LeetCode 例题精讲 | 01 反转链表：如何轻松重构链表

原创 nettee 面向大象编程 2019-12-26

收录于话题
#LeetCode 例题精讲

23个

本期例题：**LeetCode 206 - Reverse Linked List**[1]（Easy）

> 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

反转链表这道题是我在阿里的面试中遇到的题目。它本身也是单链表题目中非常典型的一道，不少题目的解法以反转链表为基础。这篇文章将会包含：

- 链表类题目的注意点
- 链表遍历的基本框架
- 本期例题：反转链表的解法
- 相关题目

链表类题目的注意点

在面试中涉及到的链表类题目，一定都是单链表。虽然实际中双向链表使用较多，但单链表更适合作为面试题考察。

单链表这样一个相对“简陋”的数据结构，实际上就是为了考察面试者指针操作的基本功。很多题目需要修改指针链接，如果操作不当，会造成链表结点的丢失，或者出现错误的回路。

我们早在 C/C++ 编程课上就学过链表数据结构。你一定对各种链表的变体印象深刻，单链表、双链表、循环链表……但是在面试中，请忘记你记得的各种花哨样式，只使用最简单的单链表操作。面试官很可能不希望看到你的各种“奇技淫巧”：

- 加入哑结点（即额外的链表头结点）可以简化插入操作，但面试官通常会要求你不要创建额外的链表结点，哑结点显然也属于额外的结点。
- 使用 C/C++ 的二级指针可以让删除结点的代码非常精简，但如果面试官对此不熟悉的话，会看得一头雾水。

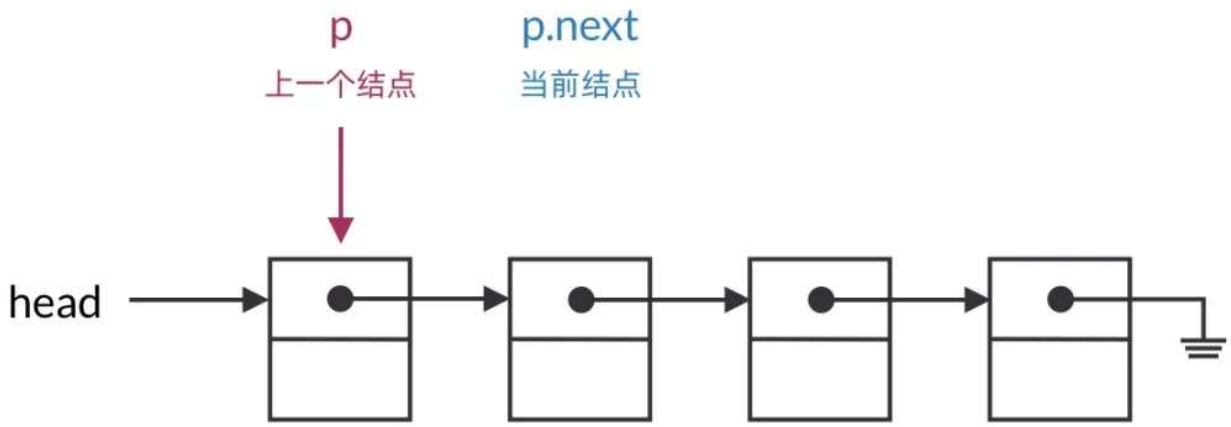
那么，如何才能简洁明了地解决单链表问题呢？实际上很多链表题目都是类型化的，都可以归结为链表的遍历，以及在遍历中做插入和删除操作。我们可以使用链表遍历的框架来解题。

