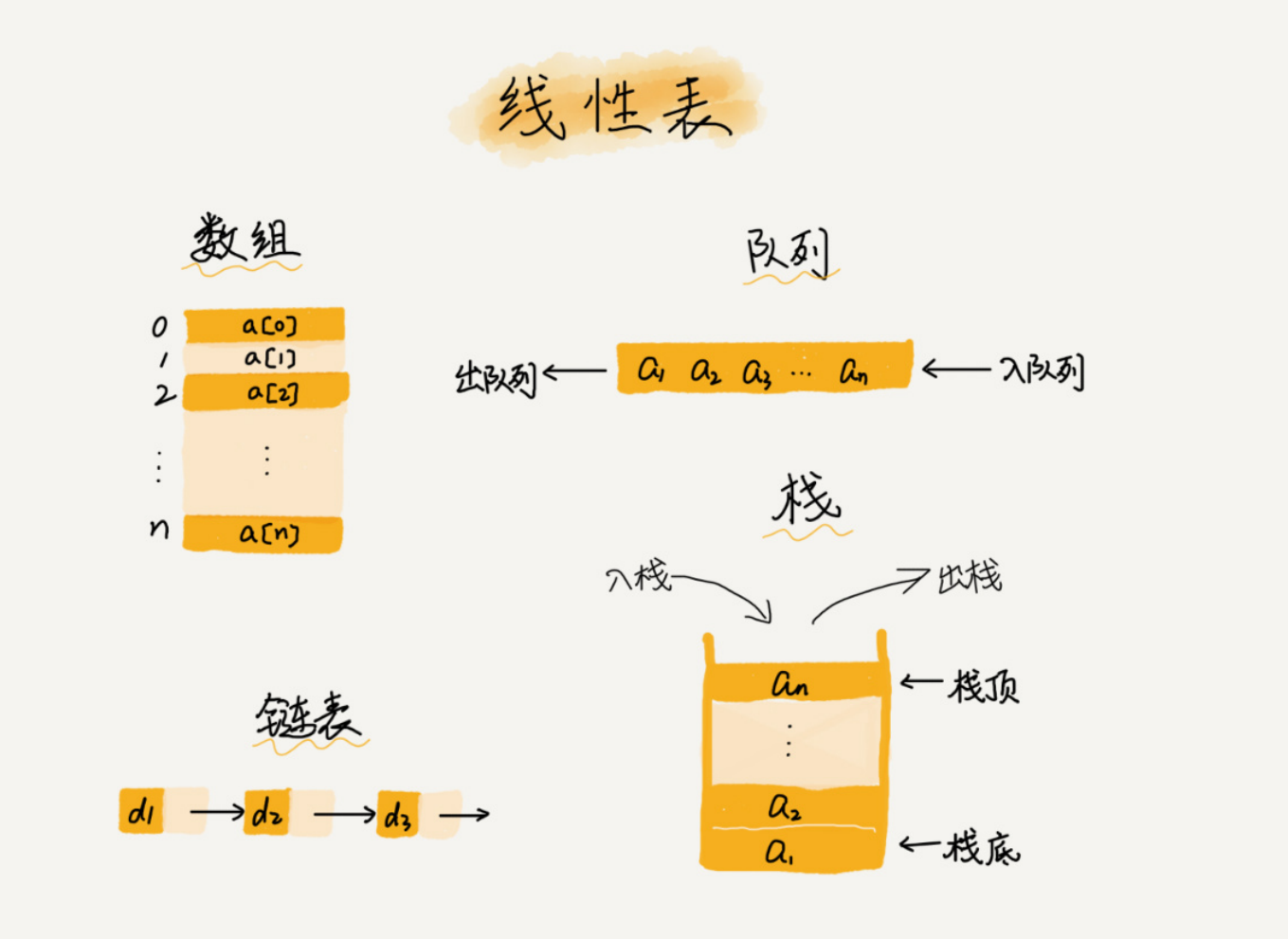
数组

题目：为什么数组要从0开始编号，而不是1开始呢？

如何实现随机访问？

