第 3 章

链表

3-1 链表数据形式与内存概念

3-2 链表的数据读取

3-3 新数据插入链表

3-4 删除链表的节点元素

3-5 循环链表 (circle linked list)

3-6 双向链表

3-7 数组与链表基本操作的时间复杂度比较

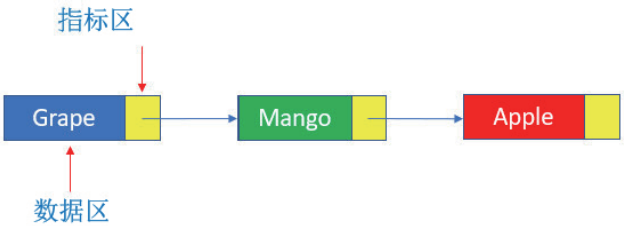
3-8 与链表有关的 Python 程序

3-9 习题

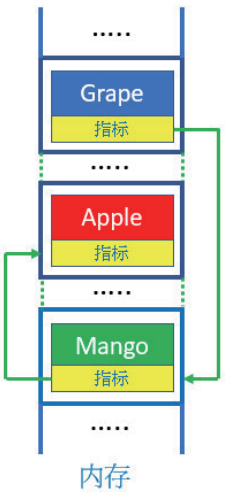
链表 (linked list) 表面上看是一串的数据，但是列表内的数据可能是散布在内存的各个地方。更明确地说，链表与数组的最大不同是，数组数据元素是放在连续的内存空间，链表数据元素是散布在内存的各个地方。

3-1 链表数据形式与内存概念

在链表中每个节点元素有 2 个区块，一个区块是数据区，主要是存放数据，另一个区块是指标区，主要是指向下一个节点元素。下列链表内有 3 个节点元素，元素内容分别是 Grape、Mango、Apple。



上述最后一个节点元素 (内容是 Apple) 的指标区没有指向任何位置，代表这是链表的最后一个节点。在链表中，因为节点元素不必放在连续内存空间，所以内存内实际的存储位置可能如下图所示：



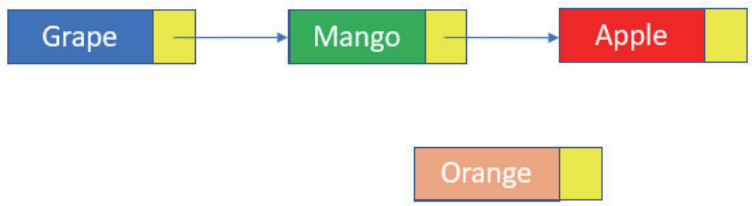
3-2 链表的数据读取

链表读取数据是使用顺序读取 (sequential access)，例如，要读取 Apple 数据，首先要从第一个节点 Grape 开始，然后经过 Mango 节点，最后连上 Apple 节点才可取得 Apple 数据。

由上图可以知道，要读取链表内容必须从头开始搜寻数据，所以整个执行的时间复杂度是 O(n)。

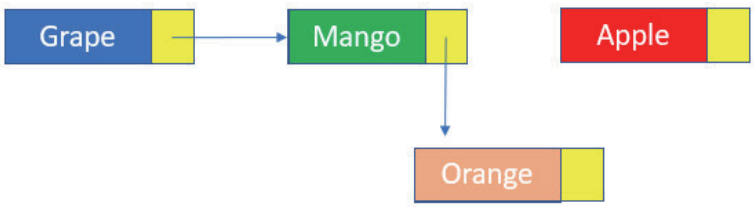
3-3 新数据插入链表

在链表中，如果要在任意位置新增节点元素，只要将前一个节点指标指向此新节点，然后将新节点指标指向下一个节点就可以了。例如，想要在链表内的 Mango 节点和 Apple 节点间增加 Orange，整个步骤如下：



