前端程序员 >

面试笔试真题与解析

< / > 在这里 /
有技术大咖面试笔试经验与技巧的提炼与总结；
< / > 在这里 /
有各大IT名企高频的前端面试笔试真题；
< / > 在这里 /
有作者团队庖丁解牛式的解析与答案。

猿媛之家 / 组编　平文 楚秦 等 / 编著

PROGRAMMER

> INTERVIEW QUESTIONS AND ANALYSIS

本书覆盖了近3年前端程序员面试笔试中超过98%的高频考题

当你细细品读完本书后，各类企业的offer将任由你挑选

一 书 在 手 / 工 作 不 愁 > .

机械工业出版社
CHINA MACHINE PRESS

前端程序员面试笔试真题与解析

猿媛之家　组编
平　文　楚　秦　等编著

机械工业出版社

本书针对当前各大 IT 企业面试笔试中的特点与侧重点，精心挑选了 3 年来近百家典型 IT 企业的前端面试笔试真题。这些企业涉及的业务包括系统软件、搜索引擎、电子商务、手机 APP、安全关键软件等，所提供的前端面试笔试真题非常具有代表性与参考性。同时，本书对这些题目进行了合理的划分与归类，并且对其进行了庖丁解牛式的分析与讲解，针对试题中涉及的部分重难点问题，本书都进行了适当的扩展与延伸，力求对知识点的讲解清晰而不紊乱，全面而不啰嗦，使读者不仅能够通过本书获得求职的知识，还能更有针对性地进行求职准备，最终收获一份满意的工作。

本书是一本计算机相关专业毕业生面试、笔试的求职用书，同时也适合期望在计算机软、硬件行业大显身手的计算机爱好者阅读。

图书在版编目（CIP）数据

前端程序员面试笔试真题与解析 / 猿媛之家组编；平文等编著. —北京：机械工业出版社，2018.12

ISBN 978-7-111-61819-5

Ⅰ. ①前… Ⅱ. ①猿… ②平… Ⅲ. ①程序设计—资格考试—题解 Ⅳ. ①TP311.1-44

中国版本图书馆 CIP 数据核字（2019）第 034586 号

机械工业出版社（北京市百万庄大街 22 号　邮政编码 100037）

策划编辑：时　静　　责任编辑：时　静

责任校对：张艳霞　　责任印制：张　博

三河市宏达印刷有限公司印刷

2019 年 3 月第 1 版 • 第 1 次印刷

184mm×260mm • 19.5 印张 • 474 千字

0001－3000 册

标准书号：ISBN 978-7-111-61819-5

定价：69.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务

服务咨询热线：（010）88361066

读者购书热线：（010）68326294

（010）88379203

网络服务

机 工 官 网：www.cmpbook.com

机 工 官 博：weibo.com/cmp1952

教育服务网：www.cmpedu.com

金　书　网：www.golden-book.com

前　言

程序员求职始终是当前社会的一个热点，而市面上有很多关于程序员求职的书籍，例如《程序员代码面试指南》（左程云著）、《剑指 offer》（何海涛著）、《程序员面试笔试宝典》（何昊编著）、《Java 程序员面试笔试宝典》（何昊等编著）、《编程之美》（《编程之美》小组著）和《编程珠玑》（Jon Bentley 著）等。它们都是针对基础知识的讲解，各有侧重点，而且在市场上反映良好。但是，我们发现，当前市面上没有一本专门针对前端程序员面试笔试真题的分析与讲解的书。很多读者朋友们向我们反映，他们经过精心准备后，感觉自己什么知识都会了，但又感觉自己什么都不会，不知道自己是否真的能够在程序员面试笔试中得心应手，心里一点底都没有。他们偶尔会在网上搜索一些 IT 企业的面试笔试真题，但这些题大都七拼八凑，毫无系统性可言，而且答案简单，也没有详细的讲解，这就导致读者做完这些真题之后，根本就不知道自己做得是否正确。如果下一次这个题目再次被考查，自己还是不会。更有甚者，网上的答案还有可能是错误的，误导读者。

针对这种情况，我与我的创作团队（猿媛之家）经过精心准备，从互联网上的海量面试笔试真题中，选取了当前典型企业（包括微软、谷歌、百度、腾讯、阿里巴巴、360 和小米等）的面试笔试真题，挑选出其中最典型、考查频率最高、最具代表性的真题，做到难度适宜，兼顾各层次读者的需求，同时对真题进行知识点的归类，做到层次清晰、条理分明、答案简单明了，最终形成了这本《前端程序员面试笔试真题与解析》。本书特点鲜明，所选真题以及写作手法具有以下特点：

第一，考查率高。本书中所选真题全是前端程序员面试笔试常考点，例如 JavaScript、HTML、CSS、网络、数据结构和算法等。

第二，行业代表性强。本书中所选真题全部来自于典型知名企业，它们是行业的风向标，代表了行业的高水准，其中绝大多数真题因为题目难易适中，而且具有非常好的区分度，通常会被众多小企业全盘照搬，因而具有代表性。

