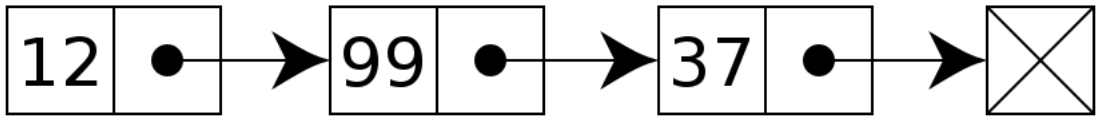
链表 Linked List

无隅

什么是链表

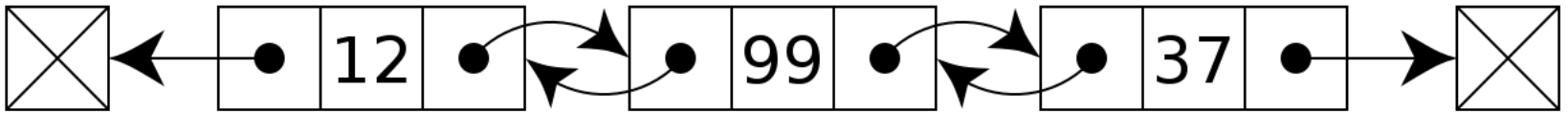
- 链表（Linked list）是一种常见的基础数据结构，是一种线性表，但是并不会按线性的顺序存储数据，而是在每一个节点里存到下一个节点的指针 (Pointer)

- 单链表



一个单向链表包含两个值: 当前节点的值和一个指向下一个节点的链接

- 双向链表



一个双向链表有三个整数值: 数值, 向后的节点链接, 向前的节点链接

链表特性

- 每个节点都知道它下一个节点的地址
- 链表的第一个节点可以代表整个链表
- 查找一个节点或者访问特定编号的节点则需要 O(n) 的时间

链表定义

```java
public class ListNode<T> {
    T data;
    ListNode<T> next;

    public ListNode(T data) {
        this.data = data;
    }
}
```

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
```

为啥要用链表？

- 不确定数据结构的容量时

（1） 数组大小调整的成本非常大，所以我们需要提前设置容量

（2）通常我们不知道我们需要多少空间花费

- 常用于组织**删除、检索较少，而添加、遍历较多**的数据

如何实现链表以及基础操作

```
1
2   public class LinkedList {
3       private ListNode head;
4       //Todo
5       public int get(int index);
6       public void set(int index, int value);
7       public void add(int index, int value);
8       public void remove(int index);
9       public int getLength();
10  }
```

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
```

Dummy Node 哨兵

- 简化边界情况

1. 使得链表原头节点不在特殊

2. 使代码更短，更少的出错

- 链表总是存在至少一个节点，但链表的真正元素是从哨兵节点的下一个节点开始。

链表基本操作总结

- 如果链表的数据结构发生变化，则需要考虑使用 dummy node ，
  例如 add 和 remove 的操作

- 链表节点只能通过前一个节点的指针访问
  在将当前节点分配给新节点之前，请不要更改上一个节点的 next 指针，这样会丢失当前节点

链表高频面试题

- 两类问题

1. 与计数或位置相关的问题
2. 与链表结构变化相关的问题

- 三种武器

1. Dummy node 哨兵节点
2. 链表基本操作（插入，删除，翻转）
3. 双指针

与计数或位置相关的问题

- 得到链表中点元素
- 得到链表的倒数第 N 个节点
- 环形链表 Ⅰ & Ⅱ

得到链表中点元素

给定一个链表，编写一个函数返回链表的中间节点

例：

Input: 1 -> 3 -> 5 -> 7　　return 3

Input: 1 -> 3 -> 5 -> 7 -> 9　return 5

```java
public ListNode getMiddleNode(ListNode head) {
    int length = getLength(head);
    int index = (length - 1) / 2;
    ListNode curNode = head;
    while (index-- != 0) {
        curNode = curNode.next;
    }
    return curNode;
}

private int getLength(ListNode head) {
    ListNode cur = head;
    int length = 0;
    while (cur != null) {
        length++;
        cur = cur.next;
    }
    return length;
}
```

```java
public ListNode getMiddleNode(ListNode head) {
    ListNode fast = head;
    ListNode slow = head;
    while (fast.next != null && fast.next.next != null) {
        fast = fast.next.next;
        slow = slow.next;
    }
    return slow;
}
```

