是在考察你作为一个架构师的潜质。不用认为架构师有多么高不

可攀和遥不可及，任何一个复杂的系统都是由不同的子系统组合而来。每一个子系统的设计人员在该领域里都扮演着架构师的角色。

架构师的核心任务并不是去设计或者实习系统的每个细节，而是应该关注与解决方案。一个好的解决方案往往比精妙的实现更重要。例如你的解决方案在完成相同的任务时，能够减少一次服务本身的重启或者中断，很有可能比具体实现上精巧的排序算法减少更多时间成本的消耗。系统设计问题往往考察应聘者的过往经验积累以及灵活应用知识（如算法、数据结构）的能力。接触过或者研究过大量优秀设计的软件项目的应聘者往往更容易做好这类问题，因为他们可以从以往成功的案例中获取灵感。因此要真正对系统设计问题做到游刃有余，更多的是需要应聘者在日常工作学习中，开阔自己的眼界，多了解一些热门软件项目的设计思路，修炼好内功，没有所谓的速成法。

14.1 实战技巧

技巧 1：不要惊慌

笔者之所以把不要惊慌作为第一条实战技巧，是因为很多人对系统设计问题有一种莫名的恐惧，一旦遇到面试官提出这类问题，在心理上已经放弃了，其实大可不必。因为系统设计问题根本就没有所谓的正确答案。这些系统往往需要整个团队工作好几周甚至好几个月，面试官怎么可能期望应聘者在 20~30 分钟内解决呢？系统设计问题更多是考察应聘者能否将现实问题抽象化，又能落地到具体计算机技术上，并且可以模块化实现的能力。既然没有标准答案，同一系统就可以有多种实现方式。只要应聘者能够向面试官清晰地展示自己的设计思路是如何满足需求的就足够。

技巧 2：与面试官积极交流

系统设计问题往往都是开放性问题。其重点并非系统本身，而是应聘者完成设计的过程。正如前面提到的，面试官并没有指望应聘者能够在 20~30 分钟中设计出完美的解决方案，而是更愿意看到应聘者在短短半个小时之内展示自己如何思考解决新的问题。正是因为系统设计并非结果导向而是过程导向，应聘者需要积极地与面试官沟通，时不时地停下来为面试官解释一下自己为什么要这么设计，这样设计带来哪些好处，又有什么可能的限制。系统设计问题没有简单的标准答案，整个面试过程应该被看作一个与面试官技术讨论和交流的过程，是一个思想交流的平台。不同面试官的风格可能不同。有些面试官喜欢追问具体实现细节，而有些面试官更喜欢不停加入新的因素和需求。应聘者时刻准备被要求提供替代方案、厘清解决性能瓶颈。

技巧 3：厘清需求

对于系统设计问题，面试官往往一开始会给出一个比较简单的描述，例如设计一个文本搜索引擎。这个问题的描述是很抽象的，没有提及任何性能需求和所面对的数据特征，如该搜索引擎适用的文档总集的规模有多大，这个因素将决定你设计的系统是使用长驻内存的数据结构还是需要借助硬盘。需求决定数据（结构），数据（结构）决定算法。所以应聘者不要一上来就跳入具体设计中，而是应该积极与面试官沟通，了解这套系统使用的目标场景是怎样的，有哪些性能上的需求。你只有针对这些具体的需求来设计系统，才会符合面试官的要求，也避免了过度设计。

需求分析能力也是系统设计问题面试过程中面试官考察的重点之一。常见的需求如：每秒请求数量、请求类型特征、数据读写速度等。以文本搜索引擎的系统需求为例。

- 搜索文档总量大小 10~30G

- 搜索需求 > 1000 次 / 秒
- 系统内存 5G

技巧 4：先框架再细节

一旦完成了需求分析的工作，你就可以开始勾勒系统的总体框架，包括系统大致划分为多少个子系统，每个子系统负责的功能是什么，以及各个子系统之间是如何协同合作的。在面试的过程中，你可以画一个简单的框图来向面试官展现这些设计。面试官通常会针对你的框架设计给出一些意见。这时还不要急急忙忙地跳入各个子系统具体的设计中，而是应该向面试官简单解释各个子系统如何协作来满足他提出的需求，并针对面试官给出的反馈修改调整系统框架。这样可以保证我们整个设计方案向正确的方向前行，不会出现重要功能缺失的错误。

