

精选380余个C/C++技术面试真题，详解应聘C/C++程序员的常见考点
揭秘知名IT企业用人的核心准则，解读应聘IT企业的成功要诀



C/C++程序员 面试宝典

另赠超值
学习视频



16.5小时多媒体教学视频

梁镇宇 等编著

清华大学出版社

第17章 思维拓展

本章主要介绍一些面试时与思维拓展相关的内容，例如一些经典常见的问题的算法的实现、面试经验分享以及群体面试这种情况地介绍解答。通过本章可以知道除了专业知识以外，在面试时也会遇到其他的情况，例如多个考官的群体面试如何应对等。应试者可以根据自己的实际情况，有目的地加强这些方面能力。

17.1 经典试题

经典试题主要是一些经典的算法题。例如，八皇后问题，经典矩形生成和汉诺塔大数乘法等。通过本小节可以了解这些经典题目的一种求解方式。

面试题 194 八皇后问题

【出现频率】★★★

【关键考点】

- ☐ 八皇后问题介绍；
- ☐ 回溯法介绍；
- ☐ 八皇后问题算法分析。

【考题分析】

八皇后问题是一个古老而著名的问题。该问题是 19 世纪著名的数学家高斯 1850 年提出：在一个 8×8 国际象棋盘上，有 8 个皇后，每个皇后占一格；要求皇后之间不会出现相互“攻击”的现象，即不能有两个皇后处在同一行、同一列或同一对角线上。问共有多少种不同的方法？

回溯算法也叫试探法，它是一种搜索问题的解的方法。回溯算法的基本思想是在一个包含所有解的解空间树中，按照深度优先的策略，从根结点出发搜索解空间树。算法搜索至解空间树的任意结点时，总是先判断该结点是否肯定不包含问题的解。如果肯定不包含，则跳过对以该结点为根的子树的系统搜索，逐层向其祖先结点回溯。否则，进入该子树，继续按深度优先的策略进行搜索。回溯法在用来求问题的所有解时，要回溯到根，且根结点的所有子树都已被搜索遍才结束。

八皇后问题有很多中解法，其中使用回溯法进行求解是其中一种。而回溯法也是最直接的一种解法，也较容易理解。

八皇后问题的回溯法算法，可以采用一维数组来进行处理。数组的下标 i 表示棋盘上的第 i 列， $a[i]$ 的值表示皇后在第 i 列所放的位置。例如， $a[1]=5$ ，表示在棋盘的第一列的

第五行放一个皇后。程序中首先假定 $a[1]=1$ ，表示第一个皇后放在棋盘的第一列的第一行的位置上，然后试探第二列中皇后可能的位置，找到合适的位置后，再处理后续的各列，这样通过各列的反复试探，可以最终找出皇后的全部摆放方法。

