
链表Linked List

1. 链表以节点的方式来存储
2. 每个节点包含
   data域：储存数据
   next域：指向下一个节点
3. 链表的各个节点不一定连续存储
4. 链表分为有头结点的链表和没有头结点的链表

单链表 Single Linked-List

1.创建过程

1. 创建头结点：

不存放数据

仅表示单链表头

有一个next域，指向第一个结点

2.创建数据结点

存放数据

有一个next域，指向下一个节点

3.单链表的功能

1. 加入在尾部加入元素

   设置一个节点来记录尾部元素

2. 根据某元素排序，在某个位置插入

3. 根据编号删除或者更新节点

4.代码实现

1.Node

```
package 线性结构.链表LinkedList.单链表;
public class Node {
    private int no;
    private String name;
    private String nickname;
    Node next;

    public Node(int no, String name, String nickname) {
        this.no = no;
        this.name = name;
        this.nickname = nickname;
    }
    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

```
    public String getNickname() {
        return nickname;
    }

    public void setNickname(String nickname) {
        this.nickname = nickname;
    }



    @Override
    public String toString() {
        return "Node{" +
                "no=" + no +
                ", name='" + name + '\'' +
                ", nickname='" + nickname + '\''+'}';
    }
}
```

2.SingleLinkedList

```
package 线性结构.链表LinkedList.单链表;

import com.sun.org.apache.xalan.internal.xsltc.compiler.Template;
import com.sun.org.apache.xpath.internal.WhitespaceStrippingElementMatcher;
import jdk.nashorn.internal.ir.WhileNode;
import sun.plugin2.message.ShowDocumentMessage;

import java.util.TooManyListenersException;
public class SingleLinkedList {
    private Node headNode = new Node(0, null, null);
    private Node footNode = new Node(0, null, null);

    public SingleLinkedList(Node headNode) {
        this.headNode = headNode;
        Node node=headNode;
        while (node.next!=null)
        {
            node=node.next;
        }
        footNode=node;
    }

    public Node getHeadNode() {
        return headNode;
    }
```

```
public Node getFootNode() {
    return footNode;
}

public SingleLinkedList() {
    headNode.next = footNode;
}

/**
 * 我的思路：通过一个内置节点记录最后一个节点better
 * @param node
 */
public void add(Node node) {
    if (footNode.getNo() == 0) {
        headNode.next = node;
        footNode = node;
        return;
    }
    footNode.next = node;
    footNode = node;
}

/**
 * 老师：遍历链表，得到最后一个节点
 *
 * @param node
 */
public void add1(Node node) {
    Node temp = headNode;
    while (true) {
        if (temp.next == null) {//
            break;
        }
        temp = temp.next;
    }

    temp.next = node;
}
/**
 * 自己的写的根据no排名添加节点到指定位置。专门用一个节点来记录上一个位置
 * @param node
 */
public void insertByNo(Node node) {
    if (footNode.getNo() == 0) {
        headNode.next = node;
        footNode = node;
        return;
```

```
    }
    Node temp = headNode.next;
    Node temp2 = headNode.next;
    while (temp != null) {
        int no = temp.getNo();
        int newNo = node.getNo();
        if (newNo == no) {
            System.out.println("添加失败");
            return;
        } else if (newNo > no) {
            if (temp.next == null) {
                temp.next = node;
                footNode = node;
                return;
            }
            temp2 = temp;
            temp = temp.next;
        } else {
            temp2.next = node;
            node.next = temp;
            return;
        }
    }
}
/**
 * 老师写的，用temp.next来寻找插入位置better
 * @param node
 */
public void insertByNo2(Node node) {
    if (footNode.getNo() == 0) {
        headNode.next = node;
        footNode = node;
        return;
    }
    Node temp = headNode;
    while (temp.next != null) {
        int no = temp.next.getNo();
        int newNo = node.getNo();
        if (newNo == no) {
            System.out.println("添加失败");
            return;
        } else if (newNo < no) {
            node.next = temp.next;
            temp.next = node;
            return;
        }
        temp = temp.next;
```