完成框架设计之后，你就需要实现各个子系统。在这个阶段往往可以借鉴很多已有的成熟的设计模式或者技术，如生产者消费者模式、Map-Reduce、LRU cache 等等。但是请确保你已经熟练掌握这些知识，因为面试官很有可能对这类成熟技术很熟悉，并针对你的系统提出很具体的问题。所以系统设计问题对应聘者知识的广度和深度都提出了很高的要求，本书后面的章节，笔者罗列了一些解决系统设计问题需要的基础知识以供读者参考。

技巧 5：留意错误处理

在系统设计问题中，各种错误情况的处理，是面试官特别喜欢考察的点。例如某些子系统停止响应，其他子系统应该如何应对。又如数据完整性校验失败，系统该如何处理。这类问题很容易就能看出应聘者逻辑思维是否细致严密。这类问题通常出现在系统细节设计上，这要求应聘者在实现各个子模块时对容易出错的地方有足够的敏感，例如网络不稳定、非法数据、系统阻塞等等。这种错误易发的地方往往也是系统不稳定和性能瓶颈集中所在。而性能瓶颈是面

试官另一个喜欢挖掘的点。

系统初级版本一般都会有一些性能瓶颈制约着整个系统的处理能力或者响应速度，如硬盘读写速度的限制、网络传输延时等。例如对于海量数据，你设计的系统可能需要分布式存储以及分布式处理方式带来的数据读写延时的增加、数据同步的复杂度提高。为了解决新的问题，你可能需要增加设计辅助模块来满足新的需求。在面试的过程，如果应聘者能够按照这种方式一步步地完善自己的设计系统，其严谨的思维方式无疑会给面试官留下深刻的印象。另外一方面，在处理性能瓶颈这类问题时，应聘者往往需要综合考虑多方面的因素，平衡不同性能指标。

14.2 常见知识点

真实世界的系统是复杂的，涵盖了各式各样的技术细节。在众多真实系统中，会面对一些共同或者类似的技术难题。在你设计自己的系统中，往往也会碰到它们。下面笔者列举其中一部分。

并发问题

（1）并行运算（parallel computing）：通过对计算任务的分组和计算资源的分配来减小系统平均响应时间。例如常见的多线程编程、map-reduce、分布式计算等都属于这个范围。它可以说是设计现代大型计算系统的核心问题。

（2）数据一致性 (data consistency)：在多线程或者多个计算结点的分布式系统中，如何保持计算结果的正确性。其核心是如何协调同步不同计算资源对共享资源的访问。最常见的一个数据一致性问题是当多个请求同时尝试修改数据库中同一条记录时，如何保证结果正确。

（3）分布式系统 (distributed system)：对于海量的数据或者请求，单一服

务器是很难完全满足性能需求的。将需求进行拆分重组，然后分配给不同的结点以获得性能的提升。分布式系统可以简单看作多线程编程更高级的一种展现。但是分布式的通信问题更加复杂。例如网络延时也必须考虑在内。

网络问题

（1）数据吞吐量

（2）传输延时

性能问题

（1）计算机硬件性能指标，包括内存、机械硬盘、固态硬盘、L1/2/3 CPU缓存

（2）操作系统，包括文件系统、常见调度算法

（3）常见的缓存技术

（4）负载均衡

（5）可扩展

健壮性问题

（1）异常处理：常见的就是数据异常。不符合期望的数据格式，网络通信超时或者非法的数值。这些都是考验系统稳定性的常见问题。尽量避免不可逆转的错误发生。

（2）恢复：系统可能因为某些原因产生了不可逆转的错误时，系统应该有能力恢复到之前某个正常的状态，以继续提供后续的服务。例如，针对可能出现本地数据的丢失或者被破坏的问题，备份是常见的解决方案。

14.3 如何准备

准备系统设计问题比算法数据结构更加困难。通常我们可以借助一些在线算法测试的网站来练习。但是对于系统设计问题，其本身没有所谓的标准或者最优解答，也就没有类似的在线测试资源供应聘者平时练习。由于系统设计题重视过程不重视结果以及往往涉及一些经典技术问题，应聘者还是可以提前做一些准备工作的。笔者建议应聘者按照如下几个方面进行准备。

