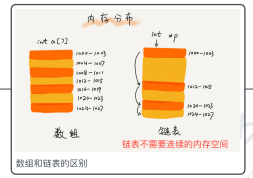
3.数据结构--链表

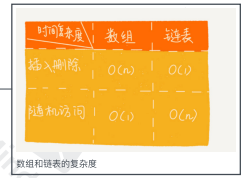
1. 链表与数组的比较

数组和链表的对比
数组的缺点是大小固定，一经声明就要占用整块连续内存空间。连续空间不够也不行，声明时不够用需要重新声明，把原数组拷贝进去，非常费时。链表本身没有大小的限制，天然地支持动态扩容，我觉得这也是它与数组最大的区别

除此之外，如果你的代码对内存的使用非常苛刻，那数组就要适合。因为链表中的每个结点都需要消耗额外的存储空间去存储一份指向下一个结点的指针，所以内存消耗会翻倍。而且，对链表进行频繁的插入、删除操作，还会导致频繁的内存申请和释放，容易造成内存碎片，如果是 Java 语言，就有可能会导致频繁的 GC（Garbage Collection，垃圾回收）



数组和链表的区别



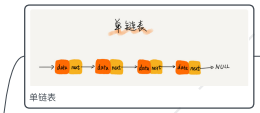
数组和链表的复杂度

2.常见链表的结构及操作

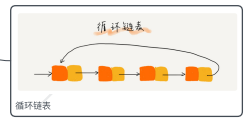
a 单链表
　头结点用来记录链表的基地址。有了它，我们就可以遍历得到整条链表。而尾结点特殊的地方是：指针不是指向下一个结点，而是指向一个空地址 NULL，表示这是链表上最后一个结点
　优缺点：支持快速插入和删除，链表的访问需要从头到尾依次查询 复杂度是O(n)

b 循环链表
　如图

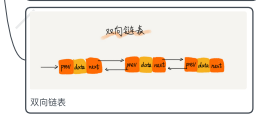
c 双向链表
　双向链表和单链表的对比，例如删除场景
　*删除结点中"值等于某个给定值"的结点：无论单双链表，在查询这个值时复杂度都是O(n)
　*删除给定指针指向的结点：单链表需要知道被删除节点的前向节点，所以仍需从头查找来实现删除。而双向链表中可以以O(1)的复杂度来完成删除动作
　*除了删除场景，当链表为有序时，查询场景中，由于双向链表的特点，可以保存上次的查找位置，直接从该位置开始查找，这样只需要查找一半的数据



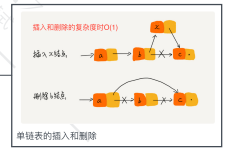
单链表



循环链表



双向链表



单链表的插入和删除

3. 链表常见操作

链表代码技巧

a 指针的含义：
　将某个变量赋值给指针，实际上就是将这个变量的地址赋值给指针，或者反过来说，指针中存储了这个变量的内存地址，指向了这个变量。通过指针就能找到这个变量

b 新增节点时：例如之前是a—>b,中间新增节点X，先将X—>b，再将a—>X

c 利用哨兵：head 指针都会一直指向这个哨兵结点。我们也把这种有哨兵结点的链表叫带头链表。相反，没有哨兵结点的链表就叫作不带头链表

d 画图：将各个情况举个例子，画出之前和之后的变化示意图

e 常见的链表操作：单链表反转 链表中环的检测 两个有序的链表合并 删除链表倒数第 n 个结点 求链表的中间结点

4.典型场景举例

如何基于链表实现 LRU 缓存淘汰算法？
　LRU 缓存淘汰算法，三种常见的策略：先进先出策略 FIFO（First In，First Out）、最少使用策略 LFU（Least Frequently Used）、最近最少使用策略 LRU（Least Recently Used）

　思路是这样的：我们维护一个有序单链表，越靠近链表尾部的结点是越早之前访问的。当有一个新的数据被访问时，我们从链表头开始顺序遍历链表。
1. 如果此数据之前已经被缓存在链表中了，我们遍历得到这个数据对应的结点，并将其从原来的位置删除，然后再插入到链表的头部。
2. 如果此数据没有在缓存链表中，又可以分为两种情况：
　*如果此时缓存未满，则将此结点直接插入到链表的头部；
　*如果此时缓存已满，则链表尾结点删除，将新的数据结点插入链表的头部