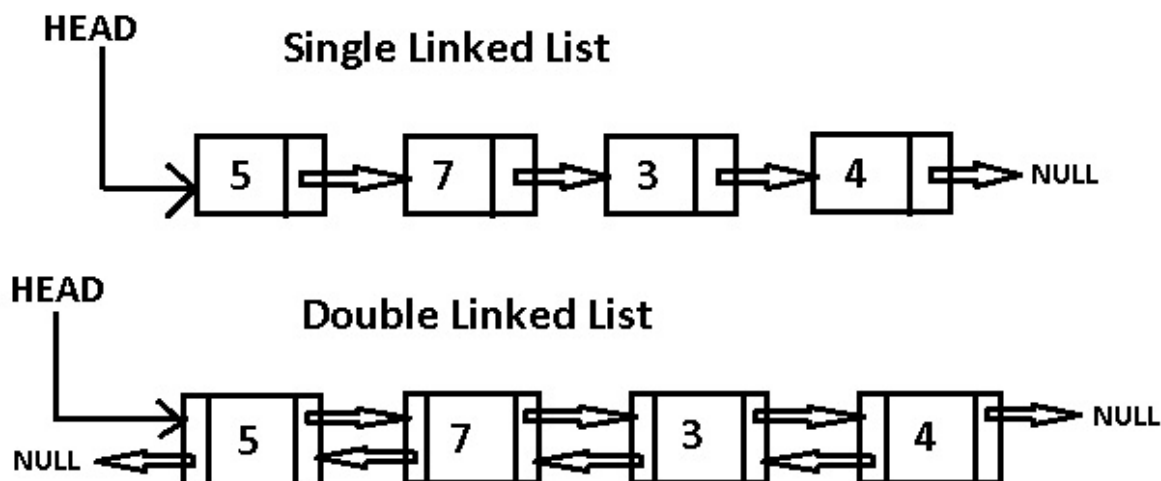


# 作业

用面向对象实现LinkedList链表

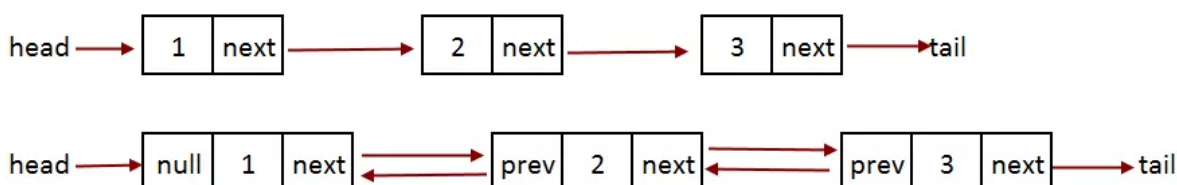
- 单向链表实现append、iternodes方法
- 双向链表实现append、pop（尾部弹出）、insert、remove（使用索引移除）、iternodes方法
- 为链表提供\_\_getitem\_\_、\_\_iter\_\_、\_\_setitem\_\_等方法



## 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

```
1 # 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
2 from __future__ import annotations
3
4
5
6 class ListNode:
7     """结点保存内容和下一跳"""
```

```

8     def __init__(self, item, next: ListNode = None): # 注解延后评估
9         self.item = item
10        self.next = next
11
12        def __repr__(self):
13            return str(self.item)
14
15    class LinkedList:
16        """容器，有头尾"""
17        def __init__(self):
18            self.head = None
19            self.tail = None # 单向链表为什么需要保存这个尾巴？
20
21        def append(self, item):
22            node = ListNode(item)
23            if self.head is None: # 0个元素
24                self.head = node
25            else: # 多于1个元素
26                self.tail.next = node
27            self.tail = node # 设置新tail
28            return self # return self的好处？
29
30        def iternodes(self):
31            current:ListNode = self.head
32            while current:
33                yield current
34                current = current.next
35
36    ll = LinkedList()
37    ll.append(1).append(2).append(3)
38    ll.append('abc').append('def')
39    print(ll.head)
40    print(ll.tail)
41
42    print('-' * 30)
43    for i, item in enumerate(ll.iternodes()):
44        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

```

1    # 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
2    from __future__ import annotations
3
4
5
6    class ListNode:
7        """结点保存内容和下一跳"""
8        def __init__(self, item, prev:ListNode=None, next:ListNode=None): # 注
9            解延后评估
10            self.item = item
11            self.prev = prev # 增加上一跳
12            self.next = next

```

