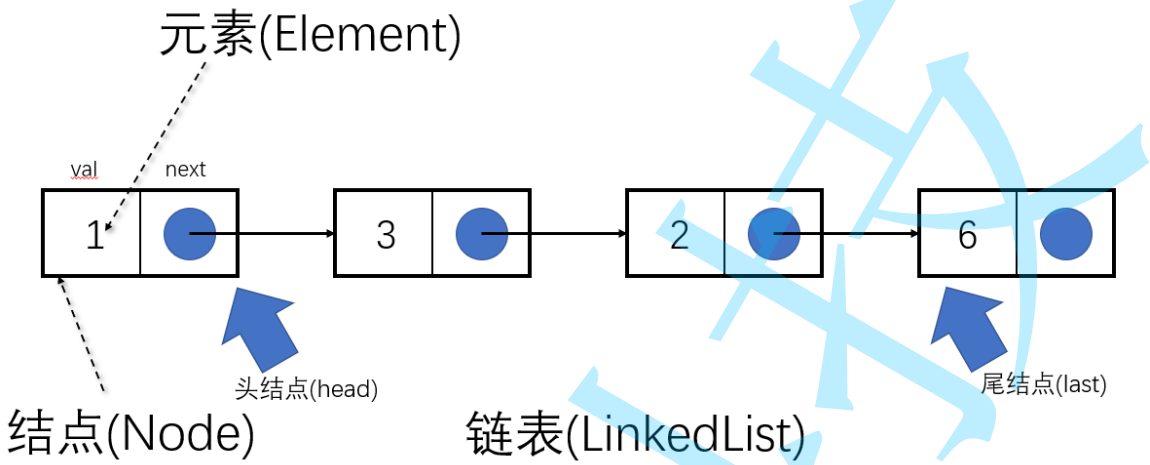
线性表-链表

1. 链表的原理



元素(element): 真实存于线性表中的内容，是我们关心的核心内容。

结点(node): 为了组织链表而引入的一个结构，除了保存我们的元素之外，还会保存指向下一个结点的引用

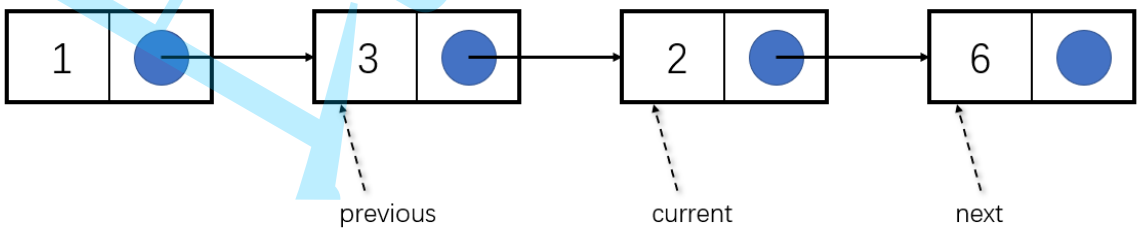
```
class Node {
    int val;    // 保存我们的元素
    Node next;  // 保存指向下一个结点的引用；其中尾节点的 next == null
}
```

链表(linked list): 最终的线性表，表示逻辑上的 [1 3 2 6]

目前，我们通过链表的头结点，来代表一整条链表

```
// head 是一条链表的头结点；通过 head 我们可以找到所有的结点；所以用头节点完全代表链表
Node head = ...;

// 某条链表的头结点是 null，表示头结点不存在。进一步可以表示是一条没有头结点的链表，也就是一条空链表
Node head = null;
```



当前结点(current / cur): 表示链表中某个结点。

前驱结点(previous / prev): 表示链表中某个结点的前一个结点；头结点没有前驱结点。

后继结点(next): 表示链表中某个结点的后一个结点；尾结点没有后继结点。

2. 代码表示 + 画图分析过程

2.1 链表的结点定义

```java
public class Node {
    int val;
    Node next;

    public Node(int val) {
        this.val = val;
        this.next = null;
    }

    @Override
    public String toString() {
        return "Node{" + val + "}";
    }
}
```

2.2 链表的手工创建

创建一个 [1 3 2 6] 的链表

```java
Node n1 = new Node(1);
Node n3 = new Node(3);
Node n2 = new Node(2);
Node n6 = new Node(6);

n1.next = n3;
n3.next = n2;
n2.next = n6;
n6.next = null;

Node head = n1;
```

创建一个空链表

```java
Node head = null;
```

2.3 链表的遍历

遍历之前创建的 [1 3 2 6] 链表

```java
Node cur = head;
while (cur != null) {
    cur = cur.next;
}
```

练习 —— 代码+画图分析：

已知一条链表，不知道其长度的情况下：

通过遍历，打印链表的每个元素。

通过遍历，找到链表的最后一个结点。

通过遍历，找到链表的倒数第二个结点。

通过遍历，找到链表的第 n 个结点。（链表的长度 >= n）

通过遍历，计算链表中元素的个数。

通过遍历，找到链表中是否包含某个元素。

2.4 链表的元素插入和删除

给定前驱结点后的插入

```
Node prev = ...;
Node node = new Node(v);

node.next = prev.next;
prev.next = node;
```