链表遍历的基本框架

单链表操作的本质难度在哪里？相比于双向链表，单链表缺少了指向前一个结点的指针，所以在删除结点时，还需要持有前一个结点的指针。这让遍历过程变得麻烦了许多。

比较容易想到的方法是将遍历的指针指向“前一个结点”，删除结点时使用 `p.next = p.next.next` 。但这个方法会带来一些心智负担：

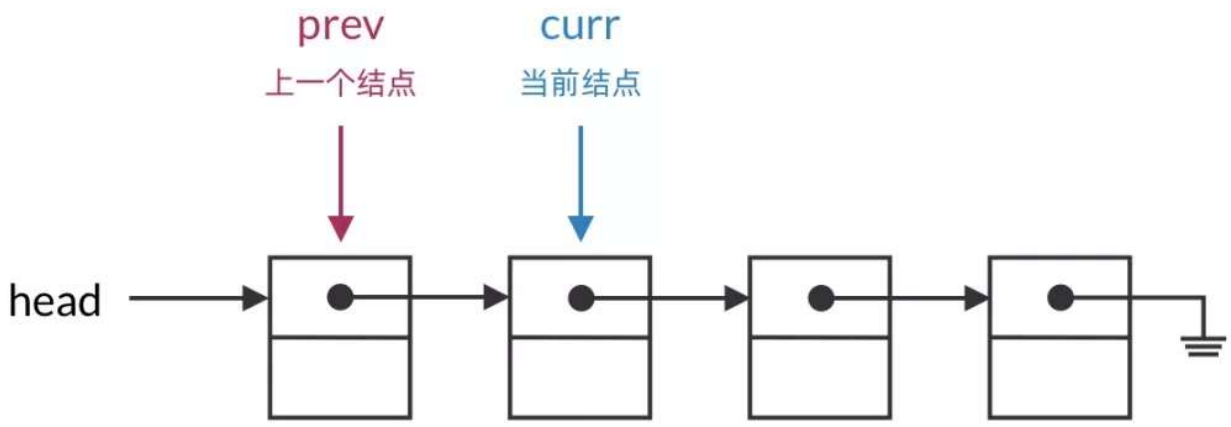
- 每次要查看的结点是 `p.next` ，也就是下一个结点，别扭
- 循环终止条件不是 `p == null` 而是 `p.next == null` ，容易出错



不是很好的链表遍历方式，有一定心智负担

实际上，这就是单链表操作的复杂性所在。我们前面也否定了使用二级指针这样的高级技巧来简化操作的方法，那么，有没有更简单明了的遍历方式呢？答案是有的。这里隆重推荐我一直在使用的链表遍历框架：

当删除链表结点时，既需要访问当前结点，也需要访问前一个结点。既然这样，我们不妨使用两个指针来遍历链表， `curr` 指针指向当前结点， `prev` 指针指向前一个结点。这样两个指针的语义明确，也让你写出的代码更易理解。



