Python 数据结构之链表

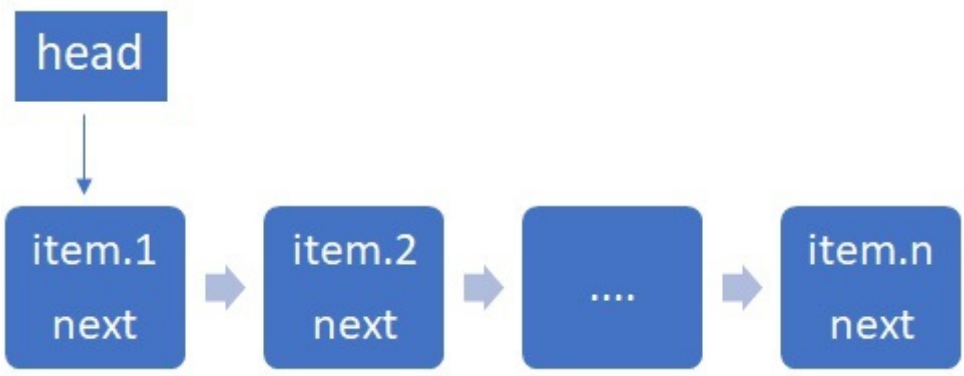
一、链表简介

链表是一种在存储单元上非连续、非顺序的存储结构。数据元素的逻辑顺序是通过链表中的指针链接次序实现。链表是由一序列的节点组成，节点可以在运行时动态生成。

每个节点包含两部分：数据域和指针域。数据域存储数据元素，指针域存储下一节点的指针。

二、单向链表

单向链表也是单链表，是链表中最简单的形式。它的每个节点包含两个域，一个信息域（元素域）和一个链接域。这个链接指向链表的下一个节点地址，而最后一个节点的链接域则指向一个空值。



head 保存首地址，item 存储数据，next 指向下一个节点地址。

链表失去了序列的随机读取有点，同时链表增加了指针域，空间开销也大，但它对存储空间的使用相对灵活。

例如：有一堆数据[1,2,3,5,6]，要做3和5之间插入4，如果用数组，需要将5之后的数据都往后退一位，然后再插入4，这样非常麻烦，但是如果用链表，直接可以在3和5之间插入4就行。

1、定义节点

节点的数据结构为数据元素（item）和指针（next）

```
class Node(object):
    '''单向链表的节点'''
    def __init__(self, item):
        # item 存放数据元素
        self.item = item
        # next 是下一个节点的地址
        self.next = None
```

2、定义单链表

链表需要具有首地址指针head

```
class SingleLinkList(object):
    '''单向链表'''
    def __init__(self):
        self._head = None
```

示例：创建链表

```
if __name__ == 'main':
    # 创建链表
    linkList = SingleLinkList()
    # 创建节点
    node1 = Node(1)
    node2 = Node(2)

    # 将节点添加到链表
    linkList._head = node1
    # 将第一个节点的next指针指向下一个节点
    node1.next = node2

    # 访问链表
    print(linkList._head.item)    # 访问第一个节点数据
    print(linkList._head.next.item) # 访问第二个节点数据
```

在链表中增加操作方法，以简化链表的操作。

- is_empty() 链表是否为空
- length() 链表长度
- items() 获取链表数据迭代器
- add(item) 链表头部添加元素
- append(item) 链表尾部添加元素
- insert(index, item) 指定位置添加元素
- remove(item) 删除节点
- find(item) 查找节点是否存在

```
class SingelLinkList(object):
    '''单链表'''
    def __init__(self, item):
        self._head = None

    def is_empty(self):
        '''判断链表是否为空'''
        return self._head is None

    def length(self):
        '''链表长度'''
        # 初始指针指向head
        cur = self._head
        count = 0
        # 指针指向None 表示到达尾部
        while cur is not None:
            count += 1
            # 指针后移
            cur = cur.next
        return count

    def items(self):
        '''遍历链表'''
        # 获取head 指针
        cur = self._head
        # 遍历循环
        while cur is not None:
```

