二叉树的遍历

1.前序遍历：(右子树先入栈，最后出栈)

```java
//循环
public List<Integer> preorderTraversal(TreeNode root) {
    ArrayList<Integer> ans = new ArrayList<Integer>();
    if( null == root )
        return ans;
    Stack<TreeNode> stack = new Stack<TreeNode>();
    stack.push(root);
    while( !stack.isEmpty() ){
        TreeNode node = stack.pop();  //1.访问节点
        ans.add(node.val);
        if( null != node.right )
            stack.push(node.right);   //2.先将右孩子入栈
        if( null != node.left )
            stack.push(node.left);    //3.最后将左孩子入栈
    }
    return ans;
}
```

2.中序遍历： 将该节点左子树全部入栈，然后(访问栈点,并将该节点右孩子及左子树全部入栈)。

```java
//循环
public List<Integer> inorderTraversal(TreeNode root) {
    ArrayList<Integer> ans = new ArrayList<Integer>();
    if( null == root )
        return ans;
    Stack<TreeNode> stack = new Stack<TreeNode>();
    TreeNode node = root;
    while( null != node ){ //1.先将左子树全部入站
        stack.push(node);
        node = node.left;
    }
    while(!stack.isEmpty()){
        node = stack.pop();
        ans.add(node.val);          //2. 访问节点
        if( null != node.right ){  //3.将该节点的右孩子及其左子书入栈(体现递归)
            node = node.right;
            while( null != node ){
                stack.push(node);
                node = node.left;
            }
        }
    }
    return ans;
}
```

3.后序遍历：从pre记录之前访问过的节点,如果该节点的左右孩子已经访问了，则可以访问该节点。

```java
//循环:
public List<Integer> postorderTraversal(TreeNode root) {
    ArrayList<Integer> ans = new ArrayList<Integer>();
    if( null == root )
        return ans;
    Stack<TreeNode> stack = new Stack<TreeNode>();
    TreeNode node = root;
    stack.push(node);
    TreeNode cur = null;
    TreeNode pre = null;
    while( !stack.isEmpty() ){
        cur = stack.peek();//访问栈顶元素,但不出栈
        if( (null == cur.left && null == cur.right) ||
            ( null != pre && ( pre == cur.left || pre == cur.right ) )){
                //1.该节点为叶子节点
                //2.上一次访问的是左孩子(说明没有右孩子)
                //3.上一次访问的是右孩子
                pre = stack.pop();//出栈并访问
                ans.add(pre.val);
            }
        else{
            if( null != cur.right )
                stack.push(cur.right);
            if( null != cur.left )
                stack.push(cur.left);
        }
    }
    return ans;
}
```

4.递归：

```java
//递归：
private void order( TreeNode node, ArrayList<Integer> ans ){
    if( null == node )
      return ;
    ans.add(node.val); //前序
    order(node.left, ans);
    ans.add(node.val); //中序
    order(node.right, ans);
    ans.add(node.val); //后续
}
```