PDF下载 代码随想录 刷题 微信群 B站 代码随想录 知识星球 代码随想录

欢迎大家参与本项目，贡献其他语言版本的代码，拥抱开源，让更多学习算法的小伙伴们收益！

> 听说这道题目把链表常见的五个操作都覆盖了？

707.设计链表

https://leetcode-cn.com/problems/design-linked-list/

题意：

在链表类中实现这些功能：

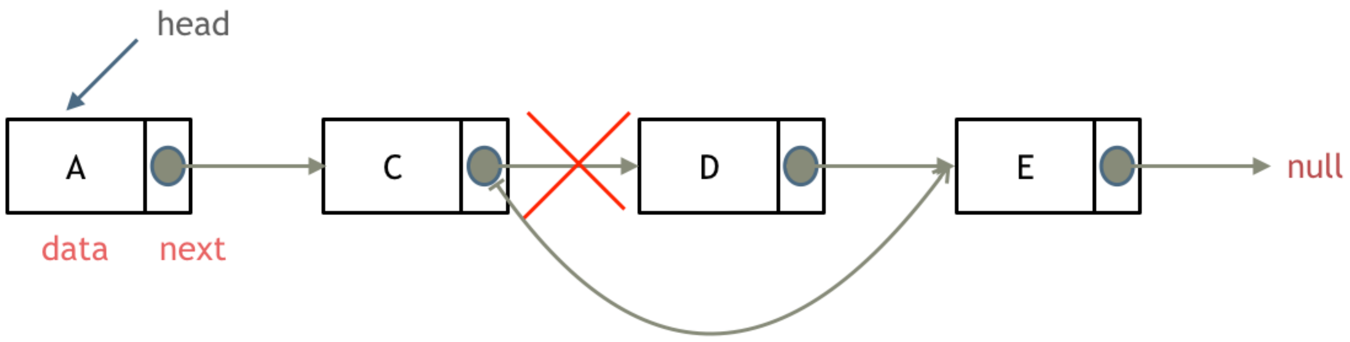
- get(index)：获取链表中第 index 个节点的值。如果索引无效，则返回-1。
- addAtHead(val)：在链表的第一个元素之前添加一个值为 val 的节点。插入后，新节点将成为链表的第一个节点。
- addAtTail(val)：将值为 val 的节点追加到链表的最后一个元素。
- addAtIndex(index,val)：在链表中的第 index 个节点之前添加值为 val 的节点。如果 index 等于链表的长度，则该节点将附加到链表的末尾。如果 index 大于链表长度，则不会插入节点。如果index小于0，则在头部插入节点。
- deleteAtIndex(index)：如果索引 index 有效，则删除链表中的第 index 个节点。

