第 3 章

二叉树问题

分别用递归和非递归方式实现二叉树先序、中序和后序遍历

【题目】

用递归和非递归方式，分别按照二叉树先序、中序和后序打印所有的节点。我们约定：先序遍历顺序为根、左、右；中序遍历顺序为左、根、右；后序遍历顺序为左、右、根。

【难度】

校 ★★★☆

【解答】

用递归方式实现三种遍历是教材上的基础内容，本书不再详述，直接给出代码实现。先序遍历的递归实现请参看如下代码中的 preOrderRecur 方法。

```
public class Node {
        public int value;
        public Node left;
        public Node right;

        public Node(int data) {
                this.value = data;
        }
}

public void preOrderRecur(Node head) {
        if (head == null) {
```

```
            return;
        }
        System.out.print(head.value + " ");
        preOrderRecur(head.left);
        preOrderRecur(head.right);
}
```

中序遍历的递归实现请参看如下代码中的 inOrderRecur 方法。

```
public void inOrderRecur(Node head) {
        if (head == null) {
            return;
        }
        inOrderRecur(head.left);
        System.out.print(head.value + " ");
        inOrderRecur(head.right);
}
```

后序遍历的递归实现请参看如下代码中的 posOrderRecur 方法。

```
public void posOrderRecur(Node head) {
        if (head == null) {
            return;
        }
        posOrderRecur(head.left);
        posOrderRecur(head.right);
        System.out.print(head.value + " ");
}
```

用递归方法解决的问题都能用非递归的方法实现。这是因为递归方法无非就是利用函数栈来保存信息，如果用自己申请的数据结构来代替函数栈，也可以实现相同的功能。

用非递归的方式实现二叉树的先序遍历，具体过程如下：

1．申请一个新的栈，记为 stack。然后将头节点 head 压入 stack 中。

2．从 stack 中弹出栈顶节点，记为 cur，然后打印 cur 节点的值，再将节点 cur 的右孩子（不为空的话）先压入 stack 中，最后将 cur 的左孩子（不为空的话）压入 stack 中。

3．不断重复步骤 2，直到 stack 为空，全部过程结束。

下面举例说明整个过程，一棵二叉树如图 3-1 所示。

节点 1 先入栈，然后弹出并打印。接下来先把节点 3 压入 stack，再把节点 2 压入，stack 从栈顶到栈底依次为 2，3。

节点 2 弹出并打印，把节点 5 压入 stack，再把节点 4 压入，stack 从栈顶到栈底为 4，5，3。

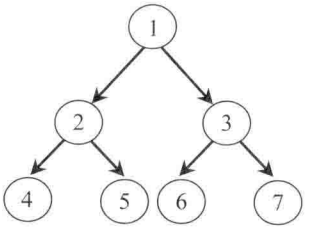

图 3-1

节点 4 弹出并打印，节点 4 没有孩子压入 stack，stack 从栈顶到栈底依次为 5，3。

节点 5 弹出并打印，节点 5 没有孩子压入 stack，stack 从栈顶到栈底依次为 3。

节点 3 弹出并打印，把节点 7 压入 stack，再把节点 6 压入，stack 从栈顶到栈底为 6，7。

节点 6 弹出并打印，节点 6 没有孩子压入 stack，stack 目前从栈顶到栈底为 7。

节点 7 弹出并打印，节点 7 没有孩子压入 stack，stack 已经为空，过程停止。

整个过程请参看如下代码中的 preOrderUnRecur 方法。

```
public void preOrderUnRecur(Node head) {
        System.out.print("pre-order: ");
        if (head != null) {
                Stack<Node> stack = new Stack<Node>();
                stack.add(head);
                while (!stack.isEmpty()) {
                        head = stack.pop();
                        System.out.print(head.value + " ");
                        if (head.right != null) {
                                stack.push(head.right);
                        }
                        if (head.left != null) {
                                stack.push(head.left);
                        }
                }
        }
        System.out.println();
}
```

