

```
26 class BinaryTreeTraversal {
27     /**
28      * @param root 树根节点
29      * 递归先序遍历
30      */
31     public static void preOrderRec(Node root){
32         if(root!=null){
33             System.out.println(root.value);
34             preOrderRec(root.left);
35             preOrderRec(root.right);
36         }
37     }
38     /**
39      * @param root 树根节点
40      * 递归中序遍历
41      */
42     public static void inOrderRec(Node root){
43         if(root!=null){
44             preOrderRec(root.left);
45             System.out.println(root.value);
46             preOrderRec(root.right);
47         }
48     }
49     /**
50      * @param root 树根节点
51      * 递归后序遍历
52      */
53     public static void postOrderRec(Node root){
54         if(root!=null){
55             preOrderRec(root.left);
56             preOrderRec(root.right);
57             System.out.println(root.value);
58         }
59     }
}
```

```

/**
 *
 * @param root 树根节点
 * 利用栈实现循环先序遍历二叉树
 * 这种实现类似于图的深度优先遍历 (DFS)
 * 维护一个栈，将根节点入栈，然后只要栈不为空，出栈并访问，接着依次将访问节点的右节
 * 这种方式应该是对先序遍历的一种特殊实现（看上去简单明了），但是不具备很好的扩展性，
 */
public static void preOrderStack_1(Node root){
    if(root==null)return;
    Stack<Node> s=new Stack<Node>();
    s.push(root);
    while(!s.isEmpty()){
        Node temp=s.pop();
        System.out.println(temp.value);
        if(temp.right!=null) s.push(temp.right);
        if(temp.left!=null) s.push(temp.left);
    }
}

/**
 *
 * @param root 树的根节点
 * 利用栈模拟递归过程实现循环先序遍历二叉树
 * 这种方式具备扩展性，它模拟递归的过程，将左子树点不断的压入栈，直到null，然后处理
 */
public static void preOrderStack_2(Node root){
    if(root==null)return;
    Stack<Node> s=new Stack<Node>();
    while(root!=null||!s.isEmpty()){
        while(root!=null){
            System.out.println(root.value);
            s.push(root);//先访问再入栈
            root=root.left;
        }
    }
}

```

```
84     */
85     public static void preOrderStack_2(Node root){
86         if(root==null)return;
87         Stack<Node> s=new Stack<Node>();
88         while(root!=null||!s.isEmpty()){
89             while(root!=null){
90                 System.out.println(root.value);
91                 s.push(root);//先访问再入栈
92                 root=root.left;
93             }
94             root=s.pop();
95             root=root.right;//如果是null，出栈并处理右子树
96         }
97     }
98     /**
```

```

90  * @param root 树根节点
91  * 利用栈模拟递归过程实现循环中序遍历二叉树
92  * 思想和上面的preOrderStack_2相同，只是访问的时间是在左子树都处理完直到null的时候
93  */
94  public static void inOrderStack(Node root){
95      if(root==null)return;
96      Stack<Node> s=new Stack<Node>();
97      while(root!=null||!s.isEmpty()){
98          while(root!=null){
99              s.push(root);//先访问再入栈
100             root=root.left;
101         }
102         root=s.pop();
103         System.out.println(root.value);
104         root=root.right;//如果是null，出栈并处理右子树
105     }
106 }
107 /**
108  *
109  * @param root 树根节点
110  * 后序遍历不同于先序和中序，它是要先处理完左右子树，然后再处理根(回溯)，所以需要一
111  */
112 public static void postOrderStack(Node root){
113     if(root==null)return;
114     Stack<Node> s=new Stack<Node>();
115     Map<Node,Boolean> map=new HashMap<Node,Boolean>();
116     s.push(root);
117     while(!s.isEmpty()){
118         Node temp=s.peek();
119         if(temp.left!=null&&!map.containsKey(temp.left)){
120             temp=temp.left;
121             while(temp!=null){
122                 if(map.containsKey(temp))break;
123                 else s.push(temp);
124                 temp=temp.left;

```

```

0         temp=temp.left;
1         while(temp!=null){
2             if(map.containsKey(temp))break;
3             else s.push(temp);
4             temp=temp.left;
5         }
6         continue;
7     }
8     if(temp.right!=null&&!map.containsKey(temp.right)){
9         s.push(temp.right);
10        continue;
11    }
12
13    Node t=s.pop();
14    map.put(t,true);
15    System.out.println(t.value);
16
17    }
18
19    }
20
21    /**
22     *
23     * @param root 树根节点
24     * 层序遍历二叉树，用队列实现，先将根节点入队列，只要队列不为空，然后出队列，并访问。
25     */
26    public static void levelTravel(Node root){
27        if(root==null)return;
28        Queue<Node> q=new LinkedList<Node>();
29        q.add(root);
30        while(!q.isEmpty()){
31            Node temp = q.poll();
32            System.out.println(temp.value);
33            if(temp.left!=null)q.add(temp.left);
34            if(temp.right!=null)q.add(temp.right);
35        }
36    }
37
38    }
39
40    }

```

后序遍历的思想：

```
1 public static ArrayList postOrder1(TreeNode root){
2     ArrayList alist = new ArrayList();
3     Stack<TreeNode> stack = new Stack<TreeNode>();
4     if(root == null)
5         return alist;
6     TreeNode cur,pre = null;
7     stack.push(root);
8     while(!stack.empty()){
9         cur = stack.peek();
10        if((cur.left == null && cur.right == null) || (pre != null &&
11            cur.val == pre.val)){
12            TreeNode temp = stack.pop();
13            alist.add(temp.val);
14            pre = temp;
15        }
16        else{
17            if(cur.right != null)
18                stack.push(cur.right);
19            if(cur.left != null)
20                stack.push(cur.left);
21        }
22    }
23    return alist;
}
```