10月学习笔记leetcode

Day 1

[02.01.01 链表基础知识（第 01 ~ 02 天）](https://datawhalechina.github.io/leetcode-notes/#/ch02/02.01/02.01.01-Linked-List-Basic)

**「求线性链表长度」** 的代码如下：

# 获取链表长度

def length(self):

count = 0

cur = self.head

while cur:

count += 1

cur = cur.next

return count

**「链表尾部插入元素」** 的代码如下：

# 链表尾部插入元素

def insertRear(self, val):

node = ListNode(val)

cur = self.head

while cur.next:

cur = cur.next

cur.next = node

while的地方有不同之处，上面是要遍历到最后一个节点后停止。

下面要遍历到最后一个节点时停止

**「链表尾部删除元素」** 的代码如下：

# 链表尾部删除元素

def removeRear(self):

# 如果链表只有一个元素或没有元素，则返回错误

if not self.head.next:

return 'Error'

cur = self.head

while cur.next.next:

cur = cur.next

cur.next = None

此处while要遍历到最后一个节点前的一个节点停止

**「链表中间插入元素」** 的代码（自已用for写了一遍）如下：

def insertInside(self, index, val):

cur = self.head

count = 0

for count=0 in index-1

count += 1

cur = cur.next

# 这一步是检查指针变量是否还在链表，如果不在链表，返回空，not 返回true，执行return

if not cur:

return 'Error'

node = ListNode(val)

node.next = cur.next

cur.next = node

**「链表中间删除元素」** 的代码如下：

# 链表中间删除元素

def removeInside(self, index):

count = 0

cur = self.head

while cur.next and count < index - 1:

count += 1

cur = cur.next

if not cur:

return 'Error'

# 这里为什么不能直接cur.next = cur.next.next？

del\_node = cur.next

cur.next = del\_node.next

# 尝试用for改写

def removeInside(self, index):

count = 0

cur = self.head

for count in index-1

count += 1

cur = cur.next

if not cur

return ‘Error’

cur.next = cur.next.next

Day2

## [1.](https://datawhalechina.github.io/leetcode-notes/#/ch02/02.01/02.01.02-Exercises?id=_1-0707-%e8%ae%be%e8%ae%a1%e9%93%be%e8%a1%a8)[0707. 设计链表](https://leetcode.cn/problems/design-linked-list/)

class MyLinkedList:

    def \_\_init\_\_(self) -> None:

        self.Head = None

    class MyListNode:

        def \_\_init\_\_(self, val = 0, next = None):

            self.val = val

            self.next = next

    # 获取链表长度

    def length(self):

        count = 0

        cur = self.head

        while cur:

            count += 1

            cur = cur.next

        return count

    def get(self,index: int) -> None: #这里加self的作用？让链表本身有定义  -> None的作用？

        count = 0

        cur = self.head

        while cur and count < index - 1:

            count += 1

            cur = cur.next

        if not cur:

            return '-1'

        return cur.next #错误应为：return current.val

    def addAtHead(self, val) -> None:

        node = self(val) #错误应为：node = self.MyListNode(val)

        # 错误答案 MyLinkedList.Head = node.next

        node.next = self.Head

    def addAtTail(self, val) -> None:

        node = self.MyListNode(val) #缺失判断：如果链表里没有元素，if not self.head: （下一行）self.head = new\_node

        while cur.next:

            cur = cur.next

        cur.next = node

    def addAtIndex(self, index: int, val):

        count = 0 #缺失判断：如果index不在链表里，小于零则排在表头，大于等于链表长度-1则排在表尾

        if index < 0:

            self.addAtHead(val)

        elif index == self.length(self):

            self.addAtTail(val)

        elif index < self.length(self) and index > 0:

            cur = self.Head

            while cur and count < index - 1:

                count += 1

                self.cur = self.cur.next

            node = self.MyListNode(val)

            # 错误答案MyLinkedList.cur = node

            # 错误答案node.next = MyLinkedList.cur.next

            node.next = cur.next

            cur.next = node

    def deleteAtIndex(self, index: int):

        count = 0

        cur = self.Head

        #错误答案while MyLinkedList.cur and count < index - 2:

        while cur and count < index - 1:

            count += 1

            cur = cur.next

        if not cur: #缺失：如果索引index无效？例如index小于0 应改为 if index < 0: \\return \\elif index > self.length(self) return \\elif index == 0 self.Head = self.Head.next \\else \\while count < index - 1 cur = cur.next

            return

        cur = cur.next.next

linkedList = MyLinkedList()

linkedList.addAtHead(1)

Day3

## [2.](https://datawhalechina.github.io/leetcode-notes/#/ch02/02.01/02.01.02-Exercises?id=_2-0206-%e5%8f%8d%e8%bd%ac%e9%93%be%e8%a1%a8)[0206. 反转链表](https://leetcode.cn/problems/reverse-linked-list/)

迭代法

class Solution:

    def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:

        prev = None

        current = head

        while current is not None:

            next\_node = current.next

            current.next = prev

            prev = current

            current = next\_node

        return prev

递归法

class Solution:

    def reverseList(self, head):

        if head is None or head.next is None:

            return head

        new\_head = self.reverseList(head.next)

        head.next.next = head

        head.next = None

        return new\_head