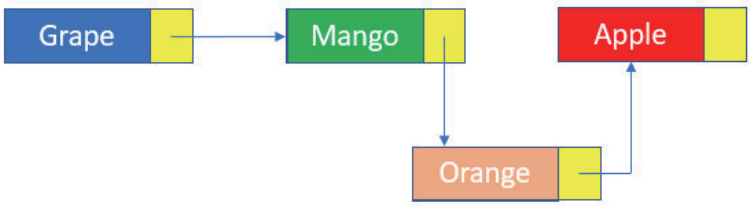
❑ 步骤 1

将 Mango 节点的指标指向 Orange 节点。



❑ 步骤 2

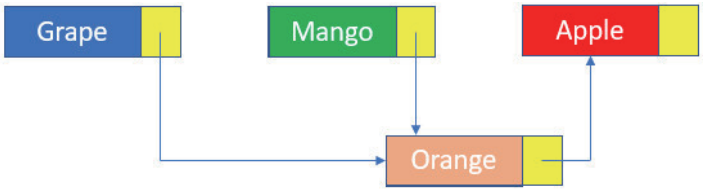
将 Orange 节点的指标指向 Apple 节点。



由于上述只更改两个指针就完成了数据插入，不需要遍历 n 个节点，所以运行时间复杂度是 O(1)。

3-4 删除链表的节点元素

链表中也可以删除某个节点元素，例如，想要删除 Mango 节点元素，只要将 Mango 前一个节点的指标从指向 Mango 改为指向 Mango 的下一个节点 Orange 即可。



虽然 Mango 节点仍存在于内存中，但此链表已经无法到达 Mango 节点，所以这个节点就算是删除了。

由于不需要遍历 n 个节点就可以删除节点元素，所以运行时间复杂度是 O(1)。

3-5 循环链表 (circle linked list)

在链表中有头尾概念，要找寻一个节点必须从头到尾搜寻，如果将一个链表在设计时将末端节点的指标指向第一个节点，这样就成了循环链表，它的特色是未来不管目前指标是指向哪一个节点，皆可以搜寻整个列表。

3-6 双向链表

截至目前为止，所有链表皆是单向搜寻，如果我们将每个节点多增加一个指标区，其中一个指标指向前面节点，另一个指标指向后面节点，这样就成了双向链表 (double linked list)，指标可以往前搜寻，也可以往后搜寻。

3-7 数组与链表基本操作的时间复杂度比较

下表是当数组与链表在相同操作环境下，执行读取、插入、删除时的运行时间复杂度比较。

时间复杂度	数组	链表
读取	O(1)	O(n)
插入	O(n)	O(1)
删除	O(n)	O(1)

由上述可知，2 个数据结构应用在不同的操作各有优缺点，未来所设计的程序应用何种算法存储数据，应由常用操作决定。

3-8 与链表有关的 Python 程序

这一节笔者将教导读者使用 Python 建立链表指标及遍历链表。

3-8-1 建立链表

想要建立链表，首先要建立此链表的节点，我们可以使用下列 Node 类别建立此节点。

```
class Node():
    ''' 节点 '''
    def __init__(self, data=None):
        self.data = data            # 数据
        self.next = None            # 指标
```

Node 类别有 2 个属性，其中 data 是存储节点数据，next 是存储指标，此指标未来可指向下一个节点，在尚未设定前我们可以使用 None。

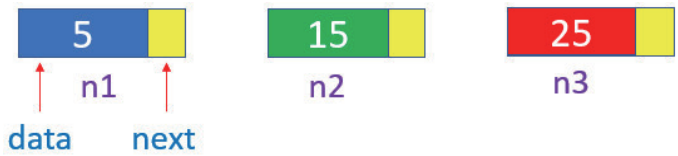
程序实例 ch3_1.py：建立一个含 3 个节点的链表，然后打印此链表。

```
# ch3_1.py
class Node():
    ''' 节点 '''
    def __init__(self, data=None):
        self.data = data            # 数据
        self.next = None            # 指标

n1 = Node(5)                        # 节点 1
n2 = Node(15)                       # 节点 2
n3 = Node(25)                       # 节点 3
n1.next = n2                        # 节点 1 指向节点 2
n2.next = n3                        # 节点 2 指向节点 3
ptr = n1                            # 建立指标节点
while ptr:
    print(ptr.data)                 # 打印节点
    ptr = ptr.next                  # 移动指标到下一个节点
```