```
        }
        temp.next = node;
        footNode = node;
    }

    public void show() {
        Node node = headNode.next;
        while (node != null) {
            System.out.println(node);
            node = node.next;
        }
    }
    /**
     * 自己写的，用temp来寻找插入位置，用temp2来记录前一个节点
     *
     * @param no
     */
    public void deleteByNo(int no) {
        if (footNode.getNo() == 0) {
            System.out.println("空链表");
            return;
        }
        Node temp = headNode.next;
        Node temp2 = headNode;
        while (temp != null) {
            int n = temp.getNo();
            if (n == no) {
                if (temp == footNode) {
                    footNode = temp2;
                }
                temp2.next = temp.next;
                return;
            }
            temp = temp.next;
            temp2 = temp2.next;
        }
        System.out.println("没有该节点");
    }

    public void updateByNo(int no, String name, String nickname) {
        Node temp = headNode.next;
        if (temp == null) {
            System.out.println("空链表");
        }
        while (temp != null) {
            int n = temp.getNo();
            if (n == no) {
```

```
                temp.setName(name);
                temp.setNickname(nickname);
                return;
            }
        }
        System.out.println("没有该节点");
    }

    /**
     * 获取单链表的有效数据节点个数
     * @return
     */
    public int getNodeCount() {
        int count=0;
        Node node = headNode.next;
        if (node == null) {
            System.out.println("空链表");
            return count = 0;
        }
        while (node != null) {
            count++;
            node=node.next;
        }
        return count;
    }
}
```

3.测试demo

```
package 线性结构.链表LinkedList.单链表;

/**
 * @author jndeng
 * @create 2019-11-06 12:14
 */
public class Demo {
    public static void main(String[] args) {
        Node node1 = new Node(1, "宋江", "及时雨");
        Node node2 = new Node(2, "卢俊义", "玉麒麟");
        Node node3 = new Node(7, "吴用", "智多星");
        Node node4 = new Node(10, "林冲", "豹子头");
        Node node5 = new Node(5, "戴宗", "神行太保");
        //直接添加到尾部
//        singleLinkedList.add(node1);
//        singleLinkedList.add(node4);
//        singleLinkedList.add(node3);
//        singleLinkedList.add(node2);
//        singleLinkedList.show();
```

```
//        System.out.println("----------------");

        //按照no排序插入
        SingleLinkedList singleLinkedList1 = new SingleLinkedList();
        singleLinkedList1.insertByNo2(node1);
        singleLinkedList1.insertByNo2(node4);
        singleLinkedList1.insertByNo2(node3);
        singleLinkedList1.insertByNo2(node2);
        singleLinkedList1.insertByNo2(node5);
        singleLinkedList1.show();
    }
}
```

4.常见面试题目

1.获取有效节点数量（不包括头结点）

遍历即可

2.查找链表的倒数第k个节点

思路方法：

1. 获取节点数量n
2. 倒数第k个节点，即为n-k个节点

3.单链表反转

思路方法：

1. 定义一个新的链表（头节点）
2. 遍历旧链表，将每个数据依次插入到新的头节点后
3. 旧的头结点取代指向第一个元素

代码实现

```
public static void reverse(Node headNode) {
    Node reverseHead = new Node(0, null, null);
    Node cur = headNode.next;
    if (cur == null) {
        System.out.println("空链表");
        return;
    }
    while (cur != null) {
        Node node=cur.next;
        cur.next=reverseHead.next;
        reverseHead.next=cur;
        cur=node;
    }
    headNode.next=reverseHead.next;
}
```

4. 从尾到头遍历打印单链表

思路方法1：

反转后打印

这样做的问题会破坏原有链表

思路方法2：

利用栈，将各个节点压入栈中，利用先进后出的特点，完成逆序打印

```
/**
 * 通过栈Stack来实现逆序打印
 *
 * @param headNode
 */
public static void reversePrint(Node headNode) {
    Stack<Node> stack = new Stack<>();
    Node node = headNode.next;
    while (node != null) {
        stack.add(node);
        node = node.next;
    }

    while (stack.size() != 0) {
        System.out.println(stack.pop());
    }

}
```

5.合并2个有序单链表，合成后依旧序