【答案】

八皇后问题可以使用回溯法进行求解，程序实现如下：

```
#include<stdio.h>
#define Queens 8          //定义结果数组的大小，也就是皇后的数目
int a[Queens +1];         //八皇后问题的皇后所在的行列位置，从 1 开始算起，所以加 1
int main()
{
    int i,k,flag,not_finish=1,count=0;
    //正在处理的元素下标，表示前 i-1 个元素已符合要求，正在处理第 i 个元素
    i=1;
    a[1]=1;                //为数组的第一个元素赋初值
    printf("The possible configuration of 8 queens are:\n");
    while(not_finish)      //not_finish=1:处理尚未结束
    {
        while(not_finish&&i<=Queens ) //处理尚未结束且还没处理到第 Queens 个元素
        {
            for(flag=1,k=1;flag&&k<i;k++) //判断是否有多个皇后在同一行
                if(a[k]==a[i])
                    flag=0;
            for(k=1;flag&&k<i;k++)          //判断是否有多个皇后在同一对角线
                if((a[i]==a[k]-(k-i)) || (a[i]==a[k]+(k-i)))
                    flag=0;
            if(!flag)                      //若存在矛盾不满足要求，需要重新设置第 i 个元素
            {
                if(a[i]==a[i-1])          //若 a[i] 的值已经经过一圈追上 a[i-1] 的值
                {
                    i--;                  //退回一步，重新试探处理前一个元素
                    if(i>1&&a[i]==Queens )
                        a[i]=1;           //当 a[i] 为 Queens 时将 a[i] 的值置
                    else
                        if(i==1&&a[i]==Queens )
                            not_finish=0; //当第一位的值达到 Queens 时结束
                    else a[i]++;           //将 a[i] 的值取下一个值
                }
                else
                {
                    if(a[i]==Queens )
                        a[i]=1;
                    else a[i]++;           //将 a[i] 的值取下一个值
                }
                else
                {
                    if(++i<=Queens )
                        if(a[i-1]==Queens )
                            a[i]=1;       //若前一个元素的值为 Queens 则 a[i]=1
                        else a[i]=a[i-1]+1; //否则元素的值为前一个元素的下一个值
                }
            }
        }
        if(not_finish)
        {
            ++count;
            printf((count-1)%3?" [%2d]: ":" \n[%2d]: ",count);
            for(k=1;k<=Queens ;k++)      //输出结果
                printf(" %d",a[k]);
            if(a[Queens -1]<Queens )

```

```

        a[Queens -1]++;           //修改倒数第二位的值
    else
        a[Queens -1]=1;
        i=Queens -1;             //开始寻找下一个满足条件的解
    }
}
}

```

面试题 195 经典矩形

【出现频率】★★★

【关键考点】

- 经典矩形问题介绍;
- 特点分析。

【考题分析】

经典矩形问题指的是利用数字生成一个矩阵，而该矩阵刚好是一个正方形。该矩阵内的数字是有规律的排序而形成矩形。比较常见的有以下几种形式。

第1种矩阵的形式，如下：

```

1   2   9  10
4   3   8  11
5   6   7  12
16  15  14  13

```

第2种矩阵的形式，如下：

```

1   2   6   7
3   5   8  13
4   9  12  14
10  11  15  16

```

第3种矩阵的形式，如下：

```

1   2   3   4
12  13  14   5
11  16  15   6
10   9   8   7

```

第4种矩阵的形式，如下：

```

7   6   5  16
8   1   4  15
9   2   3  14
10  11  12  13

```

【答案】

根据矩阵的特点，可以使用以下程序输出上述矩阵：

```

void Rectangle1()
{
    const int N=20;

```

```

int i=0,j=0,a[N][N];
int lap=1,m=1,n;
while(1)                                //开始循环
{
    cout<<"\n 请输入矩阵的行数 N(N>=2): ";
    cin>>n;
    cout<<endl;
    if(n>=2) break;
}
cout<<"第 1 种矩形: "<<endl;           //开始输出矩阵
a[i][j]=m++;
lap++;
while(lap<=n)
{
    if(lap%2==0)
    {
        for(j++;i<lap;i++)
            a[i][j]=m++;
        i--;
        for(j--;j>=0;j--)
            a[i][j]=m++;j++;
    }
    else
    {
        for(i++;j<lap;j++)
            a[i][j]=m++;
        j--;
        for(i--;i>=0;i--)
            a[i][j]=m++;i++;
    }
    lap++;
}
for(i=0;i<n;i++)
{
    for(j=0;j<n;j++)
        cout<<setw(4)<<setiosflags(ios::left)<<a[i][j];
    cout<<endl;
}
cout<<endl;
int nSnake[N][N];
cout<<"第 2 种矩形: "<<endl;           //开始输出矩阵
i=0,j=0;
int s=1,nNum=1;                          //s 标记升降方向,斜向上为升(s==1),斜向下为降(s==-1)
while(1)
{
    if(s==1)
    {
        nSnake[i][j]=nNum;
        if(i-1<0)
        {
            if(j+1==n)
                i++;
            else
                j++;
            s=-1;
        }
    }
    else
    {
        if(j+1==n)
            i++;
    }
}

