链表反转

将链表 `lt1 -> lt2 -> lt3 -> lt4` 反转成 `lt4 -> lt3 -> lt2 -> lt1`。

1. 先创建链表

明确当前节点的值，和下一个节点的值，如：`{value: 1, next: { value: 2, next: { value: 3}}}`，才能让变成有序的数据结构。

2. 反转链表

将 next 删除，将原先的 next 变成当前节点，原先的当前节点变成现在的 next 。不过这样一旦删除了 next，就会造成 next 节点丢失。所以我们可以定义三个节点，上一个节点 `prev`， 当前节点 `cur`，下一个节点 `next`，先将所有 `prev cur next` 分别向后移动一位，然后删掉原来的 `next`。

```
// 创建链表
// [1, 2, 3, 4]
/*
let obj = {
    value: 1,
    next: {
        value: 2,
        next: {
            value: 3,
            next: {
                value: 4
            }
        }
    }
}
*/
function createLinkTable(arr) {
    if (!arr) {
        throw new Error("必须传入一个数组! ");
    }
    const len = arr.length;
    if (!len) {
        return {};
    }
    if (len === 1) {
        return {value: arr[0]}
    }
    // len >= 2, 假设为 4
    let curNode = {
        value: arr.at(-1)
    };
    // 时间复杂度 O(n)
    // 空间复杂度 O(1)，就是将数组转换，没有占用多余空间
    for (let i = len - 2; i >= 0; i--) {
        curNode = {
```

```
            value: arr[i],
            next: curNode
        }
    }
    return curNode;
}
createLinkTable([1,2,3,4]);
```

```
// 反转链表
function reserveLinkTable(node) {
    // 以链表中第一个元素为 nextNode 为基准，定义 3 个指针
    let prevNode = undefined;
    let curNode = undefined;
    let nextNode = node;

    if (!node) {
        throw new Error("链表至少一个节点！");
    }

    while(nextNode) {
        // 第一个元素，删除 next，防止循环引用
        if (curNode && !prevNode) {
            delete curNode.next;
        }

        // 反转指针，箭头方向改变
        if (curNode && prevNode) {
            curNode.next = prevNode;
        }

        // 指针整体向后移动
        prevNode = curNode;
        curNode = nextNode;
        nextNode = nextNode.next;
    }

    // 当 nextNode 无值（undefined）时
    curNode.next = prevNode;
    return curNode;
}
```