算法面试——反转链表

作为计算机的内功，操作系统，计算机网络，数据结构，计算机组成原理，数据库原理。这个大家都有必要了解。无论我们处于面试也好，还是出于程序员的基本功，对于以上内容都要好好掌握。不过我呢，几天先从算法开涮，想到哪，写到哪，后续会把相关内容一并展示出来~~~

今天我们先从一个比较简单的算法题目入手——反转链表。

一、链表的基本概念

链表是一种物理存储单元上非连续、非顺序的存储结构，数据元素的逻辑顺序是通过链表中的指针链接次序实现的。
用生活中的例子就是我们的火车，一节一节的，对应就是链表中的一个又一个的节点（Node）。

这里说的物理层面非连续：指的在磁盘上，数据不是连续的。可以这有一块，那有一块，通过指针进行连接。
对于指针是C语言的精华，在这里不过多阐释，防止误导大家。在这里我们仅仅通过js角度去说明这个“指针”

用一个变量存储下一个节点的地址。

还是很好理解的对吧，接下来我们使用代码看一下，啥事节点，啥是链表。
🌰：

```
function Node(val) {
    this.value = val;
    this.next = null;
}
```

这是一个节点的构造函数。返回的实例就是一个节点。我们可以产生若干节点实例。通过他们的next进行连接。

注：链表存在多种结构：单向链表，双向链表，十字交叉链表，邻接链表（邻接矩阵）等。

当然了我们这里强调的链表是单向链表，**一条链一个方向**。

为了快速生成链表和展示链表内容，我们这里先做了一个工具（tool.js）里面返回若干的工具函数。
如下：

```
// 节点函数
function Node(val) {
    this.value = val;
    this.next = null;
}
// 产生顺序链表 createList(3) =>  1 => 2 => 3 => null
function createList(len) {
    let head = new Node(1);
    let p = head;
    for (let i = 1; i < len; i++) {
        p.next = new Node(i + 1);
        p = p.next;
    }
    return head;
}
// 将链表顺序输出
function showList (head) {
    let p = head;
    while(p) {
        console.log(p.value);
        p = p.next;
    }
}

module.exports = {
    showList, Node, createList
}
```

上面的工具函数，咱们就不多说了。我们来看看今天的几道题目

二、反向链表题目

1. 简单反转单向链表

给出一个这样的链表
1 => 2 => 3 => 4 => 5 => null
返回如下链表
5 => 4 => 3 => 2 => 1 => null

对于这种结构进行反转，基本思想：
像我们站队，原来是从小到大排，变成从大到小排序。
最简单的方式，大家先后转，很明显在计算机里面不支持这样的操作（滑稽脸）。
那就从小的开始，陆续的成新的一排，最终变成从大到小。
基本思想如下：
List1: 1 => 2 => 3 => 4 => 5 => null
List2: null

第一步：
List1: 2 => 3 => 4 => 5 => null
List:2: 1=>null
第二步：
List1: 3 => 4 => 5 => null
List:2: 2 => 1=>null
若干步骤之后
List1: null
List2: 5 => 4 => 3 => 2 => 1 => null
上代码：

```
// 循环方式实现
let reverseList1 = head => {
    let pre = null; // 新的队List2
    let cur = head; // 一个点兵人，记录当前的兵是谁
    while (cur) { // 如果当前还有兵 就要站到List2后面
        let next = cur.next; // 防止当前的兵走了就不知道下一个是谁了，我就在
```

```
    找一个点兵人看着下一个兵
7           cur.next = pre; // 当前的兵站到新队里面去
8           pre = cur; // 新队的点兵人指向第一个人
9           cur = next; // List1 点兵人后移
10      }
11      return pre;
12  }
```