第三，答案详尽。本书对每一道题目都有非常详细的解答，庖丁解牛，不只是告诉读者答案，还以示例佐证（代码可以从 https://github.com/pwstrick/FEA-Code 下载）。笔者坚持授之以鱼的同时还要授之以渔，所以也通过大量延伸性的详细讲解让读者能轻松应对同类型问题。

第四，分类清晰、条理分明。本书对各个知识点都进行了归类，有利于读者针对个人实际情况做到有的放矢，重点把握。

由于图书的篇幅所限，我们无法将所有的前端程序员面试笔试真题都写在书中，鉴于此，我们在官方网站（www.yuanyuanba.com）上提供了一个读者交流平台，读者朋友们可以在该网站上上传各类面试笔试真题，也可以查找到自己所需要的知识，同时，读者朋友们也可以向本平台提供当前最新、最热门的程序员面试笔试题、面试技巧等相关材料。除此以外，我

们还建立了公众号：猿媛之家，作为对外消息发布平台，以期最大限度地满足读者需要。欢迎读者关注来探讨新技术。

回首，我在相关技术岗位上已经有多年的经验，一直想把自己的心得分享给大家，因为一个机缘巧合促成了这本书的面世。我要感谢与出版本书有关的人，因为有你们，我才能坚持下来，完成整本书的编撰。

首先，感谢机械工业出版社时静给我写作的机会。其次，要感谢的是楚秦，他不但让我加入到这项工作中，还帮我审阅了整本书，并对文字和代码进行了矫正和润色，为改进本书提供了许多宝贵的建设性意见，这些建议与意见极大地提高了本书的质量。

除此之外，也感谢那些给予我热情帮助的人，从他们那里亦得到了很多非常好的建议，这些人是（按姓氏首字母排列）：陈安阳、陈曼杰、陈涛、潘义璠、沈哲俊、王春明、王汝婷、夏丽、赵茹林、周捷、周山、周晶。

最后，我要感谢我的家人，他们是我生命中最重要的人，感谢他们对我的理解和鼓励。尤其要感谢我的爱妻，一直陪伴在我身边，在我感到困难的时候支持我、鼓励我，为我营造了一个安心、舒适的写作环境，让我有信心完成整本书的写作。

由于编者水平有限，书中不足之处在所难免，还望读者见谅。读者如果发现问题或是有此方面的困惑，都可以通过邮箱 yuancoder@foxmail.com 联系我们。

平　文

2018 年 10 月于上海松江

目　录

面试笔试经验技巧篇

想找到一份程序员的工作，一点技术都没有显然是不行的，但是，只有技术也是不够的。面试笔试经验技巧篇主要针对程序员面试笔试中遇到的18个常见问题进行深度解析，并且结合实际情景，给出了较为合理的参考答案以供读者学习与应用。掌握这18个问题的解答精髓，对求职者大有裨益。

经验技巧 1　如何巧妙地回答面试官的问题?

所谓“来者不善，善者不来”，在程序员面试中，求职者不可避免地需要回答面试官各种刁钻、犀利的问题。回答面试官的问题千万不能简单地回答“是”或者“不是”，而应该具体分析“是”或者“不是”的原因。

回答面试官的问题是一门很深的学问。那么，面对面试官提出的各类问题，如何才能条理清晰地回答呢？如何才能让自己的回答不至于撞上枪口呢？如何才能让自己的回答结果令面试官满意呢？

谈话是一门艺术，回答问题也是一门艺术，同样的话，不同的回答方式，往往会产生不同的效果，甚至是截然相反的效果。在此，编者提出以下几点建议，供读者参考。

首先回答问题务必谦虚谨慎。既不能让面试官觉得自己很自卑，唯唯诺诺，也不能让面试官觉得自己自负，应该通过问题的回答表现出自己自信从容、不卑不亢的一面。例如，当面试官提出“你在项目中起到了什么作用”的问题时，如果求职者回答：我完成了团队中最难的工作，此时就会给面试官一种居功自傲的感觉，而如果回答：我完成了文件系统的构建工作，这个工作被认为是整个项目中最具有挑战性的一部分内容，因为它几乎无法重用以前的框架，需要重新设计。这种回答不仅不傲慢，反而有理有据，更能打动面试官。

其次，回答面试官的问题时，不要什么都说，要适当地留有悬念。人一般都有猎奇的心理，面试官自然也不例外，而且，人们往往对好奇的事情更有兴趣、更加偏爱，也更加记忆深刻。所以，在回答面试官问题时，切记说关键点而非细节，说重点而非和盘托出，通过关键点，吸引面试官的注意力，等待他们继续“刨根问底”。例如，当面试官对你的简历中一个算法问题有兴趣，希望了解时，可以如下回答：我设计的这种查找算法，对于 80%以上的情况，都可以将时间复杂度从 O(n)降低到 O(log n)，如果您有兴趣，我可以详细给您分析具体的细节。

