203-移除链表元素

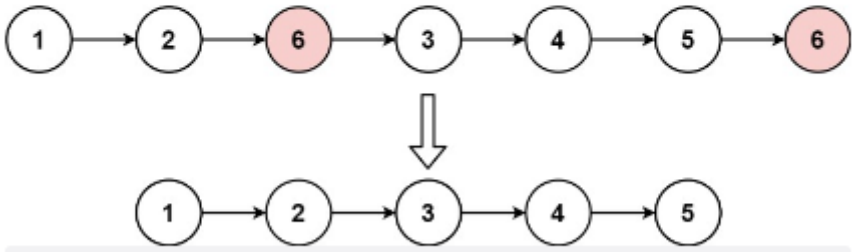
题述

203. 移除链表元素

难度 简单 763

给你一个链表的头节点 `head` 和一个整数 `val` ，请你删除链表中所有满足 `Node.val == val` 的节点，并返回 **新的头节点** 。

示例 1：




```
输入：head = [1,2,6,3,4,5,6], val = 6
输出：[1,2,3,4,5]
```

示例 2：

```
输入：head = [], val = 1
输出：[]
```

示例 3：

```
输入：head = [7,7,7,7], val = 7
输出：[]
```

提示：

- 列表中的节点数目在范围 `[0, 10^4]` 内
- `1 <= Node.val <= 50`
- `0 <= val <= 50`

思考

这种情况下的移除操作，就是让节点next指针直接指向下下一个节点就可以了，

那么因为单链表的特殊性，只能指向下一个节点，刚刚删除的是链表的中第二个，和第四个节点，那么如果删除的是头结点又该怎么办呢？

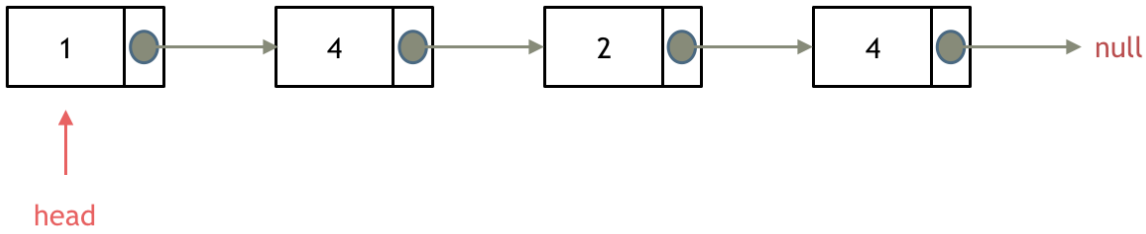
这里就涉及如下链表操作的两种方式：

- **直接使用原来的链表来进行删除操作。**

- **设置一个虚拟头结点在进行删除操作。**

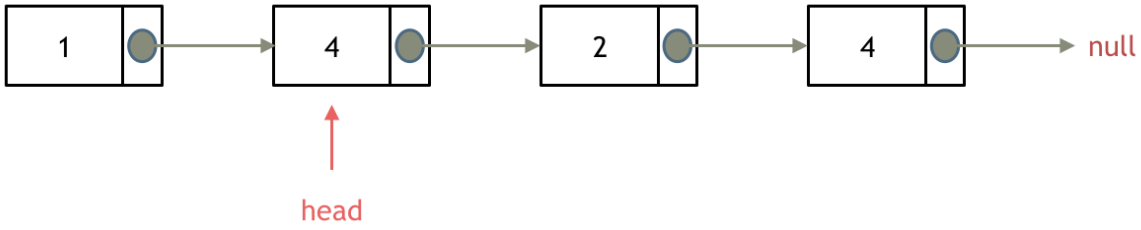
来看第一种操作：直接使用原来的链表来进行移除。

链表：1->4->2->4　　移除元素1

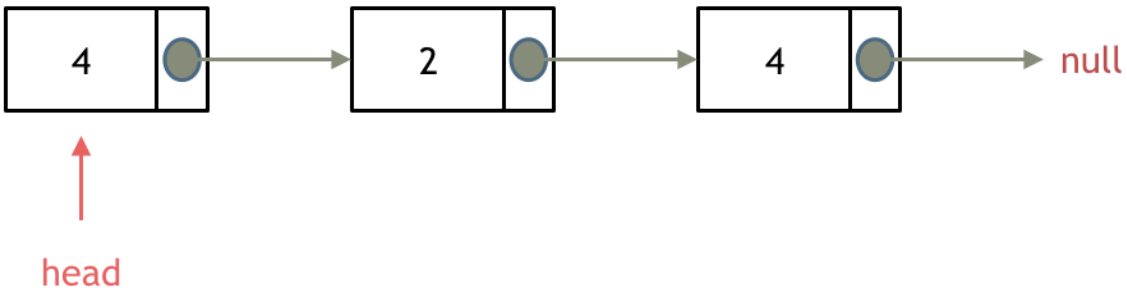


移除头结点和移除其他节点的操作是不一样的，因为链表的其他节点都是通过前一个节点来移除当前节点，而头结点没有前一个节点。

所以头结点如何移除呢，其实只要将头结点向后移动一位就可以，这样就从链表中移除了一个头结点。



依然别忘将原头结点从内存中删掉。



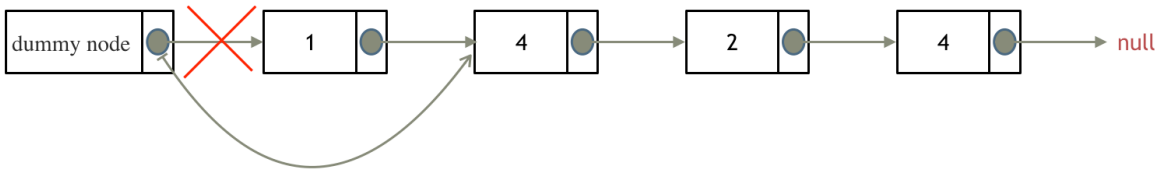
这样移除了一个头结点，是不是发现，在单链表中移除头结点 和 移除其他节点的操作方式是不一样，其实在写代码的时候也会发现，需要单独写一段逻辑来处理移除头结点的情况。

那么可不可以 以一种统一的逻辑来移除 链表的节点呢。

其实**可以设置一个虚拟头结点**，这样原链表的所有节点就都可以按照统一的方式进行移除了。

来看看如何设置一个虚拟头。依然还是在这个链表中，移除元素1。

链表：1->4->2->4　　移除元素1



这里来给链表添加一个虚拟头结点为新的头结点，此时要移除这个旧头结点元素1。

这样是不是就可以使用和移除链表其他节点的方式统一了呢?

来看一下，如何移除元素1 呢，还是熟悉的方式，然后从内存中删除元素1。

最后呢在题目中，return 头结点的时候，别忘了 `return dummyNode->next;`， 这才是新的头结点

题解

原链表删除

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* removeElements(ListNode* head, int val)
