<HTML> 随书附赠程序源代码

</ > 在这里 /
有技术大牛倾囊相授的面试经验和技巧
</ > 在这里 /
有来自各大名企的面试真题大合集
</ > 在这里 /
有专业团队倾力打造的精炼解析

# iOS 程序员 > 面试笔试真题库

猿媛之家 / 组编　蒋信厚 汪小发 楚秦 等 / 编著

PROGRAMMER

> INTERVIEW QUESTIONS LIBRARY

扫描二维码可免费使用
“程序员面试笔试真题库”微信刷题小程序

精选 18套 真 / 题 / 卷

本书聚焦名企面试笔试特点，精选18套真题卷

**单选 / 多选 / 填空 / 问答 / 编程 / 开放 / 智力，多题型练习**

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

# iOS 程序员面试笔试真题库

猿媛之家　组编
蒋信厚　汪小发　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性与侧重点，精心挑选了近 3 年以来约 20 家顶级 IT 企业的 iOS 面试笔试真题卷，这些企业涉及业务包括系统软件、搜索引擎、电子商务、手机 App、安全关键软件等，这些真题涉及的知识点包括 iOS、计算机网络、操作系统、数据结构与算法、基础数学知识、数据库、设计模式等，非常具有代表性与参考性。同时，本书对这些题目进行了详细的分析与讲解，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，使得读者能够通过本书不仅获取到求职的知识，同时更有针对性地进行求职准备，最终能够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**

iOS 程序员面试笔试真题库 / 猿媛之家组编；蒋信厚等编著. —北京：机械工业出版社，2019.6

ISBN 978-7-111-62617-6

Ⅰ. ①i… Ⅱ. ①猿… ②蒋… Ⅲ. ①移动终端－应用程序－程序设计－资格考试－试题 Ⅳ. ①TN929.53-44

中国版本图书馆 CIP 数据核字（2019）第 080851 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：尚 晨 责任编辑：尚 晨

责任校对：张艳霞 责任印制：孙 炜

北京玥实印刷有限公司印刷

2019 年 9 月第 1 版 • 第 1 次印刷

184mm×260mm • 17.75 印张 • 437 千字

0001－2500 册

标准书号：ISBN 978-7-111-62617-6

定价：69.00 元

电话服务

客服电话：010-88361066

010-88379833

010-68326294

网络服务

机 工 官 网：www.cmpbook.com

机 工 官 博：weibo.com/cmp1952

金 书 网：www.golden-book.com

机工教育服务网：www.cmpedu.com

# 前　言

在当今 IT 互联网火热的时代，iOS 技术的生命力从其诞生开始一直屡增不减，iOS 软件开发工程师的岗位也是各大互联网企业招聘必不可少的。同时，随着互联网技术的成熟以及 iOS 从业者的激增和整体水平的提高，各大企业招聘时对应聘者的要求也水涨船高，iOS 开发者必须系统地提高自身专业水平才能应对企业更加系统全面的考察，在激烈的竞争中提高获取理想工作的机率。

iOS 领域火热这么多年，之前市面上一直都没有比较系统的 iOS 面试指导书籍。“猿媛之家”作为致力于互联网面试指导的专业团队，多年来一直在为国内 IT 从业者研发整理面试指导书籍，先后由机械工业出版社出版了《程序员面试笔试宝典》《Java 程序员面试笔试宝典》等畅销书籍，受到一致好评，包括本人和身边很多同学同事都得益于此系列书籍，找到理想的工作，也帮助大家在从业前打好了相关领域的知识基础，在以后的工作中发挥长远的作用。考虑到一直以来 iOS 面试指导书籍的空缺，团队从 2016 年年底开始创作 iOS 程序员面试笔试系列书籍，历时一年半有余，于 2018 年 9 月出版了《iOS 程序员面试笔试宝典》，并按照计划于 2019 年先后出版《iOS 程序员面试笔试真题库》和《iOS 程序员面试笔试真题与解析》。

本书之前的《iOS 程序员面试笔试宝典》一书侧重点在于囊括过去几年已有的面试笔试题目，对其进行系统分类，构建知识框架，并进行知识点的辐射，深入挖掘并总结面试笔试中常考到的重要知识点，帮助应聘者快速填充基础知识空缺，同时帮助应聘者更加系统、清晰、有逻辑地组织答案和语言，取得更好的面试笔试成绩。本书的特点是整理知名企业有代表性的 iOS 面试真题，很多典型的题目经常反复考查，在面试笔试中也很容易会碰到原题或者原题的简单变型。应聘者可利用此书来进行实战模拟，快速检测出自身不足之处，及时弥补，短时间突击，以提高应聘表现成绩。另外每套真题后都进行了详细解析和知识点挖掘，同时补充了 iOS 面试笔试宝典中没有覆盖到的部分小知识点，力求全面地将 iOS 面试笔试中可能出现的考查点都总结到位。

为了更加高效地利用好这本书，建议读者自行控制时间一次性做完一套题目，遇到不会的可跳过，之后翻看答案对照，找出自身的知识盲点，并通过研读解析或者翻查宝典中的知识点进行学习，弥补自己的弱点。希望更多的 iOS 从业者通过此系列书籍，专业技能更上一层楼，历经磨练找到属于自己的理想工作，发挥自身的才能，成为优秀的 iOS 开发者乃至行业专家。另外，由于本书作者自身水平有限，难免在细节上会有疏忽的地方，希望读者谅解，

