基本解法(迭代)

Java代码

```
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> res = new ArrayList<Integer>();// 记录目标序列
        Deque<TreeNode> stk = new LinkedList<TreeNode>();// 显式模拟栈
       // 判断结点是否被访问完
       // 处理二叉树为空的特殊情况
        while (root != null || !stk.isEmpty()) {
            // 根据中序遍历顺序，第一个结点是一棵树的最左边的结点
            while (root != null) {
                stk.push(root);
                root = root.left;//
            }
            root = stk.pop();  // 出栈
            res.add(root.val); // 更新目标序列
            root = root.right; // 根据中序遍历顺序，根节点之后应该遍历右结点
        }
        return res;
    }
}
```

优化解法(递归)

Java代码

```
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> res = new ArrayList<Integer>();
        inorder(root, res);
        return res;
    }

    public void inorder(TreeNode root, List<Integer> res) {
        if (root == null) {
            return;
        }
        inorder(root.left, res);
        res.add(root.val);
        inorder(root.right, res);
    }
```

```
}
```

最优解法(莫里斯遍历)

Java代码

```java
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> res = new ArrayList<Integer>();
        TreeNode exPoint = null;
        while (root != null) {
            if (root.left != null) {
                exPoint = root.left;// exPoint 节点就是当前 root 节点向左走一步，然后一直向右走至无法走为止
                while (exPoint.right != null && exPoint.right != root) {
                    exPoint = exPoint.right;
                }
                if (exPoint.right == null) {
                    exPoint.right = root; // 让 exPoint 的右指针指向 root，继续遍历左子树
                    root = root.left;
                } else { // 说明左子树已经访问完了，我们需要断开链接
                    res.add(root.val);
                    exPoint.right = null;
                    root = root.right;
                }
            } else { // 如果没有左孩子，则直接访问右孩子
                res.add(root.val);
                root = root.right;
            }
        }
        return res;
    }
}
```

C++代码

```cpp
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> res;
        TreeNode *predecessor = nullptr;

        while (root != nullptr) {
            if (root->left != nullptr) {
                // predecessor 节点就是当前 root 节点向左走一步，然后一直向右走至无法走为止
```

```
                predecessor = root->left;
                while (predecessor->right != nullptr && predecessor->right !=
root) {
                    predecessor = predecessor->right;
                }

                // 让 predecessor 的右指针指向 root，继续遍历左子树
                if (predecessor->right == nullptr) {
                    predecessor->right = root;
                    root = root->left;
                }
                // 说明左子树已经访问完了，我们需要断开链接
                else {
                    res.push_back(root->val);
                    predecessor->right = nullptr;
                    root = root->right;
                }
            }
            // 如果没有左孩子，则直接访问右孩子
            else {
                res.push_back(root->val);
                root = root->right;
            }
        }
        return res;
    }
};
```

Python代码

```
class Solution(object):
    def inorderTraversal(self, root):
        """
        :type root: TreeNode
        :rtype: List[int]
        """
        result = []
        current = root
        while current:
            if not current.left:
                result.append(current.val)
                current = current.right
            else:
                pre = current.left
                while pre.right and pre.right != current:
                    pre = pre.right
                if not pre.right:
                    pre.right = current
```

```
                    current = current.left
                else:
                    pre.right = None
                    result.append(current.val)
                    current =current.right
        return result
```