```

```

        s=-1;
    }
    else
    {
        i--;
        j++;
    }
}
else
{
    nSnake[i][j]=nNum;
    if(j-1<0)
    {
        if(i+1==n)
            j++;
        else
            i++;
        s=1;
    }
    else
    {
        if(i+1==n)
        {
            j++;
            s=1;
        }
        else
        {
            i++;
            j--;
        }
    }
    nNum++;
    if(nNum>n*n)
        break;
}
for(i=0;i<n;i++)
{
    for(j=0;j<n;j++)
        cout<<setw(4)<<setiosflags(ios::left)<<nSnake[i][j];
    cout<<endl;
}
cout<<endl;
cout<<"第3种矩形: "<<endl; //开始输出矩阵
i=0,j=0;
int x1,x2,y1,y2;           //x1, x2, y1, y2 为上、下、左、右边界
m=1,s=1;                   //s 标记数组元素升降, s==1 为升, s==-1 为降
x1=0;y1=0;x2=n;y2=n;
while(1)
{
    if(s==1)
    {
        for(j;j<y2;j++)
            a[i][j]=m++;
        j--;i++;y2--;
        for(i;i<x2;i++)
            a[i][j]=m++;
        i--;j--;x2--;
        s=-1;
    }
    else
    {

```

```

        for (j; j >= y1; j--)
            a[i][j] = m++;
        j++; i--; y1++;
        for (i; i >= x1+1; i--)
            a[i][j] = m++;
        i++; j++; x1++;
        s = 1;
    }
    if (m > n * n)
        break;
}
for (i = 0; i < n; i++)
{
    for (j = 0; j < n; j++)
        cout << setw(4) << setiosflags(ios::left) << a[i][j];
    cout << endl;
}
cout << endl;
cout << "第 4 种矩形: " << endl;           // 开始输出矩阵
i = 0, j = 0;
m = n * n;
x1 = 0; y1 = 0; x2 = n; y2 = n;
if (n % 2 == 0)                               // 求余
{ j = n - 1; y2 = n - 1; s = 1; }
else
{ i = n - 1; y1 = 1; s = -1; }
while (1)
{
    if (s == 1)
    {
        for (i; i < x2; i++)
            a[i][j] = m--;
        i--; j--; x2--;
        for (j; j >= y1; j--)
            a[i][j] = m--;
        j++; i--; y1++;
        s = -1;
    }
    else
    {
        for (i; i >= x1; i--)
            a[i][j] = m--;
        i++; j++; x1++;
        for (j; j < y2; j++)
            a[i][j] = m--;
        j--; i++; y2--;
        s = 1;
    }
}
if (m < 1)
    break;
}
for (i = 0; i < n; i++)
{
    for (j = 0; j < n; j++)
        cout << setw(4) << setiosflags(ios::left) << a[i][j];
    cout << endl;
}
cout << endl;                               // 输出结束标志
}

```

面试题 196 汉诺塔

【出现频率】★★★★

【关键考点】

□ 汉诺塔简介;

□ 汉诺塔算法。

【考题分析】

汉诺塔问题描述: 有 3 根杆子 A、B、C。A 杆上有若干碟子, 碟子按照从小到大的顺序从上往下放。目标是将 A 杆上的碟子全部移到 C 杆上, 而且移动后碟子的顺序和原来 A 杆上一样是有序的。在移动的过程中, 每次只能移动一个碟子, 而且小的只能叠在大的上面。求如何移动碟子的方法。


汉诺塔问题是一个经典的递归问题。利用递归的算法进行实现, 比较简单。例如, 要将 m 个碟子从 A 移动到 C, 可以认为如果能够先将 $m-1$ 个碟子移动到 B, 再将 A 剩下的最大的碟子移动到 C, 然后再将 B 上的 $m-1$ 个碟子移动到 C 即可。