思路

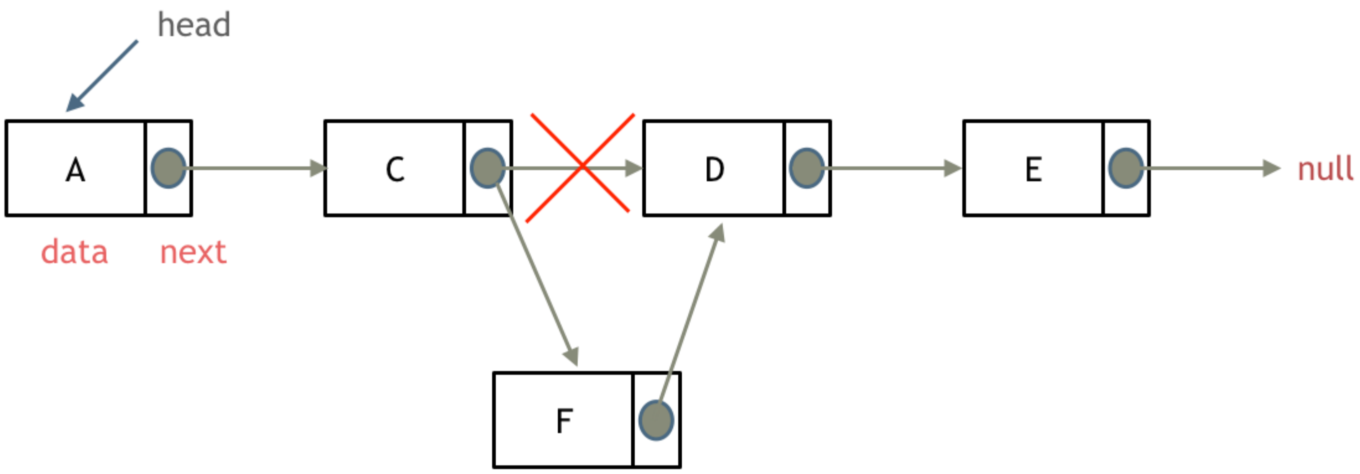
如果对链表的基础知识还不太懂，可以看这篇文章：关于链表，你该了解这些！

如果对链表的虚拟头结点不清楚，可以看这篇文章：链表：听说用虚拟头节点会方便很多？

删除链表节点：



添加链表节点：



这道题目设计链表的五个接口：

- 获取链表第index个节点的数值
- 在链表的最前面插入一个节点
- 在链表的最后面插入一个节点
- 在链表第index个节点前面插入一个节点
- 删除链表的第index个节点