1．主动了解当下热门科技公司使用的热门技术：例如可以了解一下如今十分火爆的 Uber 是如何设计调度系统来实现待运车辆和叫车乘客直接的配对。很多去应聘 Uber 的程序员都曾经在面试中被要求设计一个 Uber 叫车系统。其实网上有不少介绍 Uber 的技术资料，如果有同学或者朋友在这些热门公司工作，也是一个不错学习渠道。应聘者如果平时能够花时间搜集了解这些技术背景，那么在面试的时候遇到这类问题，往往脑子就会有一个初步较为完整的思路，而不会出现听到问题就感到一头雾水不知从何说的窘境。就像笔者在面试准备章节特别强调的，日常的积累很重要。

2．利用开源软件了解热门技术细节和瓶颈：有些时候应聘者可能感觉没有一个很好的方法去了解熟悉当前很热门的技术，而这些热门技术又往往是公司系统设计问题中经常会遇到的。好的开源项目一般都有不错的设计文档和清晰简洁的代码实现，是一个非常不错的学习渠道。应聘者通过学习真实可用的系统的设计，不仅仅学到具体问题的解决方案，更应该去理解设计者如何做到各个性能指标之间的平衡，还可以问问自己，现有系统的性能瓶颈在哪里？是否还可以优化？例如 redis 就可以用来学习如何实现常驻内存 Non-SQL 数据库，数据结构如何设计，与传统的关系性数据库相比，优势在哪里等等。又如 lucene 是学习了解文本搜索引擎实现的很好的范本，了解倒排数据是如何组织压缩的，搜索引擎是如何来实时高效地处理检索请求等等。应聘者不要简简单

单地学习这些成熟项目的实现技巧，更应该站在设计者的角度思考自己会如何设计这个系统。这样就达到了练习系统设计的目的。

3．思考日常使用系统的工作原理：其实有很多优秀的系统就在我们日常生活中经常被使用，例如 Google, Facebook, Yelp（国内也有百度、微博、大众点评)等等。我们要成为勤于思考的人。这些成功的系统都凝结了无数软件工程师们智慧的结晶。当自己在享受到这些优秀系统带来的生活便利时，也把它们作为自己系统设计的知识库和练习场。例如我们可以列出身边这样系统的列表，然后根据自己的兴趣选择其中的一个作为自己的设计目标，最后按照本章第一节提到的那些技巧，自己尝试着设计一个实现同样功能的系统。在设计时，先不要急着去查资料了解已有系统是怎么实现的。系统设计重视的是过程，而不是结果。在设计过程中，我们需要选择合适的数据结构和算法，自己去尝试平衡各个性能。当我们完成设计之后，可以向朋友或者同事同学展示自己的设计并展开讨论。在这个过程中，我们可以学习到很多相关知识，也只有在实际设计过程中，我们才会真正理解原来系统的技术难点在哪里。每次完成这样一个设计不要花太长的时间，因为我们真实的面试时间只有 1 个小时，其中还包括与面试官解释沟通的时间。所以让我们把系统设计过程控制在 30 分钟之内。最后一定要注意，这些成功的系统都是经历漫长时间的演变和一群工程师通力合作的产物。不要因为自己的设计没有达到原来系统的水准而感到沮丧，也不要花太多的精力时间在其中某些细节的设定，要关注大的结构上的设计。

4．加强系统技术的学习：一个优秀的系统设计师一定具备扎实过硬的系统技术的，这样他才能在设计时充分考虑系统各个模块的性能瓶颈所在，并有针对性地进行优化。系统技术涉及的绝大部分内容已经在本章第二节罗列的经典问题中涵盖。这些技术并不是一个个枯燥的知识点，其本身也往往是一个优秀系统设计的范例。例如传统数据库如何优雅地实现事件，操作系统是如何高效管理内存的等等。一个优秀的系统不是虚无缥缈的空中楼阁，它一定是设计师在充分权衡利用已有资源的基础上产生的。

5．模拟系统设计面试：与同学同事朋友进行一场模拟的系统设计面试，让他们要求我们设计出一个满足某种需求的系统。真实的面试其实就是应聘者和面试官一起讨论设计一个系统的过程。我们应该让自己熟悉这个过程，知道如何应对别人突然提出的各种问题。“模拟考”也是验证我们准备充分与否很好的工具。在美国有很多提供专业模拟面试的服务公司，如 CareerCenter (http://career.uga.edu/interviewing/mock_interviews). 笔者建议要使用英文进行模拟考。正确清楚地使用英语表达自己的观点是面试的第一要务。

练习系统设计的关键就是要落地，不能只是停留在理论知识的学习，而是应该真正设计一个系统并解决实际的需求。我们上面罗列关于如何准备系统设计问题的几条建议都是以真实的系统和需求为核心展开。