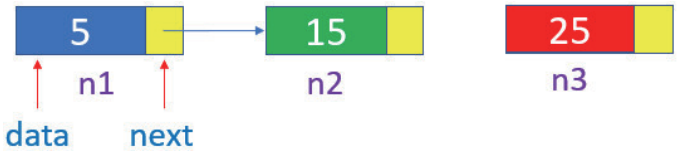
执行结果

```
====================== RESTART: D:/Algorithm/ch3/ch3_1.py ======================
5
15
25
```

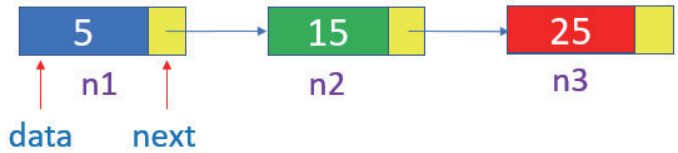
上述执行第 8 ～ 10 行后，可以在内存内建立下列 3 个节点。



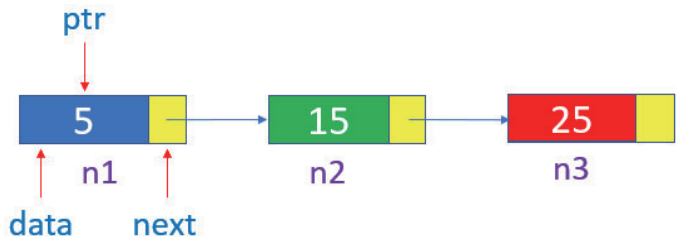
执行第 11 行后链表节点内容如下：



执行第 12 行后链表节点内容如下：



执行第 13 行后会多一个指标 ptr：



第 14 ～ 16 行可以打印此链表，得到 5、15、25。

3-8-2 建立链表类别和遍历此链表

其实前一节笔者已经用实例讲解了建立链表的方式，也说明了遍历链表，这一节主要讲解建立一个链表 Linked_list 类别，在这个类别内我们使用 __init__( ) 设计链表的第一个节点，同时使用 print_list( ) 打印链表。

程序实例 ch3_2.py：以建立 Linked_list 类别方式重新设计 ch3_1.py。

```
# ch3_2.py
class Node():
    ''' 节点 '''
    def __init__(self, data=None):
        self.data = data            # 数据
        self.next = None            # 指标

class Linked_list():
    ''' 链表 '''
    def __init__(self):
        self.head = None            # 链表第 1 个节点

    def print_list(self):
        ''' 打印链表 '''
        ptr = self.head             # 指标指向链表第 1 个节点
        while ptr:
            print(ptr.data)         # 打印节点
            ptr = ptr.next          # 移动指标到下一个节点

link = Linked_list()
link.head = Node(5)
n2 = Node(15)                       # 节点 2
n3 = Node(25)                       # 节点 3
link.head.next = n2                 # 节点 1 指向节点 2
n2.next = n3                        # 节点 2 指向节点 3
link.print_list()                   # 打印链表 link
```