可以说这五个接口，已经覆盖了链表的常见操作，是练习链表操作非常好的一道题目

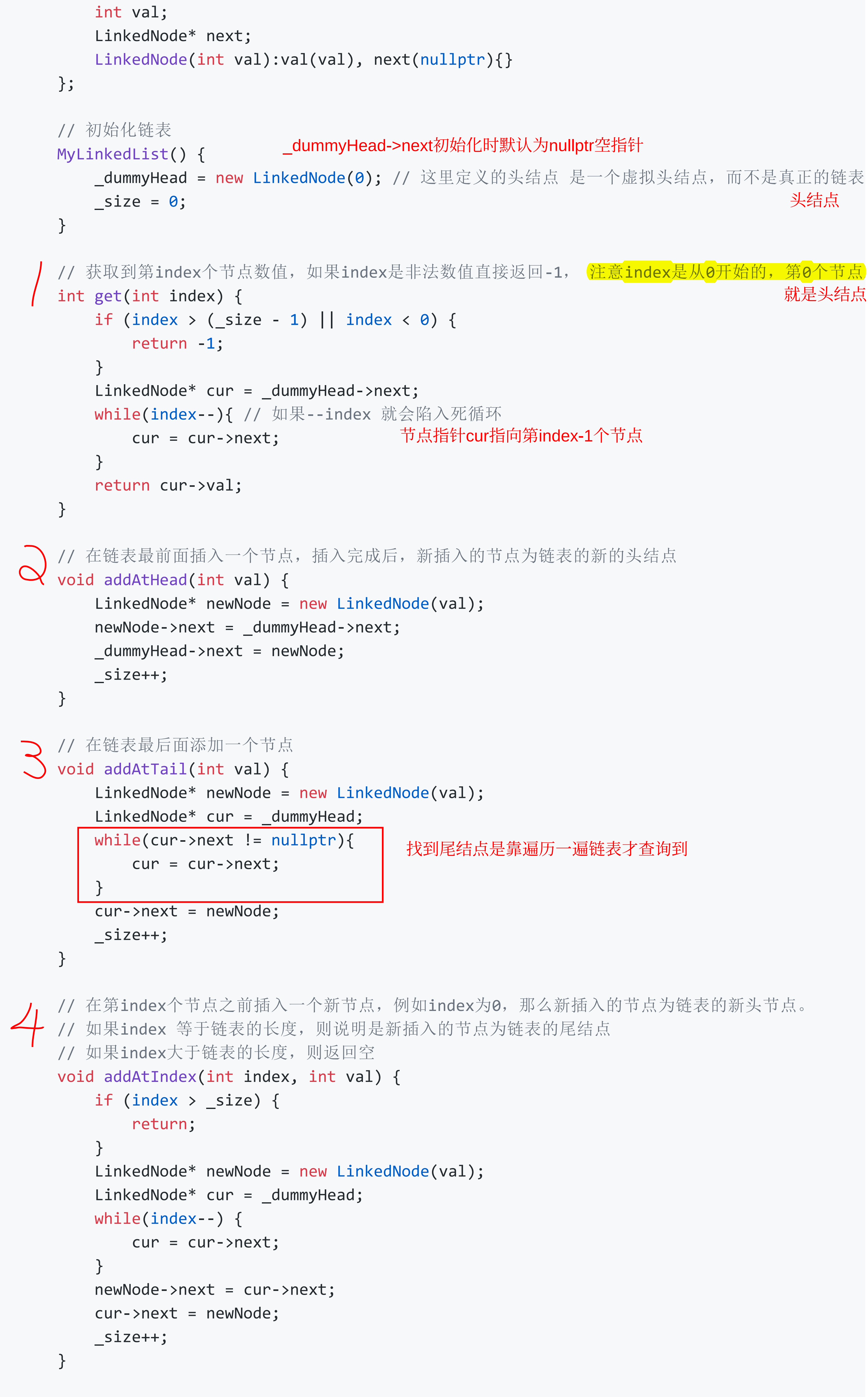
链表操作的两种方式：

1. 直接使用原来的链表来进行操作。
2. 设置一个虚拟头结点在进行操作。

下面采用的设置一个虚拟头结点（这样更方便一些，大家看代码就会感受出来）。

代码

```
class MyLinkedList {
public:
    // 定义链表节点结构体
    struct LinkedNode {
```



```
        int val;
        LinkedNode* next;
        LinkedNode(int val):val(val), next(nullptr){}
    };

    // 初始化链表
    MyLinkedList() {
```

_dummyHead->next初始化时默认为nullptr空指针

```
        _dummyHead = new LinkedNode(0); // 这里定义的头结点 是一个虚拟头结点，而不是真正的链表
        _size = 0;
    }
```

头结点

1

```
    // 获取到第index个节点数值，如果index是非法数值直接返回-1， 注意index是从0开始的，第0个节点
```

就是头结点

```
    int get(int index) {
        if (index > (_size - 1) || index < 0) {
            return -1;
        }
        LinkedNode* cur = _dummyHead->next;
        while(index--){ // 如果--index 就会陷入死循环
            cur = cur->next;
        }
        return cur->val;
    }
```

节点指针cur指向第index-1个节点

2

```
    // 在链表最前面插入一个节点，插入完成后，新插入的节点为链表的新的头结点
    void addAtHead(int val) {
        LinkedNode* newNode = new LinkedNode(val);
        newNode->next = _dummyHead->next;
        _dummyHead->next = newNode;
        _size++;
    }
```

3

```
    // 在链表最后面添加一个节点
    void addAtTail(int val) {
        LinkedNode* newNode = new LinkedNode(val);
        LinkedNode* cur = _dummyHead;
        while(cur->next != nullptr){
            cur = cur->next;
        }
        cur->next = newNode;
        _size++;
    }
```