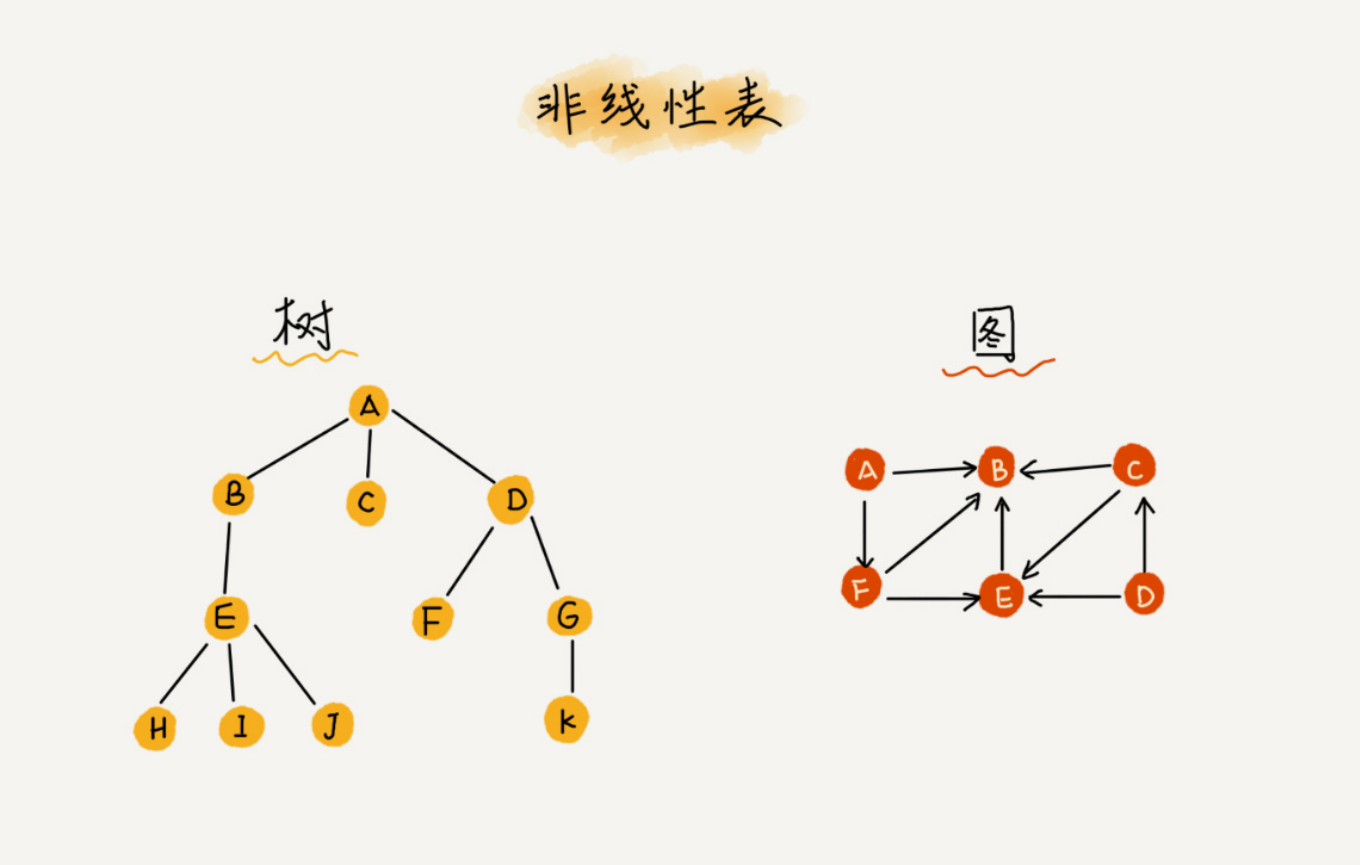
定义：数组是一种线性数据结构。它用一组连续的内存空间，来存储具有相同类型的数据。