最后，回答问题要条理清晰、简单明了，最好使用“三段式”方式。所谓“三段式”，有点类似于中学作文中的写作风格，包括“场景/任务”“行动”“结果”三部分内容。以面试官提的问题“你在团队建设中，遇到的最大挑战是什么”为例，第一步，分析场景/任务：在我参与的一个 ERP 项目中，我们团队一共四个人，其他三个人中，两个人能力很强，人也比较好相处，但有一个人却不太好相处，每次我们小组讨论问题的时候，他都不太爱说话，也很少发言，分配给他的任务也很难完成。第二步，分析行动：为了提高团队的综合实力，我决定找个时间和他好好单独谈一谈。于是我利用周末时间，约他一起吃饭。吃饭的时候，顺便讨论了一下我们的项目。我询问了一些项目中他遇到的问题，通过他的回答，我发现他并不懒，也不糊涂，只是对项目不太了解，缺乏经验，缺乏自信，所以越来越孤立，越来越不愿意讨论问题。为了解决这个问题，我尝试着把问题细化到他可以完成的程度，从而建立起他的自信心。第三步，分析结果：他是小组中水平最弱的人，但是，慢慢地，他的技术变得越来越厉害了，也能够按时完成安排给他的工作了，人也越来越自信了，也越来越喜欢参与我们的讨论，并发表自己的看法，我们也都愿意与他一起合作了。“三段式”回答的一个最明显的好处就是条理清晰，既有描述，也有结果，有根有据，让面试官一目了然。

回答问题的技巧是一门大学问。求职者完全可以在平时的生活中加以练习，提高自己与人沟通的技能，等到面试时，自然就得心应手了。

经验技巧 2　如何回答技术性的问题?

程序员面试中，面试官会经常询问一些技术性的问题，有的问题可能比较简单，都是历年的笔试面试真题，求职者在平时的复习中会经常遇到，应对自然不在话下。但有的题目可能比较难，来源于 Google、Microsoft 等大企业的题库或是企业自己为了招聘需要设计的题库，求职者可能从来没见过或者从来都不能完整地、独立地想到解决方案，而这些题目往往又是企业比较关注的。

如何能够回答好这些技术性的问题呢？编者建议：会做的一定要拿满分，不会做的一定要拿部分分。即对于简单的题目，求职者要努力做到完全正确，毕竟这些题目，只要复习得当，完全回答正确一点问题都没有（编者认识的一个朋友据说把《编程之美》《编程珠玑》《程序员面试笔试宝典》上面的技术性题目与答案全都背得滚瓜烂熟了，后来找工作简直成了“offer 杀器”，完全就是一个 Bug，无解了）；对于难度比较大的题目，不要惊慌，也不要害怕，即使无法完全做出来，也要努力思考问题，哪怕是半成品也要写出来，至少要把自己的思路表达给面试官，让面试官知道你的想法，而不是完全回答不会或者放弃，因为面试官很多时候除了关注你的独立思考问题的能力以外，还会关注你技术能力的可塑性，观察求职者是否能够在别人的引导下去正确地解决问题，所以，对于你不会的问题，他们很有可能会循序渐进地启发你去思考，通过这个过程，让他们更加了解你。

一般而言，在回答技术性问题时，求职者大可不必胆战心惊，除非是没学过的新知识，否则，一般都可以采用以下六个步骤来分析解决。

（1）勇于提问

面试官提出的问题，有时候可能过于抽象，让求职者不知所措，或者无从下手。所以，对于面试中的疑惑，求职者要勇敢地提出来，多向面试官提问，把不明确或二义性的情况都问清楚。不用担心你的问题会让面试官烦恼，影响你的面试成绩，相反，这样做还会对面试结果产生积极影响：一方面，提问可以让面试官知道你在思考，也可以给面试官一个心思缜密的好印象；另一方面，方便自己对问题的解答。

例如，面试官提出一个问题：设计一个高效的排序算法。求职者可能丈二和尚摸不到头脑，排序对象是链表还是数组？数据类型是整型、浮点型、字符型还是结构体类型？数据基本有序还是杂乱无序？数据量有多大，1000 以内还是百万以上个数？此时，求职者大可以将自己的疑问提出来，问题清楚了，解决方案自然也就出来了。

（2）高效设计

对于技术性问题，如何才能打动面试官？完成基本功能是必需的，仅此而已吗？显然不是，完成基本功能顶多算及格水平，要想达到优秀水平，还应该考虑更多的内容，以排序算法为例：时间是否高效？空间是否高效？数据量不大时也许没有问题，如果是海量数据呢？是否考虑了相关环节，例如数据的“增删改查”？是否考虑了代码的可扩展性、安全性、完整性以及鲁棒性？如果是网站设计，是否考虑了大规模数据访问的情况？是否需要考虑分布式系统架构？是否考虑了开源框架的使用？

（3）伪代码先行

有时候实际代码会比较复杂，上手就写很有可能会漏洞百出、条理混乱，所以，求职者可以首先征得面试官的同意，在编写实际代码前，写一段伪代码或者画好流程图，这样做往往会让思路更加清晰明了。