用非递归的方式实现二叉树的中序遍历，具体过程如下：

1．申请一个新的栈，记为 stack。初始时，令变量 cur=head。

2．先把 cur 节点压入栈中，对以 cur 节点为头的整棵子树来说，依次把左边界压入栈中，即不停地令 cur=cur.left，然后重复步骤 2。

3．不断重复步骤 2，直到发现 cur 为空，此时从 stack 中弹出一个节点，记为 node。打印 node 的值，并且让 cur=node.right，然后继续重复步骤 2。

4．当 stack 为空且 cur 为空时，整个过程停止。

还是用图 3-1 的例子来说明整个过程。

初始时 cur 为节点 1，将节点 1 压入 stack，令 cur=cur.left，即 cur 变为节点 2。(步骤 1+步骤 2)

cur 为节点 2，将节点 2 压入 stack，令 cur=cur.left，即 cur 变为节点 4。(步骤 2)

cur 为节点 4，将节点 4 压入 stack，令 cur=cur.left，即 cur 变为 null，此时 stack 从栈顶到栈底为 4，2，1。(步骤 2)

cur 为 null，从 stack 弹出节点 4(node)并打印，令 cur=node.right，即 cur 为 null，此时 stack 从栈顶到栈底为 2，1。(步骤 3)

cur 为 null，从 stack 弹出节点 2(node)并打印，令 cur=node.right，即 cur 变为节点 5，此时 stack 从栈顶到栈底为 1。(步骤 3)

cur 为节点 5，将节点 5 压入 stack，令 cur=cur.left，即 cur 变为 null，此时 stack 从栈顶到栈底为 5，1。(步骤 2)

cur 为 null，从 stack 弹出节点 5(node)并打印，令 cur=node.right，即 cur 仍为 null，此时 stack 从栈顶到栈底为 1。(步骤 3)

cur 为 null，从 stack 弹出节点 1(node)并打印，令 cur=node.right，即 cur 变为节点 3，此时 stack 为空。(步骤 3)

cur 为节点 3，将节点 3 压入 stack，令 cur=cur.left 即 cur 变为节点 6；此时 stack 从栈顶到栈底为 3。(步骤 2)

cur 为节点 6，将节点 6 压入 stack，令 cur=cur.left 即 cur 变为 null，此时 stack 从栈顶到栈底为 6，3。(步骤 2)

cur 为 null，从 stack 弹出节点 6(node)并打印，令 cur=node.right，即 cur 仍为 null，此时 stack 从栈顶到栈底为 3。(步骤 3)

cur 为 null，从 stack 弹出节点 3(node)并打印，令 cur=node.right，即 cur 变为节点 7，此时 stack 为空。(步骤 3)

cur 为节点 7，将节点 7 压入 stack，令 cur=cur.left，即 cur 变为 null，此时 stack 从栈顶到栈底为 7。(步骤 2)

cur 为 null，从 stack 弹出节点 7(node)并打印，令 cur=node.right，即 cur 仍为 null，此时 stack 为空。(步骤 3)

cur 为 null，stack 也为空，整个过程停止。(步骤 4)

通过与例子结合的方式我们发现，步骤 1 到步骤 4 就是依次先打印左子树，然后是每

棵子树的头节点，最后打印右子树。

全部过程请参看如下代码中的 inOrderUnRecur 方法。

```
public void inOrderUnRecur(Node head) {
        System.out.print("in-order: ");
        if (head != null) {
                Stack<Node> stack = new Stack<Node>();
                while (!stack.isEmpty() || head != null) {
                        if (head != null) {
                                stack.push(head);
                                head = head.left;
                        } else {
                                head = stack.pop();
                                System.out.print(head.value + " ");
                                head = head.right;
                        }
                }
        }
        System.out.println();
}
```

用非递归的方式实现二叉树的后序遍历有点麻烦，本书实现两种方法供读者参考。

先介绍用两个栈实现后序遍历的过程，具体过程如下：

1. 申请一个栈，记为 s1，然后将头节点 head 压入 s1 中。
2. 从 s1 中弹出的节点记为 cur，然后依次将 cur 的左孩子和右孩子压入 s1 中。
3. 在整个过程中，每一个从 s1 中弹出的节点都放进 s2 中。
4. 不断重复步骤 2 和步骤 3，直到 s1 为空，过程停止。
5. 从 s2 中依次弹出节点并打印，打印的顺序就是后序遍历的顺序。

