## 链表

* 链表是数据元素的线性集合，每个数据元素指向下一个，所以每个数据元素又可以成为“节点”
* 在链表（单向链表）中的每个节点，都包含了两个信息：*元素本身* 和 *指向下一个节点的引用*（next 指针）（双向链表还多了一个信息：*指向上一个节点的引用*（prev 指针）
* 单向链表中的每个节点只关心（知道）自己的下一个节点是什么，而不关心它的上一个或其它的节点

## js 创建单向链表节点

###### 1. 定义节点结构

const Node = function(element) {  
 this.element = element // 元素本身  
 this.next = null // 指向下一个节点  
}

###### 2. 创建节点

* 例如有以下四个节点

const Kitten = new Node('Kitten')  
const Puppy = new Node('Puppy')  
const Cat = new Node('Cat')  
const Dog = new Node('Dog')

###### 3. 添加指向下一个节点的引用

* 对以上四个节点建立链表关系

Kitten.next = Puppy  
Puppy.next = Cat  
Cat.next = Dog

## 创建单向链表结构

* 一个基本的链表结构，应该包括了以下属性和方法：
  + 属性：
    - 链表的长度 length
    - 链表的首个节点 head
    - 链表的各个节点 node
  + 方法：
    - 添加节点 add()
    - 删除节点 remove()
    - 查找给定元素在链表中的位置 indexOf()
    - 查找链表中给定位置的元素 elementAt()
    - 删除链表中指定位置的节点 removeAt()
    - 在指定位置给链表添加新节点 addAt()
* 实现
  1. 定义链表结构
* function LinkedList() {  
   let length = 0,  
   head = null  
    
   const Node = function(element) {  
   this.element = element  
   this.next = null  
   }  
    
   this.head = function() {  
   return head  
   }  
    
   this.size = function() {  
   return length  
   }  
  }
  1. 给链表添加节点 add()
* // 上述在 size() 后添加 add（）   
  this.add = function(element) {  
   const newNode = new Node(element)  
   // 如果链表中已经有节点，则要找到最后一个节点  
   if (length) {  
   let currentNode = head  
   // 借助 head 节点查找最后一个节点  
   while (currentNode.next)  
   currentNode = currentNode.next  
    
   currentNode.next = newNode  
   }   
   else // 否则将链表的头指向第一个节点  
   head = newNode  
    
   length++  
  }
  1. 移除链表中的某个节点 remove()
* this.remove = function (element) {  
   // 用两个指针分别记录上一个节点和当前节点  
   let prevNode,   
   currentNode = head  
    
   // 如果是第一个节点的元素相等  
   if (currentNode.element === element)  
   head = currentNode.next  
   else {   
   while(currentNode && currentNode.element !== element) {  
   prevNode = currentNode  
   currentNode = currentNode.next  
   }  
   if (currentNode)  
   prevNode.next = currentNode.next  
   }  
    
   length--  
  }
  1. 查找给定元素在链表中的位置 indexOf()
* this.indexOf = function(element) {  
   if (!length) return -1  
    
   let currentNode = head  
   for (let i = 0; i < length; i++) {  
   if (currentNode.element === element)   
   return i  
    
   currentNode = currentNode.next  
   }  
   return - 1  
  }
  1. 查找链表中给定位置的元素 elementAt()
* this.elementAt = function(index) {  
   if (!length) return undefined  
    
   let currentNode = head  
   for (let i = 0; i < length; i++) {  
   if (i == index)  
   return currentNode.element  
    
   currentNode = currentNode.next  
   }  
   return undefined  
  }
  1. 删除链表中指定位置的节点 removeAt()
* this.removeAt = function(index) {  
   if (index < 0 || index >= length || !length)  
   return null  
    
   let currentNode = head,  
   prevNode;  
    
   for (let i = 0; i < length; i++) {  
   if (i == index) {  
   if (i == 0)   
   head = currentNode.next  
   else   
   prevNode.next = currentNode.next  
   length--  
   return currentNode.element  
   }  
   prevNode = currentNode  
   currentNode = currentNode.next  
   }  
  }
  1. 在指定位置给链表添加新节点 addAt()
* this.addAt = function(index, element) {  
   if (index < 0 || index > length)   
   return false  
    
   const newNode = new Node(element);  
   let currNode = head,  
   prevNode;  
    
   for (let i = 0; i < length; i++) {  
   if (i == index) {  
   if (i == 0) { // 首位添加  
   newNode.next = currNode  
   head = newNode  
   } else {  
   prevNode.next = newNode  
   newNode.next = currNode  
   }  
   length++  
   break  
   }  
   else if (currNode.next == null && index == length) { // 末位添加  
   currNode.next = newNode  
   length++  
   break  
   }  
   prevNode = currNode  
   currNode = currNode.next  
   }  
   return true  
  }

### 创建双向链表

###### 1. 双向链表的节点结构

const Node = function(data, prev) {  
 this.data = data  
 this.prev = prev // 多了指向上一个节点的引用  
 this.next = null  
}

###### 2. 双向链表的属性和方法

* 一个基本的双向链表结构包含了以下的属性和方法：
  + 属性：
    - 指向第一个节点的 head
    - 指向最后一个节点的 tail
  + 方法：
    - 添加节点 add()
    - 删除节点 remove()
    - 反转双向链表 reverse()
* 实现：
  1. 添加节点 add()
* this.add = function(data) {  
   if (this.head == null) {  
   this.head = new Node(data, null)  
   this.tail = this.head  
   }  
   else {  
   const newNode = new Node(data, this.tail)  
   this.tail.next = newNode  
   this.tail = newNode  
   }  
  }
  1. 删除双向链表中的给定元素 remove()
* this.remove = function(data) {  
   if (this.head == null)   
   return null  
    
   let currNode = this.head  
   // 删除第一个节点  
   if (currNode.data === data) {  
   this.head = currNode.next  
   this.head.prev = null  
   } else {  
   while(currNode && currNode.data !== data) {  
   currNode = currNode.next  
   }  
   if (currNode) {  
   if (currNode.next) {  
   currNode.prev.next = currNode.next  
   this.tail = currNode.next  
   }  
   else { // 删除最后一个节点  
   this.tail = currNode.prev  
   currNode.prev.next = null  
   }  
   }  
   }  
  }
  1. 反转双向链表 reverse()
     + *这里换了一种思路来考虑问题：也就是不改变指针的指向，反转的实际其实也就是节点的元素的反转，所以直接将元素 reverse 就可以了*
* this.reverse = function() {  
   if (this.head == null)  
   return null  
    
   let first = this.head,  
   last = this.tail  
    
   // 用两个指针记录首尾的节点，并且两两交换元素  
   while (first !== last) {  
   [first.data, last.data] = [last.data, first.data];  
   first = first.next  
   last = last.prev  
   }  
  }