只有程序员看的懂面试圣经 | 如何拿下编程面试

博客选编

目　录

前言

> 当我最初开始参加编程面试的时候，我所有最心仪的公司都忽视了我。现在回头看那个时候，我发现自己当时去参加面试都完全没做任何准备。虽然已经有许多博客文章和书籍在讲编程面试，但现在的我作为面试官，坐在桌子的另一边，还是能看到许多来参加编程面试的人没做任何准备，或者准备得很糟糕。这也就是为什么我开始写这篇指南的原因，刚毕业时的我、第一次参加面试的我一定非常想有这么一份指南来指引自己。而从现在开始，我自己也会照着这份指南去做。

> 多年以来，我在好几家公司工作过，所以我的面试技巧得到了很好的磨炼，而且我参与面试的过程也教会了我该说什么、该做哪些准备，以及如何面试。在这篇指南里，你会了解到面试的概况、面试取得成功的六大步骤，以及我在考察数据结构和算法时所考虑的方面。这篇指南无法确保你找到工作，但它能帮助你尽最大可能给面试官留下一个好印象。

声明：本文中的观点完全出自个人视角，与我目前或者以前的雇主没有关系。

> 原文来自Medium Stefan De Clercq，翻译由is译社葛仲君提供。

> 译文：http://www.jianshu.com/p/d34c335a6cfd/comments/766555

面试过程

本节概述了硅谷公司的面试过程，仅仅是个情况介绍，大家可以跳过去往后看。

除了直接申请面试以外，一般说来，还有两种途径来获得面试的机会：由现在的雇主推荐，或者通过LinkedIn。虽然前者会快一些、更尊敬一些，但后者很可能是大部分应聘者所走的路径。事实上，每天都有无数的招聘人员趴在LinkedIn上，他们唯一的工作就是寻找和接触有可能换工作的员工，所以一定要保证自己的信息是最新的，而且要多交人脉、多请别人来认可自己的技能，并且要把你所具备的技能、做过的个人项目或者对开源软件所做的贡献加到个人页面里去。

最初的接触一般是通过电子邮件进行的，然后招聘人员会给你打电话，大概了解一下你的技术背景。如果你的技能和他们正在寻找的技能一致，他们就会安排一次电话面试，在电话面试时，你可能就会被要求在一份共享的在线文档里编程。那么你就会知道，这份文档很可能没有任何代码补全和句法高亮的功能。电话面试会持续半小时到45分钟，如果你表现不错，就会被邀请去参加现场面试。现在如果没有电话面试、或者在电话面试之外，你可能还得去参加一个小的编程项目。

现场面试由几次面试组成，总体会持续45分钟到一个小时。这些面试会和电话面试非常像，只是问题会更难——不过能亲眼见到面试官多少算是有所补偿。现场面试数周之后，所有反馈应该都被看过、招聘决定就会做出，招谁不招谁也就定了。如果你没拿到offer，也要明白面试是一个随机的过程，包含运气的成分，不妨把它看作是一次学习的经历。可能你还会想起布莱恩·阿克顿（BrianActon）面试Facebook和Twitter不成、后来成为WhatsApp联合创始人的故事。

理论上讲，用哪种编程语言并不重要，但你面试需要用某种特定语言来完成的工作时除外，比如iPhone开发者或者前端开发者。我强烈建议你用正在面试的公司所使用的一种编程语言来编程（以及练习面试问题）。

面试获得成功的六个步骤

编程面试的目的，是为了确定你的编程水平有多高。一般来说，你将被要求用编程来完成一个功能或者方法，但有时候，你会需要编辑一个类的定义，或者设计一系列相关的代码模块。在任何一种情况下，你都要有条不紊地解决问题，并遵循以下六个步骤：

1.首先，要确保你理解了面试官的问题。许多问题都是故意措辞模糊或者模棱两可，这个时候你可以请面试官把问题说清楚，从而确保你真正回答面试官的问题。你的提问同时还有一个好处，就是它能给你自己一些时间，让你的脑子转起来。