第 15 章 系统设计例题

面试题 41：大数据存储☆☆☆☆

每天有大约 2500 万条新数据产生，每条数据的大小约 100KB。数据进入系统三周之内，可能会经常被访问。一旦超过了三周，这些数据被访问的频度大大降低，而且时间越久频度越低。现在我们拥有 10 台机器，每台机器有 10TB 容量的储存空间。设计一个系统利用这些机器储存这些系统满足上述需求。

这个设计问题源自于 Twitter、Facebook 这些每天有大量数据更新的社交网站。如何储存这些数据是这些社交网站的一个重要技术挑战。具体到这道设计题本身，我们可以按照上一章提到的技巧来完成自己的设计。

1．厘清系统的需求：每天产生数据大约 100KB × 25000000 = 2.5TB，3 周一共会产生 2.5TB × 7 × 3 = 52.5TB 的数据。只需要不到 6 周的时间，所有机器就会被数据填满，无法再接受任何数据。

作为一个能够稳定运行的储存系统，显然我们需要设计一个数据退出机制，否则系统很快就会因为太多过时的数据而彻底瘫痪。另外一方面，数据存储是为了日后的读取，因此读写性能是重要的性能需求。虽然题目的描述中并没有提到这些隐形需求，但是设计者应该能够自己分析出来。这时我们就需要跟面试官进行沟通，确定我们的理解是否正确。就如上一章里面提到的，我们需要经常与面试官进行沟通，确认自己努力方向是正确的，也可以更好地让面试官了解自己的思考过程。

2．设计系统框架：一种最直接的设计就是把 10 台机器组成一个有十个元素的环形数组，其中每个元素大小为 10TB。系统保存两个机器的地址，一个是数组的起点，保存最老的数据，另外一个是数组的队尾，用于保存最新进的数据。新进的数据将全部写入数组队尾机器。队尾机器一旦写满，下一个节点变成队尾继续接受数据。直到某一天队尾和队首相邻，提前删除队首机器中的数据，把队首标志移到下一个节点。一旦队尾机器再次写满，原来的队首机器被标记为新的队尾，新的数据将写入新的队尾中。

这样的系统设计最大的优势是简单易于实现，很容易找到最过时的数据。另外，删除过时数据和写入新数据发生在不同的机器上，彼此不会冲突影响吞吐量。最后这种设计把数据按照时间顺序储存，很适合 Twitter 这种社交网络，因为大家往往只关心最近发生的事情。作为节省空间的优化，我们还可以定期把存储最老（例如超过三周）数据的机器上数据进行压缩合并，这样就可以省出部分机器用来接受新的数据。由于数据进行了压缩，删除过期数据的工作量也会相应下降。唯一的性能损失是访问这些过期数据时需要解压，会带来一定的延时。不过我们可以利用 LRU 缓存来减少这种类型访问的平均延时。这对过期数据的访问频度并不高。这种把数据按照访问冷热程度进行分级是一种常见的设计思想。

3．分析系统性能瓶颈：这个系统的设计有最大问题，每次只有一个节点接受写请求，系统的写带宽不足。按照问题的描述，一天有 2.5TB 的数据进入

系统，平均每秒 29MB 的数据写入。对于现在的服务器而言，这个速度还是能够处理一个节点，但是在峰值时可能有点难。另外，数据具有很强的时效性，那些存储老数据的节点几乎没有什么读流量，读写流量都主要集中在队列最后几台机器上，负载很不均衡。即便这样，我们不要急于否定这个设计， Twitter 早期的系统就采用了类似的设计。我们还是应该先跟面试官沟通一下自己的设计思路，看是否已经能够满足系统需求，同时也展示自己思考的过程，很多时候别出心裁的设计虽然不是最优，但是如果能引起面试官的思考也会留下不错的印象。

针对读写带宽不能满足性能需求的问题，我们可以考虑将数据均匀分布在十台机器上，这样读写带宽一下子扩大了十倍。但是也带来新的问题，现在每台机器上都储存着新老数据，那么对老数据集中进行压缩和删除的操作时，整个系统的吞吐能力严重下降。当然一种解决方案是将这种 CPU 密集型（压缩）和 I/O 密集型（删除）的操作放到网站流量低谷的时候进行，例如深夜。不过对于 Twitter、Facebook 这类全球型的网站，其流量低谷的窗口可能很小甚至没有特别明显的流量低谷。

