05.数组：为什么很多编程语言中数组都从0开始编号?

PDF版本已放至在本人github

1.什么是数组

数组（Array)是一种线性表数据结构，它用一组连续的内存空间，来存储一组具有相同类型的数据。

2.理解数组

理解以下几个关键词，就能彻底掌握数组的概念了。

2.1 线性表

线性表（Linear List):线性表就是数据排成像一条线一样的结构。每个线性表上的数据最多只有前和后两个方向。除了数组，链表、队列、栈等也是线性结构。

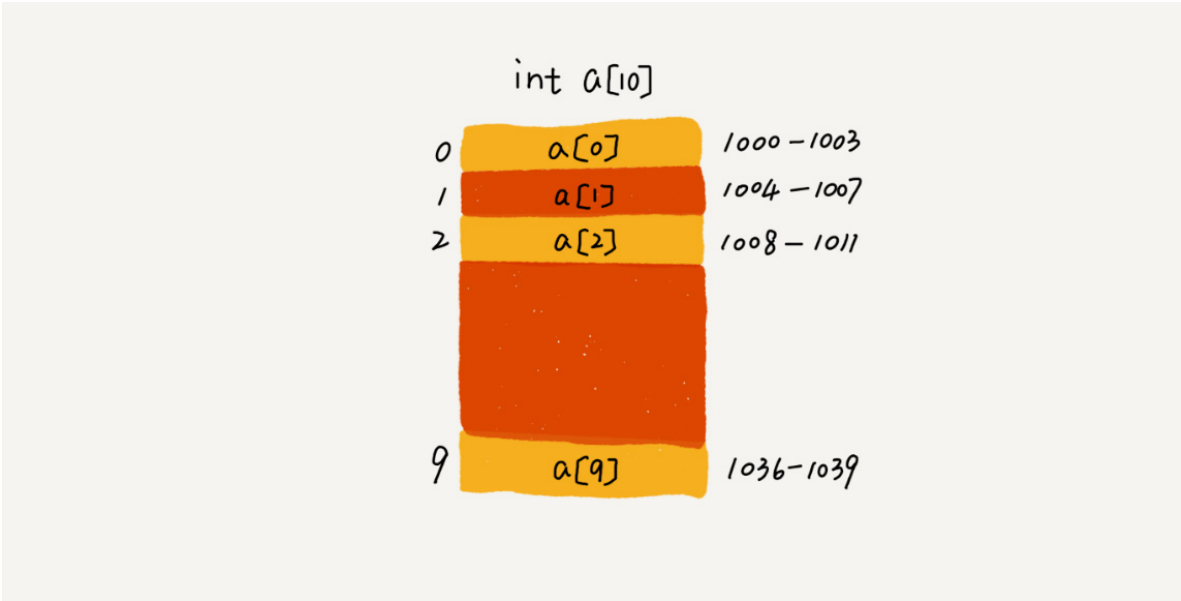
与之相对立的概念是非线性表，比如二叉树、堆、图等。在非线性表中，数据之间并不是简单的前后关系。

2.2 连续的内存空间和相同类型的数据

连续的内存空间和相同类型的数据：正是因为这个限制，使得数组有了**随机访问**的特性。同时也让数组的很多操作变得非常低效；比如要想在数组中删除、插入一个数据，为了保证连续性，就需要做大量的数据搬移工作。

3.数组如何实现根据下标随机访问数组元素

拿一个长度为10的int类型的数组$int\ []\ a = new\ int[10]$来举例。计算机给数组a[10] 分配了一块连续内存空间1000~1039，其中内存块的首地址为 base_address=1000。



计算机会给每个内存单元分配一个地址，计算机通过地址来访问内存中的数据。

当计算机需要访问数组中某个元素时，首先通过下方的寻址公式，计算出该元素存储的内存地址:

$$a[i]\_address = base\_address + i * data\_type\_size$$

其中data_type_size表示数组中每个元素的大小。这个例子中，数组中存储的是int类型，所以data_type_size就为4字节。

在面试的时候，常常会问数组和链表的区别，很多人回答“链表适合插入、删除，时间复杂度为O(1)；数组适合查找，查找时间复杂度为O(1)”。这种表述是不正确的，数组是适合查找，查找复杂度为并不为O(1).即便是排好序的数组，用二分查找时间复杂度也是O(logn)。所以正确的表述应该是：**数组支持随机访问，根据下标随机访问的时间复杂度为O(1).**

