```
        n3 = n1; // n3 -> 保存最后一个节点
        n2 = head;// n2 -> 左边第一个节点
        boolean res = true;
        while (n1 != null && n2 != null) { // 检查回文
                if (n1.value != n2.value) {
                        res = false;
                        break;
                }
                n1 = n1.next; // 从左到中部
                n2 = n2.next; // 从右到中部
        }
        n1 = n3.next;
        n3.next = null;
        while (n1 != null) { // 恢复列表
                n2 = n1.next;
                n1.next = n3;
                n3 = n1;
                n1 = n2;
        }
        return res;
}
```

将单向链表按某值划分成左边小、中间相等、右边大的形式

【题目】

给定一个单向链表的头节点 head，节点的值类型是整型，再给定一个整数 pivot。实现一个调整链表的函数，将链表调整为左部分都是值小于 pivot 的节点，中间部分都是值等于 pivot 的节点，右部分都是值大于 pivot 的节点。除这个要求外，对调整后的节点顺序没有更多的要求。

例如：链表 9->0->4->5->1，pivot=3。

调整后链表可以是 1->0->4->9->5，也可以是 0->1->9->5->4。总之，满足左部分都是小于 3 的节点，中间部分都是等于 3 的节点（本例中这个部分为空），右部分都是大于 3 的节点即可。对某部分内部的节点顺序不做要求。

进阶：

在原问题的要求之上再增加如下两个要求。

- 在左、中、右三个部分的内部也做顺序要求，要求每部分里的节点从左到右的顺序与原链表中节点的先后次序一致。

例如：链表 9->0->4->5->1，pivot=3。调整后的链表是 0->1->9->4->5。在满足原问题要求的同时，左部分节点从左到右为 0、1。在原链表中也是先出现 0，后出现 1；中间部

分在本例中为空，不再讨论；右部分节点从左到右为 9、4、5。在原链表中也是先出现 9，然后出现 4，最后出现 5。

- 如果链表长度为 N，时间复杂度请达到 $O(N)$，额外空间复杂度请达到 $O(1)$。

【难度】

尉 ★★☆☆

【解答】

普通解法的时间复杂度为 $O(N)$，额外空间复杂度为 $O(N)$，就是把链表中的所有节点放入一个额外的数组中，然后统一调整位置的办法。具体过程如下：

1．先遍历一遍链表，为了得到链表的长度，假设长度为 N。

2．生成长度为 N 的 Node 类型的数组 nodeArr，然后遍历一次链表，将节点依次放进 nodeArr 中。本书在这里不用 LinkedList 或 ArrayList 等 Java 提供的结构，因为一个纯粹的数组结构比较利于步骤 3 的调整。

3．在 nodeArr 中把小于 pivot 的节点放在左边，把相等的放中间，把大于的放在右边。也就是改进了快速排序中 partition 的调整过程，即如下代码中的 arrPartition 方法。实现的具体解释请参看本书“数组类似 partition 的调整”问题，这里不再详述。

4．经过步骤 3 的调整后，nodeArr 是满足题目要求的节点顺序，只要把 nodeArr 中的节点依次重连起来即可，整个过程结束。

全部过程请参看如下代码中的 listPartition1 方法。

```
public class Node {
        public int value;
        public Node next;

        public Node(int data) {
                this.value = data;
        }
}

public Node listPartition1(Node head, int pivot) {
        if (head == null) {
                return head;
        }
        Node cur = head;
        int i = 0;
        while (cur != null) {
                i++;
                cur = cur.next;
```

```
            }
            Node[] nodeArr = new Node[i];
            i = 0;
            cur = head;
            for (i = 0; i != nodeArr.length; i++) {
                    nodeArr[i] = cur;
                    cur = cur.next;
            }
            arrPartition(nodeArr, pivot);
            for (i = 1; i != nodeArr.length; i++) {
                    nodeArr[i - 1].next = nodeArr[i];
            }
            nodeArr[i - 1].next = null;
            return nodeArr[0];
    }

    public void arrPartition(Node[] nodeArr, int pivot) {
            int small = -1;
            int big = nodeArr.length;
            int index = 0;
            while (index != big) {
                    if (nodeArr[index].value < pivot) {
                            swap(nodeArr, ++small, index++);
                    } else if (nodeArr[index].value == pivot) {
                            index++;
                    } else {
                            swap(nodeArr, --big, index);
                    }
            }
    }

    public void swap(Node[] nodeArr, int a, int b) {
            Node tmp = nodeArr[a];
            nodeArr[a] = nodeArr[b];
            nodeArr[b] = tmp;
    }
```