面试题 42：大并发处理☆☆☆☆

设计一个类似 Flickr 的在线共享图片服务

有时候面试官不会给我们太多数据指标，而是拿一些成功的实际案例来考察应聘者。当今互联网的火爆，这类问题往往集中在如何处理大并发数据，如何快速读取数据，如何快速写入数据，如何处理冲突，以及如何备份等。上述这几个问题几乎是所有大型互联网应用或者服务都必须面对和处理的。

1．厘清系统的需求：作为图片共享服务，其必然允许用户查看（读服务）和上传（写服务）图片。作为知名的在线系统，同一时间点接受的读写请求数

目一定是巨大的，QPS（Query Per Second）常常用来衡量请求流量大小的指标，对于 Flickr 这种规模的系统，一个结点一秒之内接收到几千的请求并不值得大惊小怪。

2．设计系统框架：系统框架应该按照一步步满足系统的需求的方式来设计。对于海量的数据和请求，显然分布式是不错的选择，因此我们的系统框架也是分布式的。

单一时间点允许的连接数是有限制的，这个限制可能来自于结点上 Web 服务器的设置，也可能来自于操作系统本身（同时打开 socket 的数目也是有限制的）。如果是有经验的程序员，应该知道在一般的应用场景下，写操作比读操作要慢许多，尤其我们可以通过异步、缓存等技术极大地减少读的平均延时，而对于写操作相关的技术就少了好多。如果让读写在同一结点上不加区分地处理，我们很快就会发现所有的连接都被慢的写请求所占据。整个系统的吞吐将极大地下降，因此，在我们要设计的系统中，读写服务应该是分离的，各自拥有独立的结点来响应用户的请求。我们可以通过增加结点的数目来应付巨大的流量。

另外用户上传的图片数据是海量的，我们很难想象一台服务器能够存储所有的数据。那么我们可以采用与上一个问题类似的设计，将客户图片存放在多台机器上。为了方便读取数据，我们同时需要保存一份图片编号到存储服务地址的映射。你可以选择将这份映射分散地保存到各个结点本地，也可以选择将这份映射统一保存在一个公共的结点上。如果你选择保存在本地，那么为了避免结点之间的同步，你可以让每个结点只负责特定区间的请求，彼此不要重叠。这样每个结点只用维护自己区间范围来的映射关系，而不用担心与其他结点的不一致。对于分布式系统内配置问题，元数据（meta data）的管理已经有很多不错的工具可以选用，如 zookeeper。这部分知识笔者留给读者自己去了解。

到这里，我们就有了系统的基本框架：系统基本框架应该是有多个对外结

点，彼此独立的响应客户的读/写请求。在这些结点之后是一组存储结点。

3．分析系统性能瓶颈：对于这种系统，性能瓶颈非常明显，如何最大提高系统整体的吞吐能力是关键。常见的解决方案是异步、缓存、批处理和索引。

a．异步处理：在最开始我们已经提到了可以采用异步处理请求的方式来提高系统同一时间段接收请求的数目。异步模式特别适用于那些需要访问慢速外部资源（如远端存储的数据）的请求。因为外部资源访问速度过慢，可能导致请求线程始终处于阻塞状态，此时 CPU 使用率虽然很低，但是因为所有的线程都被阻塞而无法响应新的请求。针对处理大量连接的高并发服务器，异步模式几乎是不二选择。异步模式核心是任务队列，一类线程负责接收请求，然后把请求封装成一个个的任务，最后把这些任务压入任务队列。由于封装是很轻量级的操作，因此这些线程很快就完成了自己的任务可以继续去接收新的任务。在后台有分发线程／工作线程从任务队列中取出任务来处理请求，并把结果返回给用户。

b．缓存：缓存也是常用减小延时、提高吞吐的方法。缓存利用了数据访问的临近原则，即最近访问的数据很有可能会被再次访问。我们将最近刚被访问的数据临时存储在内存，SSD 等速度高于真正数据源的地方。当请求来时，我们先查看缓存是否已有需要的数据。如果有，就是所谓的命中缓存，直接将结果返回给用户。缓存技术广泛应用于我们日常系统中，如硬盘、操作系统、服务器等。缓存往往被置于系统的最外层离请求很近，以便减少不必要的操作带来的延时。在现实的系统中，memcached 和 redis 常常被用作缓存载体。

c．索引：索引也是另外一种使我们能够快速访问数据的常用技术。我们之前提到需要维护一张图片编号到具体存储结点路径的映射，这就是一种简单的索引。索引需要消耗额外的空间以提高查找的速度，同时也会增加写操作的延时。这是因为完成写操作的时候需要额外更新索引。通常索引都是放在内存中。索引有些时候还能完成复杂的功能，搜索引擎的核心技术之一就是所谓的

