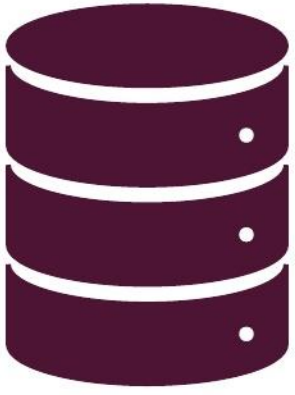


数据结构

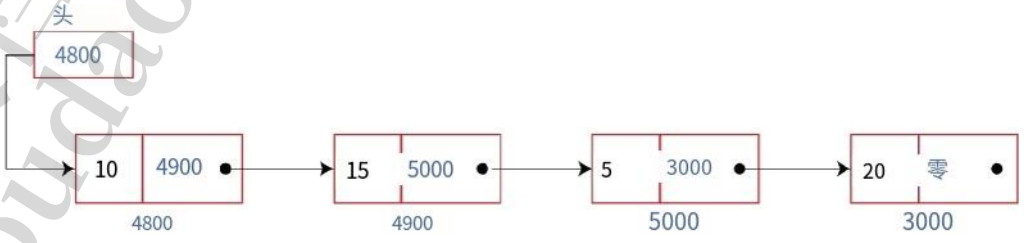
WENYE 李
CUHK-SZ

大纲

- 列表
- 实现 示例

列表

- 链表:一个节点连接到另一个节点的节点集合。
  - 像数组一样，链表是一种线性的数据结构。
  - 与数组不同，链表元素不是存储在一个连续的位置;它们使用指针进行链接。



为什么是链表?

- 数组存储类似类型的线性数据，有以下限制。
  - 数组的大小是固定的。我们必须提前知道元素数量的上限。无论使用情况如何，分配的内存都等于上限。
  - 在数组中插入一个新元素是昂贵的，因为必须为新元素创建房间，并且要创建房间，必须移动现有的元素。

- 例如，在一个系统中，如果我们在数组 id[]中维护一个有序的 id 列表。
  - id[] =[1000, 1010, 1050,2000, 2040]。
  - 并且如果我们想插入一个新的 ID 1005，那么为了保持排序顺序，我们必须将所有元素移动到 1000 之后(不包括 1000)。

  在使用数组时，删除的代价也是很高的，除非使用了一些特殊的技术。比如，要删除 id[]中的 1010,1010 之后的所有东西都要被移动。

ADVANTAGES

- 动态数据结构：
  - 链表的大小不是固定的，因为它可以根据需求变化。

- 插入和删除：
  - 链表中的插入和删除比数组更容易，因为数组中的元素存储在连续的位置。相比之下，在链表的情况下，元素被存储在随机的位置。
  - 如果我们想插入或删除数组中的元素，那么我们需要移动元素以创建空间。
  - 在链表中，我们不需要移动元素。我们只需要更新节点中指针的地址。

- 记忆效率
  - 它的内存消耗是高效的，因为链表的大小可以根据我们的需求增长或缩小。

DISADVANTAGES

- 内存使用情况
  - 链表中的节点比数组中的节点占用更多的内存，因为每个节点占用两种类型的变量，即，一种是简单变量，另一种是在内存中占用 4 字节的指针变量。

- 遍历
  - 在数组中，我们可以通过索引随机访问元素。
  - 在链表中，遍历并不容易。如果我们想要访问链表中的元素，就不能随机访问元素。
  - 示例:如果我们想访问第 3 个节点，那么我们需要遍历它之前的所有节点。所以，访问一个特定节点所需的时间是很大的。

- 反向遍历
  - 在链表中，回溯或反向遍历是困难的。
  - 在双链表中，比较容易，但存储后向指针需要更多的内存。

应用程序

输入:

第一个数字= 5x2 + 4x1 + 2x9

第二个数字= -5x1 - 5x9

输出:

5 x2-1x1-3x°

输入:

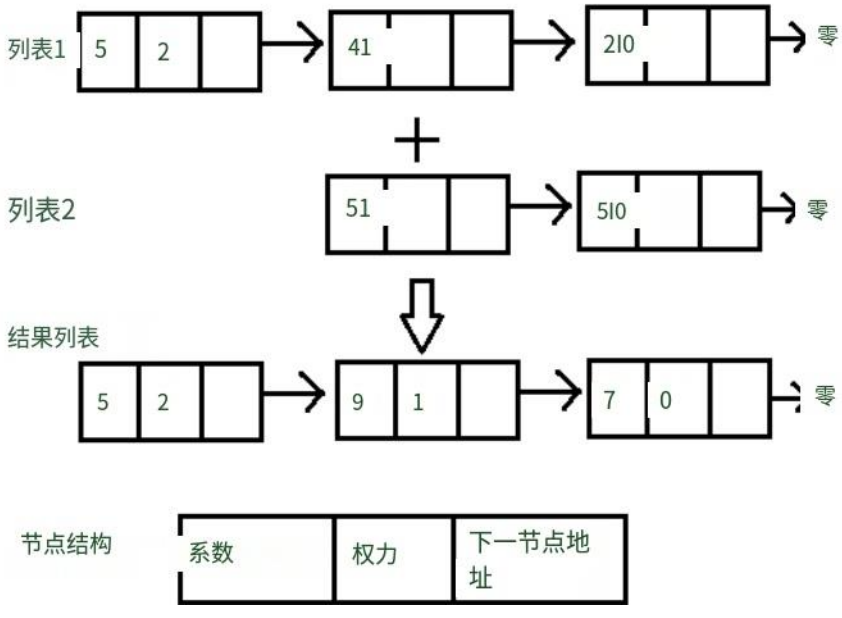
第一个数字= 5x3 + 4x2 + 2x9

第二个数= 5x^1 - 5x^0

输出:

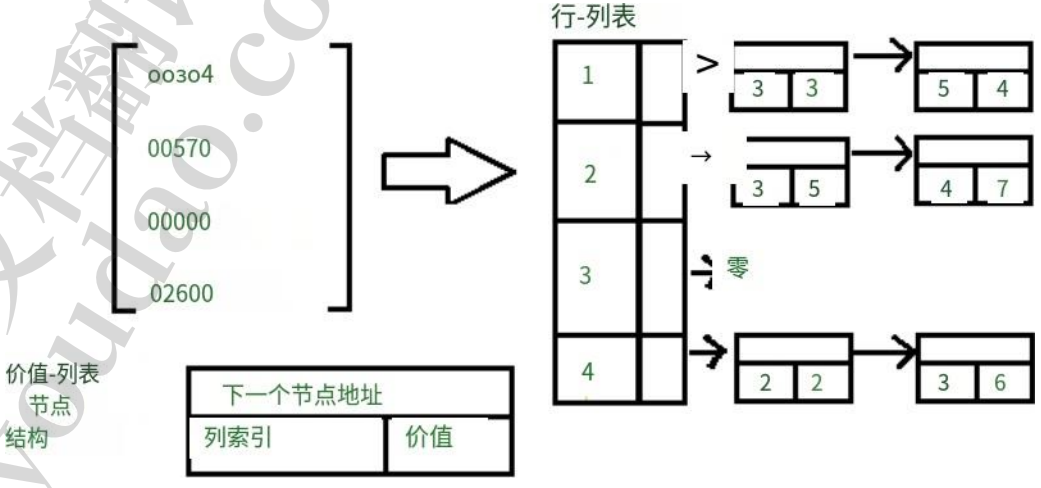
5x3 + 4x2 + 5x1 - 3x8

- 多项式是项的集合每一项包含哪些系数和权力。
- 每一项的系数和幂存储为节点和链接指针点指向链表中的下一个元素，所以链表可以用来创建、删除并显示多项式。



APPLICATIONS

- 稀疏矩阵。



APPLICATIONS

各种操作，如学生的详细信息，员工的详细信息或产品的详细信息可以使用链表实现，因为链表使用结构数据类型，可以容纳不同的数据类型。

栈，队列，树和各种其他数据结构可以使用链表实现。

图是边和顶点的集合，图可以表示为邻接矩阵和邻接表。如果我们想将图表示为邻接矩阵，那么它可以实现为数组。如果我们想将图表示为邻接表，那么它可以实现为链表。

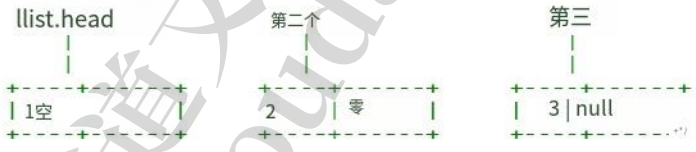
要实现哈希，我们需要哈希表。哈希表包含使用链表实现的条目。

链表可以用来实现动态内存分配。动态内存分配是在运行时完成的内存分配。

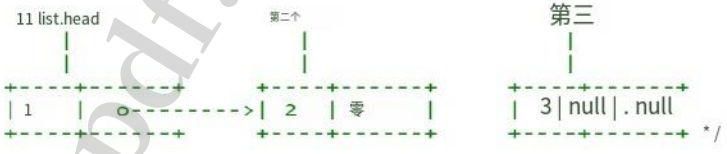
```
公共类LinkedList
{
    节点头;//列表头

    /* 链表节点*/
    类节点
    {
        int数据;节点下
        ;

        //创建新节点的构造函数Next默认初始化
        //
        // 为零
        Node(int d)
        {
            Data = d;
        }
    }
}
```


在 Java 中，LinkedList 可以表示为一个类，而节点可以表示为一个单独的类。LinkedList 类包含节点类类型的引用。

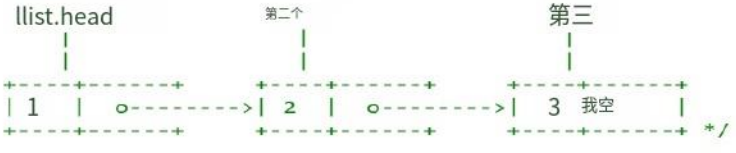
```
//一个简单的Java程序来引入链表类LinkedList{
    节点头;//列表头

    /*链表节点。这个内部类被设置为静态，以便main()可以访问它*/静态类Node{

        节点下;
        Node(int d)
        {
        Data = d: next = null;} //构造
        函数
    }

    /*方法创建一个简单的3节点链表*/l public static void main(String[] args)
    {
        /*从空列表开始。* /
        链表llist =新的链表();

        1列表。head = new Node(1);Node second =新
        Node(2);节点三=新节点(3);

        /*已经动态分配了3个节点。我们有对这三个块的引用作为head,
          第二和第三
```




```
        Llist.head.next =秒;//链接第一个节点和第二个节点

        /*现在第一个节点的next指向第二个节点。所以它们都是相连的。
```




```
        第二。下一=第三;//链接第二个节点和第三个节点

        /*现在第二个节点的next指向third。所以这三个节点都是相连的。
```



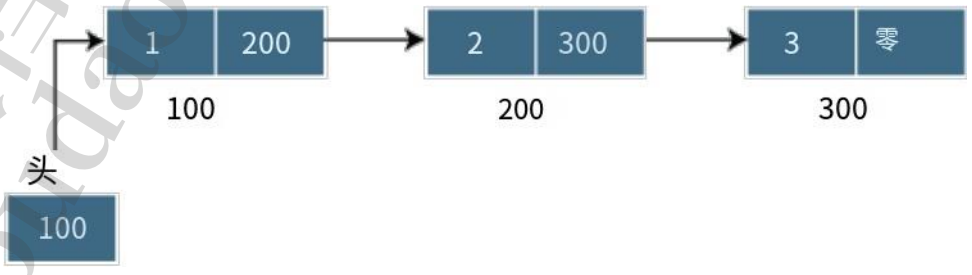

```
    }
}
```

链表的类型

- 单链表 双链表
- 循环链接列表
- 双循环链接列表

单链表

- 一个单链表在一个节点中有两个部分
  - 数据部分
  - 地址部分，包含下一个节点的地址，也称为指针。



- 只能进行前向遍历
  - 我们不能向后遍历，因为它在列表中只有一个链接。

单链表上的操作

插入

插入到单链表可以在不同的位置执行。根据被插入的新节点的位置，插入为
分为以下几类。

SN	操作	描述
-	插入 开始	它涉及到在列表的前面插入任何元素。我们只需要做一些链接调整，使新节点作为列表的头部。
2	在列表的末尾插入	它涉及到在链表的最后插入。新节点可以作为链表中唯一的节点插入，也可以作为最后一个节点插入。在每个场景中实现了不同的逻辑。
~	插入 指定的节点	它涉及到在链表的指定节点之后的插入。我们需要跳过所需的节点数量，以便到达将插入新节点的节点。