切记在写伪代码前要告诉面试官，否则他们很有可能对你产生误解，认为你只会纸上谈兵，实际编码能力却不行。只有征得了他们的允许，方可先写伪代码。

（4）控制节奏

如果是算法设计题，面试官都会给求职者一个时间限制用以完成设计，一般为 20min 左右。完成得太慢，会给面试官留下能力不行的印象，但完成得太快，如果不能保证百分百正确，也会给面试官留下毛手毛脚的印象，速度快当然是好事情，但只有速度，没有质量，速度快根本不会给面试加分。所以，编者建议，回答问题的节奏最好不要太慢，也不要太快，如果实在是完成得比较快，也不要急于提交给面试官，最好能够利用剩余的时间，认真检查一些边界情况、异常情况及极性情况等，看是否也能满足要求。

（5）规范编码

回答技术性问题时，多数都是纸上写代码，离开了编译器的帮助，求职者要想让面试官对自己的代码一看即懂，除了字迹要工整，不能龙飞凤舞以外，最好严格遵循编码规范，注意函数变量命名、换行缩进、语句嵌套和代码布局等，同时，代码设计应该具有完整性，保证代码能够完成基本功能、输入边界值能够得到正确输出、对各种不合规范的非法输入能够做出合理的错误处理，否则，写出的代码即使

无比高效，面试官也不一定看得懂或者看起来非常费劲，这些对面试成功都是非常不利的。

（6）精心测试

在软件界，有一个共识：任何软件都有 bug。但不能因此就纵容自己，允许代码错误百出。尤其是在面试过程中，实现功能也许并不十分困难，困难的是在有限的时间内设计出的算法，各种异常是否都得到了有效的处理，各种边界值是否都在算法设计的范围内。

测试代码是让代码变得完备的高效方式之一，也是一名优秀程序员必备的素质之一。所以，在编写代码前，求职者最好能够了解一些基本的测试知识，做一些基本的单元测试、功能测试、边界测试以及异常测试。

在回答技术性问题时，注意在思考问题的时候，千万别一句话都不说，面试官面试的时间是有限的，他们希望在有限的时间内尽可能地去了解求职者，如果求职者坐在那里一句话不说，不仅会让面试官觉得求职者技术水平不行，还会觉得求职者思考问题能力以及沟通能力可能都存在问题。

其实，在面试时，求职者往往会存在一种思想误区，把技术性面试的结果看得太重要了。面试过程中的技术性问题，结果固然重要，但也并非最重要的内容，因为面试官看重的不仅仅是最终的结果，还包括求职者在解决问题的过程中体现出来的逻辑思维能力以及分析问题的能力。所以，求职者在与面试官的博弈中，要适当地提问，通过提问获取面试官的反馈信息，并抓住这些有用的信息进行辅助思考，从而博得面试官的欢心，提高面试的成功率。

经验技巧 3　如何回答非技术性问题?

评价一个人的能力，除了专业能力，还有一些非专业能力，如智力、沟通能力和反应能力等，所以在 IT 企业招聘过程的笔试面试环节中，并非所有的笔试内容都是 C/C++/Java、数据结构与算法及操作系统等专业知识，也包括其他一些非技术类的知识，如智力题、推理题和作文题等。技术水平测试可以考查一个求职者的专业素养，而非技术类测试则更加强调求职者的综合素质，包括数学分析能力、反应能力、临场应变能力、思维灵活性、文字表达能力和性格特征等内容。考查的形式多种多样，但与公务员考查相似，主要包括行政职业能力测验（简称“行测”）（占大多数）、性格测试（大部分都有）、应用文和开放问题等内容。

每个人都有自己的答题技巧，答题方式也各不相同，以下是一些相对比较好的答题技巧（以行测为例）：

（1）合理有效的时间管理。由于题目的难易不同，所以不要对所有题目都“绝对公平”、都“一刀切”，要有轻重缓急，最好的做法是不按顺序回答。行测中有各种题型，如数量关系、图形推理、应用题、资料分析和文字逻辑等，不同的人擅长的题型是不一样的，因此应该首先回答自己最擅长的问题。例如，如果对数字比较敏感，那么就先答数量关系题。

（2）注意时间的把握。由于题量一般都比较大，可以先按照总时间/题数来计算每道题的平均答题时间，如 10s，如果看到某一道题 5s 后还没思路，则马上放弃。在做行测题目的时候，以在最短的时间内拿到最多分为目标。

（3）平时多关注图表类题目，培养迅速抓住图表中各个数字要素间相互逻辑关系的能力。

（4）做题要集中精力，只有集中精力、全神贯注，才能将自己的水平最大限度地发挥出来。

（5）学会关键字查找，通过关键字查找，能够提高做题效率。

（6）提高估算能力，有很多时候，估算能够极大地提高做题速度，同时保证正确率。

除了行测以外，一些企业非常相信个人性格对入职岗位的影响，所以都会引入相关的性格测试题来测试求职者的性格特性，看其是否适合所应聘的职位。大多数情况下，只要按照自己的真实想法选择就行了，千万不要弄巧成拙。因为测试是为了得出正确的结果，所以大多测试题前后都有相互验证的题目。如果求职者自作聪明，选择该职位可能要求的性格选项，则很可能导致测试前后不符，这样很容易让企业发现你是个不诚实的人，从而首先予以筛除。

经验技巧 4　如何回答快速估算类问题?