第一个关键词是： 线性表：线性表上的数据最多只有前和后两个方向。其实除了数组，链表、队列、栈等也是线性表结构。



非线性表：比如二叉树、堆、图等。之所以叫非线性表，是因为，在非线性表中，数据之间并不是简单的前后

关系。



第二个关键词是： 连续的内存空间和相同类型的数据 正因为如此所以数组具有随机访问的特性。 但也使得数组中删除、插入一个数据，需要做大量的数据搬移工作。

创建一个数组时，计算机分配了一段连续内存地址空间，如果随机访问时，首先应该通过寻址公式，计算该元素的存储的内存地址：

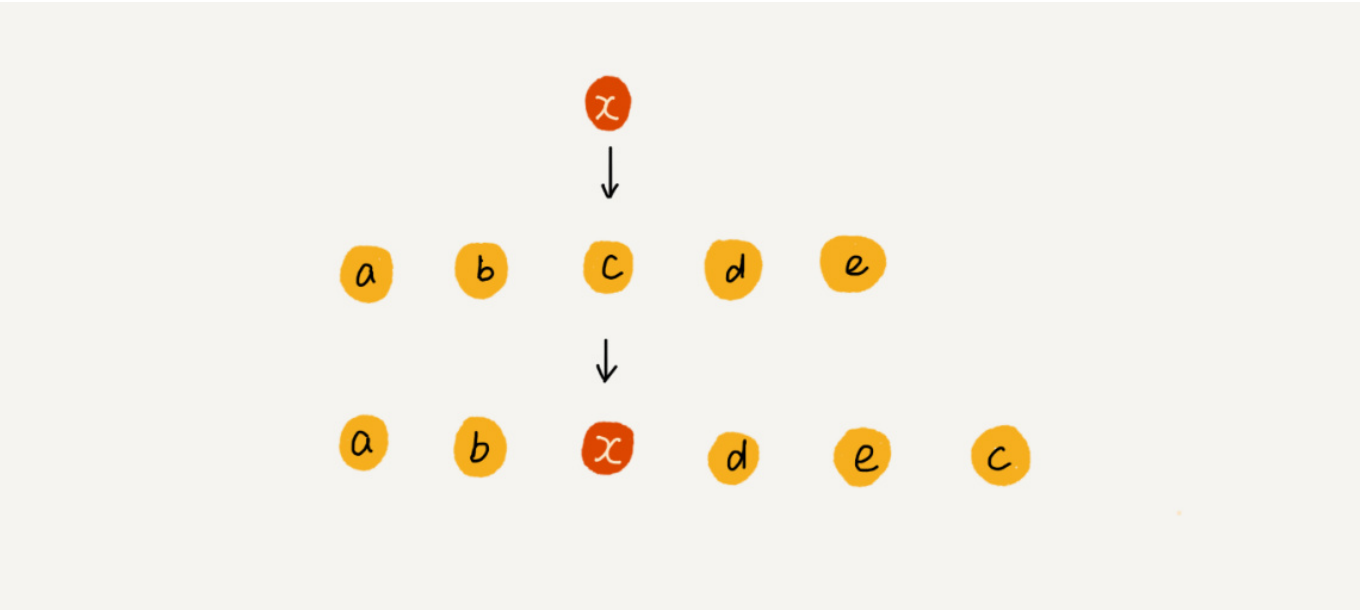
```
a[i]_address = base_address + i * data_type_size
```

数组是适合查找操作，但查找的时间复杂度并不为O(1)，即便是排好序的数组，你用二分查找，时间复杂度也是O(logn)。准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是O(1)

低效的“插入”和“删除”

插入操作 假设数组的长度是n，插入数据到k，需要将k~n都往后挪一位。所以插入的平均复杂度为(1+2+...+n)/n=O(n)