对单链表的操作

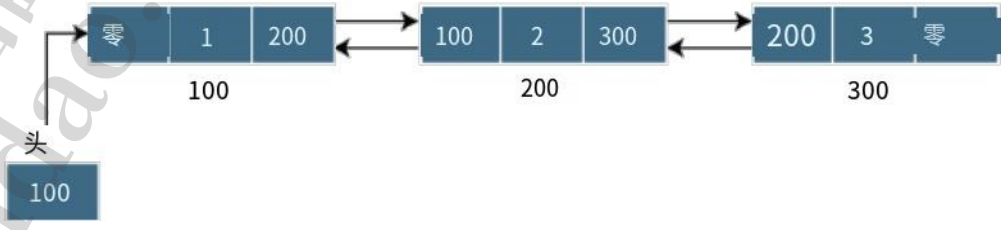
删除和遍历

从一个单链表中删除一个节点可以在不同的位置执行。根据被删除节点的位置，该操作可分为以下几类。

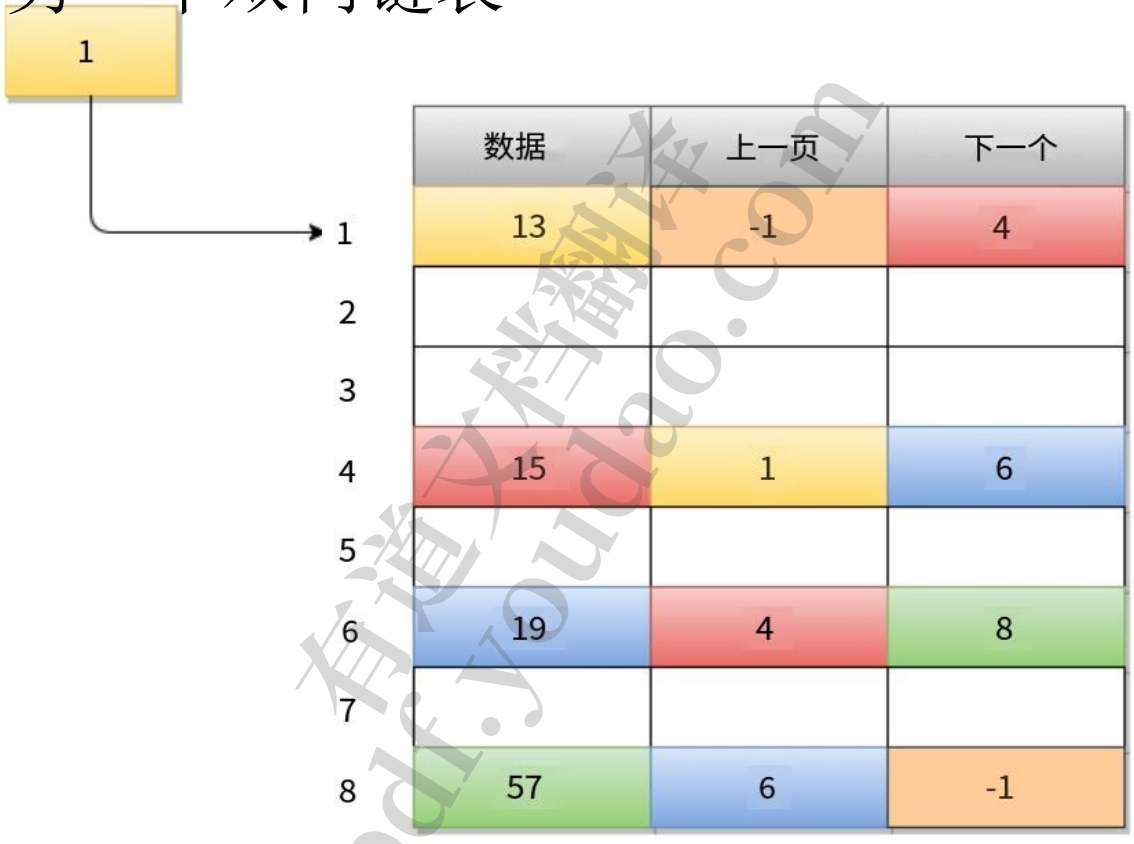
SN	操作	描述
1	删除从开始删除到列表末尾	它涉及从列表的开头删除一个节点。这是所有操作中最简单的。它只需要在节点指针上做一些调整。
2		它涉及删除列表的最后一个节点。列表可以是空的，也可以是满的。不同的场景实现了不同的逻辑。
3	删除指定节点后	它涉及删除列表中指定节点之后的节点。我们需要跳过所需的节点数量，以到达该节点之后该节点将被删除。这需要遍历列表。
4	遍历	在遍历中，我们只是简单地访问列表的每个节点至少一次，以便对其执行一些特定的操作，例如，打印列表中存在的每个节点的数据部分。
5	搜索	在搜索中，我们将列表中的每个元素与给定的元素进行匹配。如果在任意位置找到该元素，则返回该元素的位置，否则返回null。