有些大企业的面试官，总喜欢出一些快速估算类问题，对他们而言，这些问题只是手段，不是目的，能够得到一个满意的结果固然是他们所需要的，但更重要的是通过这些题目，他们可以考查求职者的快速反应能力以及逻辑思维能力。由于求职者平时准备的时候可能对此类问题有所遗漏，一时很难想起解决的方案。而且，这些题目乍一看确实是毫无头绪，无从下手，完全就是为难求职者的，其实求职者只要从惊慌失措中冷静下来，稍加分析，也就那么回事。因为此类题目比较灵活，属于开放性试题，一般没有标准答案，只要弄清楚了回答要点，分析合理到位，具有说服力，能够自圆其说，就是正确答案，一点都不困难。

例如，面试官可能会问这样一个问题："请你估算一下一家商场在促销时一天的营业额"。求职者不是统计局官员，也不是开商场的，如何能够得出一个准确的数据呢？即使求职者是商场的大当家，也不可能弄得清清楚楚明明白白吧？

难道此题就无解了吗？其实不然，本题只要能够分析出一个概数就行了，不一定要精确数据，而分析概数的前提就是做出各种假设。以该问题为例，可以尝试从以下思路入手：从商场规模、商铺规模入手，通过每平方米的租金，估算出商场的日租金，然后根据商铺的成本构成，得到全商场日均交易额，再考虑促销时的销售额与平时销售额的倍数关系，乘以倍数，即可得到促销时一天的营业额。具体而言，包括以下估计数值：

（1）以一家较大规模商场为例，商场一般按 6 层计算，每层大约长 100m，宽 100m，合计 60000m^2 的面积。

（2）商铺规模占商场规模的一半左右，合计 30000m^2。

（3）商铺租金约为 40 元/ m^2，估算出年租金为 40×30000×365=4.38 亿元。

（4）对商户而言，租金一般占销售额 20%左右，则年销售额为 4.38 亿元×5=21.9 亿元。计算平均日销售额为 21.9 亿/365=600 万元。

（5）促销时的日销售额一般是平时的 10 倍，所以大约为 600 万元×10=6000 万元。

此类题目涉及面比较广，例如：估算一下北京小吃店的数量。估算一下中国在过去一年方便面的市场销售额是多少。估算一下长江的水的质量。估算一下一个行进在小雨中的人 5min 内身上淋到的雨的质量。估算一下东方明珠电视塔的质量。估算一下中国去年一年一共用掉了多少块尿布。估算一下杭州的轮胎数量。但一般都是即兴发挥，不是哪道题记住答案就可以应付得了的。遇到此类问题，一步步抽丝剥茧，才是解决之道。

经验技巧 5　如何回答算法设计问题?

程序员面试中的很多算法设计问题，都是历年来各家企业的"炒现饭"，无论求职者以前对算法知识学习得是否扎实，理解得是否深入，只要面试前买本《程序员面试笔试宝典》（机械工业出版社出版），学习上一段时间，牢记于心，应付此类题目都应该完全没有问题。但遗憾的是，很多世界级知名企业也深知这一点，如果纯粹是出一些毫无技术含量的题目，对于考前"突击手"而言，可能会占尽便宜，但对于那些技术好的人而言是非常不公平的。所以，为了把优秀的求职者与一般的求职者更好地区分开来，他们会年年推陈出新，并倾向于出一些有技术含量的"新"题，这些题目以及答案，不再是以前的陈谷子烂芝麻了，而是经过精心设计的好题。

在程序员面试中，算法的地位就如同是 GRE 或托福考试在出国留学中的地位一样，是必须掌握但不是最重要的，它只是众多考核中的一个而已，不一定就能决定求职者的生死。但并不是说不用准备算法知识了，因为算法知识回答得好，必然会成为面试的加分项，对于求职成功，有百利而无一害。那么如何应对此类题目呢？很显然，编者不可能将此类题目在《程序员面试笔试宝典》中一一解答，一来由

于内容众多，篇幅有限，二来也没必要，今年考过了，以后一般就不会再考了，不然还是没有区分度。编者以为，靠死记硬背肯定是行不通的，解答此类算法设计问题，需要求职者具有扎实的基本功以及良好的运用能力，编者无法左右求职者的个人基本功以及运用能力，因为这些能力需要求职者“十年磨一剑”地苦学，但编者可以提供一些比较好的答题方法和解题思路，以供求职者在面试时应对此类算法设计问题。“授之以鱼不如授之以渔”。

（1）归纳法

此方法通过写出问题的一些特定的例子，分析总结其中一般的规律。具体而言就是通过列举少量的特殊情况，经过分析，找出一般的关系。例如：某人有一对兔子饲养在围墙中，如果它们每个月生一对兔子，且新生的兔子在第二个月后也是每个月生一对兔子，问一年后围墙中共有多少对兔子？