4.低效的插入和删除

数组为了保持内存数据的连续性，每次插入或者删除操作都会移动大量元素。，所以会导致低效。

4.1 插入操作

假设数组长度为n，我们需要将一个数据插入到数组第k个位置。为了把第k个位置腾出来，需要将k~n这部分的元素都顺序地往后挪一位。我们可以来分析一下它的时间复杂度。

如果在末尾插入，则不需要移动任何元素。此时时间复杂度为O(1)。

如果在开头插入，则需要移动n个元素。此时时间复杂度为O(n)。

因为我们在每个位置插入元素地概率是一样的，所以平均情况时间复杂度为（1+2+3+....n）/ n = O(n)

如果数据是有序的，我们插入数据就必须要按照上面这种方法

如果数据是无序的、没有任何规律的，数组只是被当作一个存储数据的集合。这种情况下如果要将某个数据插入到第k个位置，可以直接将第k位数据先搬到数组最后，再将新元素直接放到第k个位置。

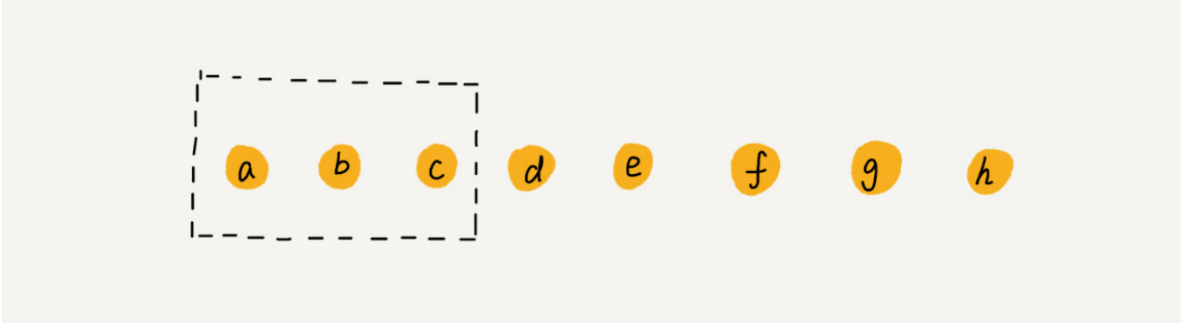
4.2 删除操作

与插入数据类似，如果要删除第k个位置地数据，为了内存地连续性，也需要搬移数据，不然中间会出现空洞，内存就不连续了。

如果删除末尾的数据，则最好的时间复杂度为O(1);如果删除开头的数据，则最坏的时间复杂度为O(n);平均情况时间复杂度为O(n)。

提高删除效率：在某些特殊场景下，我们并不一定非得追求数组中数据的连续性，如果我们将多次操作集中一起执行，删除效率会提高很多。

例如：数组a[10]中存储了8个元素：a,b,c,d,e,f,g,h。现在我们要依次删除a，b，c三个元素。



为了避免d,e,f,g,h这几个数据会被搬移三次，我们可以先记录下已经删除的数据，每次删除操作并不是真正地搬移数据，只是记录数据已经被删除。当数组没有更多空间存储数据时，在触发执行依次真正地删除操作，这样就大大减少了删除操作导致的数据搬移。

这个思想也是JVM标记清除垃圾回收算法地核心思想。

5.警惕数组越界问题

```
int main(int argc, char* argv[]){
    int i = 0;
    int arr[3] = {0};
    for(; i<=3; i++){
        arr[i] = 0;
        printf("hello world\n");
    }
    return 0;
}
```

这段代码运行结果并不是打印三行”hello world",而是会无限打印。因为for循环结束条件错写成了i<=3,当i=3时，数组a[3]访问越界。在C语言中，只要不是访问受限的内存，所有内存空间都是可以自由访问的。而a[3]刚好是存储变量 i 的内存地址，那么a[3]=0 就相当于 i=0，所以会导致代码无限循环。

这种错误debug难度非常大，很多计算机病毒也正是利用了代码中的数组越界可以访问非法地址的漏洞来攻击系统，所以写代码一定要警惕数组越界。

并非所有语言都像C一样，把数组越界检查工作丢给程序员做，像Java本身就会做越界检查。

6. 容器能否完全替代数组?

针对数组类型，很多语言提供了容器类，比如Java中的ArrayList 、C++STL中的vector。

什么时候适合用数组，什么时候适合用容器?

