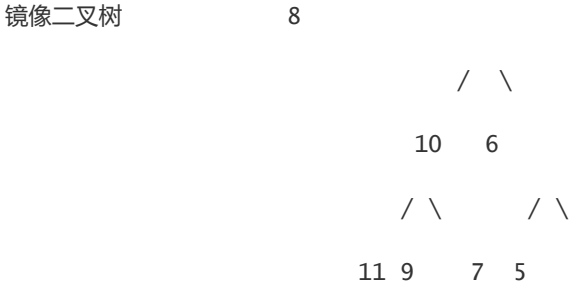
题目 二叉树的镜像

考点 面试思路 热点指数 69995 通过率 42.52%

具体题目

操作给定的二叉树，将其变换为源二叉树的镜像。

输入描述:二叉树的镜像定义：




```
public class Solution {
    public void Mirror(TreeNode root) {
        if(root != null){
            Mirror(root.left);
            Mirror(root.right);
            TreeNode temp = root.left;
            root.left=root.right;
            root.right = temp;
        }
    }
}
```

```
/* 先前序遍历这棵树的每个结点，如果遍历到的结点有子结点，就交换它的两个子节点，
当交换完所有的非叶子结点的左右子结点之后，就得到了树的镜像 */
/**
public class TreeNode {
    int val = 0;
    TreeNode left = null;
    TreeNode right = null;
    public TreeNode(int val) {
        this.val = val;
    }
}
*/
public class Solution {
    public void Mirror(TreeNode root) {
        if(root == null)
```

```
            return;
        if(root.left == null && root.right == null)
            return;


        TreeNode pTemp = root.left;
        root.left = root.right;
        root.right = pTemp;

        if(root.left != null)
            Mirror(root.left);
        if(root.right != null)
            Mirror(root.right);
        }
    }
```

```
import java.util.Stack;
public class Solution {
    public void Mirror(TreeNode root) {
        if(root == null){
            return;
        }
        Stack<TreeNode> stack = new Stack<TreeNode>();
        stack.push(root);
        while(!stack.isEmpty()){
            TreeNode node = stack.pop();
            if(node.left != null||node.right != null){
                TreeNode temp = node.left;
                node.left = node.right;
                node.right = temp;
            }
            if(node.left!=null){
                                stack.push(node.left);
            }
            if(node.right!=null){
                stack.push(node.right);
            }
        }
    }
}
```

题目描述 解题思路 我们或许还记得递归的终极思想是数学归纳法，我们思考递归的时候一定不要去一步一步看它执行了啥，只会更绕。我们牢牢记住，思考的方式是我们首先假设子问题都已经完美处理，我只需要处理一下最终的问题即可，子问题的处理方式与最终那个处理方式一样，但是问题规模一定要以1的进制缩小。最后给一个递归出口条件即可。 对于本题，首先假设root的左右子树已经都处理好了，即左子树自身已经镜像了，右子树自身也镜像了，那么最后一步就是交换左右子树，问题解决。所以我只需要将root.left和root.right交换即可。下面进入递归，就是处理子问题。子问题的处理方式与root一样，只是要缩小问题规模，所以只需要考虑root.left是什么情况，root.right是什么情况即可。 我的答案

```
public class Solution {
        public void Mirror(TreeNode root) {
            reverseTree(root);
        }
        private void reverseTree(TreeNode root){
            //为空则结束
            if(root == null){
                return;
            }
```

```
            //假设root两边的子树自己都已经翻转成功了，那么只需要再将左右子树互换一下就成功了
            //交换root的左右子树
            swap(root);
            //左右子树翻转自己去处理就行了，我们规定每个子树的root都跟最终的root处理方式一样即可
            reverseTree(root.left);
            reverseTree(root.right);
        }
        private void swap(TreeNode root){
            TreeNode node = null;
            node = root.left;
            root.left = root.right;
            root.right = node;
        }
    }
```

```
public class Solution {
    public void Mirror(TreeNode root) {
        if(root== null){
            return;
        }
        swap(root);
        Mirror(root.left);
        Mirror(root.right);
    }
    public void swap(TreeNode root){
        TreeNode node = null;
        node = root.left;
        root.left = root.right;
        root.right = node;
    }
}
```