# 257 二叉树的所有路径

Label：二叉树

给定一个二叉树，返回所有从根节点到叶子节点的路径。
说明：叶子节点是指没有子节点的节点。

输入：

```
   1
 /   \
2     3
 \
  5
```

输出：["1->2->5", "1->3"]

解释：所有根节点到叶子节点的路径为：1->2->5, 1->3

- 深度优先

```
class Solution {

    public List<String> binaryTreePaths(TreeNode root) {
        List<String> paths = new ArrayList<String>();
        constructPaths(root, "", paths);
        return paths;
    }

    public void constructPaths(TreeNode root, String path, List<String> paths) {
 // 看参数的设计
        if (root != null) {
            StringBuffer pathSB = new StringBuffer(path);  // 每个节点都构造一个
（核心思想）
            pathSB.append(Integer.toString(root.val));

            if (root.left == null && root.right == null) {  // 当前节点是叶子节点
                paths.add(pathSB.toString());  // 把路径加入到答案中  ，  不需要加连接
符
            } else {
                pathSB.append("->");  // 当前节点不是叶子节点，继续递归遍历
                constructPaths(root.left, pathSB.toString(), paths);
                constructPaths(root.right, pathSB.toString(), paths);
            }
        }
    }
}
```

- 广度优先

```
class Solution {
    public List<String> binaryTreePaths(TreeNode root) {
        List<String> paths = new ArrayList<String>();
        if (root == null) {
            return paths;
        }

        Queue<TreeNode> nodeQueue = new LinkedList<TreeNode>();
        Queue<String> pathQueue = new LinkedList<String>();

        nodeQueue.offer(root);
        pathQueue.offer(Integer.toString(root.val));

        while (!nodeQueue.isEmpty()) {
            TreeNode node = nodeQueue.poll();
            String path = pathQueue.poll();

            if (node.left == null && node.right == null) {
                paths.add(path); // 叶子节点
            } else {  // 如果不是叶子节点，则将其孩子存入队列中
                if (node.left != null) {
                    nodeQueue.offer(node.left);
                    pathQueue.offer(new StringBuffer(path).append("-
>").append(node.left.val).toString());
                }

                if (node.right != null) {
                    nodeQueue.offer(node.right);
                    pathQueue.offer(new StringBuffer(path).append("-
>").append(node.right.val).toString());
                }
            }
        }
        return paths;
    }
}
```