找到尾结点是靠遍历一遍链表才查询到

4

```
    // 在第index个节点之前插入一个新节点，例如index为0，那么新插入的节点为链表的新头节点。
    // 如果index 等于链表的长度，则说明是新插入的节点为链表的尾结点
    // 如果index大于链表的长度，则返回空
    void addAtIndex(int index, int val) {
        if (index > _size) {
            return;
        }
        LinkedNode* newNode = new LinkedNode(val);
        LinkedNode* cur = _dummyHead;
        while(index--) {
            cur = cur->next;
        }
        newNode->next = cur->next;
        cur->next = newNode;
        _size++;
    }
    // 删除第index个节点，如果index 大于等于链表的长度，直接return，注意index是从0开始的
5   void deleteAtIndex(int index) {
        if (index >= _size || index < 0) {
            return;
        }
        LinkedNode* cur = _dummyHead;
        while(index--) {
            cur = cur ->next;
        }
        LinkedNode* tmp = cur->next;
        cur->next = cur->next->next;
        delete tmp;
        _size--;
    }

    // 打印链表
    void printLinkedList() {
        LinkedNode* cur = _dummyHead;
        while (cur->next != nullptr) {
            cout << cur->next->val << " ";
            cur = cur->next;
        }
        cout << endl;
    }
private:
    int _size;          节点个数
    LinkedNode* _dummyHead;  dummy:假的，仿真的

};
```

其他语言版本

Java:

```
//单链表
class ListNode {
int val;
ListNode next;
ListNode(){}
ListNode(int val) {
this.val=val;
}
}
class MyLinkedList {
    //size存储链表元素的个数
    int size;
    //虚拟头结点
    ListNode head;

    //初始化链表
    public MyLinkedList() {
        size = 0;
        head = new ListNode(0);
    }
    //获取第index个节点的数值
    public int get(int index) {
        //如果index非法，返回-1
        if (index < 0 || index >= size) {
            return -1;
        }
        ListNode currentNode = head;
        //包含一个虚拟头节点，所以查找第 index+1 个节点
        for (int i = 0; i <= index; i++) {
            currentNode = currentNode.next;
        }
        return currentNode.val;
    }

    //在链表最前面插入一个节点
    public void addAtHead(int val) {
        addAtIndex(0, val);
    }

    //在链表的最后插入一个节点
    public void addAtTail(int val) {
        addAtIndex(size, val);
    }

    // 在第 index 个节点之前插入一个新节点，例如index为0，那么新插入的节点为链表的新头节点。
    // 如果 index 等于链表的长度，则说明是新插入的节点为链表的尾结点
    // 如果 index 大于链表的长度，则返回空
    public void addAtIndex(int index, int val) {
        if (index > size) {
            return;
        }
        if (index < 0) {
            index = 0;
        }
        size++;
        //找到要插入节点的前驱
        ListNode pred = head;
        for (int i = 0; i < index; i++) {
            pred = pred.next;
        }
        ListNode toAdd = new ListNode(val);
        toAdd.next = pred.next;
        pred.next = toAdd;
    }

    //删除第index个节点
    public void deleteAtIndex(int index) {
        if (index < 0 || index >= size) {
            return;
        }
        size--;
        ListNode pred = head;
        for (int i = 0; i < index; i++) {
            pred = pred.next;
        }
        pred.next = pred.next.next;
    }
}
```

```
//双链表
class MyLinkedList {
    class ListNode {
        int val;
        ListNode next,prev;
        ListNode(int x) {val = x;}
    }

    int size;
    ListNode head,tail;//Sentinel node

    /** Initialize your data structure here. */
    public MyLinkedList() {
        size = 0;
        head = new ListNode(0);
        tail = new ListNode(0);
        head.next = tail;
        tail.prev = head;
    }

    /** Get the value of the index-th node in the linked list. If the index is invalid, ret
    public int get(int index) {
        if(index < 0 || index >= size){return -1;}
        ListNode cur = head;

        // 通过判断 index < (size - 1) / 2 来决定是从头结点还是尾节点遍历，提高效率
        if(index < (size - 1) / 2){
            for(int i = 0; i <= index; i++){
                cur = cur.next;
            }
        }else{
            cur = tail;
            for(int i = 0; i <= size - index - 1; i++){
                cur = cur.prev;
            }
        }
        return cur.val;
    }

    /** Add a node of value val before the first element of the linked list. After the inse
    public void addAtHead(int val) {
        ListNode cur = head;
        ListNode newNode = new ListNode(val);
        newNode.next = cur.next;
        cur.next.prev = newNode;
        cur.next = newNode;
        newNode.prev = cur;
        size++;
    }

    /** Append a node of value val to the last element of the linked list. */
    public void addAtTail(int val) {
        ListNode cur = tail;
        ListNode newNode = new ListNode(val);
        newNode.next = tail;
        newNode.prev = cur.prev;
        cur.prev.next = newNode;
        cur.prev = newNode;
        size++;
    }

    /** Add a node of value val before the index-th node in the linked list. If index equal
    public void addAtIndex(int index, int val) {
        if(index > size){return;}
        if(index < 0){index = 0;}
        ListNode cur = head;
        for(int i = 0; i < index; i++){
            cur = cur.next;
        }
        ListNode newNode = new ListNode(val);
        newNode.next = cur.next;
        cur.next.prev = newNode;
        newNode.prev = cur;
        cur.next = newNode;
        size++;
    }

    /** Delete the index-th node in the linked list, if the index is valid. */
    public void deleteAtIndex(int index) {
        if(index >= size || index < 0){return;}
        ListNode cur = head;
        for(int i = 0; i < index; i++){
            cur = cur.next;
        }
        cur.next.next.prev = cur;
        cur.next = cur.next.next;
        size--;
    }
}

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * MyLinkedList obj = new MyLinkedList();
 * int param_1 = obj.get(index);
 * obj.addAtHead(val);
 * obj.addAtTail(val);
 * obj.addAtIndex(index,val);
 * obj.deleteAtIndex(index);
 */
```