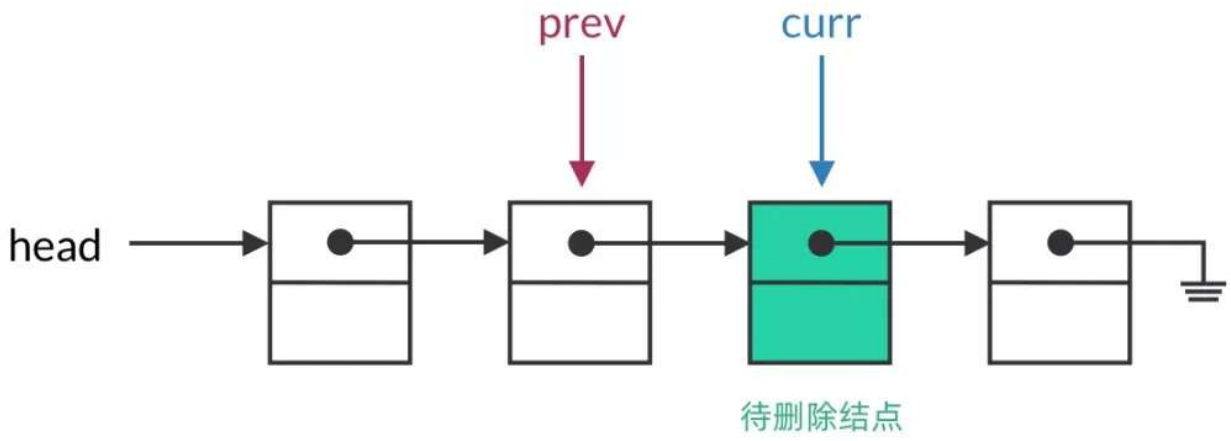
更好的链表遍历框架，指针意义清晰易懂

用代码写出来，链表遍历的框架是这样的：

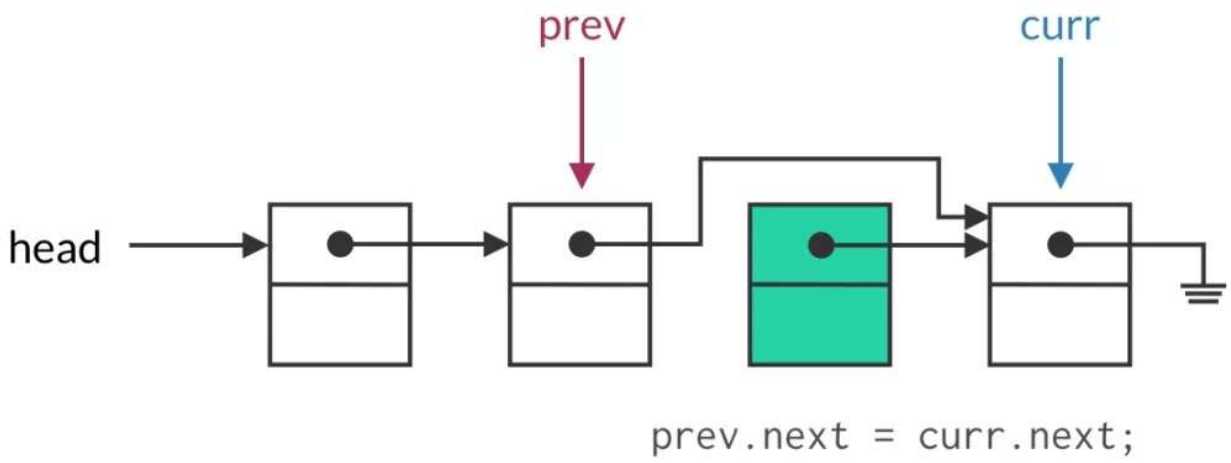
```
ListNode prev = null;
ListNode curr = head;
while (curr != null) {
    // 进行操作，prev 表示前一个结点，curr 表示当前结点
    if (prev == null) {
        // curr 是头结点时的操作
    } else {
        // curr 不是头结点时的操作
    }
    prev = curr;
    curr = curr.next;
}
```

在遍历的过程中，需要一直维护 `prev` 是 `curr` 的前一个结点。 `curr` 是循环中的主指针，整个循环的起始和终止条件都是围绕 `curr` 进行的。 `prev` 指针作为辅助指针，实际上就是记录 `curr` 的上一个值。