```

13     def __repr__(self):
14         return "{} <-- {} --> {}".format(
15             self.prev.item if self.prev else None,
16             self.item,
17             self.next.item if self.next else None
18         )
19
20 class LinkedList:
21     """容器，有头尾"""
22     def __init__(self):
23         self.head = None
24         self.tail = None # 单向链表为什么需要保存这个尾巴？
25
26     def append(self, item):
27         node = ListNode(item)
28         if self.head is None: # 0个元素
29             self.head = node
30             self.tail = node
31         else: # 多于1个元素
32             self.tail.next = node
33             node.prev = self.tail
34             self.tail = node # 设置新tail
35             return self # return self的好处？
36
37     def pop(self):
38         """尾部弹出"""
39         if self.tail is None:
40             raise Exception('Empty')
41
42         node:ListNode = self.tail
43         item = node.item
44         prev = node.prev
45         if prev is None: # only one node
46             self.head = None
47             self.tail = None
48         else:
49             prev.next = None
50             self.tail = prev
51         return item
52
53     def insert(self, index, item):
54         """指定索引插入"""
55         if index < 0:
56             raise IndexError('Not negative index:{}'.format(index))
57
58         current = None
59         for i, node in enumerate(self.iternodes()):
60             if index == i:
61                 current = node
62                 break
63         else:
64             self.append(item) # 尾部追加
65             return
66
67         node = ListNode(item)
68         prev = current.prev # 当前结点的前一个
69         next = current.next # 当前结点的后一个
70         # prev is None、current is self.head、i==0意思相同

```

```

71         if index == 0:             # 开头插入
72             self.head = node
73         else:                       # 中间插入
74             node.prev = prev
75             prev.next = node
76             node.next = current
77             current.prev = node
78
79     def remove(self, index):
80         """指定index删除"""
81         if self.tail is None:
82             raise Exception('Empty')
83
84         if index < 0:
85             raise IndexError('Not negative index:{}'.format(index))
86
87         current = None
88         for i, node in enumerate(self.iternodes()):
89             if index == i:
90                 current = node
91                 break
92         else: # Not Found
93             raise IndexError('Index out of range:{}'.format(index))
94
95         prev = current.prev
96         next = current.next
97         # 4种情况
98         if prev is None and next is None: # only one node
99             self.head = None
100             self.tail = None
101         elif prev is None: # 多于一个结点, 移除头部
102             self.head = next
103             next.prev = None
104         elif next is None: # 多于一个结点, 移除尾部
105             self.tail = prev
106             prev.next = None
107         else:                # 在中间, 头尾不变
108             prev.next = next
109             next.prev = prev
110
111         del current
112
113     def iternodes(self):
114         current:ListNode = self.head
115         while current:
116             yield current
117             current = current.next
118
119 ll = LinkedList()
120 ll.append(1).append(2).append(3)
121 ll.append('abc').append('def')
122 print(ll.head)
123 print(ll.tail)
124
125 print('-' * 30)
126 for i, item in enumerate(ll.iternodes()):
127     print(i, item)
128

```

```

129 ll.insert(0, 'start')
130 ll.insert(20, 'end')
131
132 print('~' * 30)
133 for i, item in enumerate(ll.iternodes()):
134     print(i, item)
135
136 ll.remove(5)
137 ll.remove(4)
138 ll.remove(0)
139 ll.pop()
140
141 print('=' * 30)
142 for i, item in enumerate(ll.iternodes()):
143     print(i, item)

```

## 容器化

```

1 # 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
2 from __future__ import annotations
3
4
5
6 class ListNode:
7     """结点保存内容和下一跳"""
8     def __init__(self, item, prev:ListNode=None, next:ListNode=None): # 注
9         # 解延后评估
10         self.item = item
11         self.prev = prev # 增加上一跳
12         self.next = next
13
14     def __repr__(self):
15         return "{} <-- {} --> {}".format(
16             self.prev.item if self.prev else None,
17             self.item,
18             self.next.item if self.next else None
19         )
20
21     __str__ = __repr__
22
23 class LinkedList:
24     """容器, 有头尾"""
25     def __init__(self):
26         self.head = None
27         self.tail = None # 单向链表为什么需要保存这个尾巴?
28         self._size = 0
29
30     def append(self, item):
31         node = ListNode(item)
32         if self.head is None: # 0个元素
33             self.head = node
34             self.tail = node
35         else: # 多于1个元素
36             self.tail.next = node
37             node.prev = self.tail
38             self.tail = node # 设置新tail
39         self._size += 1

```

