第3章

3 种 考 试

笔试，电话面试，面试，是顺利求职的 3 个过程。三关全过才能顺利签约，只要有一关没能通过，就会被“刷”掉。

3.1 笔试

我认为笔试是程序员面试 3 个过程中最重要的一个环节，也是最难以提升的一个环节。本书中主要叙述的也是程序员的笔试经历。不论你有多么大的才干，多么广博的知识，如果未能通过笔试，则无缘下面的进程。下面是一个表，描述了各种 IT 公司笔试所考题目的类型。

公 司 名 称	公 司 类 型	笔 试 内 容
Trend	网络公司	C++ 或 Java，网络，数据库，设计模式，智力测试，英语阅读
SAP	软件咨询，ERP，CRM	C++，概率问题，设计模式，智力测试
Advantech	硬件，自动化公司	C++（尤其是指针问题），嵌入式编程
Synopsys	电子类公司	C++（尤其是指针问题），数据结构
NEC	综合软件公司	C，数据结构
金山	综合软件公司	C++或 PHP，数据库，数据结构，设计模式
华为	通信公司	C++或 Java，数据结构，数据库
中兴	通信公司	C++或 Java，数据结构，数据库
VIA	硬件公司	C++（尤其是指针问题），嵌入式编程
华为 3COM	网络公司	C++，网络
SPSS	数据统计软件公司	C++（尤其是继承、多态问题），数据结构

（续表）

公 司 名 称	公 司 类 型	笔 试 内 容
Sybase	数据库公司	C++，Linux，UNIX
Motorola	网络公司	C++，网络
IBM	综合软件公司	C++或 Java
Oracle	数据库公司	Java，数据库
HP	综合软件公司	C++
腾讯	综合软件公司	C++
Yahoo	综合软件公司	C++或 Java 或 C#
微软	综合软件公司	C++，数据结构，智力测试
神州数码	金融软件公司	C++或 Java，数据结构，数据库（SQL）
大唐移动	通信公司	C++
Siemens	数据通信公司	C++，设计模式
Grapecity	软件公司	C++，C#，智力测验

根据上表，对各大 IT 公司的笔试题目和所考的内容，我们可以窥见一斑，并得出以下几个结论。

1．语言的偏向性

综合上表所示，IT 公司笔试在编程语言上有一定偏向性，以 C、C++为主或者是以 Java 为主。语言本身并没有什么高低贵贱之分，但相对来说，考到 Delphi 或者 VB 的可能性很小。作为应届毕业生，如果只是学过 VB、VF 却从来没有接触过 C 系语言，则在笔试中是比较吃亏的。

2．英语的重要性

我所经历过的外企的笔试卷子基本上都是英语试卷，无论从出题到解答，都是让你用英文去回答，所以必须有很好的英文阅读能力，这也是外企招人对英语非常看重的原因。其实也不需要一定通过六级，但一定要有相对多的单词量，能够看懂考题的意思。然后按自己的想法组织语言来描述就可以。

国内企业一般对外语要求不是很看重，题目也是中文的。如果不想进外企的话，也不用特别准备英语。

3．淡看智力测试

之所以要强调这一点，是和市面上过度强调外企智力测试有关。实际上笔者参加过的微软等外企笔试，智力测试只占很小的比例，约 3%～

5%左右。而华为、神州数码等国内 IT 企业基本上没有智力测试，完全是技术考试。所以奉劝大家不要把精力都投在所谓的外企智力测试上面，还是应该以准备技术方面的笔试为主。

4．有的放矢准备简历

不同的公司会考不同的内容，这就像高中时准备不同科目考试的差别。比如说神州数码不会考嵌入式编程，而 VIA 考设计模式的可能性很小。一般有点儿偏“硬”的 IT 公司会对 C++中指针的用法、数据结构考得比较多。偏“软”的企业会对设计模式、模板着重一些。所以本书分得很细，力求对各种 IT 公司的笔试题目做一个详尽的阐述。

作为求职者，笔试前你要首先搞清这个公司的基本情况，它是做什么的，它有什么产品，你是学什么方面的。有的放矢才能折桂。

5．纸上写程序

搞计算机的肯定不习惯在纸上写程序，然而技术面试的时候这是面试官最常用的一招。让写的常见程序有：数据结构书上的程序，经典 C 程序（strcmp、strcpy、atoi……），C++程序（表现 C++经典特性的）。第一次在面试官眼皮底下在纸上写程序，思路容易紊乱。建议大家事先多练习，找个同学坐在边上，在他面前写程序，把该同学当成面试官。经过多次考验，在纸上写程序就基本不慌了。

每次面试总会有些问题回答得不好，回来之后一定要总结，把不懂的问题搞明白。一个求职者就碰到两家公司问了同样的问题，第一次答不出，回去没查，第二次又被问到，当然这是很郁闷的事情。