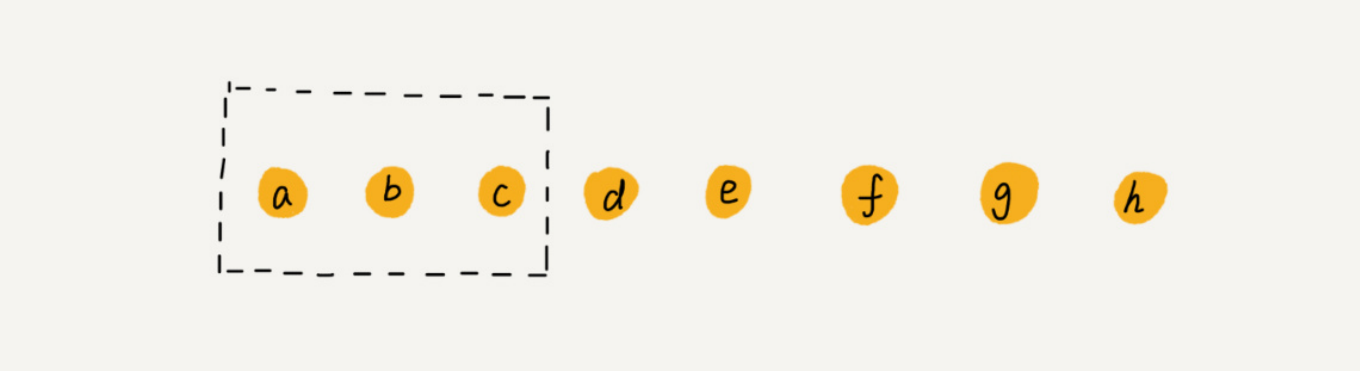
如果只是把数组当做一个集合，不考虑顺序，可以讲插入位置的数据放到最后，将插入数据插入，则O(1)



删除操作 删除第K个位置的数据，为了内存的连续性，也需要搬移数据，不然内存不连续了

删除操作，删除末尾的数据，最好时间复杂度为O(1)；删除开头位置的数据，最坏时间复杂度为O(n);平均情况复杂度也为O(n)

我们继续来看例子。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。每次的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，我们再触发执行一次真正的删除操作，这样就大大减少了删除操作导致的数据搬移。

这也就是**JVM标记清除垃圾回收算法的核心思想**

警惕数组访问越界问题

C语言不会对数组是否下标越界进行检查，除了访问受限的内存单元其他的都可以自由访问 JAVA语言会做数组下标检查。

容器是否完全代替数组

很多语言提供了容器类，比如java的ArrayList等，ArrayList最大的优势在于将很多数组操作的细节封装起来。还有一个优势就是支持动态扩容。

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。 如果事先能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。

```
ArrayList users = new ArrayList(10000);
```

总结，什么时候用数组更合适些

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，
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. 还有一个是我个人的喜好，当要表示多维数组时，用数组往往会更加直观。比如 Object[][] array；而用容器的话则需要这样定义：ArrayList > array。

解答开篇

从数组存储的内存模型上来看，“下标”最确切的定义应该是“偏移（offset）”。前面也讲到，如果用 a 来表示数组的首地址，a[0] 就是偏移为 0 的位置，也就是首地址，a[k] 就表示偏移 k 个 type_size 的位置，所以计算 a[k] 的内存地址只需要用这个公式：

```
a[k]_address = base_address + k * type_size
```

但是，如果数组从 1 开始计数，那我们计算数组元素 a[k] 的内存地址就会变为：

```
a[k]_address = base_address + (k-1)*type_size
```

所以为了减少一次减法操作，数组选择了从 0 开始编号，而不是从 1 开始。

课后思考

1. 前面我基于数组的原理引出 JVM 的标记清除垃圾回收算法的核心理念。我不知道你是否使用 Java 语言，理解 JVM，如果你熟悉，可以在评论区回顾下你理解的标记清除垃圾回收算法。

大多数主流虚拟机采用可达性分析算法来判断对象是否存活，在标记阶段，会遍历所有 GC ROOTS，将所有 GC ROOTS 可达的对象标记为存活。只有当标记工作完成后，清理工作才会开始。

不足：1.效率问题。标记和清理效率都不高，但是当知道只有少量垃圾产生时会很高效。2.空间问题。会产生不连续的内存空间碎片。

2. 前面我们讲到一维数组的内存寻址公式，那你可以思考一下，类比一下，二维数组的内存寻址公式是怎样的呢？

对于 m * n 的数组，a [i][j] (i < m,j < n)的地址为：

address = base_address + (i * n + j) * type_size