还是用图 3-1 的例子来说明整个过程。

节点 1 放入 s1 中。

从 s1 中弹出节点 1，节点 1 放入 s2，然后将节点 2 和节点 3 依次放入 s1，此时 s1 从栈顶到栈底为 3，2；s2 从栈顶到栈底为 1。

从 s1 中弹出节点 3，节点 3 放入 s2，然后将节点 6 和节点 7 依次放入 s1，此时 s1 从栈顶到栈底为 7，6，2；s2 从栈顶到栈底为 3，1。

从 s1 中弹出节点 7，节点 7 放入 s2，节点 7 无孩子节点，此时 s1 从栈顶到栈底为 6，2；s2 从栈顶到栈底为 7，3，1。

从 s1 中弹出节点 6，节点 6 放入 s2，节点 6 无孩子节点，此时 s1 从栈顶到栈底为 2；s2 从栈顶到栈底为 6，7，3，1。

从 s1 中弹出节点 2，节点 2 放入 s2，然后将节点 4 和节点 5 依次放入 s1，此时 s1 从

栈顶到栈底为 5，4；s2 从栈顶到栈底为 2，6，7，3，1。

从 s1 中弹出节点 5，节点 5 放入 s2，节点 5 无孩子节点，此时 s1 从栈顶到栈底为 4；s2 从栈顶到栈底为 5，2，6，7，3，1。

从 s1 中弹出节点 4，节点 4 放入 s2，节点 4 无孩子节点，此时 s1 为空；s2 从栈顶到栈底为 4，5，2，6，7，3，1。

过程结束，此时只要依次弹出 s2 中的节点并打印即可，顺序为 4，5，2，6，7，3，1。

通过如上过程我们知道，每棵子树的头节点都最先从 s1 中弹出，然后把该节点的孩子节点按照先左再右的顺序压入 s1，那么从 s1 弹出的顺序就是先右再左，所以从 s1 中弹出的顺序就是中、右、左。然后，s2 重新收集的过程就是把 s1 的弹出顺序逆序，所以 s2 从栈顶到栈底的顺序就变成了左、右、中。

使用两个栈实现后序遍历的全部过程请参看如下代码中的 posOrderUnRecur1 方法。

```
public void posOrderUnRecur1(Node head) {
        System.out.print("pos-order: ");
        if (head != null) {
                Stack<Node> s1 = new Stack<Node>();
                Stack<Node> s2 = new Stack<Node>();
                s1.push(head);
                while (!s1.isEmpty()) {
                        head = s1.pop();
                        s2.push(head);
                        if (head.left != null) {
                                s1.push(head.left);
                        }
                        if (head.right != null) {
                                s1.push(head.right);
                        }
                }
                while (!s2.isEmpty()) {
                        System.out.print(s2.pop().value + " ");
                }
        }
        System.out.println();
}
```

最后介绍只用一个栈实现后序遍历的过程，具体过程如下：

1．申请一个栈，记为 stack，将头节点压入 stack，同时设置两个变量 h 和 c。在整个流程中，h 代表最近一次弹出并打印的节点，c 代表 stack 的栈顶节点，初始时 h 为头节点，c 为 null。

2．每次令 c 等于当前 stack 的栈顶节点，但是不从 stack 中弹出，此时分以下三种情况。

① 如果 c 的左孩子不为 null，并且 h 不等于 c 的左孩子，也不等于 c 的右孩子，则把

c 的左孩子压入 stack 中。具体解释一下这么做的原因，首先 h 的意义是最近一次弹出并打印的节点，所以如果 h 等于 c 的左孩子或者右孩子，说明 c 的左子树与右子树已经打印完毕，此时不应该再将 c 的左孩子放入 stack 中。否则，说明左子树还没处理过，那么此时将 c 的左孩子压入 stack 中。

② 如果条件①不成立，并且 c 的右孩子不为 null，h 不等于 c 的右孩子，则把 c 的右孩子压入 stack 中。含义是如果 h 等于 c 的右孩子，说明 c 的右子树已经打印完毕，此时不应该再将 c 的右孩子放入 stack 中。否则，说明右子树还没处理过，此时将 c 的右孩子压入 stack 中。