3.2　电话面试

电话面试主要是对简历上一些模糊信息的确认、之前经历的验证、针对应聘职位简单技术问题的提问，以及英文方面的考查。

由于模式的限制，电话面试时间不会很长。在这个环节中，一定要表现得自信、礼貌、认真、严肃，这样会在声音上给对方一个良好的印象。如果声音慵懒，语气生硬，除非是技术题目及英文方面表现得足够好，否则很难予以平衡。

在回答电话面试的问题时，不要过于紧张，要留心对方的问题，这些问题也许在当面的面试中还会再出现。如果对方在电话面试中要求你做英文的自我介绍，或者干脆用英文和你对话，那在电话面试结束后一定要好好准备英文面试的内容。

笔者曾经参加过 Thoughtworks、Sybase、SAP、麒麟原创等公司的电话面试。外企一般都会要求你做一个英文自我介绍和一些小问题，总的来说不会太过涉及技术方面，因为用英语来描述技术对国人而言还是有一定困难的。国企会问到技术问题，我就曾被问到如何在 C++中调用 C 程序、索引的分类等技术问题，回答基本上要靠平时的积累和对知识的掌控能力。电话面试的具体内容可参见第 18 章。

3.3 面试

一个比较好的面试是能够问出求职者擅长哪方面而哪方面不足的面试。如果面试官针对求职者不足之处穷追猛打，或是炫耀自己的才能，这是不足取的。

对于求职者而言，面试是重点环节，要守时是当然的了。如果不能按时参加面试，最好提前通知对方。着装上不需要过分准备，舒服、干净就好了。一般的 IT 公司对技术人员都不会有很高的着装要求。虽然着装不要求，但精神状态一定要好。饱满的精神状态会显得你很自信。

有笔试的话（有时笔试和面试是同时进行的，即面试官会在提问后请你回答并写下详细描述），也无非是与应聘职位相关的技术考查或者英文考查，如英汉互译等。应视你应聘职位的等级进行准备。

应聘初级职位，会针对你的编程能力和以往的项目经验进行重点的考查。如果面试官针对你做的某个项目反复提问，那么你就需要注意了，要么面试官在这个方面特别精通，要么就是未来的职位需要用到这方面的技术。我们应该抱着一种诚恳的态度来回答，对熟悉的技术点可以详细阐述，对于不熟悉的部分可以诚实地告诉面试官，千万不要不懂装懂。不过，我们同意可以引导与面试官的谈话，把他尽量引导到我们所擅长的领域。在 SPSS 公司面试时，在回答完面试官单链表逆置和拷贝构造函数问题之后，我把话题引入了我所擅长的设计模式方面，这是一种谈话

的艺术。

应聘中级职位，不但会考查代码编写，而且会对软件架构或相关行业知识方面进行考查。代码编写方面，主要以考查某种编程技巧来判断你对代码的驾驭能力。比如某国际知名软件公司经常会让面试者编写 malloc 或 atoi 函数。越是简单的函数越能考验应聘者的编码能力。你不但要实现功能，而且还要对可能出现的错误编写防御性代码，这些经验都需要在实际编程过程中积累。

应聘高级职位，应聘者肯定对技术或某个行业有相当程度的了解，这时主要是看你与职位的契合程度、企业文化的配比性（即将人力资源及成本配比作为服务体系的重要组成部分，将公司企业文化中核心理念及价值观作为客户服务的重要媒介）及整体感觉。应聘管理职位的话，考查的更多是管理技巧、沟通技巧和性格因素。架构师一般会考查行业背景与软件架构方面的知识，比如 UML 或建模工具的使用等；技术专家的职位则会针对相关技术进行深度考查，而不会再考查一般性的编码能力。

面谈的时候，要与面试官保持目光接触，显示出你的友好、真诚、自信和果断。如果你不与对方保持目光接触，或者习惯性地瞟着左上角或者右上角的话，会传达给对方你对目前话题表现冷淡、紧张、说谎或者缺乏安全感的感觉。

如果对方问到的某个问题你不是很熟悉，有一段沉默的话，请不要尴尬和紧张。**面试过程中允许沉默，你完全可以用这段时间来思考。**可以用呼吸调整自己的状态。如果过于紧张，可以直接告诉对方。表达出自己的紧张情绪，能够起到很好的舒缓作用。而且紧张本来也是正常的表现。

在面试过程中，应聘者也保有自己的权利。比如面试时间过长，从上午一直拖到下午，而你未进午餐就被要求开始下午的面试的话，你完全可以要求进餐后再开始。面试是一个双方信息沟通及达成合作目的的会谈，是一个双方彼此考量和认知的过程。**不要忽略自己应有的权利。**

