# 题目

反转一个单链表。

**示例:**

输入: 1->2->3->4->5->NULL

输出: 5->4->3->2->1->NULL

**进阶:**

你可以迭代或递归地反转链表。你能否用两种方法解决这道题？

# 分析

## 方法一：迭代法/双指针

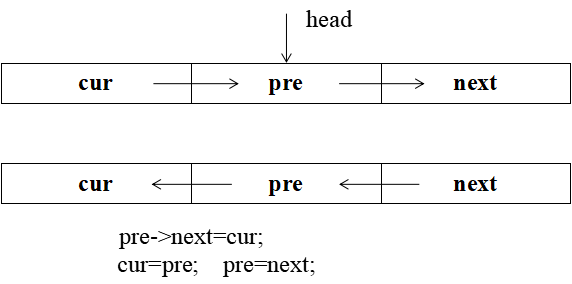
**思路：**

1、定义两个指针：pre和cur；pre在前cur在后。

2、每次让pre的next指向cur，实现一次局部反转

3、局部反转完成之后，pre和cur同时往前移动一个位置

4、循环上述过程，直至pre到达链表尾部



**代码：**

class Solution {

public:

ListNode\* reverseList(ListNode\* head) {

ListNode\* cur = NULL, \*pre = head;

while (pre != NULL) {

ListNode\* next = pre->next; //暂存下一个节点

pre->next = cur;

cur = pre; //更新pre和cur（先更新cur，然后pre，从前往后）

pre = next;

}

return cur;

}

};

或：

public ListNode reverseList(ListNode\* head) {

ListNode\* prev = null;

ListNode\* curr = head;

while (curr != null) {

ListNode nextTemp = curr->next;

curr->next = prev;

prev = curr;

curr = nextTemp;

}

return prev;

}

**复杂度分析：**

时间复杂度：O(n)，假设n是列表的长度，时间复杂度是O(n)。

空间复杂度：O(1)。

注：当然也可以采用虚节点的方式（这样需要额外申请一个节点）。

class Solution {

public ListNode reverseList(ListNode head) {

// 虚拟头，用于存放反转结果

ListNode result = new ListNode(0);

ListNode item = head;

ListNode next;

while(true){

if(item == null){

break;

}

// 为了保证循环正常后移，先把下一个元素临时取出来

next = item.next;

// 因为需要移动到result链表中最前一个，所以我们需要把result的next节点，赋值给最新元素的item

item.next = result.next;

// 然后再把item插入到result当中

result.next = item;

// 元素后移继续循环

item = next;

}

// 由于我们使用的虚拟头，第一个元素是无效的，直接取下一个节点就行了

return result.next;

}

}

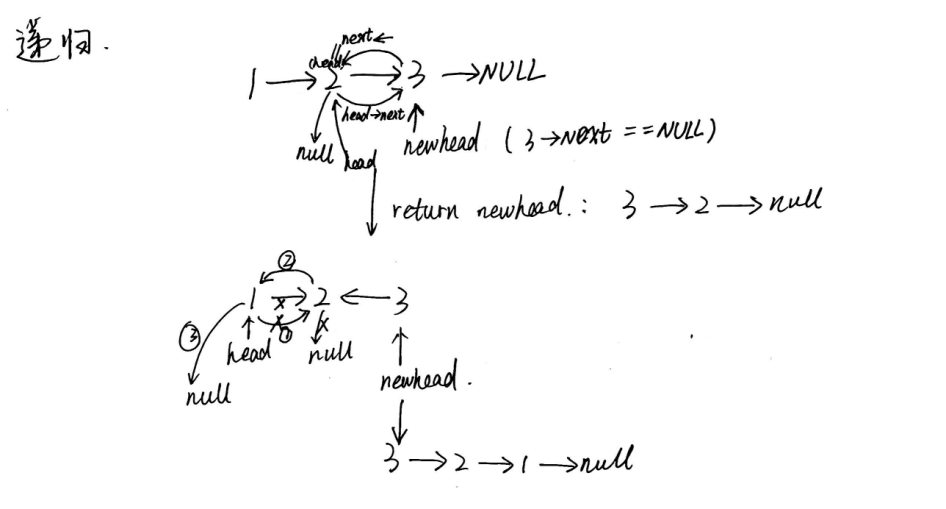
## 方法二：递归法

**思路：**

1、递归终止条件：(head == NULL) || (head->next==NULL)

2、递归对象：head->next；

3、递归过程：head->next->next = head; head->next = NULL;



**代码：**

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

ListNode\* reverseList(ListNode\* head) {

if((head == NULL) || (head->next==NULL)){

return head;

}

ListNode \*newhead;

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)，假设n是列表的长度，那么时间复杂度为O(n)。

空间复杂度：O(n)，由于使用递归，将会使用隐式栈空间。递归深度可能会达到n层。

或：

ListNode\* reverseList(ListNode\* head) {

if(!head){

return nullptr;

}

return reverse(head, head, head->next);

}

ListNode\* reverse(ListNode\* head, ListNode\* first, ListNode\* target){

if(!target){

return head;

}

first->next = target->next;

ListNode\* temp = target->next;

target->next = head;

return reverse(target, first, temp);

}

}