给定前驱结点后的删除

```
Node prev = ...;    // 待删除元素的前驱结点
prev.next = prev.next.next;
```

头插

```
public static void main(String args) {
    Node head = build();
    head = pushFront(head, 0);
    // 通过遍历打印，验证头插是否正确
}

private static Node pushFront(Node head, int v) {
    Node node = new Node(v);
    node.next = head;
    head = node;
    return head;
}
```

头删

```
public static void main(String args) {
    Node head = build();    // 需要考虑以下情况：链表为空；链表中有元素
    head = popFront(head);
    // 通过遍历打印，验证头删是否正确
}

private static Node popFront(Node head) {
    if (head == null) {
        throw new RuntimeException("链表为空");
    }

    head = head.next;
    return head;
}
```

尾插

```java
public static void main(String args) {
    Node head = build();    // 需要考虑以下情况：链表为空；链表中有元素
    head = pushBack(head, 0);
    // 通过遍历打印，验证尾插是否正确
}

private static Node pushBack(Node head, int v) {
    if (head == null) {
        Node node = new Node(v);
        return node;
    }

    Node last = head;
    while (last.next != null) {
        last = last.next;
    }

    Node node = new Node(v);
    last.next = node;

    return head;
}
```

尾删

```java
public static void main(String args) {
    Node head = build();    // 需要考虑以下情况：链表为空；链表中有一个元素；链表中有多个元素
    head = popFront(head);
    // 通过遍历打印，验证头删是否正确
}

private static Node popBack(Node head) {
    if (head == null) {
        throw new RuntimeException("链表为空");
    }

    if (head.next == null) {
        head.next = null;
        return head;
    }

    Node last2 = head;
    while (last2.next.next != null) {
        last2 = last2.next;
    }
    last2.next = null;
    return head;
}
```

练习

```java
// 请将给定数组，转换成链表
public static Node arrayToLinkedList(int[] array);
```

2.5 深入理解引用和对象的知识

```
Node p = ...;   // 已知 p 是一条链表中的某个结点
Node q = ...;   // 已知 q 是一条链表中的某个结点
```

在上述条件的基础上，请画图表示分别执行以下代码后的变化结果

```
p = q;
```

```
p = q.next;
```

```
p.next = q;
```

```
p.next = q.next;
```

请同学加深自己对引用的理解。

3. 链表 OJ 题训练

1. 删除链表中等于给定值 **val** 的所有节点。 OJ链接
2. 反转一个单链表。 OJ链接
3. 给定一个带有头结点 head 的非空单链表，返回链表的中间结点。如果有两个中间结点，则返回第二个中间结点。OJ链接
4. 输入一个链表，输出该链表中倒数第k个结点。 OJ链接
5. 将两个有序链表合并为一个新的有序链表并返回。新链表是通过拼接给定的两个链表的所有节点组成的。OJ链接
6. 编写代码，以给定值x为基准将链表分割成两部分，所有小于x的结点排在大于或等于x的结点之前。OJ链接
7. 在一个排序的链表中，存在重复的结点，请删除该链表中重复的结点，重复的结点不保留，返回链表头指针。 OJ链接
8. 链表的回文结构。OJ链接
9. 输入两个链表，找出它们的第一个公共结点。OJ链接
10. 给定一个链表，判断链表中是否有环。 OJ链接
11. 给定一个链表，返回链表开始入环的第一个节点。 如果链表无环，则返回 null OJ链接
12. 其他 。ps：链表的题当前因为难度及知识面等等原因还不适合我们当前学习，以后大家自己下去以后 Leetcode OJ链接 + 牛客 OJ链接

4. Java 中的链表 —— LinkedList

```
public class LinkedList implements List {
    public LinkedList() { ... }
}
```

LinkedList 的具备的方法，等同于 List 具备的方法。

Java 中的链表，不再使用头节点来代表链表，而是定义了一个链表类，来表示链表。

Java 中的链表采用的是一种双向链表

Java 中的链表既保存了链表的头结点，也保存了链表的尾结点

5. 实现自己的 LinkedList —— MyLinkedList

```java
public class Node {
    String val;
    Node prev;
    Node next;

    public Node(String val) {
        this.val = val;
    }

    @Override
    public String toString() {
        return "Node{" + val + "}";
    }
}
```

```java
// 为了更接近真实的 java.util.LinkedList，我们使用 String 这种引用类型来做为元素类型
// 同时请回答每个方法的时间复杂度是什么。
public class MyLinkedList {
    private Node head;
    private Node last;
    private int size;

    boolean add(String e);

    void add(int index, String e);

    String remove(int index);

    boolean remove(String e);

    String get(int index);

    String set(int index, String e);

    boolean contains(String e);

    int indexOf(String e);

    int lastIndexOf(String e);

    void clear();

    int size();

    boolean isEmpty();
}
```

6. 【面试题】顺序表 vs 链表

顺序表：一白遮百丑

白：空间连续、支持随机访问

丑：1.中间或前面部分的插入删除时间复杂度O(N) 2.增容的代价比较大。

链表：一(黑)毁所有 （双向链表）

黑：以节点为单位存储，不支持随机访问

所有：1.任意位置插入删除时间复杂度为O(1) 2.没有增容问题，插入一个开辟一个空间。