使用归纳法解答此题，首先想到的就是第一个月有多少对兔子，第一个月的时候，最初的一对兔子生下一对兔子，此时围墙内共有两对兔子。第二个月仍是最初的一对兔子生下一对兔子，共有 3 对兔子。到第三个月除最初的兔子新生一对兔子外，第一个月生的兔子也开始生兔子，因此共有 5 对兔子。通过举例，可以看出，从第二个月开始，每一个月兔子总数都是前两个月兔子总数之和，$U_{n+1}=U_n+U_{n-1}$，一年后，围墙中的兔子总数为 377 对。

此种方法比较抽象，也不可能对所有的情况进行列举，所以，得出的结论只是一种猜测，还需要进行证明。

（2）相似法

此方法考虑解决问题的算法是相似的。如果面试官提出的问题与求职者以前用某个算法解决过的问题相似，此时就可以触类旁通，尝试改进原有算法来解决这个新问题。通常情况下，此种方法都会比较奏效。

例如，实现字符串的逆序打印，也许求职者从来就没遇到过此问题，但将字符串逆序肯定在求职准备的过程中是见过的。将字符串逆序的算法稍加处理，即可实现字符串的逆序打印。

（3）简化法

此方法首先将问题简单化，例如改变一下数据类型、空间大小等，然后尝试着解决简化后的问题，一旦有了一个算法或者思路可以解决这个简化的问题，再将问题还原，尝试着用此类方法解决原有问题。

例如，在海量日志数据中提取出某日访问 xxx 网站次数最多的 IP。很显然，由于数据量巨大，直接进行排序不可行，但如果数据规模不大，采用直接排序不失为一种好的解决方法。那么如何将问题规模缩小呢？于是想到了 Hash 法，Hash 往往可以缩小问题规模，然后在简化的数据里面使用常规排序算法即可找出此问题的答案。

（4）递归法

为了降低问题的复杂度，很多时候都会将问题逐层分解，最后归结为一些最简单的问题，这就是递归。此种方法，首先要能够解决最基本的情况，然后以此为基础，解决接下来的问题。

例如，在寻求全排列的时候，可能会感觉无从下手，但仔细推敲，会发现后一种排列组合往往是在前一种排列组合的基础上进行的重新排列，只要知道了前一种排列组合的各类组合情况，只需将最后一个元素插入到前面各种组合的排列里面，就实现了目标：即先截去字符串 s[1…n]中的最后一个字母，生成所有 s[1…n-1]的全排列，然后再将最后一个字母插入到每一个可插入的位置。

（5）分治法

任何一个可以用计算机求解的问题所需的计算时间都与其规模有关。问题的规模越小，越容易直接求解，解题所需的计算时间也越少。分治法正是充分考虑到这一情况，将一个难以直接解决的大问题，分割成一些规模较小的相同问题，以便各个击破，分而治之。分治法一般包含以下三个步骤：

1）将问题的实例划分为几个较小的实例，最好具有相等的规模。

2）对这些较小的实例求解，而最常见的方法一般是递归。

3）如果有必要，合并这些较小问题的解，以得到原始问题的解。

分治法是程序员面试常考的算法之一，一般适用于二分查找、大整数相乘、求最大子数组和、找出伪币、金块问题、矩阵乘法、残缺棋盘、归并排序、快速排序、距离最近的点对、导线与开关等。

（6）Hash 法

很多面试笔试题目，都要求求职者给出的算法尽可能高效。什么样的算法是高效的？一般而言，时间复杂度越低的算法越高效。而要想达到时间复杂度的高效，很多时候就必须在空间上有所牺牲，用空间来换时间。而用空间换时间最有效的方式就是 Hash 法、大数组和位图法。当然，此类方法并非包治百病，有时，面试官也会对空间大小进行限制，那么此时，求职者只能再去思考其他的方法了。

其实，在涉及大规模数据处理的算法设计中，Hash 法就是最好的方法之一。

（7）轮询法

在设计每道面试笔试题时，往往会有一个载体，这个载体便是数据结构，例如数组、链表、二叉树或图等，当载体确定后，可用的算法自然而然地就会显露出来。可问题是很多时候并不确定这个载体是什么。当无法确定这个载体时，一般也就很难想到合适的方法了。

编者建议，此时，求职者可以采用最原始的思考问题的方法——轮询法，在脑海中轮询各种可能的数据结构与算法，常考的数据结构与算法一共就那么几种（见表 1），即使不完全一样，也是由它们衍生出来的或者相似的，总有一款适合考题的。

表 1　最常考的数据结构与算法知识点

数据结构	算　法	概　念
链表	广度（深度）优先搜索	位操作
数组	递归	设计模式
二叉树	二分查找	内存管理（堆、栈等）
树	排序（归并排序、快速排序等）	
堆（大顶堆、小顶堆）	树的插入/删除/查找/遍历等	
栈	图论	
队列	Hash 法	
向量	分治法	
Hash 表	动态规划	