    {
        //思路一：直接在原链表基础上进行删除

        //情况1 头结点一直是要删除的节点
        while(head!=nullptr && head->val==val)
        {
            ListNode* temp=head;
            head=head->next;
            delete temp;
        }

        //情况2：删除非头结点
        ListNode* p=head;
        while(p!=nullptr && p->next!=nullptr)
        {
            if(p->next->val==val)
            {
                ListNode* temp=p->next;
                p->next=p->next->next;
                delete temp;
            }
```

```
            else
            {
                p=p->next;      //链表后移
            }
        }

        return head;
    }
};
```

虚拟头结点

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* removeElements(ListNode* head, int val)
    {
        //思路二：虚拟头结点
        ListNode* fakeHead=new ListNode(0); //初始化一个虚拟头结点
        fakeHead->next=head;    //将虚拟头结点指向head，这样方面后便做删除操作
        ListNode* p=fakeHead;
        while(p->next!=nullptr)
        {
            if(p->next->val==val)
            {
                ListNode* temp=p->next;
                p->next=p->next->next;
                delete temp;
            }
            else
            {
                p=p->next;
            }
        }
        head=fakeHead->next;
        delete fakeHead;
        return head;
    }
};
```

总结

理清概念及不同节点之间的区分

- 首元节点

- 头节点

设置合适的工作指针和循环条件来对链表进行遍历即可完成题目需求