2.用一到两个例子来确定问题的限制条件和要求（在现场面试时在白板上完成这个过程，在电话面试时在笔记本上完成）。尝试用中等规模的例子，以便覆盖到一些特殊情况。如果你能想到可能相关的表格，就把它画出来。事实上，把你想到的任何东西都写下来是会有帮助的，因为它能为你提供一个视觉锚点，从而让你在走不通时或者思考过程中随时返回某一个点。

3.把话说清楚，这可能是最重要的一步。要试着让面试尽可能有更多的互动，面试官不知道你在想什么，而让他们参与到你的思考过程里，会让她给你一些有用的提示，防止你偏向错误的方向。你的目标就是要先和面试官确证你的答案，然后再去写代码，而且你考虑答案越清晰、越高效，你得到的即时反馈也就越好。

4.通过应用以下技巧来找到答案：回想一下你遇到的类似问题，再想想它们是如何被解决的，尝试各种不同的算法（分治算法、贪心算法、递归、排序，等等），把问题分解成更小的、可处理的小问题（这样你就能得到相应部分的分数），最后再通览一遍你列出的数据结构，因为有时候，只要想到了正确的数据结构，就能给出正确的答案。

5.当你向面试官问清楚了问题、并向她解释了你的答案之后，就可以开始写代码了。要记住，在共享文档里写代码的时候，你可以复制粘贴、写评论，而且能回过头来完成骨架算法和功能。但在白板上写代码就不一样了，它需要你的头脑很清醒，而且需要你具备管理白板空间的技能。如果足够幸运的话，现在当你开始在白板左上角动笔的时候，应该非常明白你要写些什么东西，而且你要确保在你写答案的时候，没有挡住面试官的视线。花点儿时间把代码写得紧凑而美观一点儿，因为你的代码也会是面试反馈的一部分。在你写代码的时候，要大声解释你在写什么，这会让你的面试官更容易地跟上你的思路。

6.最后，用不同的例子和特殊案例验证一下你的代码，并且要一行一行地过。这会展示你的思考过程，让你检查出小错误，并告诉面试官你的办法是可行的。如果你想得到额外加分的话，甚至可以把单元测试的代码写下来！最后再和面试官聊一下你的答案在空间和时间利用方面的复杂性，然后结束整场面试。

电话面试中提示出的问题

电话面试值得特别一提，因为这是大多数人失利的地方。之所以会这样，部分原因在于电话面试是招聘过程中第一道真正的关卡，但也有一部分原因在于，这种形式容易造成沟通的错误，而且缺乏可视化线索，所以电话面试是特别严酷的。

电话面试有两大障碍。第一大障碍是，在电话面试的一开始，双方都能看到的唯一的东西就是一个空白的共享文档。这会让面试者倾向于过度补偿非语言沟通的缺失，从而着急忙慌地在屏幕上进行沟通。令人遗憾的是，这么做很少会有好结果。所以当务之急并不是去关注那个正在盯着你的空白文档，而是要首先理解和评估问题（也就是完成上述六个步骤中的前四个），同时通过尽可能地沉浸到面试中来弥补现实存在感的缺失（要记住，电话的另一头是一位可以很容易就被别的事情[比如查看邮件]分心的面试官）。

电话面试的第二大障碍，就是要同时在电脑上打字和在电话上聊天的后勤保障问题。你不必一只手敲代码、一只手打电话，也不必把电话调到扬声器模式，我建议你用电脑上的Google Hangouts接面试电话（你得有一个GoogleVoice号码，而且得在面试前测试一下）。你还可以用耳麦或者耳机来进一步降低不好的接收效果、提高沟通质量。

算法+数据结构=程序

如果你正在思考为什么软件工程的面试和日常编程不一样，那你可能有兴趣读一下Quora上的这条回答。最根本的原因在于：面试是为了测试你在计算机技术方面的基础，所以会非常偏重算法和数据结构，因此你可能需要练习一些面试问题，从而让自己具备解决面试问题的心态。

从短期来看，你所能做的最好的准备工作就是买一块白板，并通读一遍《程序员面试金典》（Cracking The Code Interview），里面都是很好的建议，而且里面的许多面试问题和答案会帮助你确定问题所在，并匹配好回答模式。请参阅本指南最后列出的常用面试问题。