此种方法看似笨拙，其实实用，只要求职者对常见的数据结构与算法烂熟于心，一点都没有问题。

为了更好地理解这些方法，求职者可以在平时的准备过程中，应用此类方法去答题，做得多了，自然对各种方法也就熟能生巧了，面试的时候，再遇到此类问题，也就能够收放自如了。当然，千万不要相信有张无忌般的运气，能够在一夜之间练成乾坤大挪移这一绝世神功，称霸武林。算法设计功力的练就是平时一点一滴的付出和思维的磨炼。方法与技巧也许只是给面试打了一针“鸡血”、喂一口“大补丸”，真正的功力还是需要一个长期的积累过程的。

经验技巧 6　如何回答系统设计题？

应届生在面试的时候，偶尔也会遇到一些系统设计题，这些题目往往只是测试一下求职者的知识面，或者测试求职者对系统架构方面的了解，一般不会涉及具体的编码工作。虽然如此，对于此类问题，很多人还是感觉难以应对，也不知道从何说起。

如何应对此类题目呢？在正式介绍基础知识之前，首先罗列几个常见的系统设计相关的面试笔试题，如下所示：

1. 设计一个 DNS 的 Cache 结构，要求能够满足每秒 5000 次以上的查询，满足 IP 数据的快速插入，

查询的速度要快（题目还给出了一系列的数据，例如站点数总共为 5000 万、IP 地址有 1000 万等）。

2. 有 N 台计算机，M 个文件，文件可以以任意方式存放到任意计算机上，文件可任意分割成若干块。假设这 N 台计算机的宕机率小于 1/3，想在宕机时可以从其他未宕机的计算机中完整导出这 M 个文件，求最好的存放与分割策略。

3. 假设有 30 台服务器，每台服务器上面都存有上百亿条数据（有可能重复），如何找出这 30 台机器中，根据某关键字，重复出现次数最多的前 100 条？要求使用 Hadoop 来实现。

4. 设计一个系统，要求写速度尽可能快，并说明设计原理。

5. 设计一个高并发系统，说明架构和关键技术要点。

6. 有 25TB 的日志(query->queryinfo)，大小在不断增长，设计一个方案，给出一个 query 能快速返回 queryinfo。

以上所有问题中凡是不涉及高并发的，基本可以采用 Google 的三个技术解决，即 GFS、MapReduce 和 Bigtable，这三个技术被称为“Google 三驾马车”，Google 只公开了论文而未开源代码，开源界对此非常有兴趣，仿照这三篇论文实现了一系列软件，如 Hadoop、HBase、HDFS 及 Cassandra 等。

在 Google 这些技术还未出现之前，企业界在设计大规模分布式系统时，采用的架构往往是 database+sharding+cache，现在很多公司（例如淘宝、weibo.com）仍采用这种架构。在这种架构中，仍有很多问题值得探讨。如采用什么数据库，是 SQL 界的 MySQL 还是 NoSQL 界的 Redis/TFS，两者有何优劣？采用什么方式 sharding（数据分片），是水平分片还是垂直分片？据网上资料显示，weibo.com 和淘宝图片存储中曾采用的架构是 Redis/MySQL/TFS+sharding+cache，该架构解释如下：前端 cache 是为了提高响应速度，后端数据库则用于数据永久存储，防止数据丢失，而 sharding 是为了在多台计算机间分摊负载。最前端由大块大块的 cache 组成，要保证至少 99%的访问数据落在 cache 中，这样可以保证用户访问速度，减少后端数据库的压力。此外，为了保证前端 cache 中的数据与后端数据库中的数据一致，需要有一个中间件异步更新数据（为什么使用异步？理由简单：同步代价太高），这个功能有些人可能比较清楚，新浪有个开源软件叫 Memcachedb（整合了 Berkeley DB 和 Memcached），正是用来完成此功能的。另外，为了分摊负载压力和海量数据，会将用户微博信息经过分片后存放到不同节点上（称为“Sharding”）。

这种架构优点非常明显：简单，在数据量和用户量较小的时候完全可以胜任。但缺点是扩展性和容错性太差，维护成本非常高，尤其是数据量和用户量暴增之后，系统不能通过简单地增加计算机解决该问题。

鉴于此，新的架构应运而生。新的架构仍然采用 Google 公司的架构模式与设计思想，以下将分别就此内容进行分析。

GFS 是一个可扩展的分布式文件系统，用于大型的、分布式的、对大量数据进行访问的应用。它运行于廉价的普通硬件上，提供容错功能。现在开源界有 HDFS（Hadoop Distributed File System），该文件系统虽然弥补了数据库+sharding 的很多缺点，但自身仍存在一些问题，例如：由于采用 master/slave 架构，因此存在单点故障问题；元数据信息全部存放在 master 端的内存中，因而不适合存储小文件，或者如果存储大量小文件，存储的总数据量不会太大。