执行结果 与 ch3_1.py 相同。

3-8-3　在链表第一个节点前插入一个新的节点

在链表的应用中，常常需要插入新的节点数据，这一节重点是将新节点插入链表的第一个节点之前，也就是插在链表开头的位置。

程序实例 ch3_3.py：扩充 ch3_2.py，新建数据是 100 的节点，同时将 100 插入链表开头的位置。

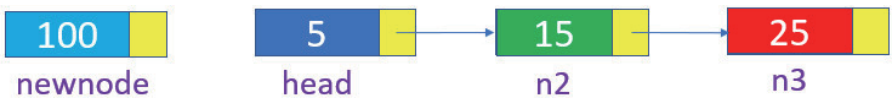
```
# ch3_3.py
class Node():
    ''' 节点 '''
    def __init__(self, data=None):
        self.data = data            # 数据
        self.next = None            # 指标

class Linked_list():
    ''' 链表 '''
    def __init__(self):
        self.head = None            # 链表第 1 个节点
```

```
13      def print_list(self):
14          ''' 打印链表 '''
15          ptr = self.head              # 指标指向链表第 1 个节点
16          while ptr:
17              print(ptr.data)          # 打印节点
18              ptr = ptr.next           # 移动指标到下一个节点
19
20      def begining(self, newdata):
21          ''' 在第 1 个节点前插入新节点 '''
22          new_node = Node(newdata)     # 建立新节点
23          new_node.next = self.head    # 新节点指标指向旧的第1个节点
24          self.head = new_node         # 更新链表的第一个节点
25
26  link = Linked_list()
27  link.head = Node(5)
28  n2 = Node(15)                        # 节点 2
29  n3 = Node(25)                        # 节点 3
30  link.head.next = n2                  # 节点 1 指向节点 2
31  n2.next = n3                         # 节点 2 指向节点 3
32  link.print_list()                    # 打印链表 link
33  print("新的链表")
34  link.begining(100)                   # 在第 1 个节点前插入新的节点
35  link.print_list()                    # 打印新的链表 link
```

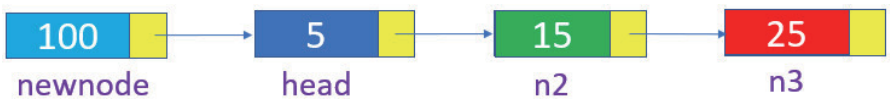
执行结果

```
====================== RESTART: D:\Algorithm\ch3\ch3_3.py ======================
5
15
25
新的链表
100
5
15
25
```

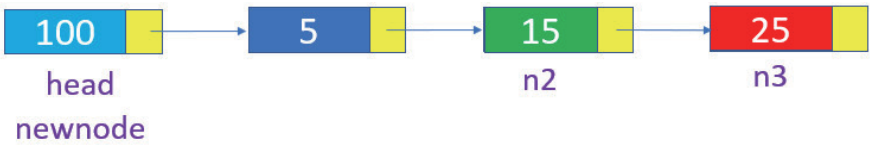
上述程序第 34 行是调用 begining( ) 方法，同时传递新节点值 100，当执行第 22 行后，链表节点内容如下：



当执行第 23 行后，链表节点内容如下：



当执行第 24 行后，链表节点内容如下：



3-8-4　在链表末端插入新的节点

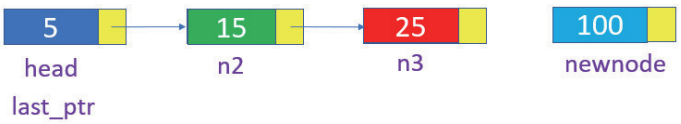
程序实例 ch3_4.py：在链表的末端插入新的节点。

```
# ch3_4.py
class Node():
    ''' 节点 '''
    def __init__(self, data=None):
        self.data = data                # 数据
        self.next = None                # 指标

class Linked_list():
    ''' 链表 '''
    def __init__(self):
        self.head = None                # 链表第 1 个节点

    def print_list(self):
        ''' 打印链表 '''
        ptr = self.head                 # 指标指向链表第 1 个节点
        while ptr:
            print(ptr.data)             # 打印节点
            ptr = ptr.next              # 移动指标到下一个节点

    def ending(self, newdata):
        ''' 在链表末端插入新节点 '''
        new_node = Node(newdata)        # 建立新节点
        if self.head == None:           # 如果是True, 表示链表是空的
            self.head = new_node        # 所以head就可以直接指向此新节点
            return
        last_ptr = self.head            # 设定最后指标是链表头部
        while last_ptr.next:            # 移动指标直到最后
            last_ptr = last_ptr.next
        last_ptr.next = new_node        # 将最后一个节点的指标指向新节点

link = Linked_list()
link.head = Node(5)
n2 = Node(15)                           # 节点 2
n3 = Node(25)                           # 节点 3
link.head.next = n2                     # 节点 1 指向节点 2
n2.next = n3                            # 节点 2 指向节点 3
link.print_list()                       # 打印链表 link
print("新的链表")
link.ending(100)                        # 在链表末端插入新的节点
link.print_list()                       # 打印新的链表 link
```

