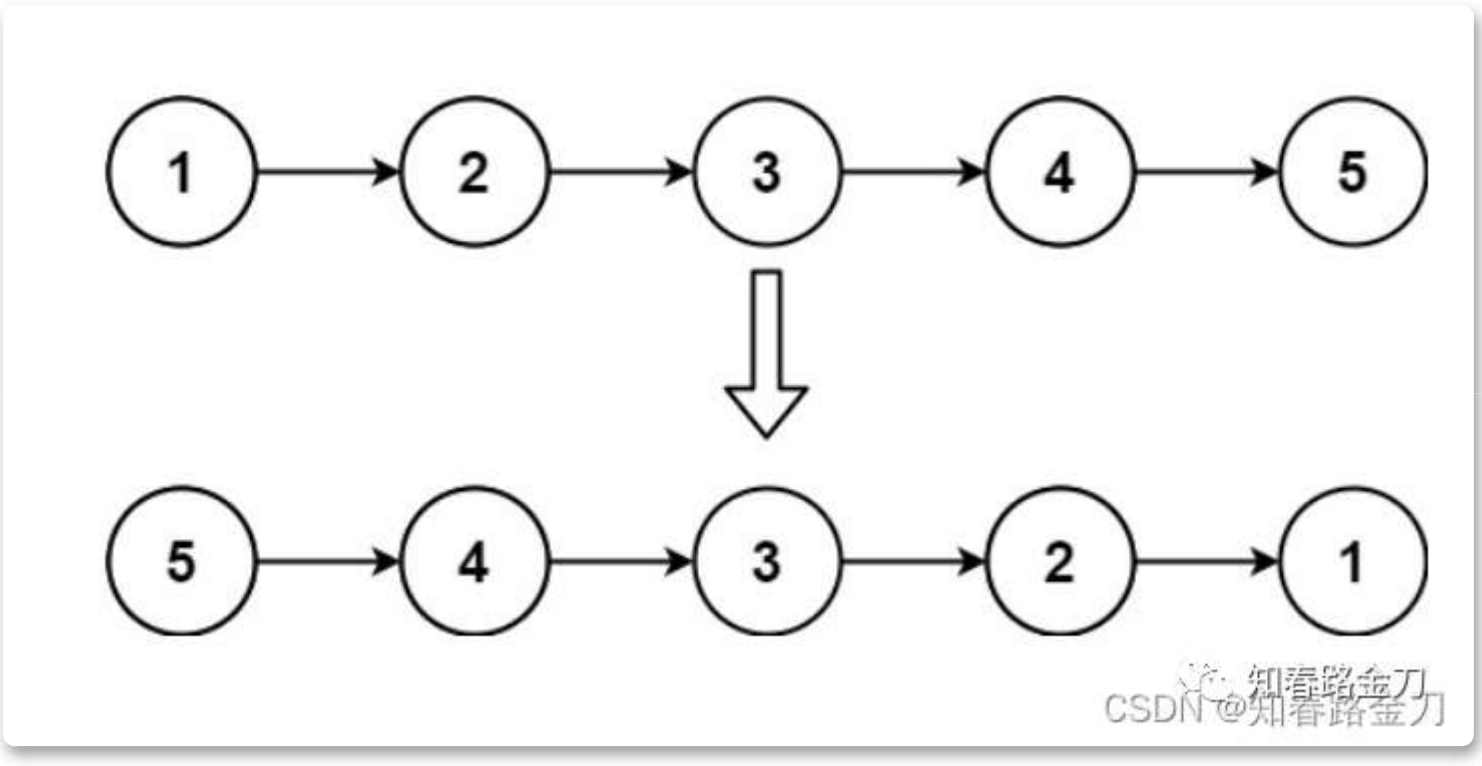
> PS：这道题目估计是很多刷leetcode小伙伴们做的第一次，别以为这个题很简单而且是esay的就不重视它。金刀之前在大厂面试中真的遇到过这个问题，我都惊呆了竟然出这个题，说来惭愧，我竟然没写对！焯！所以大家还是要多多练习，掌握好基本功才能拿下好offer。

题目介绍[1]

给你单链表的头节点 head ，请你反转链表，并返回反转后的链表。

示例1



> 输入：head = [1,2,3,4,5]
>
> 输出：[5,4,3,2,1]