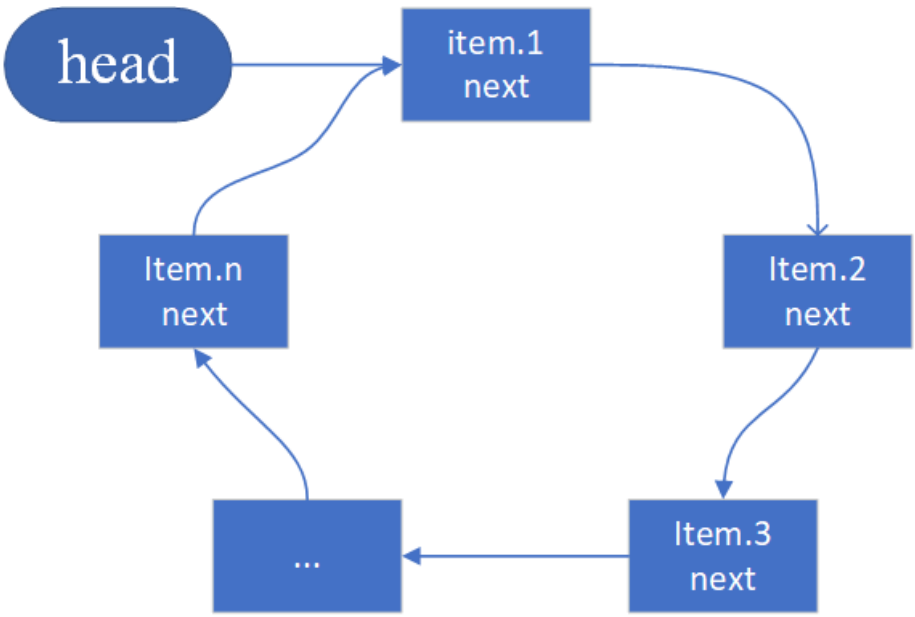
```
            # 返回生成器
            yield cur.item
            cur = cur.next

    def add(self, item):
        '''向链表头部添加元素'''
        node = Node(item)
        # 将新节点指针指向原头部节点
        node.next = self._head
        # 头部节点指针修改为新节点
        self._head = node

    def append(self, item):
        '''向链表尾部添加元素'''
        node = Node(item)
        if self.is_empty():
            # 空链表，_head 指向新节点
            self._head = node
        else:
            # 非空链表，则找到尾部， 将尾部next 节点指向新节点
            cur = self._head
            if cur.next is not None:
                cur = cur.next
            else:
                cur.next = node

    def insert(self, index, item):
        '''在单链表指定位置插入元素'''
        # 指定位置在第一个元素之前，在头部插入
        if index <= 0:
            self.add(item)
        # 指定位置超过尾部，在尾部插入
        elif index >= (self.length()-1):
            self.append(item)
        else:
            node = Node(item)
            cur = self._head
            # 循环到需要插入的位置
            for i in range(index-1):
                cur = cur.next
            node.next = cur.next
            cur.next = node

    def remove(self, item):
        '''删除节点'''
        cur = self.item
        pre = None
        while cur is not None:
            # 找到指定元素
            if cur.item == item:
                # 如果第一个元素就是删除的节点
                if not pre:
                    # 将头指针指向头节点的后一个节点
                    self._head = cur.next
                else:
                    # 将删除位置前的一个节点的next指向删除元素的后一个节点
                    pre.next = cur.next
                return True
```

```
            else:
                # 继续按链表后移节点
                pre = cur
                cur = cur.next
    def find(self, item):
        '''查找元素是否存在'''
        return item in self.items()
```

示例：操作链表

```
if __name__ == 'main':
    linkList = SingleLinkList()
    # 向链表尾部添加数据
    for i in range(5):
        linkList.append(i)
    # 向链表头部添加数据
    linkList.add(6)
    # 遍历链表数据
    for i in linkList.items():
        print(i,end='\t')
    # 链表指定位置插入元素
    linkList.insert(3,9)
    print('\n', linkList.items())
    # 删除链表数据集
    linkList.remove(0)
    # 查找链表数据
    print(linkList.find(4))
```

三、循环链表



单向循环链表为单向链表的变种，链表的最后一个next指向链表头，新增一个循环。

1、循环链表节点

```python
class Node(object):
    '''循环链表节点'''

    def __init__(self, item):
        # item 存放数据元素
        self.item = item
        # next 是下一个节点的地址
        self.next = None
```

