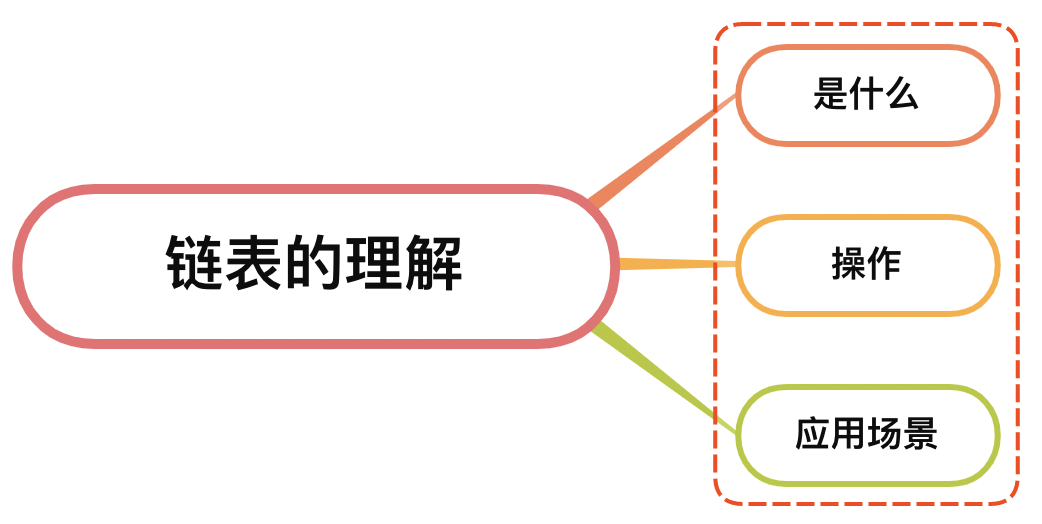
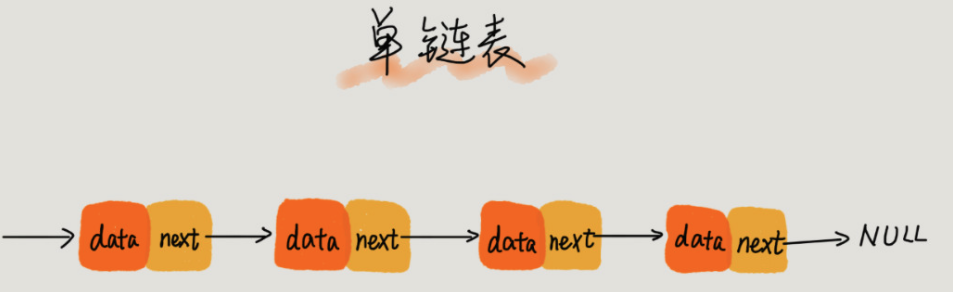
# 面试官：说说你对链表的理解？常见的操作有哪些？



## 一、是什么

链表（Linked List）是一种物理存储单元上非连续、非顺序的存储结构，数据元素的逻辑顺序是通过链表中的指针链接次序实现的，由一系列结点（链表中每一个元素称为结点）组成

每个结点包括两个部分：一个是存储数据元素的数据域，另一个是存储下一个结点地址的指针域



节点用代码表示，则如下：

class Node {  
 constructor(val) {  
 this.val = val;  
 this.next = null;  
 }  
}

* data 表示节点存放的数据
* next 表示下一个节点指向的内存空间

相比于线性表顺序结构，操作复杂。由于不必须按顺序存储，链表在插入的时候可以达到O(1)的复杂度，比另一种线性表顺序表快得多，但是查找一个节点或者访问特定编号的节点则需要O(n)的时间，而线性表和顺序表相应的时间复杂度分别是O(logn)和O(1)

链表的结构也十分多，常见的有四种形式：

* 单链表：除了头节点和尾节点，其他节点只包含一个后继指针
* 循环链表：跟单链表唯一的区别就在于它的尾结点又指回了链表的头结点，首尾相连，形成了一个环
* 双向链表：每个结点具有两个方向指针，后继指针(next)指向后面的结点，前驱指针(prev)指向前面的结点，其中节点的前驱指针和尾结点的后继指针均指向空地址NULL
* 双向循环链表：跟双向链表基本一致，不过头节点前驱指针指向尾迹诶单和尾节点的后继指针指向头节点