Python：

```
# 单链表
class Node:

    def __init__(self, val):
        self.val = val
        self.next = None


class MyLinkedList:

    def __init__(self):
        self._head = Node(0)  # 虚拟头部节点
        self._count = 0  # 添加的节点数
    def get(self, index: int) -> int:
        """
        Get the value of the index-th node in the linked list. If the index is invalid, ret
        """
        if 0 <= index < self._count:
            node = self._head
            for _ in range(index + 1):
                node = node.next
            return node.val
        else:
            return -1

    def addAtHead(self, val: int) -> None:
        """
        Add a node of value val before the first element of the linked list. After the inse
        """
        self.addAtIndex(0, val)

    def addAtTail(self, val: int) -> None:
        """
        Append a node of value val to the last element of the linked list.
        """
        self.addAtIndex(self._count, val)

    def addAtIndex(self, index: int, val: int) -> None:
        """
        Add a node of value val before the index-th node in the linked list. If index equal
        """
        if index < 0:
            index = 0
        elif index > self._count:
            return

        # 计数累加
        self._count += 1

        add_node = Node(val)
        prev_node, current_node = None, self._head
        for _ in range(index + 1):
            prev_node, current_node = current_node, current_node.next
        else:
            prev_node.next, add_node.next = add_node, current_node

    def deleteAtIndex(self, index: int) -> None:
        """
        Delete the index-th node in the linked list, if the index is valid.
        """
        if 0 <= index < self._count:
            # 计数-1
            self._count -= 1
            prev_node, current_node = None, self._head
            for _ in range(index + 1):
                prev_node, current_node = current_node, current_node.next
            else:
                prev_node.next, current_node.next = current_node.next, None


# 双链表
```

```python
# 相对于单链表，Node新增了prev属性
class Node:

    def __init__(self, val):
        self.val = val
        self.prev = None
        self.next = None


class MyLinkedList:

    def __init__(self):
        self._head, self._tail = Node(0), Node(0)  # 虚拟节点
        self._head.next, self._tail.prev = self._tail, self._head
        self._count = 0  # 添加的节点数

    def _get_node(self, index: int) -> Node:
        # 当index小于_count//2时，使用_head查找更快，反之_tail更快
        if index >= self._count // 2:
            # 使用prev往前找
            node = self._tail
            for _ in range(self._count - index):
                node = node.prev
        else:
            # 使用next往后找
            node = self._head
            for _ in range(index + 1):
                node = node.next
        return node

    def get(self, index: int) -> int:
        """
        Get the value of the index-th node in the linked list. If the index is invalid, ret
        """
        if 0 <= index < self._count:
            node = self._get_node(index)
            return node.val
        else:
            return -1

    def addAtHead(self, val: int) -> None:
        """
        Add a node of value val before the first element of the linked list. After the inse
        """
        self._update(self._head, self._head.next, val)

    def addAtTail(self, val: int) -> None:
        """
        Append a node of value val to the last element of the linked list.
        """
        self._update(self._tail.prev, self._tail, val)

    def addAtIndex(self, index: int, val: int) -> None:
        """
        Add a node of value val before the index-th node in the linked list. If index equal
        """
        if index < 0:
            index = 0
        elif index > self._count:
            return
        node = self._get_node(index)
        self._update(node.prev, node, val)

    def _update(self, prev: Node, next: Node, val: int) -> None:
        """
            更新节点
            :param prev: 相对于更新的前一个节点
            :param next: 相对于更新的后一个节点
            :param val:  要添加的节点值
        """
        # 计数累加
        self._count += 1
        node = Node(val)
        prev.next, next.prev = node, node
        node.prev, node.next = prev, next

    def deleteAtIndex(self, index: int) -> None:
        """
        Delete the index-th node in the linked list, if the index is valid.
        """
        if 0 <= index < self._count:
            node = self._get_node(index)
            # 计数-1
            self._count -= 1
            node.prev.next, node.next.prev = node.next, node.prev
```

