# 论数组思想

## 前言

我不知道读到此文的人是谁，也不知道其技术背景如何。我写下面这些内容，基于以下技术背景和知识领域：（1）C/C++（2）Java（3）数据结构（4）高等数学（5）数据库（6）概率学。如果读者对上述六个领域的知识有一定的了解，则更能体会本文内涵。如果没有，则不影响对本文的理解，但是能否体会和掌握，未可知。

## 1、数组知识上升为数组思想

有的人掌握了数组知识，觉得不过如此；有的人掌握了数组思想，觉得很伟大。我希望每个人都能重视数组的知识，进而升华成数组思想，体会数组这种数据结构的强大之处。

首先，我说一下如何让自己去产生数组思想：数组思想的产生不是凭空产生的，而是立足于丰富的数组知识之上的，是踩坑踩出来的。如果数组知识不丰富，如果在学习的过程中没有踩过坑，是不会花费时间去思考数组，没有时间的付出谈何思想的升华呢？在 Java 里面，数组的知识并不丰富，大家一学就会，自然就没有机会让人去思考数组了，而在 C/C++里面，数组的知识变化多端，在学习的过程中会踩很多的坑，坑踩的多，自然思考的机会就多，花费的时间就多，慢慢就能升华到数组思想了。

## 2、C/C++数组知识

### 2.1、数组名和数组类型的理解

因为最常见的是：int a = 10; 所以看到=，我们总是觉得：前面的变量名，后面的是变量值。但是在数组领域却不是这样的：

int a[10] = {1,2,3,4,5,6,7,8,9};
a[10]，不是变量的名称，a 才是变量的名称。
int[]，不是变量的类型，int 才是变量的类型。

a 不是普通的变量，而是指针变量，所以可以赋值给其他的指针：int *aptr = a;
a 不是普通的指针变量，即可以使用 a[i]访问，也可以使用*(a+i)访问。
aptr 不是普通的指针变量，即可以使用 aptr[i]访问，也可以使用*(aptr+i)访问。

### 2.2、定义数组

int a[10] = {1,2,3,4,5,6,7,8,9};
这种方式是完全没有必要的，这种可以说是静态数组，数组定的很死，两边必须保持一致，否则就容易出现问题。实际上应该是：
int a[] = {1,2,3,4,5,6,7,8,9};
这个绝不是省略的意思，这个暗含了深刻的思想。

**补充：**
在 Java 里面也有动态数组，静态数组
在 Java 里面 int[]表示一直类型，并不是 int 类型，这是与 C++区别所在。

### 2.3 数组名，指针，指针常量的关系

定义两个相同类型的变量，int a[5],b[5]；那么他们可以这样 b=a 赋值吗？答案是不可能。因为 a 和 b 是常量，一旦分配了内存，那么 a 和 b 值是不可能再有什么改变。

**注意：虽然数组名是指针，但是它是一个指针常量，不能被赋值。**

## 3、Java 数组知识

### 3.1、数组初始化

Java 语言的数组必须经过初始化才能够使用。所谓初始化就是为元素分配内存空间，并为每个数组元素指定初始值。

数组的初始化方式有以下两种：

**（1）静态初始化：指在定义数组变量的同时指定数组元素。**
**int [] arr = {1,2,3,4,5};**
**在静态初始化时，不需要指定数组的大小，系统会根据指定的内容自动分配大小的。**
初始化时候由程序员显示指定每个数组元素的初始值，由系统觉得初始值的长度。

**（2）动态初始化：指在定义时首先通过 new 关键字开辟指定大小的存储空间，然后再为存储单元指定内容。**
**int [] arr = new int[3];**
**arr[0]=10;**
**arr[1]=20;**

**注意：**
第一：不要同时使用静态初始化和动态初始化。也就是说不要在数组初始化时候，既指定数组的长度，也为每个数组元素分配初始值。
**第二：数组变量是存放在栈内存中的，数组对象是存放在堆内存中的。**

### 3.2、多维数组定义

在通过 new 关键字创建多维数组时，不必指定每一维大小，只需要指定最左边的维的大小即可。如果指定了某一维的大小，那么处在这一维左边的各维大小都需要指定，否则将编译出错。

