python数据结构-链表

2016-05-12

python学习笔记 链表

python学习笔记之链表

数据结构-链表

什么是链表，我对这个概念非常陌生。

链表是实现了数据之间保持逻辑顺序，但存储空间不必按顺序的方法。可以用一个图来表示这种链表的数据结构：

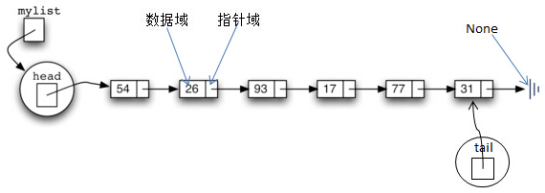


图1：链表

链表中的基本要素：

1. 结点(也可以叫节点或元素)，每一个结点有两个域，左边部份叫 值域，用于存放用户数据；右边叫 指针域，一般是存储着到下一个元素的指针
2. head结点，head是一个特殊的结节，head结点永远指向第一个结点
3. tail结点，tail结点也是一个特殊的结

点，tail结点永远指向最后一个节点

4. None，链表中最后一个结点指针域的指针指向None值，因也叫 接地点 ，所以有些资料上用电气上的接地符号代表None

链表的常用方法：

1. LinkedList() 创建空链表，不需要参数，返回值是空链表
2. is_empty() 测试链表是否为空，不需要参数，返回值是布尔值
3. append(data) 在尾部增加一个元素作为列表最后一个。参数是要追加的元素，无返回值
4. iter() 遍历链表，无参数，无返回值，此方法一般是一个生成器
5. insert(idx,value) 插入一个元素，参数为插入元素的索引和值
6. remove(idx)移除1个元素，参数为要移除的元素或索引，并修改链表
7. size() 返回链表的元素数，不需要参数，返回值是个整数
8. search(item) 查找链表某元素，参数为要查找的元素或索引，返回是布尔值

节点类

python用类来实现链表的数据结构，节点（Node）是实现链表的基本模块，每个节点至少包括两个重要部分。首先，包括节点自身的数据，称为“数据域”(也叫值

域)。其次，每个节点包括下一个节点的“引用”(也叫指针)

下边的代码用于实现一个Node类：

```
1 class Node:
2     def __init__(self, 
3         self.data = dat
4         self.next = Non
```

此节点类只有一个构建函数，接收一个数据参数，其中 `next` 表示指针域的指针，实例化后得到一个节点对象，如下：

```
1 node = Node(4)
```

此节点对象数据为 `4`，指针指向None。

这样一个节点对象可以用一个图例来更形象的说明，如下：

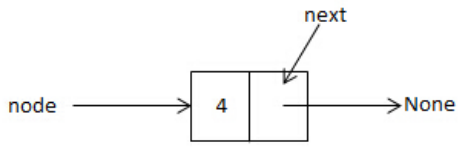


图2： 节点

链表类

先来看LinkedList类的构建函数：

```
class LinkedList:
    def __init__(self):
        self.head = Non
        self.tail = Non
```

此类实例后会生成一个链表对象，初始化了 head 和 tail 节点，且两节点都指向 None，实例化代码如下：

```
link_list = LinkedList(
```

也可以用图形象的表示这个链表对象，如下：

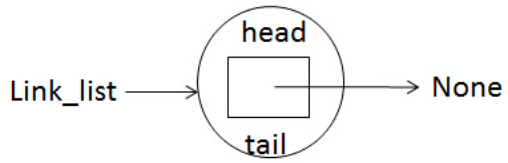


图3：空链表

is_empty方法实现

is_empty方法检查链表是否是一个空链表，这个方法只需要检查 head 节点是否指向 None 即可，代码如下：

```
def is_empty(self):
        return self.hea
```

如果是空列表返回 True，否则返回 False

append方法实现

append方法表示增加元素到链表，这和insert方法不同，前者使新增加的元素成为链表中第一个节点，而后者是根据索引值来判断插入到链表的哪个位置。代码如下：

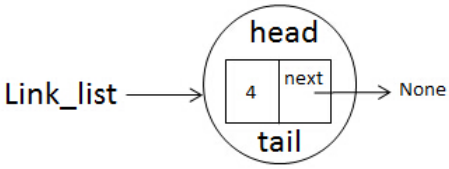
```
def append(self, data):
        node = Node(data
        if self.head is
            self.head =
            self.tail =
        else:
            self.tail.ne
            self.tail =
```

既然要新增加节点，首先把Node类实例化得到一个node对象。这里有两种情况需要考虑，一是链表是一个空链表时怎样append一个节点；二是当链表不是空链表时又怎样append一个节点?