2、定义循环单链表

```python
class SingleCycleLinkList(object):
    def __init__(self):
        self._head = None

    def is_empty(self):
        '''判断链表是否为空'''
        return self._head is None

    def length(self):
        '''获取链表长度'''
        # 链表为空
        if self.is_empty():
            return 0
        count = 0
        cur = self._head
        while cur.next != self._head:
            count += 1
            cur = cur.next
        return count

    def items(self):
        '''遍历链表'''
        # 链表为空
        if self.is_empty():
            return
        # 链表不为空
        cur = self._head
        while cur.next != self._head:
            yield cur.item
            cur = cur.next
        yield cur.item

    def add(self, item):
        '''链表头部添加节点'''
        node = Node(item)
        if self.is_empty():
            self._head = node
            node.next = self._head
        else:
            # 添加节点指向head
            node.next = self._head
            cur = self._head
            while cur.next != self._head:
                cur = cur.next
            cur.next = node
```

```
        # 修改 head 指向新节点
        self._head = node
    def append(self, item):
        # 向尾部添加元素
        node = Node(item)
        # 链表为空
        if self.is_empty():
            self._head = node
            self.next = self._head
        # 链表非空
        else:
            cur = self._head
            while cur.next != self._head:
                cur = cur.next
            cur.next = node
            # 新节点指针指向head
            node.next = self._head

    def insert(self, index, item):
        # 指定位置小于等于0，在链表头部添加
        if index <= 0:
            self.add(item)
        # 指定位置大于链表长度，在链表尾部添加
        elif index > self.length()-1:
            self.append(item)
        else:
            cur = self._head
            node = Node(item)
            for i in range(index - 1):
                cur = cur.next
            # 新节点指针指向旧节点
            node.next = cur.next
            # 旧节点指针指向新节点
            cur.next = node

    def remove(self, item):
        '''删除数据值为item的节点'''
        if self.is_empty():
            return
        cur = self._head
        pre = Node
        # 链表第一个元素为需要删除的元素
        if cur.item == item:
            # 链表不止一个元素
            if cur.next != self._head:
                while cur.next != self._head:
                    cur = cur.next
                # 尾节点指向头部节点的下一个节点
                cur.next = self._head.next
                # 删除头部节点，调整头部节点为头节点下一节点
                self._head = self._head.next
            else:
                # 只有一个元素
                self._head = None
        else:
            # 不是第一个元素
            pre = self._head
            # 遍历链表查找元素
```

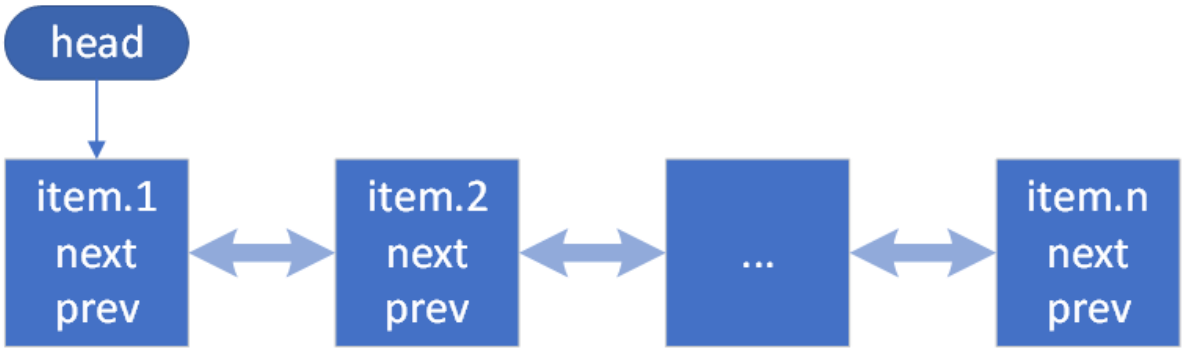
```
            while cur.next != self._head:
                if cur.item == item:
                    # 删除元素
                    pre.next = cur.next
                    return True
                else:
                    # 记录前一个指针
                    pre = cur
                    #调整指针位置
                    cur = cur.next
        # 删除元素在末尾
        if cur.item == item:
            pre.next = self._head
            return True

    def find(self, item):
        '''查找元素'''
        return item in self.items()
```

示例：操作循环单链表

```
def main():
    linkList = SingleCycleLinkList()
    print(linkList.is_empty())
    # 头部添加元素
    for i in range(5):
        linkList.add(i)
    print(list(linkList.items()))
    # 尾部添加元素
    for i in range(6):
            linkList.append(i)
    print(list(linkList.items()))
    # 添加元素
    linkList.insert(3, 45)
    print(list(linkList.items()))
    # 删除元素
    linkList.remove(5)
    print(list(linkList.items()))
    # 元素是否存在
    print(linkList.find(4))

if __name__ == '__main__':
    main()
```