倒排索引（inverted index）。

例如我们想查找某个特定用户编号最近 10 天上传的照片。一种方法是我们先通过用户编号从远端存储服务器读取用户数据，然后从用户数据中找到所有图片编号的列表，再从远端服务器读取这些图片的数据并过滤掉其中超过 10 天的部分。这个方法简单实现但是需要两次从远端读取数据并且会读入无用的数据（10 天前上传的图片）。我们其实可以利用索引来加速这类请求。在内存中，我们可以维护一张 key-value 表，键是用户编号，值是一个队列，队列中每个元素包含了用户图片编号和时间戳。这个队列是按照时间顺序排列的。那么为了获得最近 10 天的图片，我们只需要查找索引拿到队列，根据时间戳找到所有符合条件的图片，然后从远端服务器读取出来即可。

面试题 43：大数据收集☆☆☆☆

> 为了预测农作物的收成，我们在农场部署了四组不同的传感器在不停地收集相关数据。通过分析这些收集到的数据得到农作物产量的预测。请设计一系统满足上面需求。

并非所有的互联网公司都只会问互联网相关的系统设计问题。有些面试官也会提出一些比较“抽象”的问题。例如本题并没有那些大并发、高性能运算等花哨的性能需求，而只是简单地要求应聘者设计一个“略显老土”的数据分析系统。其实这些看似普通的问题才是真正意义上的系统设计问题，它不要求应聘者有多少经验，每个人都能根据自己的理解设计出系统。这种问题更关注于考察应聘者理解问题、解决问题的过程。FLAG 面试中常常会有类似的问题，本题就来自于其中一家。由于保密协议原因，笔者对问题略作修改。

回到这个问题本身，我们尝试按照前面总结的步骤来设计符合需求的系统。

1．厘清需求：如果把我们即将设计的系统作为一个黑盒，那么这个系统显然有四条输入流，有一条输出流。不要问我哪里来的输出流，题目描述没有提到。请注意这是一个数据分析预测系统，系统肯定需要把分析结果输出给用户。输入流对应着写请求，输出流则对应读请求。

题目描述很简单，并没有涉及任何计算机领域的描述。其实日常工作中，我们面对的也大部分都是这类具体的生活化的需求，而不是清楚地要求你设计一个消息队列或者一个调度算法。因此解决这类问题的先决条件就是进行抽象。

2．设计系统框架：在计算机系统中，需求决定数据，数据决定数据结构和算法。本题需求比较简单，因此框架上不需要花费太大的力气就能满足。目标系统分三个大模块，分别是数据收集、数据处理和结果输出。

3．设计系统细节：

a．数据处理模块：牢记系统的核心是数据。任何计算机系统都是在不停地把某一种数据转化为另外一种数据。在这个系统中，数据转化的过程就在数据处理模块。由于题目本身并没有提及使用何种算法来分析收集到数据，所以我们也不需要在设计时过度操心数据处理模块内部实现。我们唯一需要关心的是数据分析模块需要怎样的数据。这个细节，题目描述中并没有提及。这就要求我们去跟面试官沟通了解。笔者希望广大读者能够意识到面试和考试的区别。虽然两者都是考察我们能力的方法，但是考试过程你不能去跟考官讨论，而面试过程中往往是鼓励我们多和面试官交流。

一般的系统设计面试不会给你一张完整详细的系统需求分析报告，往往只是考官简单的几句话语的描述。这种时候，很多隐含的信息需要你自己去挖掘，需要你多思考题目背后隐含的东西。这些隐含的东西并不神秘，往往就是一些实际工作生活中你自己要面对的。关键是你在面试时能不能多想一步。《孙子兵法》有一句很精辟的话语很能概括这种情景：“多算胜，少算不胜，而况于