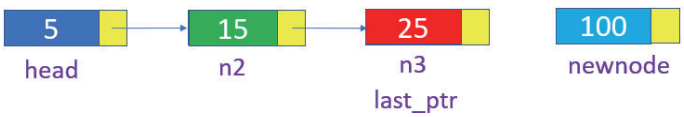
执行结果

```
========== RESTART: D:\Algorithm\ch3\ch3_4.py ==========
5
15
25
新的链表
5
15
25
100
```

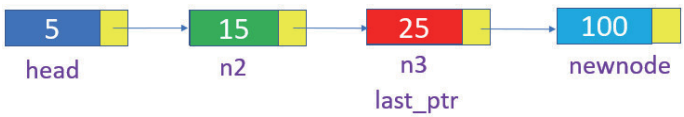
对于在链表末端插入节点，程序在第 20 ～ 29 行使用了 ending( ) 方法，当执行第 26 行后，链表节点内容如下：



当执行第 27 ～ 28 行后，链表节点内容如下：



当执行第 29 行后，链表节点内容如下：



3-8-5 在链表中间插入新的节点

程序实例 ch3_5.py：在链表 n2 节点的后面插入新的节点。

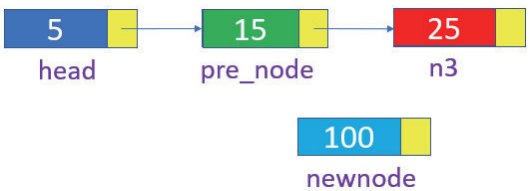
```
# ch3_5.py
class Node():
    ''' 节点 '''
    def __init__(self, data=None):
        self.data = data            # 数据
        self.next = None            # 指标

class Linked_list():
    ''' 链表 '''
    def __init__(self):
        self.head = None            # 链表第 1 个节点

    def print_list(self):
        ''' 打印链表 '''
        ptr = self.head             # 指标指向链表第 1 个节点
        while ptr:
```

```
17              print(ptr.data)          # 打印节点
18              ptr = ptr.next           # 移动指标到下一个节点
19
20      def between(self, pre_node, newdata):
21          ''' 在链表两个节点间插入新节点 '''
22          if pre_node == None:
23              print("缺插入节点的前一个节点")
24              return
25          # 建立和插入新节点
26          new_node = Node(newdata)     # 建立新节点
27          new_node.next = pre_node.next # 新建节点指向前一节点的下一节点
28          pre_node.next = new_node     # 前一节点指向新建节点
29
30  link = Linked_list()
31  link.head = Node(5)
32  n2 = Node(15)                        # 节点 2
33  n3 = Node(25)                        # 节点 3
34  link.head.next = n2                  # 节点 1 指向节点 2
35  n2.next = n3                         # 节点 2 指向节点 3
36  link.print_list()                    # 打印链表 link
37  print("新的链表")
38  link.between(n2, 100)                # 在链表n2插入新的节点
39  link.print_list()                    # 打印新的链表 link
```

