4.5 小结

1. 重点回顾

- 数组和链表是两种基本的数据结构，分别代表数据在计算机内存中的两种存储方式：连续空间存储和分散空间存储。两者的特点呈现出互补的特性。
- 数组支持随机访问、占用内存较少；但插入和删除元素效率低，且初始化后长度不可变。
- 链表通过更改引用（指针）实现高效的节点插入与删除，且可以灵活调整长度；但节点访问效率低、占用内存较多。常见的链表类型包括单向链表、环形链表、双向链表。
- 列表是一种支持增删查改的元素有序集合，通常基于动态数组实现。它保留了数组的优势，同时可以灵活调整长度。
- 列表的出现大幅提高了数组的实用性，但可能导致部分内存空间浪费。
- 程序运行时，数据主要存储在内存中。数组可提供更高的内存空间效率，而链表则在内存使用上更加灵活。
- 缓存通过缓存行、预取机制以及空间局部性和时间局部性等数据加载机制，为 CPU 提供快速数据访问，显著提升程序的执行效率。
- 由于数组具有更高的缓存命中率，因此它通常比链表更高效。在选择数据结构时，应根据具体需求和场景做出恰当选择。

2. Q & A

Q：数组存储在栈上和存储在堆上，对时间效率和空间效率是否有影响？

存储在栈上和堆上的数组都被存储在连续内存空间内，数据操作效率基本一致。然而，栈和堆具有各自的特点，从而导致以下不同点。

1. 分配和释放效率：栈是一块较小的内存，分配由编译器自动完成；而堆内存相对更大，可以在代码中动态分配，更容易碎片化。因此，堆上的分配和释放操作通常比栈上的慢。
2. 大小限制：栈内存相对较小，堆的大小一般受限于可用内存。因此堆更加适合存储大型数组。
3. 灵活性：栈上的数组的大小需要在编译时确定，而堆上的数组的大小可以在运行时动态确定。

Q：为什么数组要求相同类型的元素，而在链表中却没有强调相同类型呢？

链表由节点组成，节点之间通过引用（指针）连接，各个节点可以存储不同类型的数据，例如 `int`、`double`、`string`、`object` 等。

相对地，数组元素则必须是相同类型的，这样才能通过计算偏移量来获取对应元素位置。例如，数组同时包含 `int` 和 `long` 两种类型，单个元素分别占用 4 字节 和 8 字节，此时就不能用以下公式计算偏移量了，因为数组中包含了两种“元素长度”。

```
# 元素内存地址 = 数组内存地址（首元素内存地址） + 元素长度 * 元素索引
```

Q：删除节点 `P` 后，是否需要把 `P.next` 设为 `None` 呢？

不修改 `P.next` 也可以。从该链表的角度看，从头节点遍历到尾节点已经不会遇到 `P` 了。这意味着节点 `P` 已经从链表中删除了，此时节点 `P` 指向哪里都不会对该链表产生影响。

从数据结构与算法（做题）的角度看，不断开没有关系，只要保证程序的逻辑是正确的就行。从标准库的角度看，断开更加安全、逻辑更加清晰。如果不断开，假设被删除节点未被正常回收，那么它会影响后继节点的内存回收。

Q：在链表中插入和删除操作的时间复杂度是 $O(1)$ 。但是增删之前都需要 $O(n)$ 的时间查找元素，那为什么时间复杂度不是 $O(n)$ 呢？

如果是先查找元素、再删除元素，时间复杂度确实是 $O(n)$ 。然而，链表的 $O(1)$ 增删的优势可以在其他应用上得到体现。例如，双向队列适合使用链表实现，我们维护一个指针变量始终指向头节点、尾节点，每次插入与删除操作都是 $O(1)$ 。

Q：图“链表定义与存储方式”中，浅蓝色的存储节点指针是占用一块内存地址吗？还是和节点值各占一半呢？

该示意图只是定性表示，定量表示需要根据具体情况进行分析。

- 不同类型的节点值占用的空间是不同的，比如 `int`、`long`、`double` 和实例对象等。
- 指针变量占用的内存空间大小根据所使用的操作系统及编译环境而定，大多为 8 字节或 4 字节。

Q：在列表末尾添加元素是否时时刻刻都为 $O(1)$ ？

如果添加元素时超出列表长度，则需要先扩容列表再添加。系统会申请一块新的内存，并将原列表的所有元素搬运过去，这时候时间复杂度就会是 $O(n)$ 。

Q：“列表的出现极大地提高了数组的实用性，但可能导致部分内存空间浪费”，这里的空间浪费是指额外增加的变量如容量、长度、扩容倍数所占的内存吗？

这里的空间浪费主要有两方面含义：一方面，列表都会设定一个初始长度，我们不一定需要用这么多；另一方面，为了防止频繁扩容，扩容一般会乘以一个系数，比如 $\times 1.5$ 。这样一来，也会出现很多空位，我们通常不能完全填满它们。

Q：在 Python 中初始化 `n = [1, 2, 3]` 后，这 3 个元素的地址是相连的，但是初始化 `m = [2, 1, 3]` 会发现它们每个元素的 id 并不是连续的，而是分别跟 `n` 中的相同。这些元素的地址不连续，那么 `m` 还是数组吗？

假如把列表元素换成链表节点 `n = [n1, n2, n3, n4, n5]` ，通常情况下这 5 个节点对象也分散存储在内存各处。然而，给定一个列表索引，我们仍然可以在 $O(1)$ 时间内获取节点内存地址，从而访问到对应的节点。这是因为数组中存储的是节点的引用，而非节点本身。

与许多语言不同，Python 中的数字也被包装为对象，列表中存储的不是数字本身，而是对数字的引用。因此，我们会发现两个数组中的相同数字拥有同一个 id ，并且这些数字的内存地址无须连续。

Q：C++ STL 里面的 `std::list` 已经实现了双向链表，但好像一些算法书上不怎么直接使用它，是不是因为有什么局限性呢？

一方面，我们往往更青睐使用数组实现算法，而只在必要时才使用链表，主要有两个原因。

- 空间开销：由于每个元素需要两个额外的指针（一个用于前一个元素，一个用于后一个元素），所以 `std::list` 通常比 `std::vector` 更占用空间。
- 缓存不友好：由于数据不是连续存放的，因此 `std::list` 对缓存的利用率较低。一般情况下， `std::vector` 的性能会更好。

另一方面，必要使用链表的情况主要是二叉树和图。栈和队列往往会使用编程语言提供的 `stack` 和 `queue` ，而非链表。

Q：初始化列表 `res = [0] * self.size()` 操作，会导致 `res` 的每个元素引用相同的地址吗？

不会。但二维数组会有这个问题，例如初始化二维列表 `res = [[0] * self.size()]` ，则多次引用了同一个列表 `[0]` 。

上一页 ← 4.4 内存与缓存 *

下一页 第 5 章 栈与队列 →

欢迎在评论区留下你的见解、问题或建议