双链表

- 一个双链表有三个部分在一个节点
  - 一个数据部分
  - 一个指向上一个节点的指针 一个指向下一个节点的指针



另一个双向链表



双链表的内存表示

对双链表的操作

SN	操作	描述
1	开始插入	开始时将节点添加到链表中。
2	末端插入	将节点添加到链表的末尾。
3	在指定节点之后插入	在指定节点之后将节点添加到链表中。
4	开始删除	从列表开始删除节点
5	最后删除	从列表末尾删除节点。
6	删除给定数据的节点	删除位于包含给定数据的节点之后的节点。
7	搜索	将每个节点数据与要搜索的项进行比较，如果找到项则返回该项在列表中的位置，否则返回null。
8	遍历	访问列表的每个节点至少一次，以便执行一些特定的操作，如搜索、排序、显示等。

循环链表

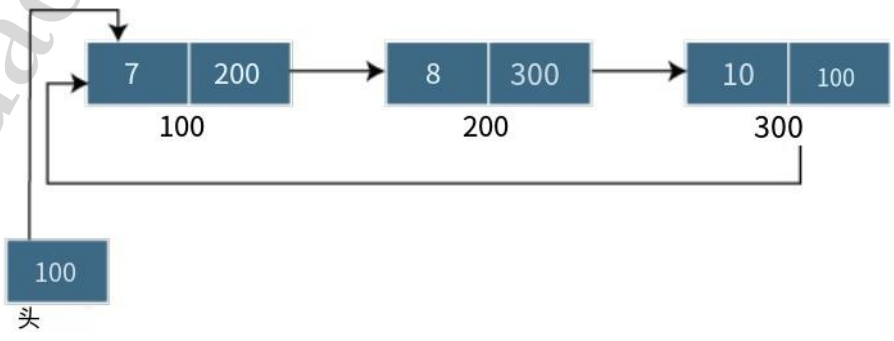
循环链表是单链表的一种变体。

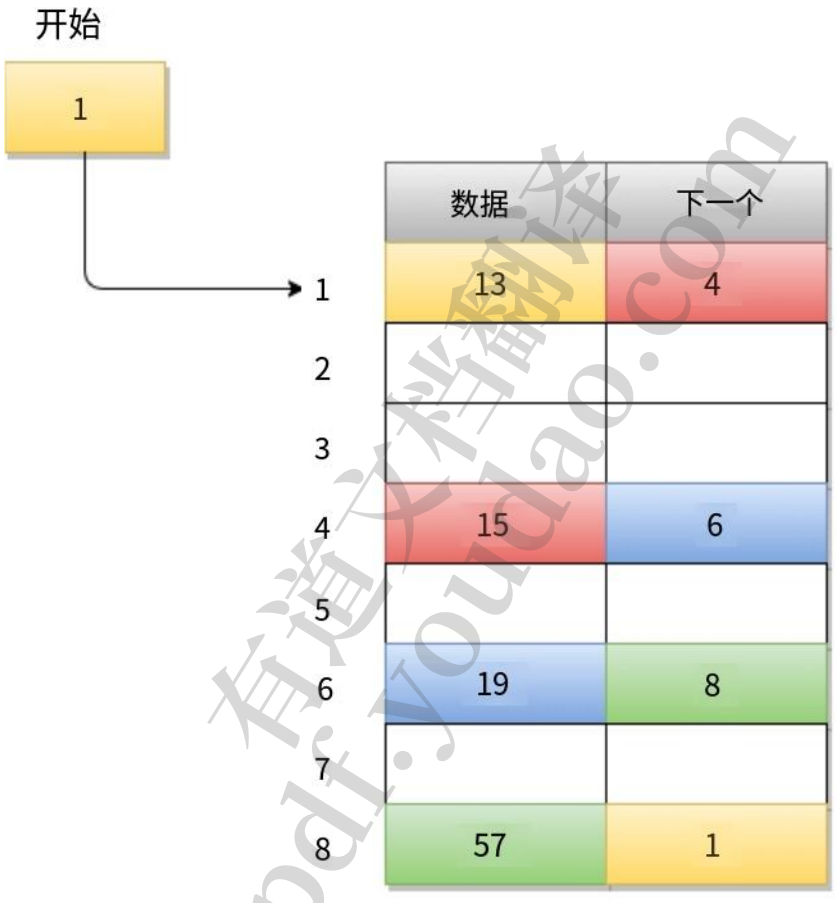