当然了，长远来看，我们都会死掉，所以我会把事情搞简单，说一些你绝对应该复习一下的关键概念。

数组/字符串

大部分数组和字符串是可互换的，事实上，你遇到的大部分字符串处理的问题，都可以在理解数组的基础上得到解决。记住这一点之后，你应该懂得如何遍历数组，知道如何访问、转换和调换其中的每一个元素，而且要懂得如何对它们进行各种不同的集合运算。和其他算法相比，二分法检索（Binary search）可能会更多地成为面试问题的核心内容（如果你曾经碰到过有分类数组的问题，那么二分法检索有可能应该是你答案的一部分），你绝对必须知道如何使用它。

排序

和数组密切相关的，是排序算法。你不大可能会被要求重复使用一个排序算法，但很可能你至少知道排序是如何在O(nlogn)的时间里完成的就行。不过你应该大概知道归并排序（merge sort）或者快速排序（quicksort）和基数排序（radix sort）的执行细节。

动态数组/可增数组

动态数组可以按需重新调整自己的大小，同时依然提供分时平摊的持续时间访问。一种典型的做法是，当在一个全排列数组中增加一个元素的时候，会形成一个新的、更大的数组，而旧数组中的元素也会被复制到新数组里。你应该在面试时做到完成一个动态数组。

如果你拿到一个非数组类问题，但你在答题中需要用到像数组结构这样的数组，不妨少给自己惹麻烦，直接用动态数组吧。

哈希映射/哈希表/词典/哈希集合

哈希表（Hash tables）是编程时的瑞士军刀，很多不同类型的问题（检查存在、计算频率、排序，等等）都能用哈希表来完美解决。它几乎肯定会出现在你的面试中，而你应该理解它的原理（哈希功能的角色、冲突如何解决、什么时候要调整大小、为什么）以及如何运用它们。

链表

链表问题在C和C++的面试中最常见，因为它们是弄清楚应聘者是否理解指针的一种简单的办法。不过这个点太初级、太基础了，所以不管用哪种语言，你都应该知道该如何从零做起应用它们。而且由于大部分链表问题不过是与人所周知的遍历还有删除和插入相关的问题的变体，所以链表问题准备起来很容易，你没有理由拿不到这部分分数。

许多链表问题中都会用到一个小技巧，那就是慢速/快速指针技术。它的简单版含义如下：使用两个指针迭代生成一个列表，其中一个指针在另一个指针的前面。快速模式下的指针可能会是一个位于前面的固定数值（它有助于确定列表有无循环，或者找到列表中的第k个元素），或者也可能会跳过慢速指针经过的多个结点（打个比方，如果快速指针的速度是慢速指针的两倍，那么当它到达列表末尾时，慢速指针将会位于列表的中间）。

请注意，当面试官谈到链表时，他们常常指的是单链表，但你无论如何都应该问清楚。

栈/队列

栈和队列一般会是你用来解题的数据结构的一部分。你应该知道如何用链表和数组两种方式来实现它们。

加练两道题：利用两个队列实现一个栈，以及利用两个栈来实现一个队列。

树/二叉树/二叉搜索树（BST）/字典树/堆

你可能不会每天都见到树和图，但你很可能会在面试时遇到它们，所以你要彻底地看一下这些数据结构。

树最一般的定义，是和其他结点没有或者有一个以上关系的结点的集合，但在实践中，当面试官说“树”的时候，他们指的是一种叫二叉树的东西。二叉树是一种树的类型，它的每个结点都至多有两个子树，一般被称为左子树和右子树。

你不应该把二叉树和二叉搜索树混淆起来，后者是一种特殊的二叉树，它的左子树结点上的值都比父结点小，而右子树结点上的值都比父结点大或者相等。二叉搜索树的优点是，如果树的结构相对平衡（向面试官问清楚这个问题），那么查找、插入和删除就可以在O(log n)的时间里完成。二叉搜索树的其他重要属性，就是你跟着所有的左子树走，就能得到这个树上最小的元素，而跟着所有的右子树走，就能得到这个树上最大的元素。

请注意，是有办法让树一直保持平衡的，最常用的办法就是红黑树和AVL树。我不会去弄清楚它具体实现的细节，只要知道有这些数据结构就行。

不过你绝对必须知道遍历树（tree traversal）算法：广度优先搜索（breadth-first-search）、深度优先搜索（depth-first-search），以及中序遍历、后序遍历和前序遍历之间的差别。