在每一轮遍历中，可以根据需要决定是否使用 `prev` 指针。注意 `prev` 可能为 null（此时 `curr` 是头结点），在使用前需要先进行判断。



使用两个指针让删除结点非常容易：待删除

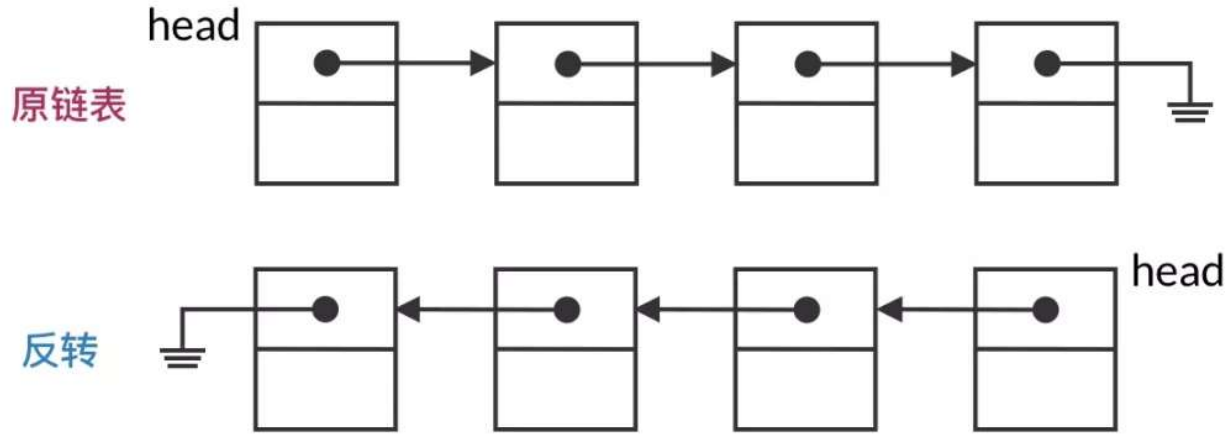


使用两个指针让删除结点非常容易：已删除

下面，我们看一看如何用这个链表遍历的框架来解决本期的例题：反转链表。

本期例题：反转链表的解法

反转链表的题目会有一个隐藏的要求：不能创建新的链表结点，只是在原有结点上修改链表指针。这样的原地操作会比生成一个新的链表要难很多。



反转链表的目标：链表结点不变，修改链表指针

Step 1 套用框架

这道题实际上就是一个典型的链表的遍历-处理的操作，于是我们套用使用上面所讲的链表遍历框架。要反转链表，实际上就是要反转所有相邻结点之间的指针。那么，整体的代码框架应该是：

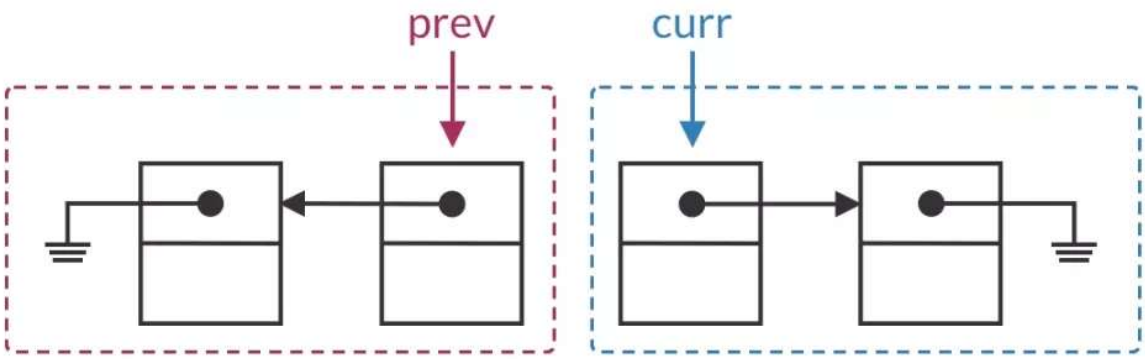
```
ListNode prev = null;
ListNode curr = head;
while (curr != null) {
    // 反转 prev 和 curr 之间的指针
    prev = curr;
    curr = curr.next;
}
```

可以看到，遍历的框架已经将整体的思路架构了出来，我们知道按照如此的方式一定能遍历到所有相邻的结点对，也知道遍历结束后循环一定能正常退出。接下来只需要关注每一步如何反转结点之间的指针即可。