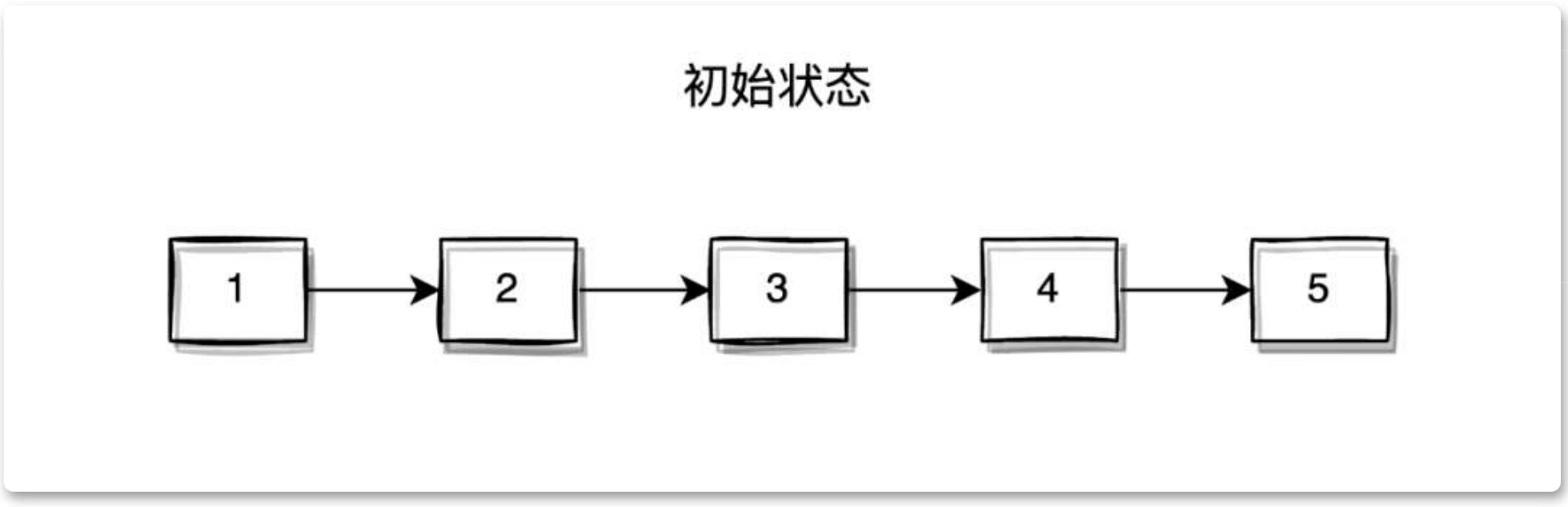
题目解答

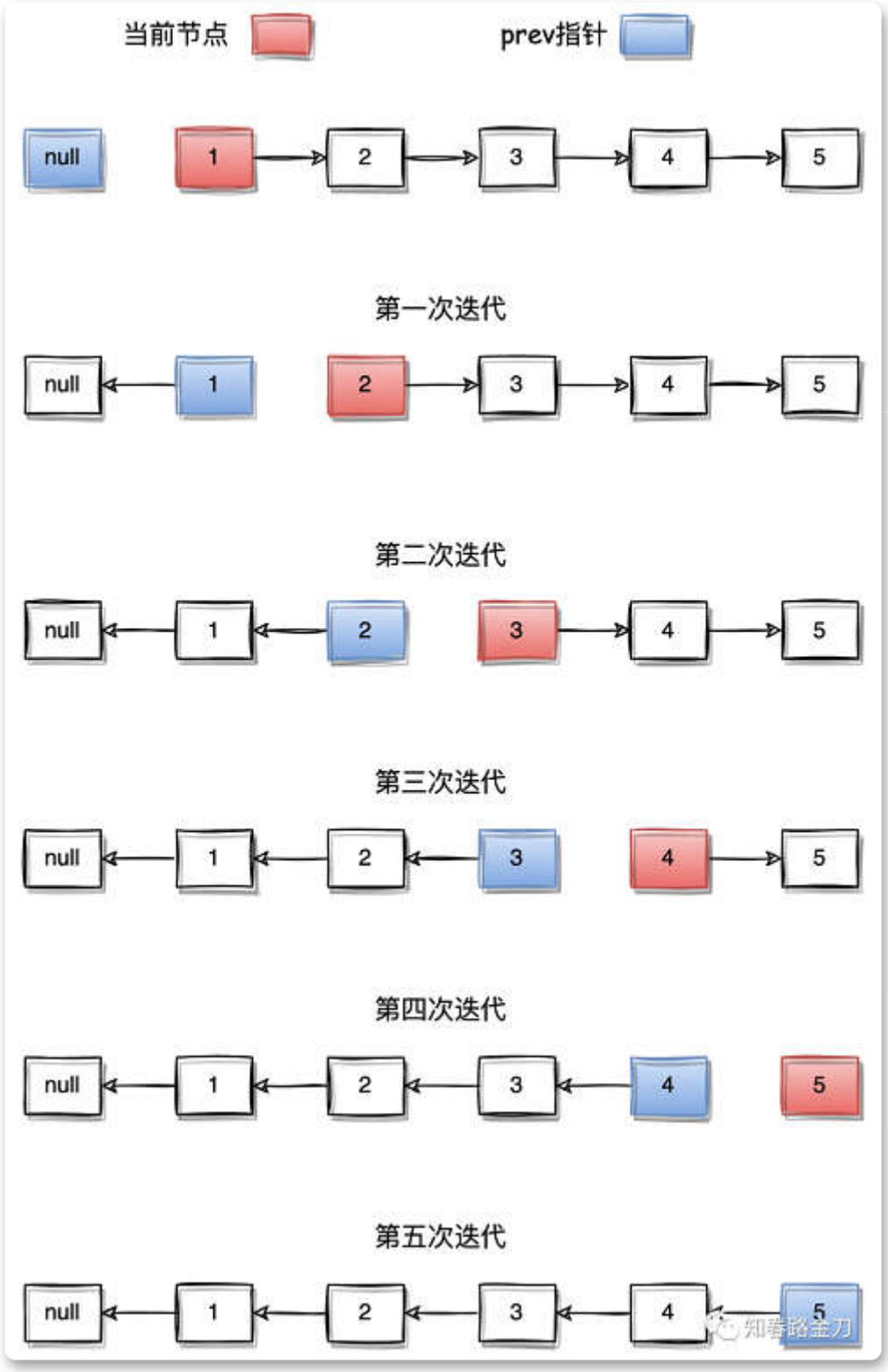
方法一：迭代法

思路和算法

基本思路大家应该很容易想出来，每遍历一个节点的时候，把当前节点的next指针指向前一个节点，在指向前一个节点之前需要先临时存储当前节点的next节点。

图解





代码实现

```
class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode prev = null;
        while (head != null) {
            ListNode t = head.next;
            head.next = prev;
            prev = head;
            head = t;
        }
        return prev;
    }
}
```

