1.144. 二叉树的前序遍历

递归

```java
public List<Integer> preorderTraversal(TreeNode root) {
    List<Integer> list = new ArrayList<>();
    preOrder(root,list);
    return list;
}
public void preOrder(TreeNode root,List<Integer> list){
    if(root==null) return;
    list.add(root.val);
    preOrder(root.left,list);
    preOrder(root.right,list);
}
```

普通迭代

```java
    public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        if (root == null) return result;
        Deque<TreeNode> stack = new LinkedList<>();
        stack.push(root);
        while (!stack.isEmpty()) {
            TreeNode node = stack.pop();
            result.add(node.val);
            if (node.right != null) stack.push(node.right);
            if (node.left != null) stack.push(node.left);
        }
        return result;
    }
```

船长亲传法

```java
    public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        if (root == null) return result;
        Deque<TreeNode> stack = new LinkedList<>();
        Deque<Integer> statusStack = new LinkedList<>();
        stack.push(root);
        statusStack.push(2);
        while (!stack.isEmpty()) {
            switch (statusStack.pop()) {
                case 0: {
                    statusStack.push(1);
                    if (stack.peek().right != null) {
                        stack.push(stack.peek().right);
                        statusStack.push(2);
                    }
                    break;
                }
                case 1: {
                    stack.pop();
                    break;
                }
                case 2: {
                    result.add(stack.peek().val);
                    statusStack.push(0);
                    if (stack.peek().left != null) {
                        stack.push(stack.peek().left);
                        statusStack.push(2);
                    }
                    break;
                }
            }
        }
        return result;
    }
```

2.589. N 叉树的前序遍历

递归

```java
    public List<Integer> preorder(Node root) {
        List<Integer> result = new ArrayList<>();
        myPreOrder(root, result);
        return result;
    }

    public void myPreOrder(Node root, List<Integer> result) {
        if (root == null) return;
        result.add(root.val);
        for (Node children : root.children) {
            myPreOrder(children, result);
        }
    }
```

迭代

```java
public List<Integer> preorder(Node root) {
    Deque<Node> stack = new LinkedList<>();
    List<Integer> result = new LinkedList<>();
    if (root == null) return result;
    stack.push(root);
    while (!stack.isEmpty()) {
        Node node = stack.pop();
        result.add(node.val);
        for (int i = node.children.size() - 1; node.children != null && i >= 0; i--) {
            stack.push(node.children.get(i));
        }
    }
    return result;
}
```

3.226. 翻转二叉树

```java
    public TreeNode invertTree(TreeNode root) {
        if (root == null) return null;
        TreeNode temp = root.right;
        root.right = root.left;
        root.left = temp;
        invertTree(root.left);
        invertTree(root.right);
        return root;
    }
```

4.剑指 Offer 32 - II. 从上到下打印二叉树 II

```java
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> result = new ArrayList<>();
        getResult(root, 0, result);
        return result;
    }

    public void getResult(TreeNode root, int k, List<List<Integer>> result) {
        if (root == null) return;
        if (k == result.size()) result.add(new ArrayList<Integer>());
        result.get(k).add(root.val);
        getResult(root.left, k + 1, result);
        getResult(root.right, k + 1, result);
    }
```

队列

```java
public List<List<Integer>> levelOrder(TreeNode root) {
    List<List<Integer>> result = new ArrayList<>();
    Queue<TreeNode> queue = new LinkedList<>();
    queue.offer(root);
    myLevelOrder(root, result, queue, 0);
    return result;
}

public void myLevelOrder(TreeNode root, List<List<Integer>> result, Queue<TreeNode> queue, int k) {
    if (root == null) return;
    if (k == result.size()) result.add(new ArrayList<Integer>());
    while (!queue.isEmpty()) {
        result.get(k).add(queue.poll().val);
    }
    if (root.left != null) queue.offer(root.left);
    if (root.right != null) queue.offer(root.right);
    myLevelOrder(root.left, result, queue, k + 1);
    myLevelOrder(root.right, result, queue, k + 1);
}
```

5.107. 二叉树的层序遍历 II

```
    public List<List<Integer>> levelOrderBottom(TreeNode root) {
        List<List<Integer>> result = new ArrayList<>();
        getResult(root, 0, result);
        Collections.reverse(result);
        return result;
    }

    public void getResult(TreeNode root, int k, List<List<Integer>> result) {
        if (root == null) return;
        if (k == result.size()) result.add(new ArrayList<Integer>());
        result.get(k).add(root.val);
        getResult(root.left, k + 1, result);
        getResult(root.right, k + 1, result);
    }

```

6.103. 二叉树的锯齿形层序遍历

```
    public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
        List<List<Integer>> result = new ArrayList<>();
        getResult(root, 0, result);
        for (int i = 1; i < result.size(); i += 2) {
            Collections.reverse(result.get(i));
        }
        return result;

    }

    public void getResult(TreeNode root, int k, List<List<Integer>> result) {
        if (root == null) return;
        if (k == result.size()) result.add(new ArrayList<Integer>());
        result.get(k).add(root.val);
        getResult(root.left, k + 1, result);
        getResult(root.right, k + 1, result);
    }
```

7.110. 平衡二叉树

```
    public boolean isBalanced(TreeNode root) {
        return getHeight(root) >= 0;
    }

    public int getHeight(TreeNode root) {
        if (root == null) return 0;
        int leftHeight = getHeight(root.left);
        int rightHeight = getHeight(root.right);
        if (leftHeight < 0 || rightHeight < 0) return -2;
        if (Math.abs(leftHeight - rightHeight) > 1) return -2;
        return Math.max(leftHeight, rightHeight) + 1;
    }
```

彩蛋

第五周作业彩蛋（Java学科）

题目描述：将数字1-10000（包含1和10000），奇数入栈，偶数入队列。然后将栈和队列中元素分别出栈和出队，将依次出栈和出队的数字分别两两相乘，然后将乘积累加求和，最后将和除以18，即为答案。