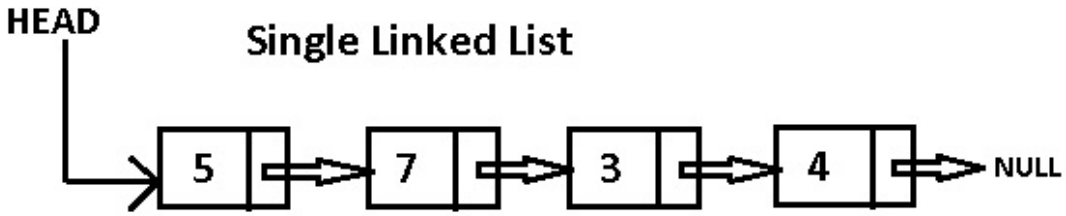
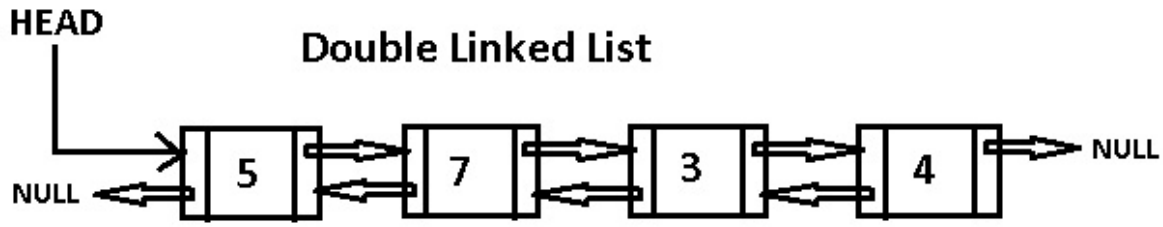
作业

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（尾部弹出）、insert、remove（使用索引移除）、iternodes方法
- 为链表提供__getitem__、__iter__、__setitem__等方法

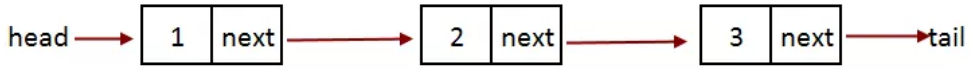


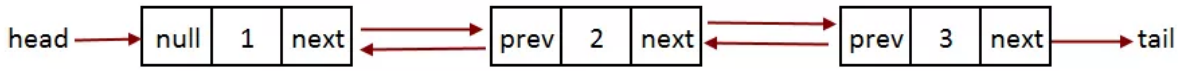


链表

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，结点对象有内容，还知道下一跳是什么。
- 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。

如同一个箱子是容器，里面放的小球就是一个个节点。
如同一串珠子，节点对象是一个个珠子，每一颗珠子**关联**其前后的珠子，所有珠子形成串珠的概念，相当于容器。一串珠子可以从固定一头提起，这是单向链表；如果这串珠子可以从两头中的任意一头提起，这就是双向链表。

单向链表

```python
# 解决ListNode注解延后评估，3.10之前使用注解延后评估功能必须有下一句
from __future__ import annotations



class ListNode:
    """结点保存内容和下一跳"""
```

```
    def __init__(self, item, next: ListNode = None): # 注解延后评估
        self.item = item
        self.next = next

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

class LinkedList:
    """容器，有头尾"""
    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: # 0个元素
            self.head = node
        else: # 多于1个元素
            self.tail.next = node
        self.tail = node # 设置新tail
        return self # return self的好处？

    def iternodes(self):
        current:ListNode = 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 i, item in enumerate(ll.iternodes()):
    print(i, 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

```
# 解决ListNode注解延后评估，3.10之前使用注解延后评估功能必须有下一句
from __future__ import annotations



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

```

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

class LinkedList:
    """容器，有头尾"""
    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: # 0个元素
            self.head = node
            self.tail = node
        else: # 多于1个元素
            self.tail.next = node
            node.prev = self.tail
        self.tail = node # 设置新tail
        return self # return self的好处?

    def pop(self):
        """尾部弹出"""
        if self.tail is None:
            raise Exception('Empty')

        node:ListNode = 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 insert(self, index, item):
        """指定索引插入"""
        if index < 0:
            raise IndexError('Not negative index:{}'.format(index))

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

        node = ListNode(item)
        prev = current.prev # 当前结点的前一个
        next = current.next # 当前结点的后一个
        # prev is None、current is self.head、i==0意思相同
```

```
        if index == 0:          # 开头插入
            self.head = node
        else:                   # 中间插入
            node.prev = prev
            prev.next = node
        node.next = current
        current.prev = node

    def remove(self, index):
        """指定index删除"""
        if self.tail is None:
            raise Exception('Empty')

        if index < 0:
            raise IndexError('Not negative index:{}'.format(index))

        current = None
        for i, node in enumerate(self.iternodes()):
            if index == i:
                current = node
                break
        else:# Not Found
            raise IndexError('Index out of range:{}'.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: # 多于一个结点，移除头部
            self.head = next
            next.prev = None
        elif next is None: # 多于一个结点，移除尾部
            self.tail = prev
            prev.next = None
        else:               # 在中间，头尾不变
            prev.next = next
            next.prev = prev

        del current

    def iternodes(self):
        current:ListNode = 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 i, item in enumerate(ll.iternodes()):
    print(i, item)