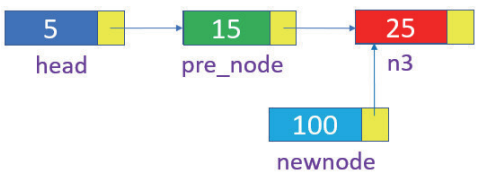
执行结果

```
==================== RESTART: D:\Algorithm\ch3\ch3_5.py ====================
5
15
25
新的链表
5
15
100
25
```

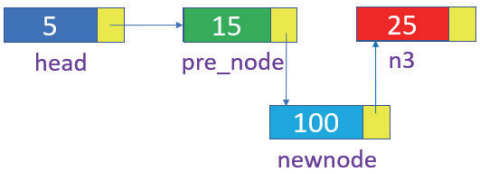
对于在链表中间插入节点，程序在第 20 ～ 28 行使用了 between( ) 方法，调用这个方法需要使用 2 个参数，第 1 个参数 pre_node 是指出要将新数据插入哪一个节点，第 2 个参数是新节点的值，当执行第 26 行后，链表节点内容如下：



当执行第 27 行后，链表节点内容如下：



当执行第 28 行后，链表节点内容如下：



3-8-6 在链表中删除指定内容的节点

程序实例 ch3_6.py：在链表中删除指定的节点前，先建立链表，此链表含有 5、15、25 这 3 个节点，然后删除 15 这个节点。

```
# ch3_6.py
class Node():
    ''' 节点 '''
    def __init__(self, data=None):
        self.data = data            # 数据
        self.next = None            # 指标

class Linked_list():
    ''' 链表 '''
    def __init__(self):
        self.head = None            # 链表第 1 个节点

    def print_list(self):
        ''' 打印链表 '''
        ptr = self.head             # 指标指向链表第 1 个节点
        while ptr:
            print(ptr.data)         # 打印节点
            ptr = ptr.next          # 移动指标到下一个节点

    def ending(self, newdata):
        ''' 在链表末端插入新节点 '''
        new_node = Node(newdata)    # 建立新节点
        if self.head == None:       # 如果是True，表示链表是空的
            self.head = new_node    # 所以head就可以直接指向此新节点
            return
        last_ptr = self.head        # 设定最后指标是链表头部
        while last_ptr.next:        # 移动指标直到最后
            last_ptr = last_ptr.next
        last_ptr.next = new_node    # 将最后一个节点的指标指向新节点

    def rm_node(self, rmkey):
        ''' 删除值是rmkey的节点 '''
        ptr = self.head             # 暂时指标
        if ptr:
            if ptr.data == rmkey:
                self.head = ptr.next  # 将第1个指标指向下一个节点
                return
        while ptr:
            if ptr.data == rmkey:
                break
```

```
41              prev = ptr                  # 设定前一节点指标
42              ptr = ptr.next              # 移动指标
43          if ptr == None:                 # 如果ptr已经是末端
44              return                      # 找不到所以返回
45          prev.next = ptr.next            # 找到了所以将前一节点指向下一个节点
46
47  link = Linked_list()
48  link.ending(5)
49  link.ending(15)
50  link.ending(25)
51  link.print_list()                       # 打印链表 link
52  print("新的链表")
53  link.rm_node(15)                        # 删除值是15的节点
54  link.print_list()                       # 打印新的链表 link
```

执行结果

```
====================== RESTART: D:\Algorithm\ch3\ch3_6.py ======================
5
15
25
新的链表
5
25
```

上述程序第 33 行是建立暂时指标 ptr，指向链表的第一个节点，第 41 ～ 42 行是建立暂时指标的前一个指标 prev，未来找到删除节点时 (ptr 所指的节点)，prev.next 指向 ptr.next，这样就算是删除暂时指标 ptr 所指的节点了，可以参考第 45 行。第 43 ～ 44 行主要是用在找不到指定节点时，可以直接返回。