无算乎”。面试中，要想象自己真的在解决一个实际客户的需求而不是在考试答题。这也就是系统设计问题难的地方，不是知识点的考察，而是某种意义上方法论的考察。

让我们还是回到问题本身。经过沟通，面试官会告诉你：这个分析算法一次需要至少 10 天的 4 组传感器的数据才能进行分析，而且运算的过程比较耗时。这句话里面包含了两个很重要的信息：1）分析过程比较耗时意味着我们不能进行实时的运算，也不能进行过于频繁的运算。2）分析算法需要一次性得到完整的数据之后才能进行分析。我们需要保证数据的完整性后才能交付分析算法进行计算，避免出现无效数据导致计算资源的浪费。

我们在前面已经提到了，容错处理在系统设计中需要时刻放在心上，也是面试官最喜欢考察的点。另外在本题中，传感器这种硬件的存在大大提高了数据不完整的可能性，所以应聘者必须对这种薄弱点足够敏感。这种敏感来自于你自己日常的积累和经验。理清了这些，也就弄明白了数据收集模块和数据分析模块之间的数据通信格式。至于数据处理模块本身，以简单归纳为一个生产者消费者模式架构。数据收集模块把通过完整性测试的数据封装成一个任务对象，将其压入任务队列中，等待后台数据处理线程处理。这样我们就基本完成了数据处理模块的设计。

b．数据收集模块：根据前面的描述，分析算法需要 10 天 4 组传感器收集的所有数据。因此数据收集模块需要判断收集的数据是否符合这种对时效性的要求。其实很简单，为每一组传感器设计一个队列，例如传感器 A 对应队列 A，传感器 B 对应队列 B，依次类推。当数据收集模块接收到传感器 A 来的数据时，将数据和当前时间一起封装好压入队列。由于有可能在传输的过程中发生数据丢失或者某个传感器发生硬件故障，数据收集模块需要能够及时发现这类情况的发生。一个最简单但实用的方法就是增加超时机制。我们可能设置一个可配置的超时参数，如果某一个队列因为某种原因超过一定时间没有收到新数据，我们就认为现在已收集数据的完整性被破坏。收集模块可以简单地把所有

队列的数据丢弃掉。这样就能保证队列中始终都是完整的数据。然后当收集超过 10 天的数据，就可以将这些队列中的数据一次性大包好发送给数据处理模块。这样一个系统基本设计完毕。

由于传感器的数据可能因为某种网络因素被延时送达，在现实网络中，我们经常遇到类似的情况。在当前系统中，所有已搜集的数据都会因为一个小小的延时而被丢弃。这个成本可能有点太高了。笔者将这个问题留给读者们，自己偿试写写如何升级系统，更高效地处理这种情况。请记住，只要你的方法能够解决问题就足够，不要担心它看上去有多蠢或者不够炫，能稳定工作的系统才是好的系统。

系统知识阅读

系统设计问题各式各样，我们很难在一本书里罗列出所有类型。授人以鱼，不如授人以渔。有很多很不错的网站或者论坛都会定期更新实现热门技术或者公司的文章。这些都是很好的阅读材料，可以用来提高自己对复杂系统设计的感觉。在这里笔者为广大读者推荐一些不错的内容。

1．Worker, Queues and Cache：这段视频讲述了如何使用工作进程、消息队列和缓存使软件变得可扩展。http://www.infoq.com/presentations/workers-queues-cache

2．Fast Queries via Approximation Algorithm：这段视频讲述如何使用近似算法提高数据的处理速度，很有启发性的报告。很多时候，我们其实不需要那么精确或者说绝对正确的结果，例如搜索引擎。http://www.infoq.com/presentations/ approximation-algorithms

3．Spring Batch Performance Tuning：这段视频讲述了如何一步步提升系统

批处理的能力。这个过程很好地展现了真实工作中，工程师们如何设计优化自己的系统。这个过程正是我们在程序设计面试中需要模拟的。http://www.infoq.com/presentations/spring- batch-xd-tuning

4．How NGINX Achieves Performance and Scalability：一篇介绍 nginx 的技术文章，文章名字很好地解释了其内容。作为当前最流行的 Web 服务器之一，其本身就集中了很多优秀的架构设计和技术实现。就像笔者在前面提到的，多了解这些优秀的项目，你自身的设计水平也会潜移默化地受到影响。http://www.infoq.com/news/2015/06/ nginx- design-performance-scale-

5．YouTubeArchitecture：很不错的阅读材料，上面简明扼要地解释了 YouTube 的基本框架要点。我们不能指望作者会为你一点一滴地展示 YouTube 框架内部的细节，但是文章提到了很多具体的问题和解决方法。这些都是我们在系统设计时可以借鉴的好素材。文章本身也提供了很不错的相关阅读链接。http://highscalability.com/youtube- architecture