### 3.3、数组默认初始值

使用 new 关键字创建数组对象时，在分配存储空间后，系统会为每一个存储单元默认初始化，其规则遵循成员变量默认初始化规则，例如，int 初始化为 0，boolean 初始化为 false。

```
public class Test{
static int arr[] = new int[10];
public static void main(String a[])
{
      System.out.println(arr[1]);
      }
}
```

### 3.4、数组的遍历

第一：原有的 for 循环

```
String [] arr = {"tom","rose","sunny"};
for(int i = 0; i<arr.length; i++){
System.out.println(arr[i]);
}
```

第二：从 Java JDK 5.0 开始提供了新式的 for 循环

```
String [] arr = {"tom","rose","sunny"};
for(String s: arr){
System.out.println(s);
}
```

### 3.5、数组的复制

**使用 system 类中一个静态方法** *arraycopy*()该方法的定义如下：

*static void arraycopy*(Object src, int srcPos, Object dest, int destPos, int length);

参数的含义如下：

scr：源数组

srcPos：源数组的起始位置

dest：目标数组

destPos：目标数组的起始位置

length：复制的数组元素的数量

## 4、数组与哈希的关系

接下来，我说一下如何判断一个人是否具备扎实的数组知识和良好的数组思想，我想出了一个不错的面试题。我在面试的过程中，经常喜欢问的一个问题是：哈希和数组有什么关系呢？可惜很多人都没有答出来。我觉得，大家不应该做一个只会搬砖的码农，只会复制粘贴，只会做开发 Web 页面，只会写后台 action、service，只会操作 sql，只会搭建分布式框架的码农，更应该学会思考，做一个有思想的工程师。只有多思考，才能将数据结构知识深度地掌握，才能提升自我能力，这才是一个程序员的核心竞争力。客观的说，凡是对这个问题没有答出来的人，很难被录取的，因为我觉得连基本的数组都没有掌握好，很难成为成长起来的。这个问题的答案是这样的：哈希表的本质就是数组，它是数组的升华。查找元素的过程分为

两大步骤：
第一：定位 key 的哈希值
第二：根据哈希值定位元素的位置
哈希表的查找过程，耗时在计算哈希上面，然后得出哈希值之后定位元素的过程是非常快的。哈希值本质就是数组的下标索引。在此强调一点：哈希表的本质就是数组，所以它也是连续的，就完全把它当做普通数组对待就行了，不要被哈希表这个名词所干扰，误认为是高大上的东西。

### 5、数组思想的内涵

最后说一下，既然提到了数组思想，那么什么是数组思想呢？我觉得应该包括以下几点：
（1）深刻认识到数组查找的快。**数组的快，是世界第一快。**（至于第二快是谁，我后面有补充资料）。假如有一堆数据，总量为 10000，如果采用链表形式去存储，现在想找到中间的数据，则需要查找 5000 次，而数组则需要 1 次。如此大的差距，数组查找的快可不是快一点，而是非常快，需要深刻的认知，需要有主观感性的认识。
（2）明白数组查找快的根本原因。
2.1 数组是顺序存储，在内存中是一个整块；
2.2 数组元素的定位分为两步：第一步，计算下标 i*数组元素大小求出偏移地址；第二步：数组首地址+偏移地址就是元素 i 的地址
2.3 链表的内存不是连续的，只能一个一个查找，速度自然慢很多了。
（3）认识到哈希表的本质就是数组，它是数组的升华，这就是上面提到的面试题目。

哈希表的本质就是数组，所以它也是连续的，就完全把它当做普通数组对待就行了。不要被哈希表这个名词所干扰，误认为是高大上的东西。

### 6、数组思想体会与 C/C++

上文提到了丰富的数组知识，才有助于掌握深刻的数组思想。其中举出了 Java，C/C++的数组知识。对于 C/C++语言的学习，我这里再做些补充：这个 IT 行业有个潜规则，所有的公司招聘都会明示：xxx，yyy，zzz 开发语言任选一个。但是面试官会死磕数据结构和算法，而它们和 C/C++这样的底层语言结合最密切，尤其是数组和哈希。C/C++这门语言的学习，可以归入数据结构这个大类，以数组相关的知识为核心，构造一个聚类来学习，不必可以单独刻意去学习。