也欢迎读者们的热心反馈或者技术交流。

最后诚挚感谢家人和身边朋友们的支持和鼓励，使我们能够坚持将此系列书籍按时保质保量地完成。

对于书中的任何问题或困惑，读者都可以通过邮件联系我们：yuancoder@foxmail.com。期待你们的来信。

编　者

# 目　录

## 真题详解篇

# 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提供 iOS 程序员面试笔试经验、面试笔试问题方法讨论等。通过本篇的学习，求职者必将获取到丰富的应试技巧与方法。

# 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”，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”的理由。

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？

谈话是一种艺术，回答问题也是一种艺术，同样的话，不同的回答方式，往往也会产生出不同的效果，甚至是截然不同的效果。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清高自负，而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(logn)，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”和“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 4 个人，除了我以外的其他 3 个人中，两个人能力很强，人也比较好相处，但有一个人不太好相处，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，缺乏自信而已，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强了，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，有理有据，让面试官一

目了然。

回答问题的技巧，是一门大的学问。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，都是历年的面试笔试真题，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问题呢？编者建议：会做的题目一定要拿满分，不会做的题目一定要拿部分分。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（编者的一个朋友把《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，后来找工作无往不利）；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求职者独立思考问题的能力以外，还会关注求职者技术能力的可塑性，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 6 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，相反它可能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，方便后续自己对问题的解答。

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，解决方案也自然就出来了。

（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，完成基本功能顶多只能算及格水平，要想达到优秀水平，至少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、完整性以及顽健性？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，求职者可以首先征求面试官的同意，在编写实际代码前，写一个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，如果不能保证百分之百正确，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，认真仔细地检查一些边界情况、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，除了要字迹工整，最好能够严格遵循编码规范：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地输出、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

在软件行业有一句真理：任何软件都有缺陷（Bug）。但不能因为如此就纵容自己的代码，允许错误百出。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，会让面试官觉得求职者不仅技术水平不行，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，求职者在面试的过程中，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，从而给面试官留下好印象，进而提高面试的成功率。

## 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++、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“绝对的公平”、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，而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。

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匹配的影响，所以都会引入相关的性格测试题用于测试求职者的性格特性，看其是否适合所投递的职位。大多数情况下，只要按照自己的真实想法选择就行了，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，其实求职者只要从惊慌失措中冷静下来，稍加分析，会发现这些问题并非看上去那样可怕。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。”求职者又不是统计局官员，家里也不是开商场的，如何能够得出一个准确的数据呢？即使求职者是商场的经理，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，再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）以一家较大规模的商场为例，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 元=4.38 亿元。

4）对商户而言，租金一般占销售额的 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 亿元/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 2017 年一年一共用掉了多少块尿布，估算一下杭州的轮胎数量。但这些题一般都是即兴发挥，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”，不管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》（编者早前编写的一本书，由机械工业出版社出版）学习上一段时间，牢记于心，应付此类题目就完全没有问题。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能够更好地区分开来，企业会推陈出新，越来越倾向于出一些有技术含量的“新”题，这些题目以及答案，不再是陈芝麻烂谷子了，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 GRE 或托福考试在出国留学中的地位，必需但不是最重要的，它只是众多考核方面中的一个而已，不一定就能决定求职者的“生死”。虽然如此，但并非说就不用去准备算法知识了，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，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都在《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，正所谓“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，分析总结其中的一般规律。具体而言就是通过列举少量的特殊情况，经过分析，最后找出一般的关系。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，此时此刻就可以触类旁通，尝试改进原有算法来解决这个新问题。而通常情况下，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过的问题，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某网站次数最多的那个 IP 地址。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时，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？于是想到了哈希（Hash）法，Hash 法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。而分治法正是充分考虑到这一内容，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 3 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，此类方法并非万能，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，凡是涉及大规模数据处理的算法设计，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暴露出来。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，也是由此衍生出来的或者相似的，总有一种是适合考题的。

**表 1　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 | — |
| 哈希（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，就没有问题。

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。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。方法与技巧只是给面试打了一针“鸡血”、喂一口“大补丸”，真正的功力还需要长期的积累。

## 经验技巧 6　如何回答系统设计题

应届毕业生在面试的时候，偶尔也会遇到一些系统设计题，而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，比如站点数总共为 5000 万、IP 地址有 1000 万等）。

2）有 N 台机器，M 个文件，文件可以以任意方式存放到任意机器上，文件可任意分割成若干块。假设这 N 台机器的宕机率小于 1/3，想在宕机时可以从其他未宕机的机器中完整导出这 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），如何根据关键字找出这 30 台机器中，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 log(query->queryinfo)，log 在不断地增长，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，基本可以采用谷歌（Google）公司的三个技术解决，即 GFS、MapReduce 和 BigTable，这三个技术被称为“Google 三驾马车”，Google 公司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 公司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（比如新浪微博和淘宝）仍采用这种架构。在这种架构中，仍有很多问题值得去探讨。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？采用什么方式数据分片（Sharding），是水平分片还是垂直分片？据网上资料显示，新浪微博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：前端缓存（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 是为了在多台机器间分摊负载。最前端由大块大块的 cache 组成，要保证至少 99%（该数据在新浪微博架构中的是自己猜的，而淘宝图片存储模块是真实的）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（为什么使用异步？理由为同步代价太高。异步有缺点，如何弥补？）

数据，这个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），具备此功能。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结点上（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；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机器解决该问题。

鉴于此，“Google 三驾马车”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，比如：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，或者说如果存储大量小文件，那么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），自动容错和隐藏跨机器间的通信。在 Hadoop 中，MapReduce 作为分布计算框架，而 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 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 在开源界比较流行，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 和电驴两种下载引擎中，也采用了类似算法。

虚拟结点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）台机器上，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，即增加或者减少机器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 台机器上，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结点存储的 key 的模值。

node1：0～1000000

node2：1000001～2000000

……

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