得到链表的倒数第 N 个节点

给定一个链表，得到链表的倒数第 n 个节点并返回。
例如，
给定一个链表 : 1->2->3->4->5, 并且 n = 2. 倒数第两个节点为 4
Input: 1->2->3->4->5, 2
Output: 2
Input: 3->5->9->6->8, 3
Output: 9
说明 :
给定的 n 始终是有效的。
尝试一次遍历实现。

```java
public ListNode getKthToLast(ListNode head, int k) {
    int length = getLength(head);
    int index = length - k;
    ListNode cur = head;
    while (index-- != 0) {
        cur = cur.next;
    }
    return cur;
}
```

```java
public ListNode getKthToLast(ListNode head, int k) {
    ListNode first = head;
    while (k-- != 0) {
        first = first.next;
    }
    ListNode second = head;
    while (first != null) {
        first = first.next;
        second = second.next;
    }
    return second;
}
```

```java
class Index {
    int value = 0;
}

public ListNode getKthToLast(ListNode head, int k) {
    Index index = new Index();
    return kthToLast(head, k, index);
}

private ListNode kthToLast(ListNode head, int k, Index index) {
    if (head == null) {
        return null;
    }

    ListNode node = kthToLast(head.next, k, index);
    index.value = index.value + 1;
    if (index.value == k) {
        return head;
    }
    return node;
}
```

环形链表 I

给定一个链表，判断链表中否有环

（

```
public boolean hasCycle(ListNode head) {
    if (head == null) {
        return false;
    }
    ListNode fast = head;
    ListNode slow = head;
    while (fast != null && fast.next != null) {
        fast = fast.next.next;
        slow = slow.next;
        if (slow == fast) {
            return true;
        }
    }
    return false;
}
```

环形链表 II

给一个链表，返回链表开始入环的第一个节点。 如果链表无环，则返回 null。

说明：不应修改给定的链表

(
https://leetcode.com/problems/linked-list-cycle-ii/description/
)

```java
public ListNode detectCycle(ListNode head) {
    ListNode fast = head, slow = head;
    while (fast != null && slow != null) {
        if (fast.next != null) {
            fast = fast.next.next;
        } else {
            return null;
        }
        slow = slow.next;
        if (fast == slow) {
            ListNode temp = head;
            while (temp != slow) {
                temp = temp.next;
                slow = slow.next;
            }
            return temp;
        }
    }
    return null;
}
```

与链表结构变化相关的问题

- 反转链表 I/II
- 交换相邻结点
- 两数相加
- 删除排序链表中的重复元素 I/II
- 合并两个有序链表

反转链表 I

反转一个单链表。

(
https://leetcode-cn.com/problems/reverse-linked-list/description
)

```
public ListNode reverseList(ListNode head) {
    if (head == null || head.next == null){
        return head;
    }
    ListNode prev = null;
    while (head != null){
        ListNode temp = head.next;
        head.next = prev;
        prev = head;
        head = temp;
    }
    return prev;
}
```

反转链表 II

反转从位置 m 到 n 的链表。用一次遍历在原地完成反转。

例如：
给定 1->2->3->4->5->NULL, m = 2 和 n = 4,

返回 1->4->3->2->5->NULL.

注意：
给定 m， n 满足以下条件：

1 ≤ m ≤ n ≤ 列表长度

（
https://leetcode-cn.com/problems/reverse-linked-list-ii/description/
）

```java
public ListNode reverseBetween(ListNode head, int m, int n) {
    if (m >= n || head == null) {
        return head;
    }

    ListNode dummy = new ListNode(0);
    dummy.next = head;
    head = dummy;

    for (int i = 1; i < m; i++) {
        if (head == null) {
            return null;
        }
        head = head.next;
    }

    ListNode premNode = head;
    ListNode mNode = head.next;
    ListNode nNode = mNode, postnNode = mNode.next;
    for (int i = m; i < n; i++) {
        if (postnNode == null) {
            return null;
        }
        ListNode temp = postnNode.next;
        postnNode.next = nNode;
        nNode = postnNode;
        postnNode = temp;
    }
    mNode.next = postnNode;
    premNode.next = nNode;

    return dummy.next;
}
```

交换相邻结点

给定一个链表，对每两个相邻的结点作交换并返回头节点。

例如：
给定 1->2->3->4，你应该返回 2->1->4->3。

你的算法应该只使用额外的常数空间。不要修改列表中的值，只有节点本身可以更改

(
https://leetcode-cn.com/problems/swap-nodes-in-pairs/description/
)

```java
public ListNode swapPairs(ListNode head) {
    ListNode dummy = new ListNode(-1);
    dummy.next = head;
    ListNode pre = dummy;
    while (pre.next != null && pre.next.next != null) {
        ListNode first = pre.next, second = pre.next.next;
        first.next = second.next;
        second.next = first;
        pre.next = second;
        pre = first;
    }
    return dummy.next;
}
```