【答案】

前面在第 4 章曾经介绍过汉诺塔的递归代码, 这里使用最简略的代码如下:

```
#include <iostream>
#include <string>
void Hanno(int i,string a,string b,string c);
int main()
{
    cout<<"输入要移动的数目:";
    int n;
    cin>>n;
    cout<<endl;
    Hanno(n,"A","B","C");
}

void Hanno(int i,string a,string b,string c)
{
    if (i==1)
    {
        cout<<a<<">"<<c<<endl;    //当只有一个碟子时直接将其从 A 移动到 C
    }
    else
    {
        Hanno(i-1,a,c,b);            //将 i-1 个碟子从 A 借助 C 移动到 B
        cout<<a<<">"<<c<<endl;    //将 A 剩下的最大的碟子移动到 C
        Hanno(i-1,b,a,c);            //将 i-1 个碟子从 B 借助 A 移动到 C
    }
}
```

 **注意:** 虽然递归算法是最直接的算法, 也是比较容易理解的。但是在一些系统资源紧缺的设备上进行开发时, 使用递归算法很可能导致资源耗尽。因此需要根据具体的情况对算法进行优化, 减少递归的使用。

面试题 197 新娘和新郎问题

【出现频率】★★★

【关键点】

□ 问题介绍;

□ 算法分析。

【考题分析】

问题介绍:

3 对情侣参加婚礼, 3 个新郎为 A、B、C, 3 个新娘为 X、Y、Z。有人不知道谁和谁结婚, 于是询问了 6 位新人中的 3 位, 但听到的回答是这样的: A 说他将和 X 结婚; X 说她的未婚夫是 C; C 说他将和 Z 结婚。这人听后知道他们在开玩笑, 全是假话。请编程找出谁将和谁结婚。

算法分析:

将 A、B、C 这 3 人用 1、2、3 表示, 将 X 和 A 结婚表示为“X=1”, 将 Y 不与 A 结婚表示为“Y!=1”。按照题目中的叙述可以写出表达式: $x!=1$ A 不与 X 结婚 $x!=3$ X 的未婚夫不是 C $z!=3$ C 不与 Z 结婚 题意还隐含着 X、Y、Z 这 3 个新娘不能结为配偶, 则有: $x!=y$ 且 $x!=z$ 且 $y!=z$ 穷举以上所有可能的情况, 代入上述表达式中进行推理运算, 若假设的情况使上述表达式的结果均为真, 则假设情况就是正确的结果。

根据算法分析, 可以利用计算机程序对这些情况进行穷举, 然后得出正确的结果

【答案】

根据算法分析, 该问题的程序实现如下:

```
#include<stdio.h>
int main()
{
    Marry();
}
void Marry()
{
    int x,y,z;
    for(x=1;x<=3;x++)           //穷举 x 的全部可能配偶
    for(y=1;y<=3;y++)           //穷举 y 的全部可能配偶
    for(z=1;z<=3;z++)           //穷举 z 的全部可能配偶
    if (x!=1&&x!=3&&z!=3&&x!=y&&x!=z&&y!=z) //判断配偶是否满足题意
    {
        printf("X 和%c 结婚\n",'A'+x-1); //打印判断结果
        printf("Y 和%c 结婚\n",'A'+y-1);
        printf("Z 和%c 结婚\n",'A'+z-1);
    }
}
```

面试题 198 大数乘法

【出现频率】★★★

【关键考点】

- 什么是大数乘法;
- 大数乘法的算法。

【考题分析】

在计算机中,由于计算机所表示的位数是有限的,而在一些科学应用中经常会出现两个很大的数进行相乘的情况。这时就不能直接使用 C++ 内置的数据类型进行运算,因为很大可能会导致溢出。而此时一般都为大数乘法专门设计一种算法。

