```
/*
给定一个单链表，把所有的奇数节点和偶数节点分别排在一起。请注意，这里的奇数节点和偶数节点指的是节点编号的奇偶性，而不是节点的值的奇偶性。
请尝试使用原地算法完成。你的算法的空间复杂度应为 O(1)，时间复杂度应为 O(nodes)，nodes 为节点总数。
示例 1:
输入: 1->2->3->4->5->NULL
输出: 1->3->5->2->4->NULL
示例 2:
输入: 2->1->3->5->6->4->7->NULL
输出: 2->3->6->7->1->5->4->NULL
说明:
应当保持奇数节点和偶数节点的相对顺序。
链表的第一个节点视为奇数节点，第二个节点视为偶数节点，以此类推。
来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/odd-even-linked-list
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
*/
```

分析:

- 方法一:辅助队列法,在遍历链表的过程中,按奇偶分别将对应节点放入两个队列,然后先奇队列后偶队列依次全部出对组件新的链表;
- 方法二:双指针重组法,在遍历链表的过程中,按奇偶分别添加到奇链表和偶链表中去，然后再讲奇链表的尾巴和偶链表的头部连接起来即可.

方法一:C++_辅助队列法

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution
{
    public:
        ListNode* oddEvenList(ListNode* head)
        {
            queue<ListNode*> qln_odd            ;
            queue<ListNode*> qln_even           ;
            ListNode         ret_val(0)         ;
            ListNode*        cur                ;
            int              count      = 0     ;

            if(head)
            {
                while(head)
                {
                    if(++count%2)
```

```
                    {
                        qln_odd.push(head);
                    }
                    else
                    {
                        qln_even.push(head);
                    }
                    head = head->next;
                }

                cur = &ret_val;

                while(qln_odd.size())
                {
                    cur->next   =   qln_odd.front();
                    cur         =   cur->next;
                    qln_odd.pop();
                }

                while(qln_even.size())
                {
                    cur->next   =   qln_even.front();
                    cur         =   cur->next;
                    qln_even.pop();
                }
                cur->next = NULL;
            }


            return ret_val.next;
        }
};



/*
执行结果：
通过
显示详情
执行用时 :28 ms, 在所有 cpp 提交中击败了39.56%的用户
内存消耗 :10.1 MB, 在所有 cpp 提交中击败了5.59%的用户
*/
```

方法一:C++_双指针重组法

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution
{
    public:
```

```
        ListNode* oddEvenList(ListNode* head)
        {
            ListNode    head_odd(0)                 ;
            ListNode*   tail_odd        =   NULL    ;
            ListNode    head_even(0)                ;
            ListNode*   tail_even       =   NULL    ;
            int         count           =   0       ;

            if(head)
            {
                tail_odd    =   &head_odd;
                tail_even   =   &head_even;

                while(head)
                {
                    /*奇*/
                    if(++count %2)
                    {
                        tail_odd->next  =   head            ;
                        tail_odd        =   tail_odd->next  ;
                    }
                    /*偶*/
                    else
                    {
                        tail_even->next =   head;
                        tail_even       =   tail_even->next;
                    }
                    head = head->next;
                }
                tail_odd->next  = head_even.next;
                tail_even->next = NULL;
            }

            return head_odd.next;
        }
};

/*
执行结果:
通过
显示详情
执行用时 :32 ms, 在所有 cpp 提交中击败了32.78%的用户
内存消耗 :9.7 MB, 在所有 cpp 提交中击败了78.19%的用户
*/
```

AlimyBreak
2019.10.22