Go:

```
//循环双链表
type MyLinkedList struct {
        dummy *Node
}

type Node struct {
        Val  int
        Next *Node
        Pre  *Node
}

//仅保存哑节点，pre-> rear, next-> head
/** Initialize your data structure here. */
func Constructor() MyLinkedList {
        rear := &Node{
                Val:  -1,
                Next: nil,
                Pre:  nil,
        }
        rear.Next = rear
        rear.Pre = rear
        return MyLinkedList{rear}
}

/** Get the value of the index-th node in the linked list. If the index is invalid, return
func (this *MyLinkedList) Get(index int) int {
        head := this.dummy.Next
        //head == this, 遍历完全
        for head != this.dummy && index > 0 {
                index--
                head = head.Next
        }
        //否则, head == this, 索引无效
        if 0 != index {
                return -1
        }
        return head.Val
}

/** Add a node of value val before the first element of the linked list. After the insertio
func (this *MyLinkedList) AddAtHead(val int) {
        dummy := this.dummy
        node := &Node{
                Val: val,
                //head.Next指向原头节点
                Next: dummy.Next,
                //head.Pre 指向哑节点
                Pre: dummy,
        }

        //更新原头节点
        dummy.Next.Pre = node
        //更新哑节点
        dummy.Next = node
        //以上两步不能反
}

/** Append a node of value val to the last element of the linked list. */
func (this *MyLinkedList) AddAtTail(val int) {
        dummy := this.dummy
        rear := &Node{
                Val: val,
                //rear.Next = dummy(哑节点)
                Next: dummy,
                //rear.Pre = ori_rear
                Pre: dummy.Pre,
        }

        //ori_rear.Next = rear
        dummy.Pre.Next = rear
        //update dummy
        dummy.Pre = rear
        //以上两步不能反
}

/** Add a node of value val before the index-th node in the linked list. If index equals to
func (this *MyLinkedList) AddAtIndex(index int, val int) {
        head := this.dummy.Next
        //head = MyLinkedList[index]
        for head != this.dummy && index > 0 {
                head = head.Next
                index--
        }
        node := &Node{
                Val: val,
                //node.Next = MyLinkedList[index]
                Next: head,
                //node.Pre = MyLinkedList[index-1]
                Pre: head.Pre,
        }
        //MyLinkedList[index-1].Next = node
        head.Pre.Next = node
        //MyLinkedList[index].Pre = node
        head.Pre = node
        //以上两步不能反
}

/** Delete the index-th node in the linked list, if the index is valid. */
func (this *MyLinkedList) DeleteAtIndex(index int) {
        //链表为空
        if this.dummy.Next == this.dummy {
                return
        }
        head := this.dummy.Next
        //head = MyLinkedList[index]
        for head.Next != this.dummy && index > 0 {
                head = head.Next
                index--
        }
        //验证index有效
        if index == 0 {
                //MyLinkedList[index].Pre = index[index-2]
                head.Next.Pre = head.Pre
                //MyLinedList[index-2].Next = index[index]
                head.Pre.Next = head.Next
                //以上两步顺序无所谓
        }
}
```