单链表和循环链表的区别

□最后一个节点不指向

单链表中的任意一个节点。□ 最后一个节点连接到

循环链表中的第一个节点。□ 循环链表没有起始节点和结束节点。我们可以向任何方向遍历。





循环链表的内存表示

循环链表的操作

插入

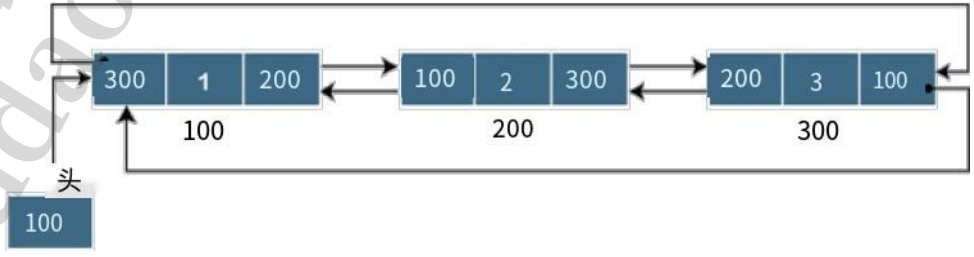
SN	操作	描述
1	开头插入	在循环单链表开始时添加一个节点。
2	结尾插入	在末尾的循环单链表中添加一个节点。

删除和遍历

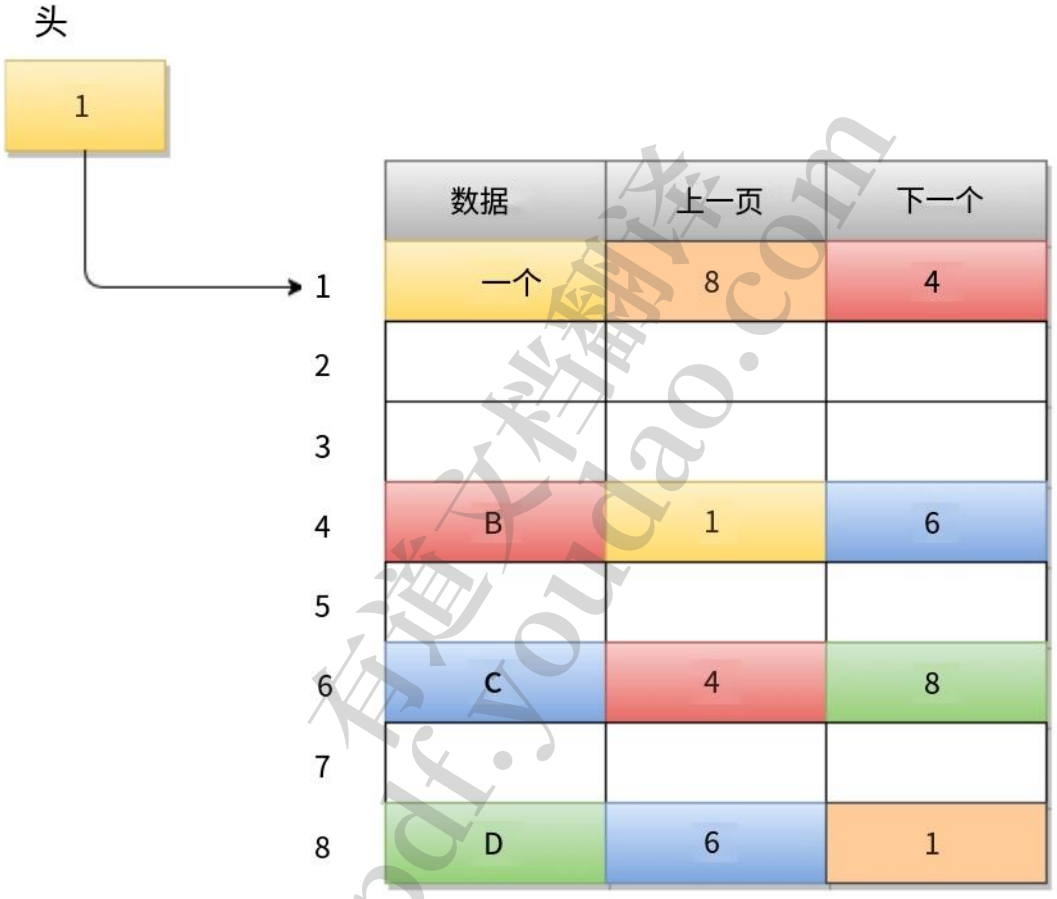
SN	操作	描述
1	删除在开始	在开始时从循环单链表中删除节点。
2	结尾删除	最后从循环单链表中删除节点。
3	搜索	将节点的每个元素与给定的项进行比较，并返回该项在列表中出现的位置，否则返回null。
4	遍历	为了执行一些特定的操作，至少访问列表中的每个元素一次。