```

39         return self # return self的好处?
40
41     def pop(self):
42         """尾部弹出"""
43         if self.tail is None:
44             raise Exception('Empty')
45
46         node:ListNode = self.tail
47         item = node.item
48         prev = node.prev
49         if prev is None: # only one node
50             self.head = None
51             self.tail = None
52         else:
53             prev.next = None
54             self.tail = prev
55         self._size -= 1
56         return item
57
58     def insert(self, index, item):
59         """指定索引插入"""
60         # if index < 0:
61         #     raise IndexError('Not negative index:{}'.format(index))
62         #
63         # current = None
64         # for i, node in enumerate(self.iternodes()):
65         #     if index == i:
66         #         current = node
67         #         break
68         # else:
69         #     self.append(item) # 尾部追加
70         #     return
71
72         if index >= len(self):
73             self.append(item)
74             return
75         if index < -len(self):
76             index = 0
77         current = self[index]
78
79         node = ListNode(item)
80         prev = current.prev # 当前结点的前一个
81         next = current.next # 当前结点的后一个
82         # prev is None、current is self.head、i==0意思相同
83         if index == 0: # 开头插入
84             self.head = node
85         else: # 中间插入
86             node.prev = prev
87             prev.next = node
88             node.next = current
89             current.prev = node
90         self._size += 1
91
92     def remove(self, index):
93         """指定index删除"""
94         if self.tail is None:
95             raise Exception('Empty')
96

```

```

97         # if index < 0:
98         #     raise IndexError('Not negative index:{}'.format(index))
99         #
100         # current = None
101         # for i, node in enumerate(self.iternodes()):
102         #     if index == i:
103         #         current = node
104         #         break
105         # else: # Not Found
106         #     raise IndexError('Index out of range:{}'.format(index))
107
108         current = self[index]
109
110         prev = current.prev
111         next = current.next
112         # 4种情况
113         if prev is None and next is None: # only one node
114             self.head = None
115             self.tail = None
116         elif prev is None: # 多于一个结点, 移除头部
117             self.head = next
118             next.prev = None
119         elif next is None: # 多于一个结点, 移除尾部
120             self.tail = prev
121             prev.next = None
122         else: # 在中间, 头尾不变
123             prev.next = next
124             next.prev = prev
125
126         del current
127         self._size -= 1
128
129     def iternodes(self, reverse=False):
130         current:ListNode = self.head if not reverse else self.tail
131         while current:
132             yield current
133             current = current.next if not reverse else current.prev
134
135     size = property(lambda self: self._size) # 只读属性
136
137     # 容器化
138     def __len__(self):
139         return self._size
140
141     # def __iter__(self):
142     #     #yield from self.iternodes()
143     #     return self.iternodes()
144     __iter__ = iternodes
145
146     def __reversed__(self):
147         # reversed内建函数优先使用__reversed__
148         # 如果不提供则使用序列协议, __len__和__getitem__方法
149         return self.iternodes(True)
150
151     def __getitem__(self, index):
152         if index >= len(self) or index < -len(self): # 正负向超界
153             raise IndexError('Index out of range:{}'.format(index))
154         # if index >= 0:

```

```

155         #         for i, node in enumerate(self.iternodes(False), 0):
156         #             if index == i:
157         #                 return node # return node.item
158         # if index < 0:
159         #         for i, node in enumerate(self.iternodes(True), 1):
160         #             if index == -i:
161         #                 return node
162         start = 0 if index >= 0 else 1
163         reverse = False if index >= 0 else True
164         for i, node in enumerate(self.iternodes(reverse), start):
165             if abs(index) == i:
166                 return node # return node.item
167
168     def __setitem__(self, index, value):
169         self[index].item = value
170
171 ll = LinkedList()
172 ll.append(1).append(2).append(3)
173 ll.append('abc').append('def')
174 print(ll.head)
175 print(ll.tail)
176
177 print('-' * 30)
178 for i, item in enumerate(ll):
179     print(i, item)
180
181 ll.insert(0, 'start')
182 ll.insert(20, 'end')
183
184 print('~' * 30)
185 for item in ll:
186     print(item)
187
188 ll.remove(5)
189 ll.remove(4)
190 ll.remove(0)
191 ll.pop()
192
193 print('=' * 30)
194 for item in ll:
195     print(item)
196
197 ll[0] = 100
198 ll[-1] = 300
199
200 print('+ ' * 30)
201 print(*reversed(ll), sep='\n')

```