四、双向链表

双向链表比单向链表更加复杂，它每个结点有两个链接：一个指向前一个节点，当此节点为第一个节点时，指向为空；而另一个链接指向下一个节点，当此节点为最后一个节点时，指向为空。



head 保存首地址，item 存储数据，next 指向下一节点地址，prev 指向上一节点地址。

1、定义双向链表节点

```
class Node(object):
    '''双向链表节点'''
    def __init__(self, item):
        # item 存放数据元素
        self.item = item
        # next 指向下一节点的地址
        self.next = None
        # prev 指向上一节点的地址
        self.prev = None
```

2、定义双向链表

```
class BilateraLinkList(object):
    '''双向链表'''
    def __init__(self):
        self._head = None

    def is_empty(self):
        '''判断链表是否为空'''
        return self._head is None

    def length(self):
        '''获取链表长度'''
        # 链表为空
        if self.is_empty():
            return 0
        # 链表非空
        count = 0
        cur = self._head
        while cur.next is not None:
            count += 1
            # 指针后移
            cur = cur.next
        return count

    def items(self):
        '''遍历链表'''
        # 获取头指针
        cur = self._head
        # 循环遍历链表
        while cur.next is not None:
```

```
            # 返回生成器
            yield cur.item
            # 指针下移
            cur = cur.next

    def add(self, item):
        '''向链表头部添加元素'''
        node = Node(item)
        # 链表为空
        if self.is_empty:
            # 头部节点指针修改为新节点
            self._head = node
        # 链表非空
        else:
            # 新节点指针指向原头部指针
            node.next = self._head
            # 原头部节点 prev 指向新节点
            self._head.prev = node
            # 修改 head 指向新节点
            self._head = node

    def append(self, item):
        '''链表尾部添加元素'''
        node = Node(item)
        # 链表为空
        if self.is_empty():
            self._head = node
        # 链表非空
        else:
            cur = self._head
            # 循环移动到尾部节点
            while cur.next is not None:
                cur = cur.next
            # 新节点上一级指针指向旧尾部
            node.prev = cur
            # 旧链表尾部指向新节点
            cur.next = node

    def insert(self, index, item):
        '''链表指定位置插入元素'''
        # 指定插入位置位于链表头部
        if index <= 0:
            self.add(item)
        # 指定插入位置位于链表尾部
        elif index > self.length()-1:
            self.append(item)
        # 指定位置位于链表之中
        else:
            cur = self._head
            node = Node(item)
            for i in range(index):
                cur = cur.next
            # 新节点的向下指针指向当前节点
            node.next = cur
            # 新节点的向上指针指向当前节点的上一节点
            node.prev = cur.prev
            # 当前节点的上一节点的向下指向新节点
            cur.prev.next = node
```

```
            # 当前节点的向上指针指向新节点
            cur.prev = node

    def remove(self, item):
        '''删除指定元素的第一个节点'''
        # 链表为空
        if self.is_empty():
            return
        # 链表非空
        cur = self._head
        if cur.item == item:
            # 删除节点为头节点
            if cur.next is None:
                # 链表只有一个元素
                self._head = None
                return True
            else:
                # head 指向下一节点
                self._head = cur.next
                # 下一节点的向上指针指向None
                cur.next.prev = None
                return True
        while cur.next is not None:
            # 删除节点非头点：遍历链表，直到找到需要删除的元素
            if cur.item == item:
                # 上一节点的next指向当前节点的下一节点
                cur.prev.next = cur.next
                # 下一节点的prev指向当前节点的上一节点
                cur.next.prev = cur.prev
                return True
            cur = cur.next
        # 删除节点为尾节点
        if cur.item == item:
            cur.prev.next = None
            return True

    def find(self, item):
        return item in self.items()
```

示例：操作双向链表

```
def main():
    linkList = BilateraLinkList()
    print(linkList.is_empty())
    # 头部添加元素
    for i in range(5):
        linkList.add(i)
    print(list(linkList.items()))
    # 尾部添加元素
    for i in range(6):
            linkList.append(i)
    print(list(linkList.items()))
    # 添加元素
    linkList.insert(3, 45)
    print(list(linkList.items()))
    # 删除元素
    linkList.remove(5)
```

```
    print(list(linkList.items()))
    # 元素是否存在
    print(linkList.find(4))

if __name__ == '__main__':
    main()
```