```

```
ll.insert(0, 'start')
ll.insert(20, 'end')

print('~' * 30)
for i, item in enumerate(ll.iternodes()):
    print(i, item)

ll.remove(5)
ll.remove(4)
ll.remove(0)
ll.pop()

print('=' * 30)
for i, item in enumerate(ll.iternodes()):
    print(i, item)
```

容器化

```
# 解决ListNode注解延后评估，3.10之前使用注解延后评估功能必须有下一句
from __future__ import annotations



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

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

    __str__ = __repr__

class LinkedList:
    """容器，有头尾"""
    def __init__(self):
        self.head = None
        self.tail = None # 单向链表为什么需要保存这个尾巴？
        self._size = 0

    def append(self, item):
        node = ListNode(item)
        if self.head is None: # 0个元素
            self.head = node
            self.tail = node
        else: # 多于1个元素
            self.tail.next = node
            node.prev = self.tail
        self.tail = node # 设置新tail
        self._size += 1
```

```
        return self # return self的好处?

    def pop(self):
        """尾部弹出"""
        if self.tail is None:
            raise Exception('Empty')

        node:ListNode = 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
        self._size -= 1
        return item

    def insert(self, index, item):
        """指定索引插入"""
        # if index < 0:
        #     raise IndexError('Not negative index:{}'.format(index))
        #
        # current = None
        # for i, node in enumerate(self.iternodes()):
        #     if index == i:
        #         current = node
        #         break
        # else:
        #     self.append(item) # 尾部追加
        #     return

        if index >= len(self):
            self.append(item)
            return
        if index < -len(self):
            index = 0
        current = self[index]

        node = ListNode(item)
        prev = current.prev # 当前结点的前一个
        next = current.next # 当前结点的后一个
        # prev is None、current is self.head、i==0意思相同
        if index == 0:        # 开头插入
            self.head = node
        else:                 # 中间插入
            node.prev = prev
            prev.next = node
        node.next = current
        current.prev = node
        self._size += 1

    def remove(self, index):
        """指定index删除"""
        if self.tail is None:
            raise Exception('Empty')

```

```
        # if index < 0:
        #     raise IndexError('Not negative index:{}'.format(index))
        #
        # current = None
        # for i, node in enumerate(self.iternodes()):
        #     if index == i:
        #         current = node
        #         break
        # else:# Not Found
        #     raise IndexError('Index out of range:{}'.format(index))

        current = self[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: # 多于一个结点，移除头部
            self.head = next
            next.prev = None
        elif next is None: # 多于一个结点，移除尾部
            self.tail = prev
            prev.next = None
        else:              # 在中间，头尾不变
            prev.next = next
            next.prev = prev

        del current
        self._size -= 1

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

    size = property(lambda self: self._size) # 只读属性

    # 容器化
    def __len__(self):
        return self._size

    # def __iter__(self):
    #     #yield from self.iternodes()
    #     return self.iternodes()
    __iter__ = iternodes

    def __reversed__(self):
        # reversed内建函数优先使用__reversed__
        # 如果不提供则使用序列协议，__len__和__getitem__方法
        return self.iternodes(True)

    def __getitem__(self, index):
        if index >= len(self) or index < -len(self): # 正负向超界
            raise IndexError('Index out of range:{}'.format(index))
        # if index >= 0:
```

```
        #     for i, node in enumerate(self.iternodes(False), 0):
        #         if index == i:
        #             return node # return node.item
        # if index < 0:
        #     for i, node in enumerate(self.iternodes(True), 1):
        #         if index == -i:
        #             return node
        start = 0 if index >= 0 else 1
        reverse = False if index >= 0 else True
        for i, node in enumerate(self.iternodes(reverse), start):
            if abs(index) == i:
                return node # return node.item

    def __setitem__(self, index, value):
        self[index].item = value

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

print('-' * 30)
for i, item in enumerate(ll):
    print(i, item)

ll.insert(0, 'start')
ll.insert(20, 'end')

print('~' * 30)
for item in ll:
    print(item)

ll.remove(5)
ll.remove(4)
ll.remove(0)
ll.pop()

print('=' * 30)
for item in ll:
    print(item)

ll[0] = 100
ll[-1] = 300

print('+' * 30)
print(*reversed(ll), sep='\n')
```