剑指Offer 24 反转链表

Label：链表 双指针 递归

反转一个单链表。

```
输入: 1->2->3->4->5->NULL
输出: 5->4->3->2->1->NULL
```

- List（用栈也是一样）

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
class Solution {
    public ListNode reverseList(ListNode head) {
        if (head == null || head.next == null) {
            return head;
        }

        List<ListNode> list = new ArrayList<>();
        while (head != null) {
            list.add(head);
            head = head.next;
        }

        for(int i = list.size() - 1; i > 0 ; i--) {
            list.get(i).next = list.get(i-1);
        }

        list.get(0).next = null;
        return list.get(list.size()-1);
    }
}
```

- 迭代

```
class Solution {
    public ListNode reverseList(ListNode head) {
        // 特殊判断
        if (head == null || head.next == null) {
            return head;
        }

        ListNode curr = head; // 1
        ListNode prev = null; // 保存新链表的 next

        while (curr != null) {
            ListNode next = curr.next; // next = 2    保存原链表的next
            curr.next = prev; // 1->null    反转

            prev = curr; // prev = 1    将当前结点置位 pre
            curr = next; // curr = 2    将下一轮的curr置位之前保存的原链表的nxext
        }
        return prev;
    }
}
```

- 递归

```
class Solution {
    public ListNode reverseList(ListNode head) {

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

        ListNode  reverseHead = reverseList(head.next);  // 要一直返回，这个值从最底层开始就是不变的

        head.next.next = head;
        head.next = null;  // 防止 顶层 出现 cycle

        return reverseHead;
    }
}
```

- 容器 API 反转（再修改指针）

```
class Solution {

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

            ArrayList<ListNode> arrayList = new ArrayList<>();

            ListNode curr = head;

            // 1.加到数组中
            while (curr != null) {
                arrayList.add(curr);
                curr = curr.next;
            }

            // 2.利用容器自身API反转（不反转，倒序遍历list也可以）
            Collections.reverse(arrayList);

            // 3.再次遍历修改指针
            int len = arrayList.size();
            for (int i = 0; i < len; i++) {
                ListNode currNode = arrayList.get(i);
                if (i < len - 1)
                    currNode.next = arrayList.get(i + 1);
                else
                    currNode.next = null;
            }

            //返回反转之后的头指针
            return arrayList.get(0);
        }

}
```

- 双端队列（遍历直接创建新节点）

```
class Solution {

    public ListNode reverseList(ListNode head) {
        if (head == null || head.next ==  null) {
            return head;
        }
        // 创建一个双端队列
        Deque<ListNode> tempDeque = new ArrayDeque<>();

        ListNode curNode = head;
        while (curNode != null) {
            tempDeque.addLast(curNode);
            curNode = curNode.next;
        }

       // 创建一个ListNode对象保存需要返回的链表
        ListNode resNode = new ListNode(0);
        ListNode newHead = resNode;
        // 遍历 创建新节点
        while (tempDeque.size() != 0) {
            resNode.next = new ListNode(tempDeque.pollLast().val); // 直接创建新节
点
            // 将resNode指针往后移动
            resNode = resNode.next;
        }
        return newHead.next;
    }
}
```