### 补充：世界第二快

上文说过，世界第一快是数组。世界第二快则是具备二分查找的数据结构。提到了二分查找，不得不提它的数学原理：指数爆炸。请读下文：《一枚硬币的两面：指数爆炸与二分之快》

### 一枚硬币的两面：指数爆炸与二分之快

## 1、折纸游戏

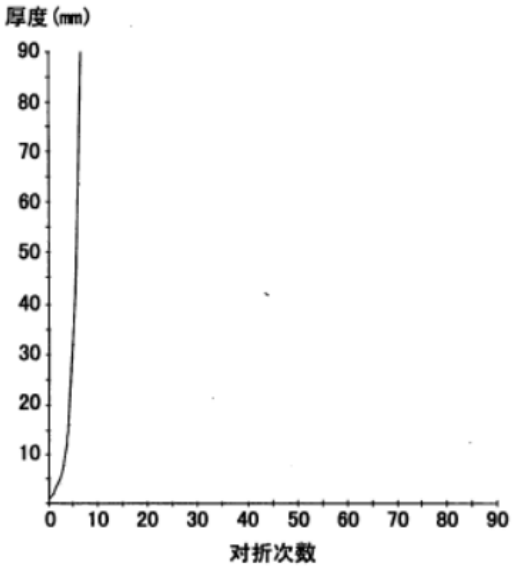
此为开篇之作，让我们来领略一下指数爆炸。请看下面的思考题：

假设现在有一张厚度为 1mm 的纸，纸质非常柔软，可以对折无数次。每对折 1 次，厚度便翻一番。已知地球距月球约 39 万公里，请问对折多少次后厚度能超过地月距离呢？

很多人会觉得这是一个非常大的数字，其实正确的答案是：39 次

足够震撼吧！仅仅对折了 39 次，就让 1mm 的纸的厚度达到了地球到月球的距离，实在是让人大吃一惊！我们把这种数字急速增长的情况称为指数爆炸。之所以称为指数爆炸是因为折纸时厚度($2^n$) 的指数 n 就是对折次数。

为使大家直观地理解指数爆炸，我们来画个图。横轴表示对折次数，纵轴表示厚度。




从图中可见，指数函数迅速攀升，其图像几乎垂直于 x 轴。

## 2、生日悖论

折纸游戏，略显无聊，谁没事去折纸玩呢！再来看一个与日常生活息息相关的现象吧：班级的人数达到多少人，至少有两个人同一天生日。所出现悖论的原因在于：这个问题的答案是一个很小的数值，让人吃惊。

正确的答案是：仅有 64 人的班级，至少有两个人的生日相同。

让我们看一些分析过程：

$$p=1-\frac{365\cdot 364\cdot\cdots\cdot(365-n+1)}{365^n}.$$

经计算可得下述结果：

| $n$ | 20 | 23 | 30 | 40 | 50 | 64 | 100 |
|---|---|---|---|---|---|---|---|
| $p$ | 0.411 | 0.507 | 0.706 | 0.891 | 0.970 | 0.997 | 0.999 999 7 |

其中，n 个人不同生日的概率为：

$$\frac{365\cdot 364\cdot\cdots\cdot(365-n+1)}{365^n}.$$

分母又是一个指数函数，将会变得巨大，其整体的值近似为 0 了。所以，当班级达到 64 人的时候，至少出现两个人生日的相同的概率将近 1。

### 3、索引的秘密武器：二分法

指数大爆炸，让我们领略了指数增长的巨大爆发力。而二分查找则是指数大爆炸的反例，在巨大海量的数据中，它能让我们领略快速查找的巨大威力。这种威力是难以想象中的震撼。

例如数据库中的查找，如果不采用任何措施的话，看似简单的查找，往往都是全表扫描，速度那可真是**龟速**。也许你觉得这是危言耸听，感觉无所谓，自认为当前的 CPU 十分的强悍，抱着一切无所谓的态度。看看下面一组数据的对比，让你对**龟速**有个直观的体验吧：

在一个 10000 条记录的数据表中，查找一次平均需要 5000 次比较，而在使用了索引的表中只需要 14 次。

看到了吧，这是 10000 对 14 的对比，这就是二分查找的巨大威力所在。