大数乘法因为数本身的表示已经超出了 C++ 内置数据类型所表示的范围,一般在处理大数乘法时都使用别的表示方式。一种方式就是通过使用链表来表示数据和结果。乘法的操作就是在该链表上进行。

【答案】

大数乘法的一个实现是:


```
#include<iostream>
using namespace std;
#define MAX 1000000
struct Node{                                //使用结构保存运算结果
    int data;
    Node *next;
};
void output(Node *head)                     //输出运算结果
{
    if(!head->next&&!head->data) return;
    output(head->next);
    cout<<head->data;
}
void Mul(char *a,char *b)
{
    char *ap=a,*bp=b;                      //如果有的话,获取参数 a 和 b 的第一个值
    Node *head=0;
    head=new Node;
    head->data=0;
    head->next=0;                            //创建运算结果的结构
    Node *p,*q=head,*p1;
    int temp=0,temp1,bit;
    while(*bp)                               //若乘数不为空,继续进行处理
    {
        p=q->next;p1=q;
        bit=*bp-48;                          //把当前结点的值转为相应的整型表示
        while(*ap||temp)                     //若被乘数不空,继续进行处理
        {
            if(!p)                            //若要操作的结点为空,则新建一个结点
            {
                p=new Node;
                p->data=0;
                p->next=0;
                p1->next=p;
            }
            if(*ap==0)
                temp1=temp;
            else
```

```

    {
        temp1=(p1->data)+(*ap-48)*bit+temp;
        ap++;
    }
    p1->data=temp1%10;                //留当前位
    temp=temp1/10;                    //进位以 int 的形式留下
    p1=p;
    p=p->next;                          //被乘数到下 1 位
}
ap=a;
bp++;
q=q->next;                            //q 进下 1 位
}
p=head;
output(p);                            //显示运算结果
cout<<endl;
while(head)                            //释放空间
{
    p=head->next;
    delete head;
    head=p;
}
}
int main()
{
    cout<<"请输入两个相乘的数"<<endl;
    char a[MAX],b[MAX];                //使用数组保存待相乘的大数
    cin.getline(a,MAX,'\n');
    cin.getline(b,MAX,'\n');
    Mul(strrev(a),strrev(b));

    return 0;
}

```

 **注意：**大数相乘是经常会遇到的问题，虽然已经有成熟的实现方案，但是如果能够通过实际动手实现大数乘法，可以更好地使用。同时在遇到现有方案不能满足要求时，应当能够进行修改以适合实际需求。

17.2 面试经验分享

本小节将介绍的是真实的面试经历的介绍。这一次面试并没有成功，但是反映出来了很多面试时需要注意的问题，非常具有代表性。在这里与大家分享，希望读者可以从中间感受到面试的很多注意事项，从而成功地通过每一次面试。

这个面试经验分享分为两部分。

- ☐ 面试的具体经过；
- ☐ 由面试想到的。

17.2.1 面试经过

应聘者接到了某家跨国企业中国分公司的面试通知，这是一个非常好的机会，接到面

试后的几天，应聘者把一些专业课温习了一遍，特别是 C++ 和数据结构。由于在大学几年里，该应聘者一直钻研这些方面，加上也顺利通过了高级程序员的考试，对于一些常用的算法应聘者可以说差不多达到了烂熟于胸的地步，当时的感觉是如果问到这些方面的问题应该是没有问题的。

应聘者被安排在 16:30 面试。由一位技术人员单独面试，在问了一些简单的问题之后考官给出了一道编程题目。题目是这样的：

写一个函数计算当参数为 n (n 很大) 时的值 $1-2+3-4+5-6+7+\dots+n$ 。

看到题目以后应聘者，有点紧张的心情顿时放松起来。

于是应聘者很快给出了自己解法代码如下：

```
long fn(long n)
{
    long temp=0;
    int i,flag=1;
    if(n<=0)
    {
        printf("error: n must > 0); //输出
        exit(1);
    }
    for(i=1;i<=n;i++)           //循环
    {
        temp=temp+flag*i;
        flag=(-1)*flag;
    }
    return temp;
}
```

