作业

将前面的链表，封装成容器

要求：提供__getitem__、__iter__、__setitem__方法

进阶题

实现类property装饰器，类名称为Property。
基本结构如下，是一个数据描述器

```
class Property: # 数据描述器
    def __init__(self):
        pass

    def __get__(self, instance, owner):
        pass

    def __set__(self, instance, value):
        pass


class A:
    def __init__(self, data):
        self._data = data

    @Property
    def data(self):
        return self._data

    @data.setter
    def data(self, value):
        self._data = value
```

链表

原双向链表源码

```
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
        )

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)

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)
```

代码实现如下

```
# 双向链表容器化

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

    __repr__ = __str__
```

```
class LinkedList:
    def __init__(self):
        self.head = None
        self.tail = None
        self._size = 0

    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

        self._size += 1
        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

        if index >= len(self):
            self.append(item)
            return

        if index < -len(self):
            index = 0

        current = self[index]

        # 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 prev == None: # 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

        self._size += 1

    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

        self._size -= 1
        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))

        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:  # 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
        self._size -= 1

    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

    size = property(lambda self:self._size)

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

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

    def __reversed__(self): # 解决reversed内建函数调用
        # 优先使用__reversed__
        # 如果没有提供，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('Wrong Index {}'.format(index))

        reverse = False if index >= 0 else True
        start = 0 if index >= 0 else 1

        for i, node in enumerate(self.iternodes(reverse), start):
            if i == abs(index):
                return node
                #return node.item

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


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)

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

```
    print(item)


print(len(ll))

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

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

print('=' * 30)
print(ll[-1], len(ll), ll[6])
print(ll[-2], ll[5])
print(ll[-7], ll[0])
ll[6] = 6
ll[-7] = 0

print('==============')

for x in reversed(ll):
    print(x)

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

属性装饰器的实现

```
class Property: # 数据描述器
    def __init__(self, fget, fset=None):
        self.fget = fget # 存getter函数
        self.fset = fset

    def __get__(self, instance, owner):
        if instance is not None:
            return self.fget(instance) # 调用getter, 传入实例
        return self
```

```
    def __set__(self, instance, value):
        if instance is not None:
            self.fset(instance, value)


    def setter(self, fn):
        self.fset = fn # 注意这是实例增加了属性，没有绑定效果
        return self


class A:
    def __init__(self, data):
        self._data = data

    @Property # data = Property(data) 描述器对象
    def data(self):
        return self._data

    @data.setter # data = data.setter(data)
    # 本质是 描述器实例.setter(data)，返回还是描述器对象，本质还是data = 描述器实例
    def data(self, value):
        self._data = value

a = A(1)
print(a.data)
a.data = 100
print(a.data)
print(a.__dict__)
```