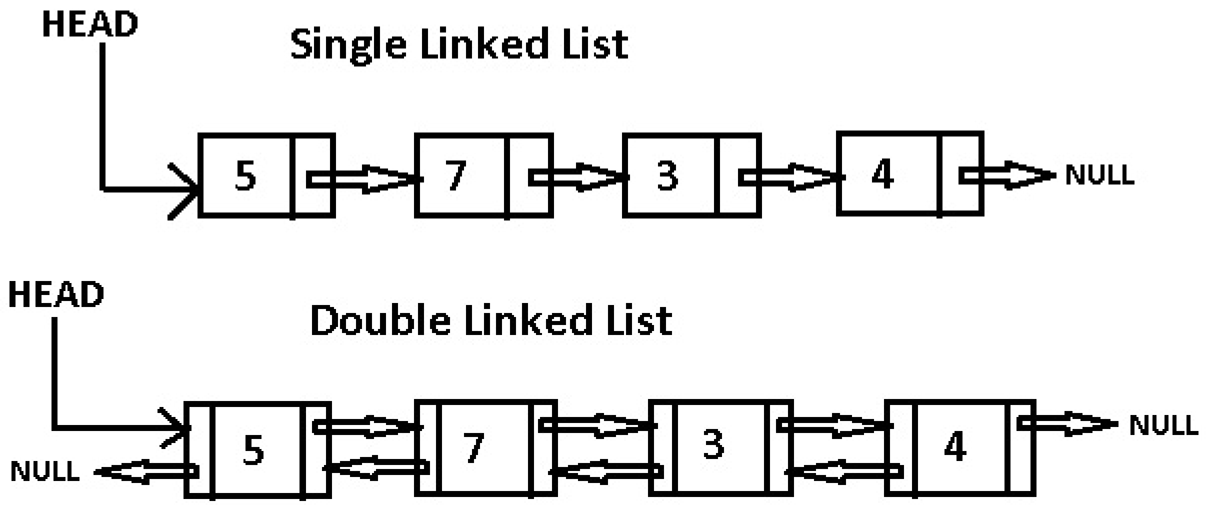
## 作业

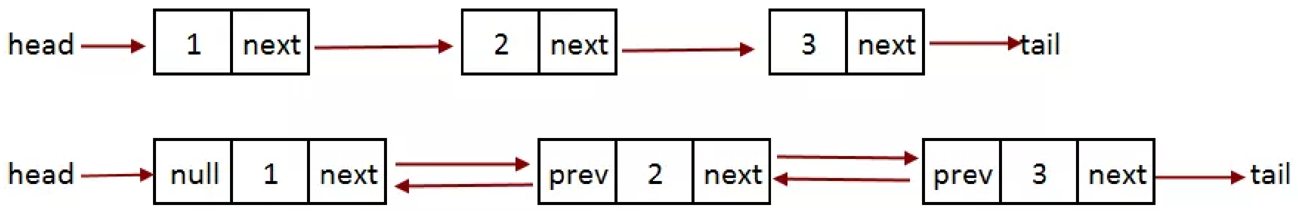
用面向对象实现LinkedList链表
单向链表实现append、iternodes方法
双向链表实现append、pop、insert、remove、iternodes方法



## 参考

实现LinkedList链表

链表有单向链表、双向链表



对于链表来说

- 每一个结点是一个独立的对象，结点自己知道内容是什么，下一跳是什么。
- 链表则是一个容器，它内部装着一个个结点对象。

所以，建议设计2个类，一个是结点Node类，一个是链表LinkedList类。

## 单向链表

```
class ListNode: # 结点保存内容和下一跳
    def __init__(self, item, next=None):
        self.item = item
        self.next = next

    def __repr__(self):
        return repr(self.item)


class LinkedList:
    def __init__(self):
        self.head = None
        self.tail = None # 单向链表为什么需要保存这个尾巴?

    def append(self, item):
        node = ListNode(item)
        if self.head is None:
            self.head = node # 设置开头结点，以后不变
        else:
            self.tail.next = node # 更新当前tail结点的next
        self.tail = node # 设置新tail
        return self # return self的好处?

    def iternodes(self):
        current = self.head
        while current:
            yield current
            current = current.next

ll = LinkedList()
ll.append(1).append(2).append(3)
ll.append('abc').append('def')

print(ll.head)
print(ll.tail)

print('-' * 30)
for item in ll.iternodes():
    print(item)
```