以下是在Java实现中序遍历的例子，它可以打印出一个树的所有值（前序遍历和后序遍历几乎和这个一样）：

```
void inOrderTraversal(Node root) {
 if (root == null) return;
 inOrderTraversal(root.getLeft());
 // Do something with the value
 System.out.println(root.getValue());
 inOrderTraversal(root.getRight());
}
```

字典树（trie，读** “tree” **）常常被用在字符串问题里，它是一个n元树，除了根结点以外的每个结点都代表一个字符或者部分或完整的单词，而且沿着树的每一条路径都代表一个单词。实际上它真的没有听起来那么复杂，只要读一下维基百科上的页面、了解该如何构建一个字典树以及如何查询其中的数值就行。请注意，你可以通过前序遍历输出字典树中的所有键。作为一个练习，你可以想一想自己会如何利用字典树实现自动完成功能。

最后是堆（heaps），它也被称为优先队列，是你应该了解的最后一种数据结构。它们通常都是满足堆属性的二叉树：每个结点的子树的值都比结点本身的值小，或者与它相等。所以根结点的值总是最大的，也就是说你总能找到最大值，但代价就是寻找其他任何一个值所需的时间都是O(n)。插入和删除所需的时间依然是O(logn)。

有向图/无向图/加权图

和树一样，图也是由带子集的结点组成的，但和树不一样的地方在于，这些结点可以有多个父结点，所以可能会形成自环（loop）或者圈（cycle）。除了链接——也被称作边（edges）——之外，两个结点之间可能地有比指针更多的信息，而且可能会有值和权重。边有方向的图被称为有向图，而只有双向指针的图被称为无向图。边上有权重的图被称为加权图。

有三种方法来表示图，但你只要搞清楚邻接矩阵（adjacency matrices）和邻接表（adjacency lists）就行了。你应该了解它们计算的复杂程度、它们需要折衷的地方，以及如何在现实的代码中实现它们。用哪种方法取决于你有的图的类型，比如连接完整的简单图可能用邻接矩阵来实现更好，而稀疏一些的图则可能用邻接表来表示更好。

请注意，如果你是在实现加权图，很可能需要定义一个Edge类。

图论是一个非常宽泛的话题，所以很难知道一个人应该为一场面试去熟悉多少种图论算法，所以我只是列出了我认为可以覆盖90%图论问题的内容：你绝对必须知道该如何遍历一个图（深度优先或者广度优先），以及如何做拓扑排序（topological sorting），你应该知道如何实现迪杰斯特拉（Dijkstra）的最短路径算法（这里有一个制作精巧的视频解释了这一算法），同时也要知道如何实现普里姆（Prim’s）算法。最后，如果你还知道如何实现A搜索算法（[A](http://en.wikipedia.org/wiki/A*_search_algorithm)searchalgorithm），那就更好了。

其他数据结构

使用以上数据结构，你就可能解决绝大多数问题了，但也请尽管在这个部分下留言，为其他读者推荐其他数据结构。

位操作

要想处理位元，你必须先得知道在二进制补码（two’s complement）标记内部，数字是如何表示的——二进制补码和无格式二进制标记是一样的，只是负数要“进行位元翻转之后再加1”。比如要想得到数字-1，你要从用8位二进制整数表示是00000001的1开始。对每一个位元进行翻转之后的结果是11111110，再加上1就是11111111，也就成了二进制补码中的-1。

左移位运算符“<<”会把位元移向左边，用0来补上移走之后的空位。

右移位运算符“>>”会把一个位模式向右移，但当向右移动负数时，它的作用在不同编程语言中也不一样，在Java中，右移位会用符号扩充的办法，用1来填充负数中的空位。

逻辑右移位运算符“>>>”是Java和Javascript中独有的，无论数值是多少，它都用0来填充空位。

设置某一位：可以用按位或运算符（|）。

```
num |= 1 << x; //这行代码将会设置位元x
```

清除某一位：可以用按位与运算符（&），并且用取反运算符（~）来屏蔽所有你不想清除的位元。

```
num &= ~(1 << x); //这会清除位元x
```