javaScript:

```
class LinkNode {
    constructor(val, next) {
        this.val = val;
        this.next = next;
    }
}

/**
 * Initialize your data structure here.
 * 单链表 储存头尾节点 和 节点数量
 */
var MyLinkedList = function() {
    this._size = 0;
    this._tail = null;
    this._head = null;
};

/**
 * Get the value of the index-th node in the linked list. If the index is invalid, return -
 * @param {number} index
 * @return {number}
 */
MyLinkedList.prototype.getNode = function(index) {
    if(index < 0 || index >= this._size) return null;
    // 创建虚拟头节点
    let cur = new LinkNode(0, this._head);
    // 0 -> head
    while(index-- >= 0) {
        cur = cur.next;
    }
    return cur;
};
MyLinkedList.prototype.get = function(index) {
    if(index < 0 || index >= this._size) return -1;
    // 获取当前节点
    return this.getNode(index).val;
};

/**
 * Add a node of value val before the first element of the linked list. After the insertion
 * @param {number} val
 * @return {void}
 */
MyLinkedList.prototype.addAtHead = function(val) {
    const node = new LinkNode(val, this._head);
    this._head = node;
    this._size++;
    if(!this._tail) {
        this._tail = node;
    }
};

/**
 * Append a node of value val to the last element of the linked list.
 * @param {number} val
 * @return {void}
 */
MyLinkedList.prototype.addAtTail = function(val) {
    const node = new LinkNode(val, null);
    this._size++;
    if(this._tail) {
        this._tail.next = node;
        this._tail = node;
        return;
    }
    this._tail = node;
    this._head = node;
};

/**
 * Add a node of value val before the index-th node in the linked list. If index equals to
 * @param {number} index
 * @param {number} val
 * @return {void}
 */
MyLinkedList.prototype.addAtIndex = function(index, val) {
    if(index > this._size) return;
    if(index <= 0) {
        this.addAtHead(val);
        return;
    }
    if(index === this._size) {
        this.addAtTail(val);
        return;
    }
    // 获取目标节点的上一个的节点
    const node = this.getNode(index - 1);
    node.next = new LinkNode(val, node.next);
    this._size++;
};

/**
 * Delete the index-th node in the linked list, if the index is valid.
 * @param {number} index
 * @return {void}
 */
MyLinkedList.prototype.deleteAtIndex = function(index) {
    if(index < 0 || index >= this._size) return;
    if(index === 0) {
        this._head = this._head.next;
        this._size--;
        return;
    }
    // 获取目标节点的上一个的节点
    const node = this.getNode(index - 1);
    node.next = node.next.next;
    // 处理尾节点
    if(index === this._size - 1) {
        this._tail = node;
    }
    this._size--;
};

// MyLinkedList.prototype.out = function() {
//     let cur = this._head;
//     const res = [];
//     while(cur) {
//         res.push(cur.val);
//         cur = cur.next;
//     }
// };
/**
 * Your MyLinkedList object will be instantiated and called as such:
 * var obj = new MyLinkedList()
 * var param_1 = obj.get(index)
 * obj.addAtHead(val)
 * obj.addAtTail(val)
 * obj.addAtIndex(index,val)
 * obj.deleteAtIndex(index)
 */
```

- 作者微信：程序员Carl
- B站视频：代码随想录
- 知识星球：代码随想录