链滴

算法：反转单链表

作者：hudk

原文链接：https://ld246.com/article/1590642582596

来源网站：链滴

对leetCode一个算法的分析学习，支持对单链表内指定区间的反转实现。

206. 反转链表

92. 反转链表 II;

```
import java.util.ArrayList;
import java.util.IdentityHashMap;
import java.util.List;

/**
 * 对leetCode一个算法的分析学习
 * 题目：单链表的反转
 *
 * @author hudk
 * @date 2020/5/27 20:20
 */
public class Solution {


    /**
     * 单链表结点
     */
    public static class ListNode {
        int val;
        public ListNode next;

        public ListNode(int x) {
            val = x;
        }

        @Override
        public String toString() {
            return String.valueOf(val);
        }
    }


    /**
     * leetCode 题目：反转单链表1
     *
     * 示例:
     * 输入: 1->2->3->4->5->NULL
     * 输出: 5->4->3->2->1->NULL
     * 进阶:
     * 你可以迭代或递归地反转链表。你能否用两种方法解决这道题?
     *
     * 来源：力扣（LeetCode）
     * 链接：https://leetcode-cn.com/problems/reverse-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。
     * @param head
     * @return
     */

    /**
```

```
    * 方法一
    * 这个方法是本人解答方案,相对使用了比较多的空间
    * 空间复杂度：O(n)
    * 时间复杂度：O(n)
    *
    * @param head
    * @return
    */
   public static ListNode myReverseList(ListNode head) {
       //当链表长度为零时，直接返回null
       if (head == null) {
           return null;
       }
       //引用指向头结点
       ListNode h = head;
       //遍历整个链表，统计链表长度 i
       int i = 1;
       while (h.next != null) {
           i++;
           h = h.next;
       }
       //创建一个和链表长度一样的数组，并将链表的元素按照原顺序逐个放入数组中
       ListNode[] ln = new ListNode[i];
       for (int j = 0; j < i; j++) {
           ln[j] = head;
           head = head.next;
       }
       //再从数组的尾部开始遍历，逐个该表链表元素的next指针指向前一个元素。
       for (int x = i - 1; x > 0; x--) {
           ln[x].next = ln[x - 1];
       }
       //将原来的头结点（现在转置后的尾结点next引用置空）
       ln[0].next = null;
       //返回转置后新的头结点
       return ln[i - 1];
   }

   /**
    * 方法二
    * 这个方法是leetCode上的算法大神解答的方案
    * 利用递归的巧妙与优雅实现
    *
    * 作者：labuladong
    * 链接：https://leetcode-cn.com/problems/reverse-linked-list-ii/solution/bu-bu-chai-jie-ru
he-di-gui-di-fan-zhuan-lian-biao/
    * 来源：力扣（LeetCode）
    * 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
    *
    * @param head
    * @return
    */
   public static ListNode revers(ListNode head) {
       if (head.next == null) {
           return head;
```

```
    }
    ListNode last = revers(head.next);
    head.next.next = head;
    head.next = null;
    return last;
}


/**
 * 方法三
 * 这个方法是官方解答的方案
 * 利用了迭代的思想，同样简洁且高效
 * 空间复杂度：O(1)
 * 时间复杂度：O(n)
 *
 * 来源：力扣（LeetCode）
 * 链接：https://leetcode-cn.com/problems/reverse-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。
 *
 * @param head
 * @return
 */
public static ListNode reverseList(ListNode head) {
    ListNode prev = null;
    ListNode curr = head;
    ListNode nextTemp = null;
    while (curr != null) {
        nextTemp = curr.next;
        curr.next = prev;
        prev = curr;
        curr = nextTemp;
    }
    return prev;
}

/**
 * leetCode 题目：反转单链表2
 * 反转从位置 m 到 n 的链表。请使用一趟扫描完成反转。
 *
 * 说明:
 * 1 ≤ m ≤ n ≤ 链表长度。
 *
 * 示例:
 * 输入: 1->2->3->4->5->NULL, m = 2, n = 4
 * 输出: 1->4->3->2->5->NULL
 *
 * 来源：力扣（LeetCode）
 * 链接：https://leetcode-cn.com/problems/reverse-linked-list-ii
 * 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。
 */


/**
 * 方法一 begin*****************************************************************
```

```
    * <p>
    * 递归实现单链表的指定区间反转
    * 这个算法的实现，淋漓尽致的体现了递归的优雅与简洁。
    * 适用于链表长度比较短的场景，或对性能要求不高的场景
    * <p>
    * 作者：labuladong
    * 链接：https://leetcode-cn.com/problems/reverse-linked-list-ii/solution/bu-bu-chai-jie-ru
he-di-gui-di-fan-zhuan-lian-biao/
    * 来源：力扣（LeetCode）
    * 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
    *
    * @param head
    * @param m
    * @param n
    * @return
    */
   public static ListNode reverseBetween(ListNode head, int m, int n) {
      if(head == null){
         return null;
      }
      // base case
      if (m == 1) {
         //单链表的前n个结点反转
         return reverseN(head, n);
      }
      // 前进到反转的起点触发 base case
      head.next = reverseBetween(head.next, m - 1, n - 1);
      return head;
   }

   /**
    * 递归实现单链表的前n个结点反转
    */
   static ListNode successor = null; // 后驱节点

   // 反转以 head 为起点的 n 个节点，返回新的头结点
   public static ListNode reverseN(ListNode head, int n) {
      if(head == null){
         return null;
      }
      if (n == 1) {
         // 记录第 n + 1 个节点
         successor = head.next;
         return head;
      }
      // 以 head.next 为起点，需要反转前 n - 1 个节点
      ListNode last = reverseN(head.next, n - 1);

      head.next.next = head;
      // 让反转之后的 head 节点和后面的节点连起来
      head.next = successor;
      return last;
   }
```

```
    /**方法一end**********************************************************************/


    /**
     * 方法二
     * 这是本人的解答方案，使用了迭代的思路，并将各种情况逐一考虑，分别处理。
     * 代码量比较多，但自我感觉逻辑看起来更加清晰一些。
     *
     * @param head 链表头结点
     * @param m    翻转区间开始位置