其实我们根据上面的逻辑能看出来，他其实就是把一个人站到上一个人的前面。这句话本身带有递归逻辑。我们直接上代码：

List1: 1 => 2 => 3 => 4 => 5 => null
👆
cur
List2: null
👆
pre

```
1  // 递归
2  let reverseList2 = head => {
3      // cur 站到 pre 的前面。然后在重新的去找 cur 和 pre
4      let reverse = (pre, cur) => {
5          if(!cur) {
6              return pre
7          }
8          let next = cur.next;
9          cur.next = pre;
10         pre = cur;
11         cur = next;
12         return reverse(pre, cur)
13     }
```

2. 区间反转链表

给出一个这样的链表
1 => 2 => 3 => 4 => 5 => null
同时给出反转的起始点与终止点

2，4
返回如下链表
1 => 4 => 3 => 2 => 5 => null

解题思路：
还是战队，我需要三个人帮我们维护队伍：
以上面的题目为例，p
1,　　2，3，4，　5
pre　cur　　next

pre:　区间开始前的一个，可以为null
cur:　区间链表的表头元素
end:　区间链表之后的链表头结点
思路：　将 pre, cur, end 筛选出来。
　　将 cur 逆序放到end之后，然后与 pre 连接。
这里可以有多种方式实现，可以是借用一个头指针，也可以不用，直接上代码：

```
// 不借助头结点
function reverseListByAmong(head, m, n) {
    let pre = null,
        cur = null,
        end = null;
    let index = 1;
    let h = head;
    //  查找到pre, cur, end;
    while(h) {
        if(m == index + 1) {
            pre = h;
        }
        if(index == m) {
            cur = h;
        }
        if(index == n) {
            end = h.next;
            h.next = null;
        }
        index ++;
        h = h.next;
    }
```

```
    // 将cur倒叙
    while(cur) {
        let next = cur.next;
        cur.next = end;
        end = cur;
        cur = next;
    }
    // 将pre与cur与end拼接
    if(pre == null) {
        return end;
    } else {
        pre.next = end;
        return head
    }
    // 将pre与cur与end拼接
}

// 借助头结点
let reverseBetween =(head, m, n) => {
    let dutyList = {next: null};
    dutyList.next = head;
    let p = dutyList.next;
    let start=dutyList, cur=null, end=null;
    let index = 1;
    while(p) {
        if(index == m-1) {
            start = p;
        }
        if(index == m) {
             cur = p;
        }
        if(index == n) {
            end = p.next;
            p.next = null;
        }
        index ++;
        p = p.next;
    }

    // 反转cur
```

```
    while(cur) {
        let next = cur.next;
        cur.next = end;
        end = cur;
        cur = next;
    }

    start.next = end;
    return dutyList.next;
}
```

当然了，也可以使用递归，还是把反转的过程递归就好：

```
let rb1 = (head, m, n) => {
    let dutyHead = {next: null};
        dutyHead.next = head;
    let index = 1;
    let start = dutyHead, cur = null, end = null;

    while(head) {
        if(index == m-1) {
            start = head;
        }
        if(index == m) {
            cur = head;
        }
        if(index == n) {
            end = head.next;
            head.next = null;
        }
        index ++;
        head = head.next;
    }

    let reverse = (pre, cur) => {
        if(cur == null) {
            // console.log(pre)
            return pre
        }
```

```
        let next = cur.next;
        cur.next = pre;
        pre = cur;
        cur = next;
        return reverse(pre, cur);
    }
    start.next = reverse(end, cur);
    return dutyHead.next
}
```

3. 两个一组反转链表

给出一个这样的链表

1 => 2 => 3 => 4 => 5 => null

返回如下链表

2 => 1 => 4 => 3 => 5 => null

对于上面的这个问题，我们借助一个头结点帮我们，也就是领头人

L => 1 => 2 => 3 => 4 => 5 => null

我们先分析是否 这个队伍里时候还够两个人

于是有了，点兵人1（p1），和点兵人2(p2)

如果够两个人那就：

L => 1 => 2 => ....

a)

先判断

p1 = L.next;

p2 = p1 && p1.next p1 不为空 返回 p1.next;

如果满足

1 => ... (p1.next = p2.next)

2 => 1 (p2.next = p1);

领头人下移

L => 2 => 1 => ...