③ 如果条件①和条件②都不成立，说明 c 的左子树和右子树都已经打印完毕，那么从 stack 中弹出 c 并打印，然后令 h=c。

3．一直重复步骤 2，直到 stack 为空，过程停止。

依然用图 3-1 的例子来说明整个过程。

节点 1 压入 stack，初始时 h 为节点 1，c 为 null，stack 从栈顶到栈底为 1。

令 c 等于 stack 的栈顶节点——节点 1，此时步骤 2 的条件①命中，将节点 2 压入 stack，h 为节点 1，stack 从栈顶到栈底为 2，1。

令 c 等于 stack 的栈顶节点——节点 2，此时步骤 2 的条件①命中，将节点 4 压入 stack，h 为节点 1，stack 从栈顶到栈底为 4，2，1。

令 c 等于 stack 的栈顶节点——节点 4，此时步骤 2 的条件③命中，将节点 4 从 stack 中弹出并打印，h 变为节点 4，stack 从栈顶到栈底为 2，1。

令 c 等于 stack 的栈顶节点——节点 2，此时步骤 2 的条件②命中，将节点 5 压入 stack，h 为节点 4，stack 从栈顶到栈底为 5，2，1。

令 c 等于 stack 的栈顶节点——节点 5，此时步骤 2 的条件③命中，将节点 5 从 stack 中弹出并打印，h 变为节点 5，stack 从栈顶到栈底为 2，1。

令 c 等于 stack 的栈顶节点——节点 2，此时步骤 2 的条件③命中，将节点 2 从 stack 中弹出并打印，h 变为节点 2，stack 从栈顶到栈底为 1。

令 c 等于 stack 的栈顶节点——节点 1，此时步骤 2 的条件②命中，将节点 3 压入 stack，h 为节点 2，stack 从栈顶到栈底为 3，1。

令 c 等于 stack 的栈顶节点——节点 3，此时步骤 2 的条件①命中，将节点 6 压入 stack，h 为节点 2，stack 从栈顶到栈底为 6，3，1。

令 c 等于 stack 的栈顶节点——节点 6，此时步骤 2 的条件③命中，将节点 6 从 stack 中弹出并打印，h 变为节点 6，stack 从栈顶到栈底为 3，1。

令 c 等于 stack 的栈顶节点——节点 3，此时步骤 2 的条件②命中，将节点 7 压入 stack，h 为节点 6，stack 从栈顶到栈底为 7，3，1。

令 c 等于 stack 的栈顶节点——节点 7，此时步骤 2 的条件③命中，将节点 7 从 stack 中弹出并打印，h 变为节点 7，stack 从栈顶到栈底为 3，1。

令 c 等于 stack 的栈顶节点——节点 3，此时步骤 2 的条件③命中，将节点 3 从 stack 中弹出并打印，h 变为节点 3，stack 从栈顶到栈底为 1。

令 c 等于 stack 的栈顶节点——节点 1，此时步骤 2 的条件③命中，将节点 1 从 stack 中弹出并打印，h 变为节点 1，stack 为空。

过程结束。

只用一个栈实现后序遍历的全部过程请参看如下代码中的 posOrderUnRecur2 方法。

```
public void posOrderUnRecur2(Node h) {
        System.out.print("pos-order: ");
        if (h != null) {
                Stack<Node> stack = new Stack<Node>();
                stack.push(h);
                Node c = null;
                while (!stack.isEmpty()) {
                        c = stack.peek();
                        if (c.left != null && h != c.left && h != c.right) {
                                stack.push(c.left);
                        } else if (c.right != null && h != c.right) {
                                stack.push(c.right);
                        } else {
                                System.out.print(stack.pop().value + " ");
                                h = c;
                        }
                }
        }
        System.out.println();
}
```

打印二叉树的边界节点

【题目】

给定一棵二叉树的头节点 head，按照如下两种标准分别实现二叉树边界节点的逆时针打印。

标准一：

1．头节点为边界节点。