面谈后，如果对方觉得你技术、沟通、态度各方面都不错，也许会增加一个素质测评确认一下对你的判断。

素质测评一般考查性格、能力、职业等方面，以判断你的价值观是否与企业相符。我们不需要去猜测这些题目到底要考查些什么，凭着你

的第一感觉填写就可以了。在几十道甚至上百道题目中，都有几道题是从不同角度考查一个方向的，凭猜测答题反而会前后有悖。

当然，要先看清楚题目，搞清楚是选择一个最适合你自己的，还是描述得最不恰当的。在通过面试之后，如果有多家公司和职位的 Offer 可以选择的话，我们可以将公司的行业排名、公司性质、人员规模、发展前景、企业文化、培训机制，结合自身的生活水平、职业生涯发展规划来进行排列，选出最适合自己的公司和职位。

建议准备一个日程本，记录每一次宣讲会、笔试和面试的时间，这样一旦公司打电话来预约面试，可以马上查找日程本上的空闲时间，不至于发生时间上的冲突。每投一份简历，记录下公司的职位和要求，如果一段时间以后（1 个月或更长）有面试机会，可以翻出来看看，有所准备。根据不同的公司，准备不同的简历，千万不要一概而论，不同的公司 care（在意）的东西不一样。每参加完一次笔试或面试，把题目回忆一下，核对一下答案，不会做的题目更要好好弄懂。同学们之间信息共享，总有人有你没有的信息。如果投了很多份简历，一点儿回音都没有，你得好好看看简历是否有问题，增加一些吸引 HR 眼球的东西。

第 8 章

循环、递归与概率

8.3 打靶

面试例题 1：一个射击运动员打靶，靶一共有 10 环，连开 10 枪打中 90 环的可能性有多少种？请用递归算法编程实现。[中国某著名通信企业 H 面试题]

解析：靶上一共有 10 种可能——1 环到 10 环，还有可能脱靶，那就是 0 环，加在一起共 11 种可能。这是一道考循环和递归的面试题。我们在这个程序中将利用递归的办法实现打靶所有可能的演示，并计算出结果。读者会问，难道一定要使用递归？当然不是。我们也可以连续用 10 个循环语句来表示程序，代码如下：

```
for (i1=0;i1<=10;i1++)
 {
    for (i2=0;i2<=10;i2++)
    {
       for (i3=0;i3<=10;i3++)
       {
          ......
            for (i10=0;i10<=10;i10++)
             {
                if(i1+i2+i3+...+i10=90)
                Print();
             }
          ......
       }
    }
 }
```

但是，上面的循环程序虽然解决了问题，但时间复杂度和空间复杂度无疑是很高的。比较好的办法当然是采用递归的方式，事实上公司也就是这么设计的。递归的条件由以下 4 步完成：

（1）如果出现这种情况，即便后面每枪都打 10 环也无法打够总环

数 90，在这种情况下就不用再打了，则退出递归。代码如下：

```
if(score < 0 || score > (num+1)*10 )  //次数num为0～9
        {
            return;
        }
```

（2）如果满足条件且打到最后一次（因为必须打 10 次），代码如下：

```
  if(num == 0)
    {
        store2[num] = score;
        Output( store2);
        return;

    }
```

（3）如果没有出现以上两种情况则执行递归，代码如下：

```
   for(int i = 0; i <= 10; ++i)
        {
         //这里实际上为了方便把顺序倒了过来，store2[9]是第 1 回
         //store2[8]是第 2 回……store2[0]是第 10 回
            store2[num] = i;
            Cumput(score - i, num - 1,store2);
        }
```

（4）打印函数，符合要求的则把它打印出来。代码如下：

```
    public static void Output(int[] store2)
    {

        for(int i = 9; i>=0; --i)
        {
            Console.Write("   {0}",store2[i]);

        }
        Console.WriteLine();
        sum++;

    }
```