6．Scaling Twitter: Making Twitter 10000 Percent Faster：Twitter 是很典型的互联网公司，具有大并发请求、海量数据处理等特点。这篇文章提到了很多当前常见的互联网技术，如常驻内存数据库、各种缓存的组合、数据分片等。http://highscalability.com/scaling- twitter-making-twitter-10000-percent-faster

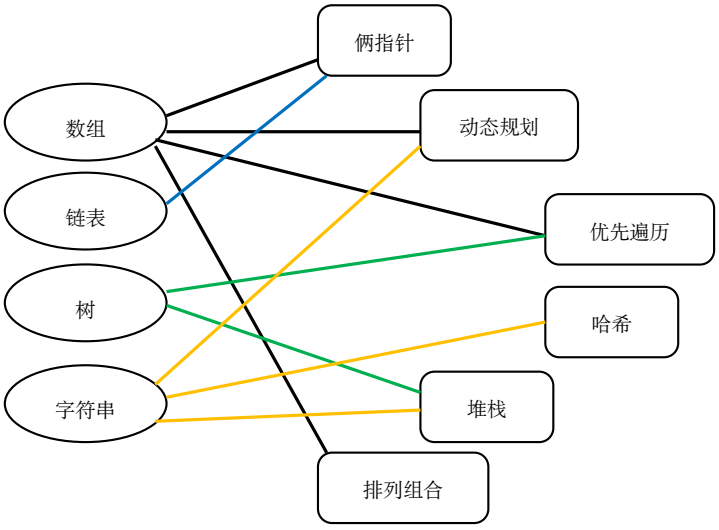
7．17 Techniques Used To Scale Turntable.Fm And Labmeeting To Millions Of Users：文章总结了 17 条设计实现高效系统的经验，每条都不长但是确实是作者多年工作经验的总结。例如其中一条是“Cache answers to expensive computations.”http://highscalability.
com/blog/2011/9/26/17-techniques-used-to-scale-turntablefm-and-labmeeting-to-mi
.html

8．8 Commonly Used Scalable System Design Patterns：文章很短，总结了一下常见设计可扩展系统的诀窍，提到很多东西都值得读者拓展开来。http://highscalability.com /blog/2010/12/1/8-commonly-used-scalable- system-design-patterns.html

9．ZooKeeper - A Reliable, Scalable Distributed Coordination System：分布式系统设计时往往会遇到一个棘手的问题，如何让各个节点进行协作，分享配置信息。ZooKeeper 就是为此而诞生的。在现在很多真实的分布式系统中，我们都能看到 ZooKeeper 身影。当你设计的系统面对类似问题，ZooKeeper 说不定就是一条不错的捷径。http://highscalability.com/blog/2008/7/15/ zookeeper-a-reliable-scalable-distributed-coordination-syste.html

附录 A

数据结构与算法



附 录 B

海量数据结构

数据结构	应用场景	示　　例
哈希表	要求所有键值对放入内存；查找可在常数时间内完成。	• 提取某日访问百度次数最多的 IP • 统计不同电话号码的个数
堆	插入和堆调整需要 O(logN)时间，N 为堆元素的个数，而获取堆顶元素只需要常数时间。	• 求出海量数据前 K 大的数 • 求出海量数据流的中位数
BitMap	通常记录整数出现情况，用来快速查找、数字判重、删除元素等。	• 统计不同电话号码的个数 • 2.5 亿个整数中查出不重复的整数的个数
双层桶	两次寻址方式以节省内存，通常用在求第 K 大、中位数和数字判重。	• 5 亿个整数找出中位数 • 海量数据的第 K 大的值
反向索引（Inverted Index）	通过单词—文档，属性—实体建索引，方便后续查找。	• 基于关键词的搜索 • 搜索框输入的自动补全

续表

数据结构	应用场景	示　　例
外排	借用硬盘空间实现海量数据的排序。	• 1GB 大小的文件，每行是一个词，内存 1MB，返回频率最高的 100 个词
前缀树（Trie）	为集合内所有单词建立前缀树。	• 求出热门的查询字符串 • 求重复率高的词
MapReduce	分布式处理，将数据交给不同机器去处理，划分数据，然后归约结果。	• 海量日志分析 • 数据挖掘 • 智能推荐系统