两数相加

给定两个非空链表来代表两个非负数，位数按照逆序方式存储，它们的每个节点只存储单个数字。将这两数相加会返回一个新的链表。

可以假设除了数字 0 之外，这两个数字都不会以零开头

例：

输入：(2 -> 4 -> 3) + (5 -> 6 -> 4) 输出：7 -> 0 -> 8 原因：342 + 465 = 807

输入：(7 -> 1 -> 6) + (5 -> 9 -> 2) 输出：2 -> 1 -> 9 原因：617 + 295 = 912

(https://leetcode-cn.com/problems/add-two-numbers/description/)

```java
public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
    int carry = 0;
    ListNode dummy = new ListNode(-1);
    ListNode p = dummy;
    while (l1 != null && l2 != null) {
        int val = (l1.val + l2.val + carry) % 10;
        carry = (l1.val + l2.val + carry) / 10;
        p = appendToTail(p, val);
        l1 = l1.next;
        l2 = l2.next;
    }
    while (l1 != null) {
        int value = (l1.val + carry) % 10;
        carry = (l1.val + carry) / 10;
        p = appendToTail(p,value);
        l1 = l1.next;
    }
    while (l2 != null) {
        int value = (l2.val + carry) % 10;
        carry = (l2.val + carry) / 10;
        p = appendToTail(p,value);
        l2 = l2.next;
    }
    if (carry != 0) {
        p.next = new ListNode(carry);
    }
    return dummy.next;
}

public ListNode appendToTail(ListNode head, int value) {
    head.next = new ListNode(value);
    head = head.next;
    return head;
}
```

删除排序链表中的重复元素 I

给定一个排序链表，删除所有重复的元素使得每个元素只留下一个

例如：

给定 1->1->2，返回 1->2

给定 1->1->2->3->3，返回 1->2->3

(
https://leetcode-cn.com/problems/remove-duplicates-from-sorted-list/description/
)

```java
public ListNode deleteDuplicates(ListNode head) {
    if (head == null) {
        return null;
    }
    ListNode dummy = new ListNode(Integer.MIN_VALUE);
    dummy.next = head;
    head = dummy;
    while (head != null) {
        while (head.next != null && head.next.val == head.val) {
            head.next = head.next.next;
        }
        head = head.next;
    }
    return dummy.next;
}
```

删除排序链表中的重复元素 II

给定一个有序的链表，删除所有有重复数字的节点，只保留原始列表中唯一的数字。

例如：
给定 1->2->3->3->4->4->5 ，则返回 1->2->5
给定 1->1->1->2->3 ，则返回 2->3

(
https://leetcode-cn.com/problems/remove-duplicates-from-sorted-list-ii/description/
)

```java
public ListNode deleteDuplicates(ListNode head) {
    if(head == null || head.next == null) {
        return head;
    }
    ListNode dummy = new ListNode(Integer.MIN_VALUE);
    dummy.next = head;
    head = dummy;

    while (head.next != null && head.next.next != null) {
        if (head.next.val == head.next.next.val) {
            int val = head.next.val;
            while (head.next != null && head.next.val == val) {
                head.next = head.next.next;
            }
        } else {
            head = head.next;
        }
    }
    return dummy.next;
}
```

合并两个有序链表

合并两个已排序的链表，并将其作为一个新列表返回。新列表应该通过拼接前两个列表的节点来完成。

示例：

输入：1->2->4, 1->3->4 输出：1->1->2->3->4->4

(
https://leetcode-cn.com/problems/merge-two-sorted-lists/description/
)

```java
public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
    ListNode dummy = new ListNode(Integer.MIN_VALUE);
    ListNode cur = dummy;
    while (l1 != null && l2 != null) {
        if (l1.val < l2.val) {
            cur.next = new ListNode(l1.val);
            l1 = l1.next;
        } else {
            cur.next = new ListNode(l2.val);
            l2 = l2.next;
        }
        cur = cur.next;
    }
    while (l2 != null) {
        cur.next = new ListNode(l2.val);
        l2 = l2.next;
        cur = cur.next;
    }
    while (l1 != null) {
        cur.next = new ListNode(l1.val);
        l1 = l1.next;
        cur = cur.next;
    }
    return dummy.next;
}
```

总结

- 链表概念 - 链表实现与基本操作
- 常用技巧： Dummy Node 哨兵节点
- 链表高频面试题
  - 两类问题
  1. 与计数或位置相关的问题
  2. 与链表结构变化相关的问题
  - 三种武器
  1. Dummy node
  2. 链表基本操作（插入，删除，翻转）
  3. 双指针