拿Java语言举例，如果你是Java工程师，几乎天天都在用ArrayList。其最大的优势就是**可以将很多数组操作的细节封装起来**，比如前面提到的数组插入、删除数据时需要搬移其他数据等。另外还有一个优势：**支持动态扩容。**

数组在定义时需要预先指定大小，因为需要分配连续的内存空间。如果我们申请了大小为10的数组，当第11个数据需要存储到数组中时，我们就需要从新分配一块更大的空间，将原来的数据复制过去，然后再将新的数据插入。

如果使用ArrayList,我们就完全不需要关心底层的扩容逻辑，ArrayList已经帮我们实现好了，每次存储空间不够的时候，它都会将空间自动扩容为1.5倍大小。

注意:因为扩容操作涉及内存申请和数据搬移，是比较耗时的。所以，如果事先能够确定需要存储的数据大小，最好在**创建ArrayList的时候事先指定数据大小。**可以省掉很多次内存申请和数据搬移操作。

经验:

1.Java ArrayList 无法存储基本类型，比如 int、long,需要封装为Integer、Long类，而Autoboxing、Unboxing则有一定的性能消耗。所以如果特别关注性能，或者希望使用基本类型，就可以使用数组。

2.如果数据大小事先已知，并且对数据的操作非常简单，用不到ArrayList提供的大部分方法，也可以直接使用数组。

3.当要表示多维数组时，用数组往往会更加直观。如：Object [] [] array;而用容器的话则需要这样定义：ArrayList<ArrayList> array。

对于业务开发，直接使用容器就够了，省时省力。毕竟损耗一丢丢性能，完全不会影响到系统整体的性能。但如果你是做一些非常底层的开发，比如开发网络框架，性能的优化需要做到极致，这个时候数组就会由于容器，成为首选。

7.解答开篇

为什么大多数编程语言中数组要从 0 开始编号，而不是从1 开始呢？

从数组存储的内存模型上来看，”下标“最确切的定义是”偏移（offset)“。a表示数组的首地址，a[0]就是偏移为0的位置，a[k]表示偏移k个type_size的位置，所以计算a[k]内存地址的公式为
$a[k]\_address = base\_address + k * type\_size$

但是如果从1开始编号，计算数组元素a[k]的内存地址就会变为
$a[k]\_address = base\_address + (k-1) * type\_size$

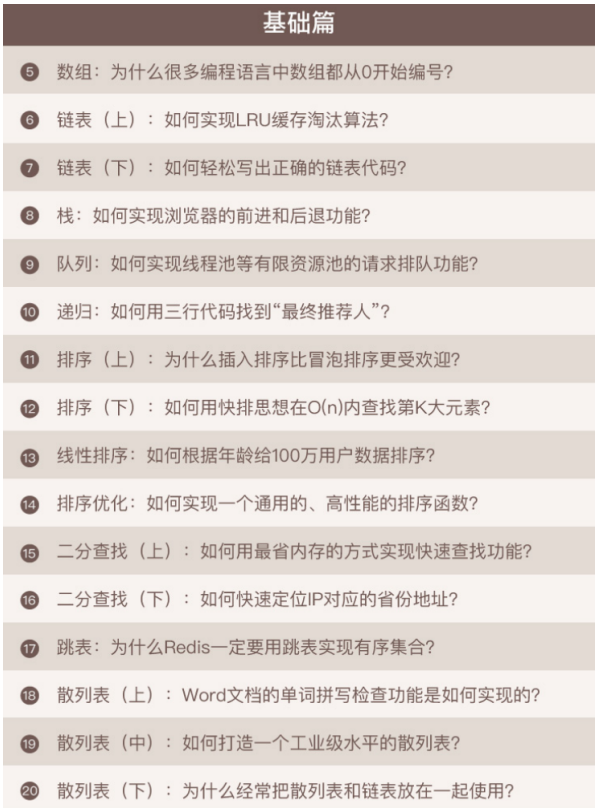
对比两个公式不难发现，从1开始编号，每次随机访问数组都多了一次减法运算，对于CPU来说，就是多了一次减法指令。数组作为非常基础的数据结构，通过下标随机访问元素又是非常基础的编程操作，效率优化要做到极致。

历史也有一定原因:C语言设计者用0开始计数数组下标，之后的Javad等高级语言也都效仿C语言，或者说减少C语言程序员学习Java的学习成本，因此继续沿用了从0计数的习惯。

8.参考

这个是我学习王争老师的《数据结构与算法之美》所做的笔记，王争老师是前谷歌工程师，该课程截止到目前已有87244人付费学习，质量不用多说。

截取了课程部分目录，课程结合实际应用场景，从概念开始层层剖析，由浅入深进行讲解。本人之前也学过许多数据结构与算法的课程，唯独王争老师的课给我一种茅塞顿开的感觉，强烈推荐大家购买学习。课程二维码我已放置在下方，大家想买的话可以扫码购买。



本人做的笔记并不全面，推荐大家扫码购买课程进行学习，而且课程非常便宜，学完后必有很大提高。