MapReduce 是针对分布式并行计算的一套编程模型。其最大的优点是：编程接口简单，自动备份（数据默认情况下会自动备份三份），自动容错和隐藏跨计算机的通信。在 Hadoop 中，MapReduce 作为分布计算框架，HDFS 作为底层的分布式存储系统，但 MapReduce 不是与 HDFS 耦合在一起的，完全可以使用自己的分布式文件系统替换掉 HDFS。当前 MapReduce 有很多开源实现，如 Java 实现 Hadoop MapReduce、C++实现 Sector/sphere 等，甚至有些数据库厂商将 MapReduce 集成到数据库中了。

BigTable 俗称“大表”，是用来存储结构化数据的，编者觉得，BigTable 在开源界最火爆，其开源实现最多，包括 HBase、Cassandra 和 levelDB 等，使用也非常广泛。

除了 Google 的这“三驾马车”以外，还有其他一些技术可供学习与使用：

Dynamo：亚马逊的 key-value 模式的存储平台，可用性和扩展性都很好，采用 DHT（Distributed Hash Table）对数据分片，解决单点故障问题，在 Cassandra 中，也借鉴了该技术，在 BT 和电驴这两种下载引擎中，也采用了类似算法。

虚拟节点技术：该技术常用于分布式数据分片中。具体应用场景是：有一大块数据（可能 TB 级或者 PB 级），需按照某个字段（key）分片存储到几十（或者更多）台计算机上，同时想尽量负载均衡且容易扩展。传统的做法是：Hash(key) mod N，这种方法最大的缺点是不容易扩展，即增加或者减少计算机均会导致数据全部重分布，代价太大。于是新技术诞生了。其中一种是上面提到的 DHT，现在已经被很多大型系统采用。还有一种是对“Hash(key) mod N”的改进：假设要将数据分布到 20 台计算机上，传统做法是 Hash(key) mod 20，而改进后，N 取值要远大于 20，比如是 20000000，然后采用额外一张表记录每个节点存储的 key 的模值，例如：

node1：0~1000000

node2：1000001~2000000

……

这样，当添加一个新的节点时，只需将每个节点上部分数据移动给新节点，同时修改一下该表即可。

Thrift：Thrift 是一个跨语言的 RPC 框架。RPC 是远程过程调用，其使用方式与调用一个普通函数一样，但执行体发生在远程计算机上。跨语言是指不同语言之间进行通信，例如 C/S 架构中，Server 端采用 C++编写，Client 端采用 PHP 编写，怎样让两者之间通信，Thrift 是一种很好的方式。

最前面的几道题均可以映射到以上几个系统的某个模块中，如：

1. 关于高并发系统设计，主要有以下几个关键技术点：缓存、索引、数据分片及锁粒度尽可能小。

2. 题目 2 涉及现在通用的分布式文件系统的副本存放策略。一般是将大文件切分成小的 block（如 64MB）后，以 block 为单位存放三份到不同的节点上，这三份数据的位置需根据网络拓扑结构配置，一般而言，如果不考虑跨数据中心，可以这样存放：两个副本存放在同一个机架的不同节点上，而另外一个副本存放在另一个机架上，这样从效率和可靠性上，都是最优的（Google 公布的文档中有专门的证明，有兴趣的读者可参阅一下）。如果考虑跨数据中心，可将两份存在一个数据中心的不同机架上，另一份存放到另一个数据中心。

3. 题目 4 涉及 BigTable 的模型。主要思想是将随机写转化为顺序写，进而大大提高写速度。由于磁盘物理结构的独特设计，其并发的随机写（主要是因为磁盘寻道时间长）非常慢，考虑到这一点，在 BigTable 模型中，首先会将并发写的大批数据放到一个内存表（称为“memtable”）中，当该表大到一定程度后，会顺序写到一个磁盘表（称为“SSTable”）中，这种写是顺序写，效率极高。此时可能有读者问，随机读可不可以这样优化？答案是：看情况。通常而言，如果读并发度不高，则不可以这么做，因为如果将多个读重新排列组合后再执行，系统的响应时间太慢，用户可能接受不了，而如果读并发度极高，也许可以采用类似机制。

经验技巧 7　如何解决求职中的时间冲突问题？

对于求职者而言，求职季就是一个赶场季，一天少则几家、十几家企业入校招聘，多则几十家、上百家企业招兵买马，企业多，选择项自然也多，这固然是一件好事情，但由于招聘企业实在是太多，自然会导致另外一个问题的发生：同一天企业扎堆，且都是自己心仪的大牛企业。如果不能提前掌握企业的宣讲时间、地点，是很容易迟到或错过的。但有时候即使掌握了宣讲时间、笔试和面试时间，还是有可能错过，为什么呢？时间冲突。人不可能同一时间做两件不同的事情，所以，很多时候就必须有所取舍了。

到底该如何取舍呢？该如何应对这种时间冲突的问题呢？在此，编者将自己的一些想法和经验分享出来，以供读者参考：

1. 如果多家心仪企业的校园宣讲时间发生冲突（前提是只宣讲，不笔试，否则请看后面的建议），