当 `if self.head is None:` 为 `True` 时，把链表的 `head` 和 `tail` 都指向了 `node`，假如我们执行了

```
link_list(append(4))
```

此时的链表结构如下图：



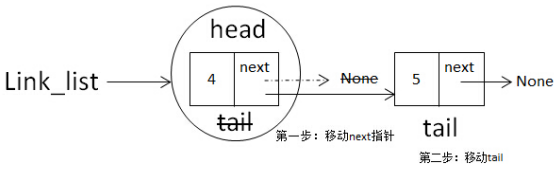
图

4：append-1

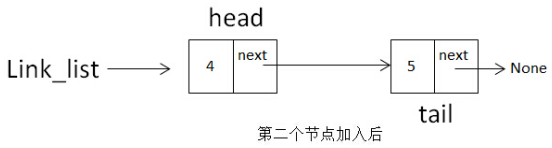
当 `if self.head is None:` 为 `False` 时,说明链表已经增加了一个节点了，再增加一个节点时 `head` 已经指向了第一个节点，所以不为 `None`，比如增加的第二个节点为：

```
1 link_list(append(5))
```

增加第二个节点的操作需要分两步完成，第一步： `self.tail.next = node`，即把上一个节点的 `next` 指针指向当前 `node`；第二步： `self.tail = node`，把 `tail` 移动到 `node`，如下图：



移动完成后就成这样了：



第二个节点加入后

当增加第三个、第四个等节点时，按照上边的操作依次类推。

iter方法实现

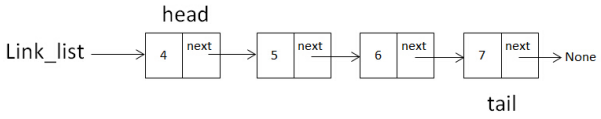
iter方法表示遍历链表。在遍历链表时也要首先考虑空链表的情况。遍历链表时从 `head` 开始，直到一个节点的 `next` 指向 `None` 结束，代码如下：

```
1 def iter(self):
2         if not self.hea
3             return
4         cur = self.head
5         yield cur.data
6         while cur.next:
7             cur = cur.n
8             yield cur.d
```

当是遍历一个空链表时， `if not self.head:` 为 `True` ，直接返回 `None` ；如果不是空链表就让一个局部变量 `cur` 指向 `head` ,并把 `head` 的 `data` 属性 `yield` 出来，再对 `cur` 的 `next` 指针指向的对象做 `while` 循环，直到 `next` 指向 `None` ，这样就遍历了链表。

insert方法实现

假如采取 append 方法又增加了两个节点，增加完成后如下图：

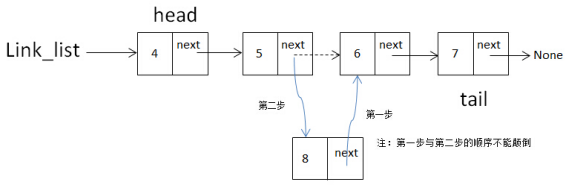


如果想在数据域为 6 的那节点处插入一个节点，需要做的操作有两步：

1. 把新节点的next指针指向数据域为 6 的这个节点，即为数据域为 5 节点的next指向指向的对象
2. 把数据域为 5 节点的next指针指向新加的节点

注： 这两个步骤不能颠倒，如果颠倒，数据域为 6 的节点会被丢失，数据域为 7 的节点不再是链表的节点。

示意图如下：

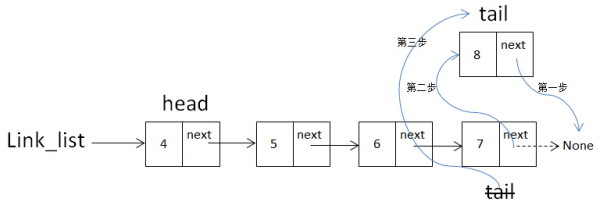


还要额外考虑两种情况：

1. 空链表时
2. 插入位置超出链表节点的长度时
3. 插入位置是链表的最后一个节点时，需要移动tail

当是在链表最后一个节点插入时，示意图

如下：



要在指定的索引位置插入一个节点，前提是需要找到这个位置，在链表中只有采用遍历的方式，具有O(n)的速度，最糟糕时会遍历链表的所有节点，而当找到插入点时，我们并不需要当前节点的信息，而是需要前一个节点的信息，所以代码中巧妙的使用了`while cur_idx < idx-1:`的方式，这样能使用`cur`这个变量能指向插入点上一个节点对象。

