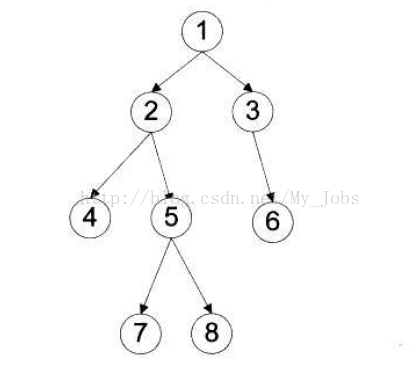
四种主要的遍历思想为：

前序遍历：根结点 ---> 左子树 ---> 右子树

中序遍历：左子树---> 根结点 ---> 右子树

后序遍历：左子树 ---> 右子树 ---> 根结点

层次遍历：只需按层次遍历即可



前序遍历：1  2  4  5  7  8  3  6

中序遍历：4  2  7  5  8  1  3  6

后序遍历：4  7  8  5  2  6  3  1

层次遍历：1  2  3  4  5  6  7  8

# 前序遍历：（深度优先遍历）

1. 递归版本

public void preOrder(TreeNode root) {  
 if(root == null){  
 return;  
 }  
 System.*out*.println(root.val);  
 preOrder(root.left);  
 preOrder(root.right);  
}

1. 迭代版本

if(root == null){  
 return;  
}  
Stack<TreeNode> stack = new Stack<>();  
TreeNode node = root;  
while (node!=null || !stack.isEmpty()){  
 if(node!=null){  
 //先序遍历，直接访问当前结点  
 System.*out*.println(node.val);  
 //把当前结点放入栈中，以便访问右子树  
 stack.push(node);  
 //先->左->右，访问完当前结点，然后访问左子树  
 node = node.left;  
 }else {  
 //返回上一层，访问右子树  
 node = stack.pop();  
 node = node.right;  
 }  
}

# 中序遍历

1. 递归版本

public void midOrder(TreeNode root) {  
 if (root != null) {  
 midOrder(root.left);  
 System.*out*.print(root.val+" ");  
 midOrder(root.right);  
 }  
}

2） 迭代版本

public void midOrder2(TreeNode root) {  
 LinkedList<TreeNode> stack = new LinkedList<>();  
 TreeNode node = root;  
 while (node != null || !stack.isEmpty()){  
 //当前树不为空，先遍历左子树  
 if(node != null){  
 stack.push(node);  
 node = node.left;  
 }else {  
 //栈中记录了当前子树和左子树，弹出后先访问,访问完访问右子树  
 TreeNode cur = stack.pop();  
 System.*out*.println(cur.val);  
 node = cur.right;  
 }  
 }  
}

# 后序遍历

1. 递归版本

public void afterOrder(TreeNode root) {  
 if (root != null) {  
 midOrder(root.left);  
 midOrder(root.right);  
 System.*out*.print(root.val+" ");  
 }  
}

1. 迭代版本

# 层序遍历（广度优先遍历）

1. 层序遍历（从上往下遍历），如果是从下往上，其实就是从上往下的结果的逆序。

public void afterOrder2(TreeNode root) {  
 if (root == null) {  
 return;  
 }  
 //队列实现层序遍历  
 LinkedList<TreeNode> queue = new LinkedList<>();  
 //尾插入根节点  
 queue.offer(root);  
 while (!queue.isEmpty()) {  
 int curLevelSize = queue.size();  
 for (int i = 0; i < curLevelSize; i++) {  
 //先访问当前层，再访问下一层  
 TreeNode node = queue.pop();  
 System.*out*.println(node.val);  
 if(node.left != null) {  
 queue.push(node.left);  
 }  
 if(node.right != null){  
 queue.push(node.right);  
 }  
 }  
  
 }  
}