L1

再重复上面的过程就可以了，同样有递归写法也有，循环写法。

先上循环：

```
function reverseBetween2(head) {
    let emptyNode = new Node('empty');
```

```
        emptyNode.next = head;
    let flag = true;
    let e = emptyNode
    while (flag) {
        let p1 = e.next;
        let p2 = p1 && p1.next;
        if(p1&&p2) {
            p1.next = p2.next;
            p2.next = p1;
            e.next = p2;
            e = p1;
        } else {
            flag = false;
        }
    }
    return emptyNode.next;
}
```

我们循环的过程就是一个递归：

```
let rb23 = head => {
    if(head == null || head.next == null) {
        return head;
    }
    let p1 = head;
    let p2 = p1.next;
    p1.next = rb23(p2.next);
    p2.next = p1;
    return p2
}
```

是不是很简洁，好好体会之后，会发生质的变化。

接下来我们将进行再次升级。

4. k个一组反转链表

题目和上一个一样，只不过多了一个 参数 k 分组内容：

给出一个这样的链表

1 => 2 => 3 => 4 => 5 => null

还有一个分组个数 k = 3

返回如下链表

3 => 2 => 1 => 4 => 5 => null

解题思路：

这个无非就是一个 分组+倒叙的过程。

1. 看看能分为几组
2. 讲每组分别倒叙

来看代码：

```
// 循环
let reverseBetweenGroup = (head, len) => {
    let p = head;
    let groups = [];
    let count = 0;
    // 分组
    while (p) {
        if (count == 0) {
            groups.push(p);
        }
        if (++count == len) {
            let p1 = p.next;
            p.next = null;
            p = p1;
            count = 0;
            if(p == null) {
                groups.push(p)
            }
        } else {
            p = p.next;
        }
    }
    if(groups.length == 1) {
        return head
    }
    // console.log(groups);
```

```
    // i逆序 插入到 i+1前面。
    let e = groups[groups.length-1]
    for(let i=groups.length-1; i>0; i--) {
         let cur = groups[i-1];
         while(cur) {
             var next = cur.next;
             cur.next = e;
             e = cur;
             cur = next;
         }
    }
    return e;
}
```

里面的实现和上面的思路很像，这里就不多说了，上面理解的话，这道题很好实现的。

再看看递归
我写了好几版，也是陆续优化的，
Version 1
基于上面的版本修改的递归

```
let rbg2 = (head, len) => {
    let groups = [];
    let p = head;
    let count = 0;
    while(p) {
        if(count == 0) {
            groups.push(p);
        }
        if(++count == len) {
            count = 0;
            let p1 = p.next;
            p.next = null;
            p = p1;
            if(p == null) {
                groups.push(null)
            }
        } else {
            p = p.next;
```

```
        }
    }
    if(groups.length ==1) {
        return head;
    }
    let e = groups.pop();
    let reverse = () => {
        let cur = groups.pop();
        if(cur == undefined) {
            return
        }
        while(cur) {
            let next = cur.next;
            cur.next = e;
            e = cur;
            cur = next;
        }
        reverse();
    };
    reverse()
    return e;

}
```

Version 2

上面的那个是先分组，后倒序，这个是边分组，边倒序

```
let rbg3 = (head, len) => {
    let pre = null;
    let cur = head;
    let p = head;
    // 是否符合len个;
    for(let i=0; i<len; i++) {
        if(p == null) {
            return head;
        }
        p = p.next;
    }
```

```
    // 倒置
    for(let i=0; i<len; i++) {
        let next = cur.next;
        cur.next = pre;
        pre = cur;
        cur = next;
    }
    head.next = rbg3(cur, len);
    return pre;
}
```

以上就是反转链表的经典题目是不是很烧脑，确实掌握不易，知行合一。回去要好好思考，逐渐消化最终掌握。