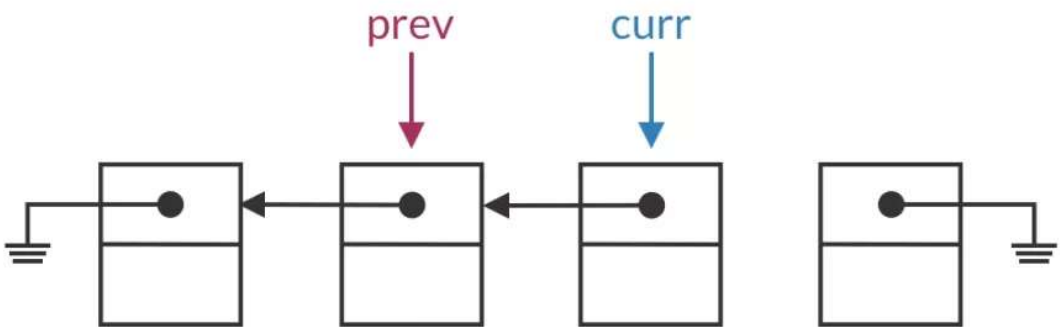
Step 2 写好单步操作

单步操作是“反转 `prev` 和 `curr` 之间的指针”。这里涉及到指针指向的改变，需要小心指针丢失的问题。在思考的时候，要考虑到和前一步、后一步的链接。

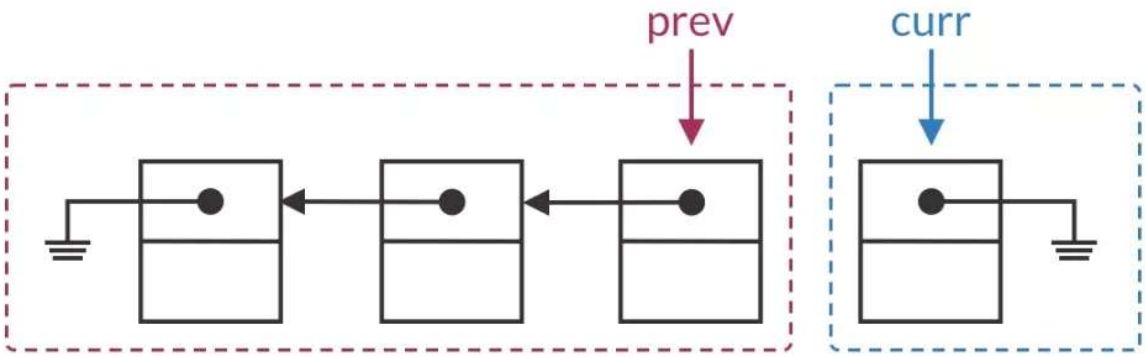
假设现在遍历到了链表中部的某个结点。链表应该会分成两个部分： `prev` 指针之前的一半链表已经进行了反转； `curr` 之后的一半链表还是原先的顺序。这次循环将让 `curr` 的指针改为指向 `prev` ，就将当前结点从后一半链表放进了前一半链表。