答案：

用 C#编写的完整代码如下：

```
using System ;

public class M
{

    //public static int[] store;
    //相当于设置了全局变量
    //这个全局变量 sum 是包含在 M 类中的
    public static int sum;
    public M()
    {
        int sum =0;
```

```
            //  int[] store =  {1,2,3,4,5,6,7,8,9,0};

        }

         //打印函数
        //符合要求的则把它打印出来
        public static void Output(int[] store2)
        {

            for(int i = 9; i>=0; --i)
            {
                Console.Write("   {0}",store2[i]);

            }
            Console.WriteLine();
            sum++;

        }

        //计算总数，返回 sum 值
        public static int sum2()
        {
            return sum;
        }

        public  static void Cumput(int score, int num, int[] store2 )
        {

            //如果总的成绩超过了 90 环（也就是 score<0），或者如果剩下要打靶
            //的成绩大于 10 环乘以剩下要打的次数，也就是说即便后面的都打 10 环
            //也无法打够次数，则退出递归
            if(score < 0 || score > (num+1)*10 )  //次数 num 为 0～9
            {
                return;
            }

            //如果满足条件且达到最后一层
            if(num == 0)
            {
                store2[num] = score;
                Output( store2);
                return;

            }

            for(int i = 0; i <= 10; ++i)
            {
                store2[num] = i;
                Cumput(score - i, num - 1,store2);
            }
            //Console.Write("   {0}",store2[5]);
        }
    }

    public class myApp
    {
        public static void Main( )
        {
```

```
            int[] store;
            store = new int[10];
            int sum = 0;
            //int a=90;
            //int b=9;
            //Output();
            M.Cumput(90,9,store);
            sum = M.sum2();

            //M.Cumput2(a,b,store);
            //Console.Write("   {0}",store[3]);
            //cout<<"总数:"<<sum<<endl;
            Console.Write(" 总数:   {0}",sum);

        }
    }
```

程序结果一共有 92 378 种可能。

也可以用 C++编写，代码如下：

```
#include <iostream>
using namespace std;
int sum;
int store[10];
void Output()
{
    for(int i = 9; i>=0; --i)
   {
     cout<<store[i]<<" ";
   }
   cout<<endl;
   ++sum;
}

void Cumput(int score, int num)
{
   if(score < 0 || score > (num+1)*10 )  //次数num 为0～9
     return;
   if(num == 0)
    {
       store[num] = score;
      Output();
      return;
    }
   for(int i = 0; i <= 10; ++i)
    {
       store[num] = i;
      Cumput(score - i, num - 1);
    }
}

int main(int argc, char* argv[])
   {
     Cumput(90, 9);
     cout<<"总数:"<<sum<<endl;
```

```
    return 0;
}
```

面试例题 2：八皇后问题是一个古老而著名的问题，是回溯算法的典型例题。该问题是 19 世纪著名的数学家高斯 1850 年提出：在 8×8 格的国际象棋盘上摆放 8 个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上，问有多少种摆法。[英国某著名计算机图形图像公司面试题]

解析：递归实现 *n* 皇后问题。

算法分析：

数组 a、b、c 分别用来标记冲突，a 数组代表列冲突，从 a[0]~a[7] 代表第 0 列到第 7 列。如果某列上已经有皇后，则为 1，否则为 0。

数组 b 代表主对角线冲突，为 b[i-j+7]，即从 b[0]~b[14]。如果某条主对角线上已经有皇后，则为 1，否则为 0。

数组 c 代表从对角线冲突，为 c[i+j]，即从 c[0]~c[14]。如果某条从对角线上已经有皇后，则为 1，否则为 0。

代码如下：

```
#include <stdio.h>

static char Queen[8][8];
static int a[8];
static int b[15];
static int c[15];
static int iQueenNum=0; //记录总的棋盘状态数

void qu(int i);       //参数 i 代表行

int main()
{
  int iLine,iColumn;

  //棋盘初始化，空格为*，放置皇后的地方为@
  for(iLine=0;iLine<8;iLine++)
  {
    a[iLine]=0; //列标记初始化，表示无列冲突
    for(iColumn=0;iColumn<8;iColumn++)
      Queen[iLine][iColumn]='*';
  }

  //主、从对角线标记初始化，表示没有冲突
  for(iLine=0;iLine<15;iLine++)
    b[iLine]=c[iLine]=0;

  qu(0);
  return 0;
```

```
}
void qu(int i)
{
  int iColumn;
  for(iColumn=0;iColumn<8;iColumn++)
  {
    if(a[iColumn]==0&&b[i-iColumn+7]==0&&c[i+iColumn]==0)
    //如果无冲突
    {
      Queen[i][iColumn]='@';    //放皇后
      a[iColumn]=1;             //标记，下一次该列上不能放皇后
      b[i-iColumn+7]=1;         //标记，下一次该主对角线上不能放皇后
      c[i+iColumn]=1;           //标记，下一次该从对角线上不能放皇后
      if(i<7) qu(i+1);          //如果行还没有遍历完，进入下一行
      else //否则输出
      {
        //输出棋盘状态
        int iLine,iColumn;
        printf("第%d 种状态为：\n",++iQueenNum);
        for(iLine=0;iLine<8;iLine++)
        {
          for(iColumn=0;iColumn<8;iColumn++)
            printf("%c ",Queen[iLine][iColumn]);
          printf("\n");
        }
        printf("\n\n");
      }
      //如果前次的皇后放置导致后面的放置无论如何都不能满足要求，则回溯，重置
      Queen[i][iColumn]='*';
      a[iColumn]=0;
      b[i-iColumn+7]=0;
      c[i+iColumn]=0;
    }
  }
}
```