Reverse Linked List

Created	@2025年5月11日 上午1:00
Question Type	Linked List
Difficulty	Easy
LeetCode Question Link	https://leetcode.com/problems/reverse-linked-list/description/
Need to Redo?	No

1. Question Self-understanding:

1.1 Description:

We need to reverse a given singly linked list. Specifically, we take a linked list and transform it so that the head node becomes the last node, and all pointers are reversed.

1.2 Input:

The input is the first node (head) of the linked list. It could be `None` (an empty list).

1.3 Input Assumption

- A linked list node is defined as follows:

```
class ListNode:
    def __init__(self, val=0, next=None):
```

```
        self.val = val
        self.next = next
```

1.4 Output:

- Return the new head of the reversed linked list. If the list is empty (head is `None`), return `None`.

1.5 Example:

```
Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]
```

1.6 Other Q&A:

- None for this question.

2. Attempt 1:

2.1 Thought:

- We can iterate through the linked list using a pointer, and for each node, we redirect its `next` pointer to the previous node.
- By the time we reach the end of the list, the last node we process will become the new head (because all pointers are reversed).

2.2 Pseudo-Code: (Ignore this part. It's a draft for brainstorming.)

```
REVERSE-LINKED-LIST(head)
1  prev ← NIL
2  curr ← head
3  while curr ≠ NIL
4      next ← curr.next
5      curr.next ← prev
6      prev ← curr
7      curr ← next
8  return prev
```

2.3 Implementation through python:

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:

        curr = head
        prev = None
        while curr is not None:

            curr.next, prev, curr = prev, curr, curr.next

        return prev
```

2.4 Time Complexity and Space Complexity

2.4.1 Time Complexity:

- We visit each node exactly once, so the time complexity is $O(n)$.

2.4.2 Space Complexity:

- No additional data structure is used (we only store a few pointers), so the space complexity is $O(1)$.