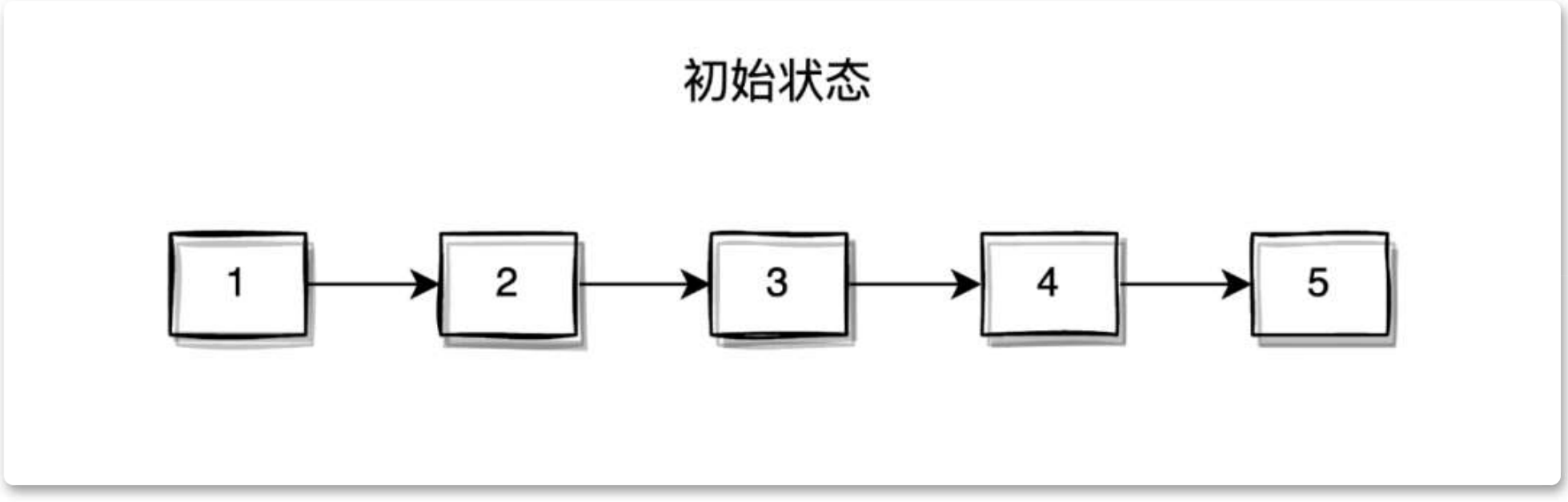
复杂度分析

- 时间复杂度：O(n)，其中 n 是链表的长度。需要遍历链表一次。
- 空间复杂度：O(1)。

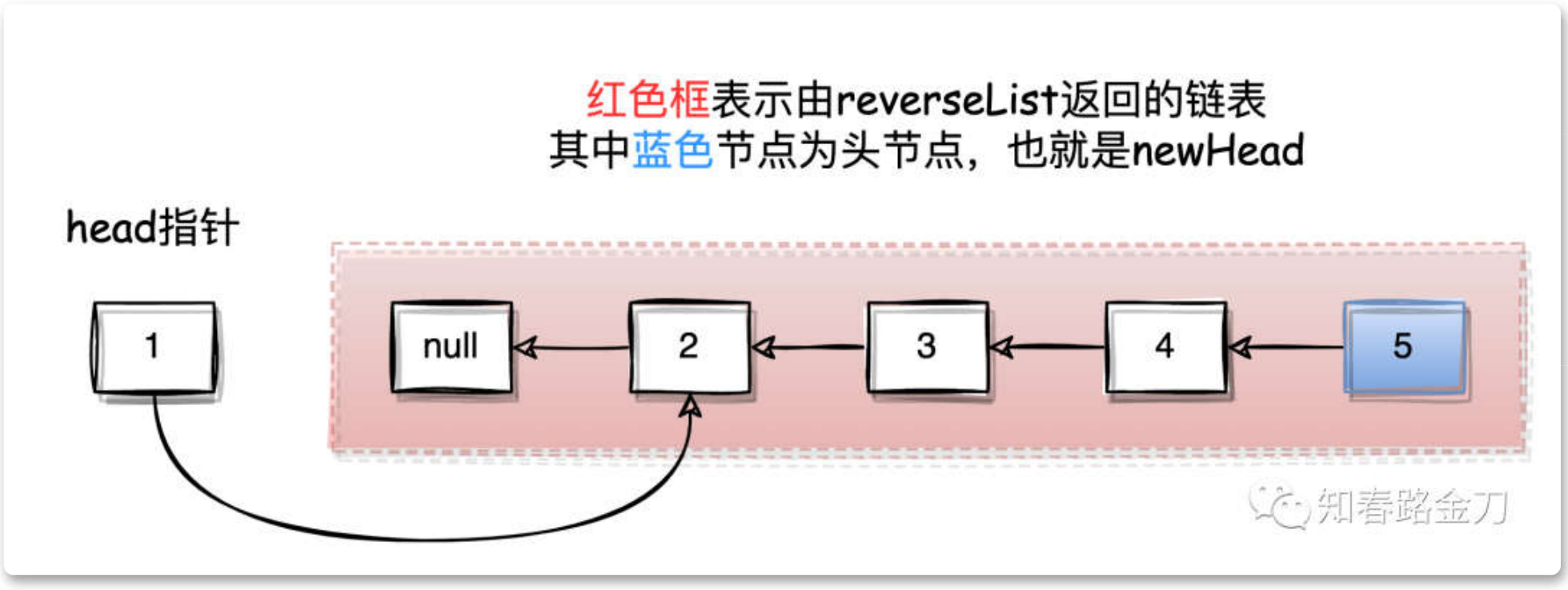
方法二：递归法

思路和算法

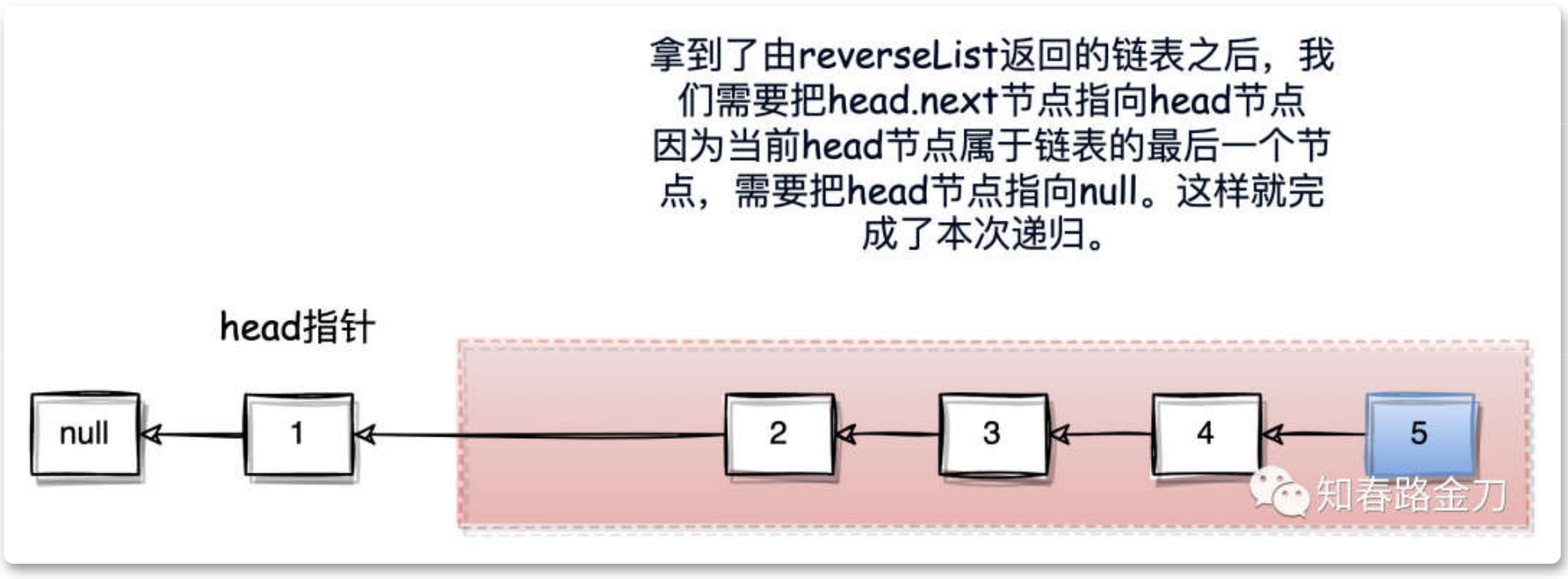
首先我们改如何理解递归呢？我们假设reverseList函数可以把一个链表进行反转。



红色框表示由reverseList递归返回的链表 其中蓝色节点为头节点，也就是newHead



拿到了由reverseList返回的链表之后，我们需要把head.next节点指向head节点。因为当前head节点属于链表的最后一个节点，需要把head节点指向null，这样就完成了本次递归。



另外需要注意一下递归的终止条件，如果head == null 或者 head.next == null 递归就终止了。

代码实现

```
class Solution {
    public ListNode reverseList(ListNode head) {
        if(head == null || head.next == null) {
            return head;
        }
        ListNode newHead = reverseList(head.next);
        head.next.next = head;
        head.next = null;
        return newHead;
    }
}
```

复杂度分析

- 时间复杂度：O(n)，其中 n 是链表的长度。需要对链表的每个节点进行反转操作。
- 空间复杂度：O(n)，其中 n 是链表的长度。空间复杂度主要取决于递归调用的栈空间，最多为 n 层。

参考资料

[1] **原题链接:**
https://leetcode-cn.com/problems/reverse-linked-list/