3-8-7　建立循环链表

如果想要建立循环链表，只要将链表末端节点指向第 1 个节点即可。

程序实例 ch3_7.py：建立循环链表，此列表有 3 个节点，打印 6 次。

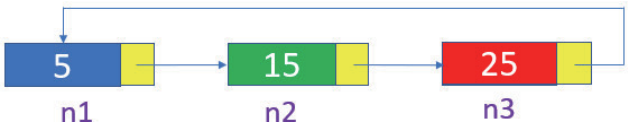
```
1   # ch3_7.py
2   class Node():
3       ''' 节点 '''
4       def __init__(self, data=None):
5           self.data = data            # 数据
6           self.next = None            # 指标
7
8   n1 = Node(5)                        # 节点 1
9   n2 = Node(15)                       # 节点 2
10  n3 = Node(25)                       # 节点 3
11  n1.next = n2                        # 节点 1 指向节点 2
12  n2.next = n3                        # 节点 2 指向节点 3
13  n3.next = n1                        # 末端节点指向起始节点
14  ptr = n1                            # 建立指标节点
15  counter = 1
16  while counter <= 6:
17      print(ptr.data)                 # 打印节点
18      ptr = ptr.next                  # 移动指标到下一个节点
19      counter += 1
```

执行结果

```
====================== RESTART: D:/Algorithm/ch3/ch3_7.py ======================
5
15
25
5
15
25
```

上述执行第 12 行后链表节点如下所示：

上述执行第 13 行后链表节点如下所示：



这样就完成了循环链表。

3-8-8 双向链表

如果要建立双向链表，每个节点必须有向前指标和向后指标，可以使用下列方式定义此节点。

```
class Node():
    ''' 节点 '''
    def __init__(self, data=None):
        self.data = data            # 数据
        self.next = None            # 向后指标
        self.previous = None        # 向前指标
```

程序实例 ch3_8.py：建立双向链表，在建立节点过程中，每次均从头部打印一次双向链表，最后从尾部打印一次双向链表。

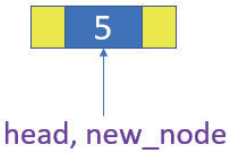
```
1  # ch3_8.py
2  class Node():
3      ''' 节点 '''
4      def __init__(self, data=None):
5          self.data = data            # 数据
6          self.next = None            # 向后指标
7          self.previous = None        # 向前指标
8
9  class Double_linked_list():
10     ''' 链表 '''
11     def __init__(self):
12         self.head = None            # 链表头部的节点
13         self.tail = None            # 链表尾部的节点
```

```
14
15      def add_double_list(self, new_node):
16          ''' 将节点加入双向链表 '''
17          if isinstance(new_node, Node):       # 先确定这item是节点
18              if self.head == None:            # 处理双向链表是空的
19                  self.head = new_node         # 头是new_node
20                  new_node.previous = None     # 指向前方
21                  new_node.next = None         # 指向后方
22                  self.tail = new_node         # 尾节点也是new_node
23              else:                            # 处理双向链表不是空的
24                  self.tail.next = new_node    # 尾节点指标指向new_node
25                  new_node.previous = self.tail   # 新节点前方指标指向前
26                  self.tail = new_node         # 新节点成为尾部节点
27          return
28
29      def print_list_from_head(self):
30          ''' 从头部打印链表 '''
31          ptr = self.head              # 指标指向链表第 1 个节点
32          while ptr:
33              print(ptr.data)          # 打印节点
34              ptr = ptr.next           # 移动指标到下一个节点
35
36      def print_list_from_tail(self):
37          ''' 从尾部打印链表 '''
38          ptr = self.tail              # 指标指向链表尾部节点
39          while ptr:
40              print(ptr.data)          # 打印节点
41              ptr = ptr.previous       # 移动指标到前一个节点
42
43  double_link = Double_linked_list()
44  n1 = Node(5)                         # 节点 1
45  n2 = Node(15)                        # 节点 2
46  n3 = Node(25)                        # 节点 3
47
48  for n in [n1, n2, n3]:
49      double_link.add_double_list(n)
50      print("从头部打印双向链表")
51      double_link.print_list_from_head()  # 从头部打印双向链表
52
53  print("从尾部打印双向链表")
54  double_link.print_list_from_tail()  # 从尾部打印双向链表
```