循环开始时，prev 和 curr 分别指向链表的前半部分和后半部分

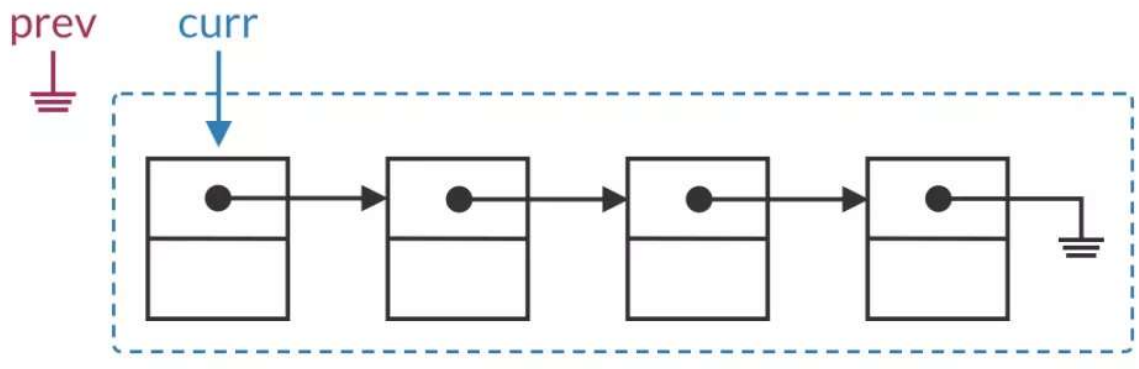


将当前结点放入前一半链表

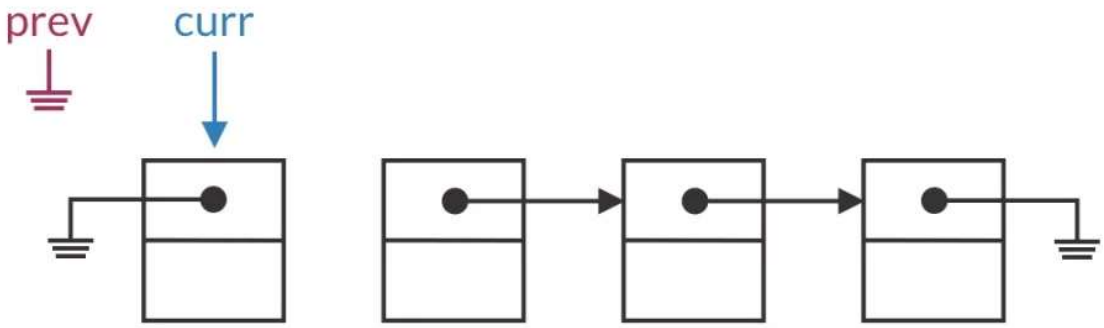


下一轮循环时，prev 和 curr 仍然分别指向链表的前半部分和后半部分

而头结点的特殊情况是，全部链表都还未进行反转，即前一半链表为空。显然 `curr.next` 应当置为 null。



当前结点为头结点时，前一半链表为空



将 curr.next 置空，当前结点成为前一半链表的唯一结点

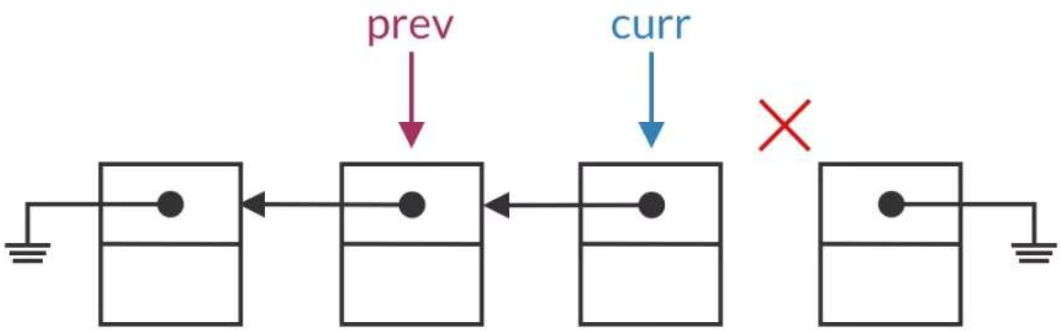
将单步操作放入代码框架，我们就得到了一份初版的解题代码：

```
ListNode prev = null;
ListNode curr = head;
while (curr != null) {
    if (prev == null) {
        curr.next = null;
    } else {
        curr.next = prev;
    }
    prev = curr;
    curr = curr.next;
}
```

