## 单链表反转

方法一：逆指针法;使用3个指针遍历单链表，逐个链接点进行反转

p=head;

q = head.next;//初始化

While(q.next){

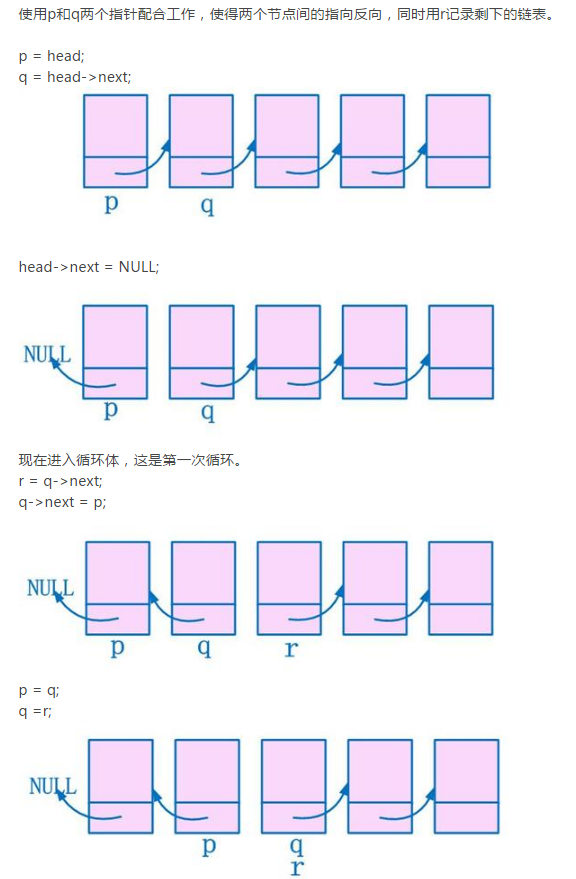
r = q.next;//临时指针r存储着下一个要反转的节点

q.next = p;//完成指向的反转

p = q;//p和q赋值为下次循环要反转指向的两个节点

q = r;

}



方法二：头插法 每次将下一个节点放到第一个位置

p = head;

pre.next = head;

while(p.next){

q = p.next;//每次将q移到第一个位置

p.next = q.next;

q.next = pre.next;//pre.next一直存储着当前链表的第一个节点

pre.next = q;

}

## 二叉树

由数组构建二叉树:如果二叉树的某个节点在数组中的下标为n，那么其左右子节点下表分为为2n+1,2n+2. 所以构建思路为取数组中所有的父节点，完善其子节点信息。而叶子节点的左右子节点都为null。

### 建立二叉树

public class BinTreeTraverse2 {

private int[] array = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

private static List<Node> nodeList = null;

private static class Node {//内部类：节点

Node left;

Node right;

int data;

Node(int newData) {

left = null;

right = null;

data = newData;

}

}

public void createBinTree() {

nodeList = new LinkedList<Node>();//树的所有节点存在list中

// 将一个数组的值依次转换为Node节点

for (int Index = 0; Index < array.length; Index++) {

nodeList.add(new Node(array[Index]));

}

// 对前lastIndex-1个父节点按照父节点与孩子节点的数字关系建立二叉树

for (int index = 0; index < array.length / 2 - 1; index++) {

// 左孩子

nodeList.get(index).left = nodeList.get(index \* 2 + 1);

// 右孩子

nodeList.get(index).right = nodeList.get(index \* 2 + 2);

}

// 最后一个父节点:因为最后一个父节点可能没有右孩子，所以单独拿出来处理

int lastIndex = array.length / 2 - 1;

// 左孩子

nodeList.get(lastIndex).left = nodeList.get(lastIndex \* 2 + 1);

// 右孩子,如果数组的长度为奇数才建立右孩子

if (array.length % 2 == 1) {

nodeList.get(lastIndex).right = nodeList.get(lastIndex\*2+2);

}

}

### 遍历二叉树

先序：先访问父节点然后左右子节点。使用递归方式,中序后序同理。

public static void preOrderTraverse(Node node) {

if (node == null)

return;

System.out.print(node.data + " ");

preOrderTraverse(node.left);

preOrderTraverse(node.right);

}

### 求二叉树节点个数

递归方式，如果二叉树为空，则节点数为0;如果不为空，二叉树节点个数为左节点个数+右节点个数+1。

public int nodeNum(Node root){

if(root == null){

return 0;

}else{

return nodeNum(root.left) + nodeNum(root.right) +1;

}

}

迭代方式：

    public static int getNodeNum(Node root) {

        if(root == null){

            return 0;

        }

        int count = 1;

//LinkedList实现了Queue接口

        Queue<Node> queue = new LinkedList<TreeNode>();

        queue.add(root);

        while(!queue.isEmpty()){

            Node cur = queue.remove();      // 从队头位置移除

            if(cur.left != null){           // 如果有左孩子，加到队尾

                queue.add(cur.left);

                count++;

            }

            if(cur.right != null){      // 如果有右孩子，加到队尾

                queue.add(cur.right);

                count++;

            }

        }

        return count;

    }