双循环链表

- 双循环链表兼具循环链表和双链表的特点。
  - 最后一个节点附在第一个节点上，创建一个圆。
  - 每个节点都保存着前一个节点的地址。



- 一个双循环链表在一个节点中有三个部分
  - 两个地址部分
  - 一个数据部分
  - 所以，它的表示类似于双链表。



循环双链表的内存表示

对双循环链表的操作

SN	操作	描述
-	开头插入	开始时在循环双链表中添加一个节点。
2	末端插入	在循环双链表的末尾添加一个节点。
3	开始删除	从头删除循环双链表中的一个节点。
4	结束删除	在循环双链表的末尾删除一个节点。

循环双链表中的遍历和搜索类似于循环单链表中的遍历和搜索。

大纲

- 列表
- 实现 示例

类节点

公共 int 数据;公
共节点 next;

public void displayNodeData(){系统退出。
Printin(“{” + data + “}”);

公有类 SinglyLinkedList{私有节
点头;

public boolean isEmpty()
{return (head == nul1);

//用于在链表的开头插入一个节点
节点 newNode=新节点(;
newNode。data =数
据)
newNode.next=头部
head =newNode;

/用于从链表开始删除节点 public node
deleteFirst() {
Node temp =head;
Head = Head .next;

返回临时;

```
32      //用于删除特定节点后面的节点
33      删除后(Node after) {
34          节点 temp=头;
35          While (temp.next != null && temp.data != after.data) {temp
36              =temp.next;
37
            If (temp.next != nul1)
                temp.next =



        //用于在链表的开头插入一个节点
        插入 last (int data) {
            Wode current =head
45          (当前。 Next != nul1) {
46
47
            节点 newNode=新节点
48          (;newNode。 Data 数据流。
            next =newNode



        //打印链表
        public void printLinkedList() {
            系统退出。 打印("打印链接列表(head -> last) "):节点当前
            头部
            While (current != null) {
```

```
            current.displayNodeData
59          ();Current  = Current  .next;
68
        System.out.println();
61
62
```

LinkedListMain.java

```
公共类 LinkedListMain  f

  public static void main(字符串 args[])

        Singlylinkedlist  myLinkedlist=新的 Singlylinkedlist();
    myLinkedlist.insertFirst(5)
          myLinkedlist。
           insertFirst(6)
          myLinkedlist。
           insertFirst(7)
    myLinkedlist.insertFirst(1)
          myLinkedlist。
           insertLast(2);
          链表将被
     / / 1 - > 2 - > 7 - > 6 - >
                  5
       Node  Node  -new
            Node()
    节点。data =
          1;
     myLinkedl坚持。删除后(node);
    //                删除 1后的节点后，链表将被
       /2-> 1 -> 6 -> 5
     myLinkedlist.printLinkedlist);
```


当你运行上面的程序时，你会得到以下输出:

打印链接列表(head -> Last) {13

{63

{5

{2

H 2 m 4 □

7

9
日

```
//使用迭代方法查找链表的长度 public int
lengthoflinklist () {
        节点 temp-head;
        Int count = e;
        而(临时! = nul1) {
                temp-temp.next;
                数+ +


        返回计数。
```

```
公共节点 nthFromLastNode(节点头， int n){节点
    firstPtr-head;
    节点 secondPtr-head;

    For (int I = e;I < n;我+ +){
        firstPtr-firstPtr.next


    而(firstPtr ! = nul1) {
        firstptr-firstPtr 。 下 一
        个          secondPtr-
        secondPtr.next;


    返回 secondPtr;
```