## 二、操作

关于链表的操作可以主要分成如下：

* 遍历
* 插入
* 删除

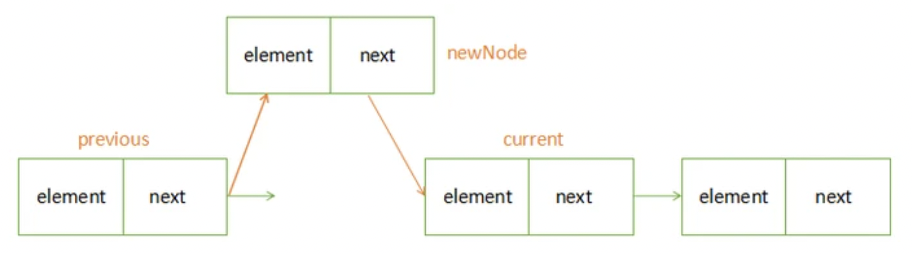
### 遍历

遍历很好理解，就是根据next指针遍历下去，直到为null，如下：

let current = head  
while(current){  
 console.log(current.val)  
 current = current.next  
}

### 插入

向链表中间插入一个元素，可以如下图所示：



可以看到，插入节点可以分成如下步骤：

* 存储插入位置的前一个节点
* 存储插入位置的后一个节点
* 将插入位置的前一个节点的 next 指向插入节点
* 将插入节点的 next 指向前面存储的 next 节点

相关代码如下所示：

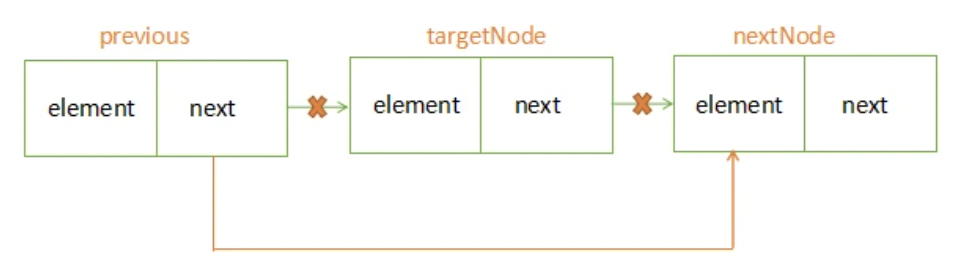
let current = head  
while (current < position){  
 pervious = current;  
 current = current.next;  
}  
pervious.next = node;  
node.next = current;

如果在头节点进行插入操作的时候，会实现previousNode节点为undefined，不适合上述方式

解放方式可以是在头节点前面添加一个虚拟头节点，保证插入行为一致

### 删除

向链表任意位置删除节点，如下图操作：



从上图可以看到删除节点的步骤为如下：

* 获取删除节点的前一个节点
* 获取删除节点的后一个节点
* 将前一个节点的 next 指向后一个节点
* 向删除节点的 next 指向为null

如果想要删除制定的节点，示意代码如下：

while (current != node){  
 pervious = current;  
 current = current.next;  
 nextNode = current.next;  
}  
pervious.next = nextNode

同样如何希望删除节点处理行为一致，可以在头节点前面添加一个虚拟头节点

## 三、应用场景

缓存是一种提高数据读取性能的技术，在硬件设计、软件开发中都有着非常广泛的应用，比如常见的CPU缓存、数据库缓存、浏览器缓存等等

当缓存空间被用满时，我们可能会使用LRU最近最好使用策略去清楚，而实现LRU算法的数据结构是链表，思路如下：

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，我们从链表头部开始顺序遍历链表

* 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的对应结点，并将其从原来的位置删除，并插入到链表头部
* 如果此数据没在缓存链表中
* 如果此时缓存未满，可直接在链表头部插入新节点存储此数据
* 如果此时缓存已满，则删除链表尾部节点，再在链表头部插入新节点

由于链表插入删除效率极高，达到O(1)。对于不需要搜索但变动频繁且无法预知数量上限的数据的情况的时候，都可以使用链表

## 参考文献

* https://zh.wikipedia.org/zh-hans/%E9%93%BE%E8%A1%A8
* https://mah93.github.io/2019/07/19/js-linked/