当应聘者用期待的目光看着面试官的时候，考官微笑着说，执行结果肯定是没有问题。但当 n 很大的时候这个程序执行效率很低。在嵌入式系统的开发中，程序的运行效率很重要，能让 CPU 少执行一条指令都是好的，考官让应聘者看看这个程序还有什么可以修改的地方，把程序优化一下。之后应聘者就对程序进行了严格的分析，给出了改进了的方案，示例代码如下：

```
long fn(long n)
{
    long temp=0;
    int j=1,i=1,flag=1;
    if(n<=0)
    {
        printf("error: n must > 0); //输出
        exit(1);
    }
    while(j<=n)                 //循环
    {
        temp=temp+i;
        i=-i;
        i>0?i++:i--;            //三元运算
        j++;
    }
    return temp;
}
```

虽然应聘者并不敢保证这个算法是最优的，但是比起上一个程序，将所有涉及到乘法指令的语句改为执行加法指令，既达到了题目的要求在运算时间上又缩短了很多。而代价

仅仅是增加了一个整型变量。但是由于现在的信心已经受了一点打击，应聘者将信将疑的看者考官，他还是微笑着跟我说：“不错，这个程序确实在效率上有的很大的提高。”应聘者心里一阵暗喜！但考官接着说这个程序仍然不能达到要求，要求给出更优的方案。应聘者当时情绪真的有些低落了，开始紧张起来，想了一会后，应聘者请求考官给出他的方案，然后他很爽快的给出了程序详细如下所示：

```
long fn(long n)
{
    if(n<=0)
    {
        printf("error: n must > 0); //输出
        exit(1);
    }
    if(0==n%2)                //判断整除
        return (n/2)*(-1);
    else
        return (n/2)*(-1)+n;
}
```

当时应聘者目瞪口呆，没想到考官考查的是这个意思，这么简单的代码真的不会写吗？但是为什么没有往那方面上想呢？考官的意思没有错，在 n 很大很大的时候这 3 个程序运行时间的差别简直是天壤之别。当应聘者刚想说点什么的时候，考官已经先说了：“不要认为 CPU 运算速度快就把所有的问题都推给它去做，程序员应该将代码优化再优化，程序员自己能做的决不要让 CPU 做，因为 CPU 是为用户服务的，不是为程序员服务的。多么精辟的语言，应聘者已经被深深触动了。接着是第二个问题：使用一种技巧性的编程方法来用一个函数实现两个函数的功能 n 为如： $fn1(n)=n/2!+n/3!+n/4!+n/5!+n/6!$ ， $fn2(n)=n/5!+n/6!+n/7!+n/8!+n/9!$ ，现在用一个函数 $fn(int n,int flag)$ 实现，当 $flag$ 为 0 时，实现 $fn1$ 功能，如果 $flag$ 为 1 时实现 $fn2$ 功能。他的要求还是程序的执行效率。说实话，如果应聘者心情好的话应该能给出一种比较好的算法，但那时应聘者已经手忙脚乱，没有心思再想了，于是应聘者在纸上随便写了一些诸如 $6!=6*5!$ 的公式后，直截了当的跟考官说请求给出他的答案。考官也没有说什么，给出了他的思路：

定义一个二维数组 `float t[2][5]` 存入 $\{2!,3!,4!,5!,6!\},\{5!,6!,7!,8!,9!\}$ 然后给出一个循环，示例如下：

```
for(i=0;i<6;i++)
{
    temp=temp+n/t[flag];
}
```