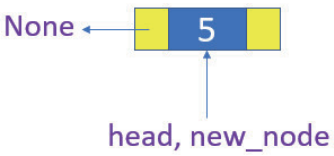
执行结果

```
========== RESTART: D:\Algorithm\ch3\ch3_8.py ==========
从头部打印双向链表
5
从头部打印双向链表
5
15
从头部打印双向链表
5
15
25
从尾部打印双向链表
25
15
5
```

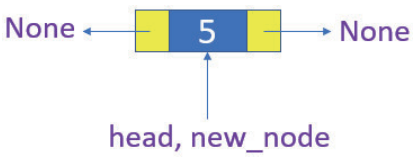
这个程序第 15 ～ 27 行使用了 add_double_list( ) 方法，将每个节点加入链表，第 17 行主要是确定所增加的数据是双向链表的节点，再执行 18 ～ 26 行。其中 19 ～ 22 行是增加第一个节点，当执行完第 19 行，链表节点内容如下：



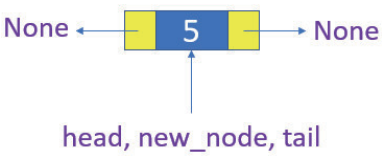
当执行完第 20 行，链表节点内容如下：



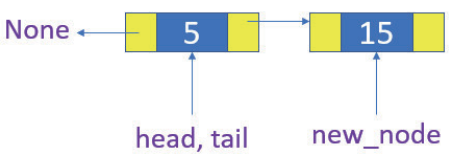
当执行完第 21 行，链表节点内容如下：



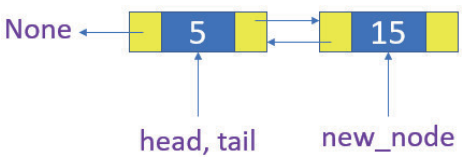
当执行完第 22 行，链表节点内容如下：



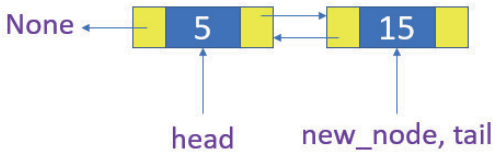
上述就是建立双向链表的第一个节点过程。程序第 24 ～ 26 行是建立双向链表第 2 个 (含) 以后的节点过程，当执行完第 24 行，链表节点内容如下：



当执行完第 25 行，链表节点内容如下：



当执行完第 26 行，链表节点内容如下：



程序第 29 ～ 34 行的 print_list_from_head( ) 是从双向链表**前端打印到末端**，程序第 36 ～ 41 行的 print_list_from_tail( ) 是从双向链表**末端打印到前端**。

3-9 习题

1. 请修改 ch3_2.py，在 Linked_list 类别内增加 length( ) 方法，计算链表的长度（也可想成节点数量）。

```
====================== RESTART: D:\Algorithm\ex\ex3_1.py ======================
链表长度是 :  3
```

2. 请建立链表，列表节点有 3 个，内容分别是 5、15、5，同时设计一个搜寻方法，然后用参数 5、15、20 测试此搜寻方法，此程序会列出 5、15、20 在链表内各出现几次。

```
====================== RESTART: D:\Algorithm\ex\ex3_2.py ======================
所建的链表
5
15
5
分别列出数值5, 15, 20的出现次数
5  出现 2 次
15 出现 1 次
20 出现 0 次
```

3. 为星期的英文缩写建立双向链表，然后分别从头打印和从尾打印。

```
====================== RESTART: D:\Algorithm\ex\ex3_3.py ======================
从头部打印双向链表
Sun
Mon
Tue
Web
Thu
Fri
Sat
从尾部打印双向链表
Sat
Fri
Thu
Web
Tue
Mon
Sun
```