## 双向链表

双向链表实现append、pop、insert、remove、iternodes方法
实现单向链表没有实现的pop、remove、insert方法，补上。
双向链表的iternodes要实现两头迭代

```
class ListNode: # 结点保存内容和下一跳
    def __init__(self, item, next=None, prev=None):
        self.item = item
        self.next = next
        self.prev = prev  # 增加上一跳

    def __repr__(self):
        return "{} <== {} ==> {}".format(
            self.prev.item if self.prev else None,
            self.item,
            self.next.item if self.next else None
        )
```

```python
class LinkedList:
    def __init__(self):
        self.head = None
        self.tail = None

    def append(self, item):
        node = ListNode(item)
        if self.head is None:
            self.head = node # 设置开头结点，以后不变
        else:
            self.tail.next = node # 更新当前tail结点的next
            node.prev = self.tail
        self.tail = node # 设置新tail
        return self # return self的好处？

    def insert(self, index, item):
        if index < 0: # 不支持负索引
            raise IndexError('Not Negative index {}'.format(index))

        current = None
        for i, node in enumerate(self.iternodes()):
            if i == index: # 找到了
                current = node
                break
        else:
            self.append(item)
            return

        # break，找到了
        node = ListNode(item)
        prev = current.prev # node的前一个就是当前的前一个
        next = current # node的后一个就是当前

        # prev == None 或 current == self.head 或 i == 0 都相同
        if i == 0: # 如果是开头，head要更新，但prev是None
            self.head = node
        else: # 不是首元素，prev不是None
            prev.next = node
            node.prev = prev
        node.next = next
        next.prev = node

    def pop(self): # 尾部移除
        if self.tail is None: # 空链表
            raise Exception('Empty')

        node = self.tail
        item = node.item
        prev = node.prev
        if prev is None: # only one node
            self.head = None
            self.tail = None
        else:
            prev.next = None
            self.tail = prev
        return item

    def remove(self, index):
        if self.tail is None: # 空链表
            raise Exception('Empty')

        if index < 0: # 不支持负索引
            raise IndexError('Not Negative index {}'.format(index))

        current = None
        for i, node in enumerate(self.iternodes()):
            if i == index: # 找到了
                current = node
                break
        else: # Not Found
            raise IndexError('Wrong index {}'.format(index))

        prev = current.prev
        next = current.next

        # 4种情况
        if prev is None and next is None: #only one node
            self.head = None
            self.tail = None
        elif prev is None:  # 头部
            self.head = next
            next.prev = None
        elif next is None: # 尾部
            self.tail = prev
            prev.next = None
        else: # 在中间
            prev.next = next
            next.prev = prev

        del current

    def iternodes(self, reverse=False):
        current = self.tail if reverse else self.head
        while current:
            yield current
            current = current.next if not reverse else current.prev

ll = LinkedList()
ll.append('abc')
ll.append(1).append(2).append(3).append(4).append(5)
ll.append('def')

print(ll.head)
print(ll.tail)
```

```python
print('-' * 30)
for item in ll.iternodes():
    print(item)

print('=' * 30)

ll.remove(6)
ll.remove(5)
ll.remove(0)
ll.remove(1)

print('-' * 30)
for item in ll.iternodes():
    print(item) # 1,3,4

ll.insert(3, 5)
ll.insert(20, 'end')
ll.insert(1, 2)
ll.insert(0, 'start')

print('-' * 30)
for item in ll.iternodes(True):
    print(item)
```