     * @param n    翻转区间结束位置
     * @return
     */
    public static ListNode myReverseBetween(ListNode head, int m, int n) {
        if(m <= 0){
            m = 1;
        }
        int size = size(head);
        if(n > size){
            n = size;
        }
        //如果传入为空，直接返回空
        if (head == null) {
            return null;
        }
        //如果m = n,说明转置后等于没转置，所以不做处理直接返回原链表
        if (m == n) {
            return head;
        }
        //1、当m等于1时，倒序之后，第一个结点一定会和第n+1个结点相连
        //2、并且第n个结点，会成为新链表的头结点
        if (m == 1) {
            ListNode perv = null;
            ListNode crr = head;
            ListNode nodeOne = null;
            int i = 1;
            while (crr != null) {
                ListNode next = crr.next;
                if (i == 1) {
                    //暂时记住第一个结点，后面它将会与第n+1个结点相连
                    nodeOne = crr;
                    nodeOne.next = null;
                    //为迭代做准备，从第2个结点开始迭代地做“翻转”动作，故将第1个结点当做下次循
时的“前置结点”
                    perv = crr;
                }
                //从第二个结点开始，一直到第n个结点，逐一"翻转"他们的next
                if (i > m && i < n) {
                    crr.next = perv;
                    perv = crr;
                }
                //第n个结点
                if (i == n) {
                    //第一个结点的的next引用指向了第n+1个结点
```

```
                nodeOne.next = crr.next;
                //翻转第n个结点的next
                crr.next = perv;
                //第n个结点，会成为新链表的头结点
                head = crr;
                //由于后面得结点不需要做处理了，故跳出循环即可
                break;
            }
            //迭代
            crr = next;
            i++;
        }
    }
    //1、如果m>1,第m-1个结点会与第n个结点相连，第m个结点会与第n+1个结点相连
    //2、然后，第m+1个到第n个结点的next依次“翻转”
    //3、头结点不变
    if (m > 1) {
        ListNode perv = null;
        ListNode crr = head;
        ListNode nodeMp = null;//第m个结点的前一个结点
        ListNode nodeM = null;//第m个结点
        int i = 1;
        while (crr != null) {
            ListNode next = crr.next;
            if (i == m - 1) {
                //暂时记住第m-1个结点,后面它将会与第n个结点相连
                nodeMp = crr;
                nodeMp.next = null;
            }
            if (i == m) {
                //暂时记住第m个结点,后面它将会与第n+1个结点相连
                nodeM = crr;
                nodeM.next = null;
                //为迭代做准备，从m+1个结点开始迭代地做“翻转”动作，故将第m个结点当做下次
环时的“前置结点”
                perv = crr;
            }
            //第m+1个到第n个结点的next依次“翻转”
            if (i > m && i < n) {
                crr.next = perv;
                perv = crr;
            }
            if (i == n) {
                //第m个结点的next引用指向了第n+1个结点
                nodeM.next = crr.next;
                //翻转第n个结点的next
                crr.next = perv;
                //第m-1个结点的next引用指向了第n个结点
                nodeMp.next = crr;
                //由于后面得结点不需要做处理了，故跳出循环即可
                break;
            }
            //迭代
            crr = next;
```

```
            i++;
        }
    }
    return head;
}

/**方法二end*****************************************************************/


/**
 * 测试用例
 * @param args
 */
public static void main(String[] args) {
    //生成一个长度为10的单链表
    ListNode head = createRandomSingleLinkList(10);
    //打印初始链表
    printLinkList(head);
    //反转测试
    ListNode head1 = myReverseList(head);
    printLinkList(head1);
    //迭代方式反转测试
    ListNode head2 = reverseList(head1);
    printLinkList(head2);
    //递归方式反转测试
    ListNode head3 = revers(head2);
    printLinkList(head3);
    //迭代方式反转指定区间测试
    ListNode head4 = myReverseBetween(head3,2,8);
    printLinkList(head4);
    //递归方式反转指定区间测试
    ListNode head5 = reverseBetween(head4,4,5);
    printLinkList(head5);
}


/**
 * 生成一个指定长度的链表
 * @param size
 * @return
 */
public static ListNode createRandomSingleLinkList(int size){
    if(size == 0){
        return null;
    }
    ListNode head = new ListNode(1);
    ListNode crr= head;
    for(int i=1; i<size; i++){
        ListNode node = new ListNode(i+1);
        crr.next = node;
        crr = node;
    }
    return head;
}
```

```
    /**
     * 生成 0 - 100 范围内的随机正式
     * @return
     */
    public static int randomInt(){
        return (int)Math.floor(Math.random()*100);
    }

    /**
     * 打印链表
     * @param head
     */
    public static void printLinkList(ListNode head){
        List<ListNode> nodes = new ArrayList<>();
        if(head == null){
            System.out.println(nodes);
            return;
        }
        nodes.add(head);
        while (head.next != null){
            nodes.add(head.next);
            head = head.next;
        }
        System.out.println(nodes);
    }

    /**
     * 计算单链表长度
     * @param head
     * @return
     */
    public static int size(ListNode head){
        if(head == null){
            return 0;
        }
        int i = 1;
        while (head.next != null){
            head = head.next;
            i++;
        }
        return i;
    }

}
```