最后得到计算值。这是一个非常典型的空间换时间的算法。

这次面试总共花了 50 分钟的时间。还有 10 分钟时间，应聘者和考官很随意地聊了一些关于编程和生活方面的问题，就结束了面试。

17.2.2 由面试想到的

开始的时候自信满满，结果却是很失败。应聘者记得那天下好大的雨，气温也很低，回去的路上边走边想，从早上 5:30 一直到 19:30，全身都湿透了，又冷又饿，但是也只是走，脑子里面充满了疑惑，当时应聘者也想让雨把自己淋醒。看到这里，有些朋友可

能觉得那些面试题目不算什么如果让自己做的话肯定能全部答对。笔者也相信，因为笔者从未怀疑过中国程序员的能力，中国有着世界上最好的程序员，这位应聘者做不出来不代表中国大陆程序员比我国台湾或者别的地方的程序员差，所以从该应聘者的角度，谈一些感想：

应聘者从学习编程以来，不管是自学还是老师指导，从来都是解决问题就好，编出程序来就行，但是这样是没有真正的强调过程序的效率，程序的质量。我们有没有经常仔细分析过写的东西，看看有没有可以改进的地方，看看有没有简单的方法来达到同样的目的。扪心自问，应聘者发现，自己从来没有对写出来的程序进行过优化，最多就是进行详细的测试，然后进行调试。

还有一个疑惑就是平时工作的时候，说的和做的真的一样吗？应聘者在学校的时候曾经受学院指派承办过一个计算机大赛，请了一个老师出决赛的题目。主要是一些算法题目，这个老师可能是应聘者上大学以来唯一敬佩的老师了，从程序调试到打分，对于每个程序都仔细分析其时间效率和空间效率，然后综合打分，40个人的卷子，老师从下午3点一直调试到晚上十点，在有些写的精彩的语句后还加上批注。当时真是很高兴遇到这样的老师并且和他作了深入的交流，但在事后，却发生了一件不愉快的事，在比赛中获得第二名的学生找到了该应聘者，说他程序全部调试成功应该给他满分，并且应该得第一，当时调出了他的原程序和第一名的原程序对比，不错，两个程序都运行的很好，这时，那个同学开口了：“我的程序写的十分简捷明了，仅仅数行就完成了题目要求，而他的却写了一大堆，为什么给他的分数多过给我的分数。”当时很是郁闷，如果不是老师负责的话，那么现在第一名和第二名的位置真的要互调了，并不是程序的行数越少程序的质量就越高，应聘者记得跟他大谈这方面的道理，最后说服他了。这些只能说说而已，也许还有不少人也这样，说起来能够头头是道，但心里却从未重视过一些实际问题。

17.3 群体面试

很多用人单位的面试都是由一组人员负责。因而面试时，应聘者要同时面对一群职务和年龄及想法都不相同的人，怎样才能让他们对你产生好印象？

通常面试开始时都会有人先介绍主考人的姓名及职位，你最好记住面试负责人及其他面试者的名字或姓氏，好在应答问题时有针对性。若申请的职位需要专门技能，往往有专家在场，以判断申请人的专门技能。

面对一群考官，应聘者往往不知把目光投向哪一位，最好望着发问者；自己发问时可向主考官提出要求，假如你希望某一成员回答，可面向他发问，并说明希望由他来回答。

主试人之间未必事先都好好安排，所以当他们互投眼光时，不要觉得过分紧张。如果其中一个主试人对你特别挑剔或表示不满，也无需紧张。通常一群主试人中总有一个是会特意去充当这样的角色的，你只需从容应付即可。每个主试人都可能有偏见，你得小心应付。要是你出言不慎，得罪了其中一个，是可能会引起全体主试者厌恶的。这样面试的效果就不会太好了。

面试时常碰到两个主试人同时向你提出不同的问题。在这个时候，你必须耐心地逐一回答，不能回答其一而理会另一个的提问。最好在回答问题时稍观察一下提问人的反应，这样你的回答就会获得好的效果。