实现`insert`方法的代码如下：

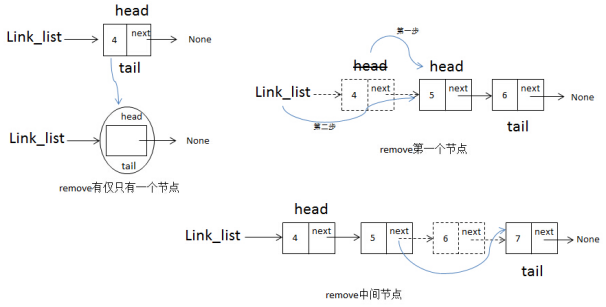
```
def insert(self, idx, 
        cur = self.hea
        cur_idx = 0
        if cur is None
            raise Excep
        while cur_idx <
            cur = cur.n
            if cur is N
                raise E
            cur_idx +=
        node = Node(va
        node.next = cu
        cur.next = node
        if node.next is
            self.tail =
```

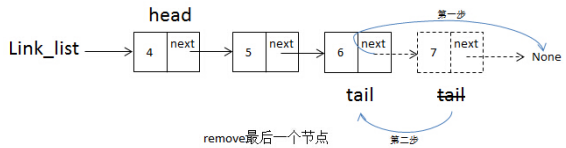
remove方法实现

remove方法接收一个idx参数，表示要删除节点的索引，此方法要考虑以下几种情况：

1. 空链表，直接抛出异常
2. 删除第一个节点时，移动head到删除节点的next指针指向的对象
3. 链表只有一个节点时，把head与tail都指向None即可
4. 删除最后一个节点时，需要移动tail到上一个节点
5. 遍历链表时要判断给定的索引是否大于链表的长度，如果大于则抛出异常信息

请看下边图例：





以下为remove函数的代码：

```
def remove(self, idx):
        cur = self.head
        cur_idx = 0
        if self.head is
            raise Except
        while cur_idx <
            cur = cur.ne
            if cur is No
                raise Ex
            cur_idx += 1
        if idx == 0:   #
            self.head =
            cur = cur.ne
            return
        if self.head is
            self.head =
            self.tail =
            return
        cur.next = cur.n
        if cur.next is N
            self.tail =
```

size函数实现

size函数不接收参数，返回链表中节点的个数，要获得链表的节点个数，必定会遍历链表，直到最后一个节点的`next`指针指向`None`时链表遍历完成，遍历时可以用一个累加器来计算节点的个数，如下代码：

```
def size(self):
        current = self.l
        count = 0
        if current is N
            return 'The
        while current i
            count += 1
            current = c
        return count
```

search函数实现

search函数接收一个item参数，表示查找节点中数据域的值。search函数遍历链表，每到一个节点把当前节点的 `data` 值与 `item` 作比较，最糟糕的情况下会全遍历链表。如果查找到有些数据则返回 `True`，否则返回 `False`，代码如下：

```
def search(self, item):
        current = self.l
        found = False
        while current i
            if current.
                found =
            else:
                current
        return found
```

Node类与LinkedList类完整代码

最后把 `Node类` 和 `LinkedList类` 的完整代码整理如下：

Node类：

```
class Node:
    def __init__(self, 
        self.data = dat
        self.next = Non
```

LinkedList类及调度代码：

```
class LinkedList:
    def __init__(self)
        self.head = No
        self.tail = No

    def is_empty(self)
        return self.he

    def append(self, d
        node = Node(da
        if self.head :
            self.head
            self.tail
        else:
            self.tail.
            self.tail

    def iter(self):
        if not self.he
            return
        cur = self.hea
        yield cur.data
        while cur.next
            cur = cur.
            yield cur.

    def insert(self, :
        cur = self.hea
        cur_idx = 0
        if cur is None
            raise Exce
        while cur_idx
            cur = cur.
            if cur is
                raise
            cur_idx +=
        node = Node(va
        node.next = cu
        cur.next = noc
        if node.next :
```

← python数据结构之栈 python的模块化 →