LeetCode 第 445 号问题：两数相加 II

> *本文首发于公众号「图解面试算法」，是 图解 LeetCode 系列文章之一。*
>
> *同步个人博客：www.zhangxiaoshuai.fun*

难度：medium　　目前通过率：57.2%

题目描述：

给你两个非空链表来代表两个非负整数。数字最高位位于链表开始位置。
它们的每个节点只存储一位数字。将这两数相加会返回一个新的链表。
你可以假设除了数字0之外，这两个数字都不会以零开头。

示例：

输入：(7 -> 2 -> 4 -> 3) + (5 -> 6 -> 4)
输出： 7 -> 8 -> 0 -> 7

显然，给定的两个“数字”都是**从左到右按照高位到低位的顺序**排列的，按照习惯我们需要将链表反转之后然后相加，再将结果链表反转之后返回。

那么顺序的问题解决了，还存在一个问题：

两个一位数相加是有可能产生一个两位数的，而我们的节点中是不能存储一个两位数字的，如果产生进位那我们需要将进位保存下来，然后将进位加到后面的两个数字相加的结果中。

下面是GIF图解：

 解法1

那么我们先来试试写出这个版本的代码：

因为在整个计算中，链表反转的这个功能用到了三次，那我们可以将这个功能单独的抽取出来写一个方法。(方法比较简单，直接附上代码，不再赘述)

```
//链表反转
private ListNode reverseListNode(ListNode head){
    if(head == null) return null;
    ListNode prev = null;
    while (head != null) {
        ListNode next = head.next;
        head.next = prev;
        prev = head;
        head = next;
    }
    return prev;
}
```

要返回两条链表相加的结果，我们需要新的链表来存储这个结果，dummy节点的值随意，因为我们最终返回的是连接在它后面的聊链表。

我们需要初始化**addOne**（代表进位）为0

```
//两条链表中的各个节点数字相加
private ListNode add(ListNode l1, ListNode l2){
    if (l1 == null && l2 == null) return null;
    ListNode dummy = new ListNode(0);
    ListNode temp = dummy;
    int addOne = 0;
    while (l1 != null || l2 != null || addOne != 0) {
        int val1 = l1 == null ? 0 : l1.val;
        int val2 = l2 == null ? 0 : l2.val;
        int sum = val1 + val2 + addOne;
        temp.next = new ListNode(sum%10);
        temp = temp.next;
        addOne = sum / 10;
        if (l1 != null) l1 = l1.next;
        if (l2 != null) l2 = l2.next;
    }
    return dummy.next;
}
```

ok，那我们现在只需要在指定的方法中调用这两个方法并传递相应的参数即可得到我们想要的。

```
public static ListNode addTwoNumbers(ListNode l1, ListNode l2){
    ListNode node1 = reverseListNode(l1);
    ListNode node2 = reverseListNode(l2);
    return reverseListNode(add(node1,node2));
}
```

如果你觉得上面的解法太过简单，那一起来看看进阶版本吧：

如果输入链表不能修改该如何处理？换句话说，你不能对列表中的节点进行翻转。 那我们上面的使用的“反转链表”的方法就无法再次使用了。 首先我们应该明确“反转链表”方法的出现是为了解决链表中“数字”的顺序问题，那么如果要得到正确的数字顺序只有这一中方法吗？ 我们还有可以依靠的数据结构：**栈**。栈的特点就是“**先进后出**”

算法思路：先将两条链表的各个数字分别压入两个栈中，然后每次弹出两个栈顶的数字进行相加，第一次相加的结点指向null结点，每次更新头节点，后面的结点依次指向新链表的头节点，那么最终我们可以得到与上面解法相同的结果。

解法2图解：

 解法2

代码：

```
public static ListNode addTwoNumbers(ListNode l1, ListNode l2){
    Stack<Integer> s1 = new Stack<>();
    Stack<Integer> s2 = new Stack<>();
    while (l1 != null) {
      s1.push(l1.val);
      l1 = l1.next;
    }
```

```
  while (l2 != null) {
    s2.push(l2.val);
    l2 = l2.next;
  }
  int addOne = 0;//进位
  ListNode head = null;
     while (!s1.isEmpty() || !s2.isEmpty() || addOne != 0) {
    int sum = addOne;
    sum += s1.isEmpty() ? 0 : s1.pop();
    sum += s2.isEmpty() ? 0 : s2.pop();
    ListNode temp = new ListNode(sum%10);
    temp.next = head;
    head = temp;
    addOne = sum/10;
  }
  return head;
}
```