数组

申请数组时 Memory Controller 在内存中给你开辟一块连续的内存地址

通过访问 Memory Controller 来访问数组 复杂度 O(1)

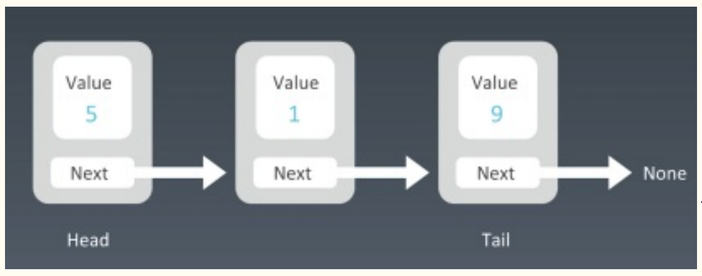
可以随机访问任何一个元素

问题在于 插入 和 删除 频繁的情况下, 数组不好用

了解这两步原理

插入需要挪动元素 O(1)、O(n)

删除也需要挪动元素, 然后给数组最后一位设置为空来触发GC, O(n)



Linked List

next 指向下一个元素, 多个元素串到一起就像类数组的结构

只有一个指针叫 单链表

两个指针叫双向链表

头指针用 Head 表示

尾指针用 Tail 表示

最后一个元素的 next 指向 None

如果最后一个元素的 next 指向 Head 就叫循环链表

添加结点 O(1)

删除节点 O(1)

访问链表的任意位置 O(n)

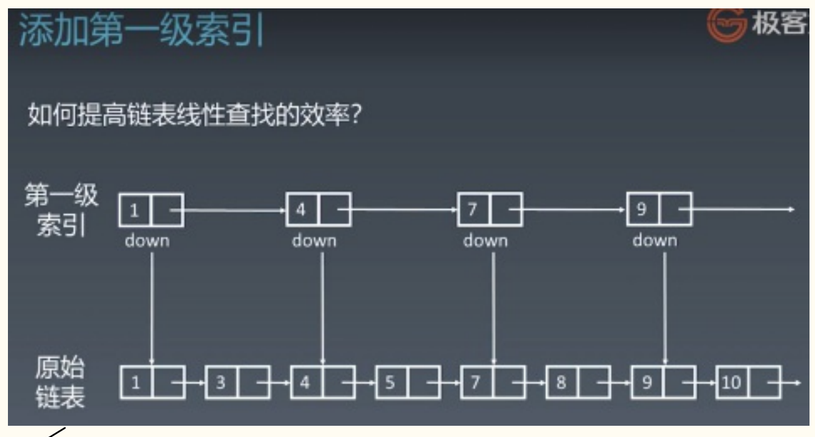
```
class LinkedList {
    Node head; // head of list

    /* Linked list Node*/
    class Node {
        int data;
        Node next;

        // Constructor to create a new node
        // Next is by default initialized
        // as null
        Node(int d) { data = d; }
    }
}
```

标准实现

Linked List 的标准实现代码

Linked List 示例代码

Java 源码分析（LinkedList）



Redis - Skip List：跳跃表、为啥 Redis 使用跳表（Skip List）而不是使用 Red-Black?

弥补 链表 的缺陷而产生

Skip List

Redis 里面运用

工程中应用

LRU Cache - Linked list：

LRU 缓存机制

如何给链表加速

普通链表时间复杂度: 查询 O(n)

简单优化: 添加头尾指针

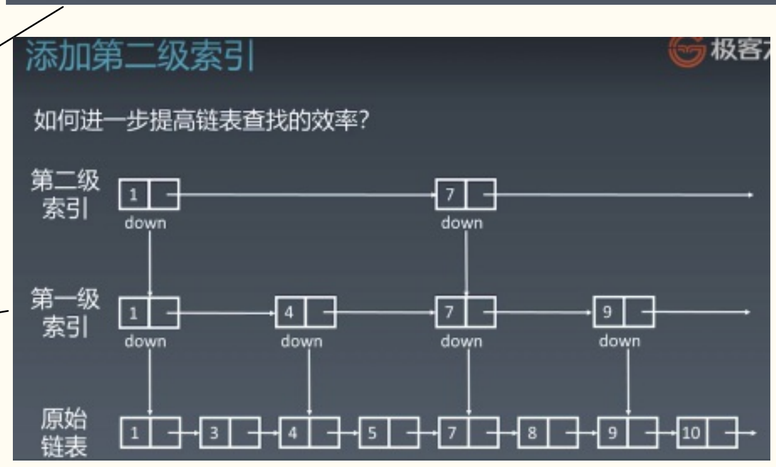
然后呢: 贯穿所有数据结构与算法的中心思想 务必记住

1.升维度 2.空间换时间

增加一级索引

增加二级索引

增加多级索引



跳表时间复杂度: 查询 O(logn)

索引高度: logn

维护成本高，增删节点都需要更新索引

增删复杂度变成: O(logn)

空间复杂度是 O(n)