下面来看看增加要求之后的进阶解法。对每部分都增加了节点顺序要求，同时时间复杂度仍然为 $O(N)$，额外空间复杂度为 $O(1)$。既然额外空间复杂度为 $O(1)$，说明实现时只能使用有限的几个变量来完成所有的调整。

进阶解法的具体过程如下：

1．将原链表中的所有节点依次划分进三个链表，三个链表分别为 small 代表左部分，equal 代表中间部分，big 代表右部分。

例如，链表 7->9->1->8->5->2->5，pivot=5。在划分之后，small、equal、big 分别为：

small：1->2->null

equal：5->5->null

big：7->9->8->null

2．将 small、equal 和 big 三个链表重新串起来即可。

3．整个过程需要特别注意对 null 节点的判断和处理。

进阶解法还是主要考查面试者利用有限几个变量调整链表的代码实现能力，全部进阶解法请参看如下代码中的 listPartition2 方法。

```
public static Node listPartition2(Node head, int pivot) {
        Node sH = null; // 小的头
        Node sT = null; // 小的尾
        Node eH = null; // 相等的头
        Node eT = null; // 相等的尾
        Node bH = null; // 大的头
        Node bT = null; // 大的尾
        Node next = null; // 保存下一个节点
        // 所有的节点分进三个链表中
        while (head != null) {
                next = head.next;
                head.next = null;
                if (head.value < pivot) {
                        if (sH == null) {
                                sH = head;
                                sT = head;
                        } else {
                                sT.next = head;
                                sT = head;
                        }
                } else if (head.value == pivot) {
                        if (eH == null) {
                                eH = head;
                                eT = head;
                        } else {
                                eT.next = head;
                                eT = head;
                        }
                } else {
                        if (bH == null) {
                                bH = head;
                                bT = head;
                        } else {
                                bT.next = head;
                                bT = head;
                        }
                }
                head = next;
        }
        // 小的和相等的重新连接
        if (sT != null) {
                sT.next = eH;
                eT = eT == null ? sT : eT;
```

```
        }
        // 所有的重新连接
        if (eT != null) {
                eT.next = bH;
        }
        return sH != null ? sH : eH != null ? eH : bH;
}
```

复制含有随机指针节点的链表

【题目】

一种特殊的链表节点类描述如下：

```
public class Node {
        public int value;
        public Node next;
        public Node rand;

        public Node(int data) {
                this.value = data;
        }
}
```

Node 类中的 value 是节点值，next 指针和正常单链表中 next 指针的意义一样，都指向下一个节点，rand 指针是 Node 类中新增的指针，这个指针可能指向链表中的任意一个节点，也可能指向 null。

给定一个由 Node 节点类型组成的无环单链表的头节点 head，请实现一个函数完成这个链表中所有结构的复制，并返回复制的新链表的头节点。例如：链表 1->2->3->null，假设 1 的 rand 指针指向 3，2 的 rand 指针指向 null，3 的 rand 指针指向 1。复制后的链表应该也是这种结构，比如，1′->2′->3′->null，1′的 rand 指针指向 3′，2′的 rand 指针指向 null，3′的 rand 指针指向 1′，最后返回 1′。

进阶：不使用额外的数据结构，只用有限几个变量，且在时间复杂度为 $O(N)$内完成原问题要实现的函数。

【难度】

尉 ★★☆☆