//查找 linkedlist 中的中间元素

2 (

节点 slowPointer，
fastPointer;slowPointer fastPointer=
头部:

while(fastPointer !=nul1) {

□

fastPointer=fastPointer.next;

如果(fastPointer != null &&①fastPointer。 下一步!=
null){slowPointer= slowPointer.next。

fastPointer fastPointer.next;

返回 slowPointer

```
/函数检查链表是否为回文或非公共静态布尔
checkPalindrome(节点头)f

    //用慢速和快速的 pointen 节点查找中间节点
    middleNode-findMiddleNode (head);

    //我们得到了第二部分的
    head

    节点 secondHead-middleNode.next;

     /它是链表第一部分的结尾
   middleNode.next=nul1;
18
    /得到反向链表的第二部分节点 reverseSecondHead-
    reverseLinkedList(secondHead);而(头!=null&&
13  reverseSecondHead!=nu11)
14  {if(head.value==reverseSecondHead.value){head-
15  head.next;
16
                reverseSecondHead-reverseSecondHead.next;
17              继续;
18
19
    返回 true;
20
```

```
//反向链表的迭代解决方案
f . public static节点reverseLinkedList(节点currentNode
        //对于第一个节点，previousNode将为空
        节点previousNode =零;节点
        nextNode;
        while(currentNode!=null) {
                nextNode=currentNode.next;
                //反向链接
                currentNode.next = previousNode;
                //将当前节点和前节点移动一个节点
                previousNode = currentNode;
                currentNode=nextNode;
        ~
        返回previousNode;
}
```

```
//反向链表的递归解决方案
public static节点reverseLinkedList(节点节点)节点。Next == null) {
        If (node == null .
                返回节点;
        ~

        节点剩余= reverseLinkedList(Node .next);
        node.next.next =节点;
        节点。Next = null;
        返回剩余;
}
```

```
public static void main(String[] args) {
    链表列表=新的链表();
    //创建链表
    节点头=new Node(5);
    list. addtolast(头部);

    列表。addtolast(新节点(6));列表。
    addtolast(新节点(7));列表。addtolast(
    新节点(1));列表。addtolast(新节点(2));


    list.printList(头);//反向链
    表
    节点reverseHead = reverseLinkedList(头);
    system . out。println(“扭转”);
    list.printList(reverseHead);
```


运行上述程序，你将得到以下输出:

```
56712
在扭转
21765
```

大纲

- 列表
- 实现 示例

例子:单链表

- 它用于实现堆栈和队列，这些类似于整个计算机科学的基本需求。
- 为了防止哈希图中数据之间的碰撞，我们使用了一个单链表。 一个随意的记事本使用单链表来执行撤销/重做功能。
- 我们可以想象它在照片查看器中的使用，在幻灯片中连续查看照片。

例如:循环链表

- 它还可以通过维护一个指向最后插入节点的指针来实现队列，并且前端总是可以作为倒数第二个节点获得。
- 双循环链表用于实现高级数据结构，如斐波那契堆。
- 它也被操作系统用来为不同的用户共享时间，一般采用轮询分时机制。
  - 每个人轮流获得均等的份额。它是最古老、最简单的调度算法，主要用于多任务处理。
- 多人游戏使用循环列表在玩家之间进行循环交换。 在 photoshop， word，或任何油漆中，我们在撤销功能中使用这个概念。

例子:双链表

- 双链表用于导航系统，用于前后导航。
- 在浏览器中当我们想要使用 back 或者 next 功能来更改选项卡时，它就使用了双向链表的概念。
- 实现其他数据结构很容易，比如二叉树、哈希表、栈等。
- 它用于音乐播放系统，你可以轻松地播放前一首或下一首歌曲的次数一个人想要。
- 在许多操作系统中，线程调度器会维护一个随时运行的所有进程的双向链表。
  - 这使得将进程从一个队列移动到另一个队列很容易。

为什么链表中的插入速度更快?

链表的每个元素都维护着两个指针(地址)指向链表中相邻的两个元素

链表有索引吗?

- 需要强调的是，与数组不同，链表没有内置索引索引。
- 为了在链表中找到一个特定的点，你需要从头开始并逐个遍历每个节点，直到找到你要找的东西。