Step 3 细节处理

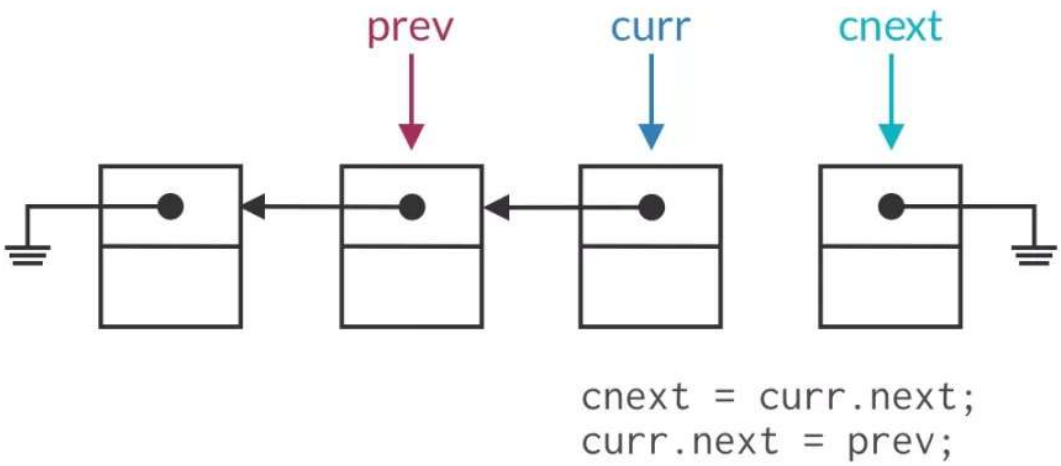
上面的代码已经基本上比较完整了，但是还存在着明显的错误，那就是存在指针丢失的问题。

我们使用 `curr.next = prev` 来反转指针，但这会覆盖掉 `curr.next` 本来存储的值。丢掉这个指针之后，链表的后续结点就访问不到了！



直接赋值 curr.next 是错误的，我们会丢掉指向下一个结点的指针

要解决指针丢失的问题也很简单，使用一个临时指针保存 `curr` 的下一个结点即可。如下图所示：



使用临时指针保存下一个结点，避免指针丢失问题

不过这样一来，我们遍历框架中更新指针的操作也要随之进行微调。框架本来就不是一成不变的，需要根据实际题目灵活调整。

根据以上两点的细节处理，我们修改得到完整版的代码：

```
ListNode reverseList(ListNode head) {
    ListNode prev = null;
    ListNode curr = head;
```

```
        while (curr != null) {
            ListNode cnext = curr.next;
            if (prev == null) {
                curr.next = null;
            } else {
                curr.next = prev;
            }
            prev = curr;
            curr = cnext;
        }
        return prev;
    }
```

上述代码中，if 的两个分支实际上可以优化合并，这里为了清晰起见仍然保留分支。

总结

总结起来，我们解决这一类单链表问题时，遵循的步骤是：

1. 判断问题是否为链表遍历-修改，套用链表遍历框架
2. 思考单步操作，将代码加入遍历框架
3. 检查指针丢失等细节

有很多更复杂的链表题目都以反转链表为基础。下面列出了 LeetCode 上几道相关的题目：

- **LeetCode 203 - Remove Linked List Elements**[2] 是一道直白的链表删除题目
- **LeetCode 445 - Add Two Numbers II**[3] 以反转链表为基础解题
- **LeetCode 92 - Reverse Linked List II**[4] 反转部分链表

希望本文的讲解能让你在写链表类题目时更得心应手。

参考资料

[1] LeetCode 206 - Reverse Linked List: *https://leetcode.com/problems/reverse-linked-list/*

[2] LeetCode 203 - Remove Linked List Elements: *https://leetcode.com/problems/remove-linked-list-elements/*

[3] LeetCode 445 - Add Two Numbers II: *https://leetcode.com/problems/add-two-numbers-ii/*

[4] LeetCode 92 - Reverse Linked List II: *https://leetcode.com/problems/reverse-linked-list-ii/*

喜欢此内容的人还喜欢

难顶，要出差成都网易了！

findyi

大疆自动驾驶，首次官宣即交货

量子位

牵手动视暴雪电竞，进击的哔哩哔哩电竞

GameLook