清除一直到**i**的所有有效位元：

```
num &= (1 << (i + 1)) -1;
```

切换某一位元：可以用按位异或运算符（^）

```
num ^= 1 << x; //这会切换位元x
```

获得一个位元：对你想检查的位元用按位与

```
bit = num & (1 << x);
```

和面向对象编程相关的问题，一般会涉及到设计相关类里的集，以便检验你对面向对象编程的熟悉程度，并了解你是如何架构代码的。你可以使用界面和/或抽象的类来说明，并记住用单例模式（Singleton）、工厂方法模式（Factory）和策略模式（Strategy）来解决这类问题，在编写优雅而可维护的代码方面，它们能对你有长久的助益。

编程应该知道的事情

要知道如何用你正在使用的编程语言来读取和写入文件，并且要知道如何生成随机数。

数学
你并不是在面试数学相关的职位，但考虑到我们被计数和测量的问题所包围，所以有一些数学概念也成了编程面试时关注的东西，比较重要的有质数、进制转换（base conversions）和一些基本的组合数学。

对于质数，要大概知道为什么它们很重要，并且要知道每一个数都可以被分解成质数的和。你还得知道如何实现埃拉托斯特尼筛法（sieve of Eratosthenes）。

对于基本的组合数学，你得知道排列和组合。

排列是对一个集合中的数按照一定的次序或者顺序进行整理。比如对于集合{1,2,3}，就有6种排列的方式，也就是(1,2,3)、(1,3,2)、(2,1,3)、(2,3,1)、(3,1,2)和(3,2,1)。n个不同数字的排列方式一共有n!种。

还有一种排列叫部分排列，也就是从n个数字的集合中取出k个不同的元素，然后再进行排序。这种排列可以用下面的公式来表达：

$$\frac{n!}{(n-k)!}$$

Partial permutation formula.

部分排列公式

组合则是从一个组里选择成员的一种方法，因此选择的顺序并不重要。比如一手牌可以被描述成是从52张一摞的牌堆（n=52）中选出5张组成一组（k=5）。从有n个元素的集合中挑出k个元素，当k>n时，不存在相应的组合，否则这k个元素的组合的数量可以用下面的公式来表达：

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

General formula for the number of k-combinations in a set that has n elements when k <= n.

当k<=n时，从有n个元素的集合中挑出k个元素的组合形式数量的一般公式。

并发

并发问题在面试中并不常见，但也确实有过，所以你肯定不想到时候毫无准备，那就再去看一下如何生成线程、使用同步以及锁定对共享资源的访问，并理解会导致死锁（deadlocks）的几种情况。准备这个话题有一个好办法，那就是去做出来一个你最喜欢的数据结构的同步版本。

面试时的行为举止

做些功课，了解一下要面试的公司，了解一下你自己，以及为什么你要去这家公司。要理解公司在做的事、你的新工作涉及哪些东西，以及它最让你激动的地方是什么。换工作是件大事，所以要认真对待它，提前做些研究。

保持积极心态。保持一个好的情绪，要微笑，不要谈论和你现在或者之前的工作有关的负面信息，当描述挑战的时候，要保持乐观的语调，并强调你从中学到的积极的东西。

本条是前一条里说的不要向面试官传递负面信息的必然结果。一些面试官会问你现在感觉如何，千万别说你之前受不了某一位或两位面试官，一定要说所有事情都非常好。

要保持激情！要让你的激动之情闪亮全场，并展示出你对软件开发、技术和解决重大问题的热情。

要问问题。要真正对你的面试官每天都在做什么抱有真正的兴趣，问问他们工作中遇到的机遇与挑战，提前准备几个程式化的问题，显示一下你对公司和这个职位的兴趣。不过无论你做什么，都别问对方“你感觉如何”。首先，你很可能会收到同样程式化的回答，其次，把面试你的人摆在那样一个位置上，也不是什么好主意。

保持亲切感，并形成闭环。当你结束面试之后，给招聘你的人发一句简短的感谢语，让他们知道你对这次面试的感觉。

回想你学到的东西。无论结果如何，你都能学到一些东西——可以是知识上的某个缺失，也可以是新的面试问题——所以要做自我反思，从自己的经历中学习。

祝各位职场和面试好运！