面试题36：二叉搜索树与双向链表

题目描述

输入一棵二叉搜索树，将该二叉搜索树转换成一个排序的双向链表。要求不能创建任何新的结点，只能调整树中结点指针的指向。

解题思路

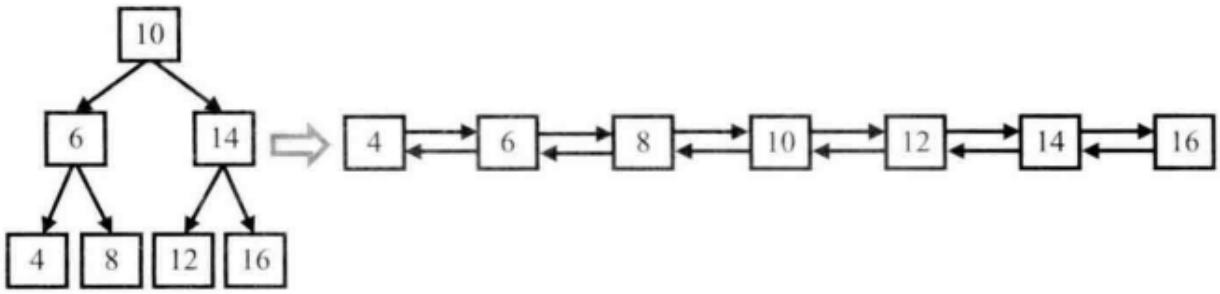

图 4.15　一棵二叉搜索树及转换之后的排序双向链表

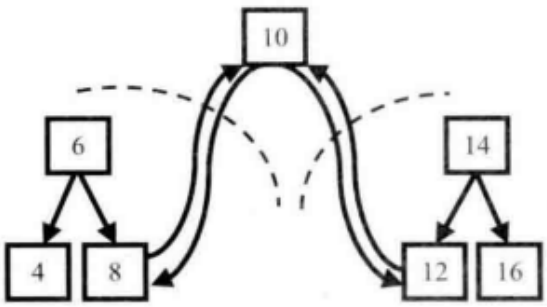

图 4.16　把二叉搜索树看成 3 部分

注：根节点、左子树和右子树。在把左、右子树都转换成排序双向链表之后再和根节点链接起来，整棵二叉搜索树也就转换成了排序双向链表。

C++

```
/*
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
    TreeNode(int x) :
            val(x), left(NULL), right(NULL) {
    }
};*/
class Solution {
public:
    TreeNode* Convert(TreeNode* pRootOfTree)
    {
        //判断根节点的指针为nullptr指针
        if(!pRootOfTree)
            return nullptr;
        //没有左子树的二叉树
        TreeNode* pPreNode = nullptr;
        ConvertCore(pRootOfTree, &pPreNode);
        TreeNode* pNewHead = pRootOfTree;
        while(pNewHead->left)
            pNewHead = pNewHead->left;
        return pNewHead;
    }
```

```
    void ConvertCore(TreeNode* pRootOfTree,TreeNode** pPreNode){
        if(pRootOfTree->left)
            ConvertCore(pRootOfTree->left,pPreNode);
        pRootOfTree->left = *pPreNode;
        if(*pPreNode)
            (*pPreNode)->right = pRootOfTree;
        *pPreNode = pRootOfTree;
        if(pRootOfTree->right)
            ConvertCore(pRootOfTree->right,pPreNode);
    }
};
```

Java

```
/**
public class TreeNode {
    int val = 0;
    TreeNode left = null;
    TreeNode right = null;

    public TreeNode(int val) {
        this.val = val;

    }
```

```
11
12  }
13  */
14  public class Solution {
15      public TreeNode Convert(TreeNode
    pRootOfTree) {
16          /**
17  方法二：递归版
18  解题思路:
19  1.将左子树构造成双链表，并返回链表头节点。
20  2.定位至左子树双链表最后一个节点。
21  3.如果左子树链表不为空的话，将当前root追加到左子树
    链表。
22  4.将右子树构造成双链表，并返回链表头节点。
23  5.如果右子树链表不为空的话，将该链表追加到root节点
    之后。
24  6.根据左子树链表是否为空确定返回的节点。
25          */
26          if(pRootOfTree==null)
27              return null;
28
     if(pRootOfTree.left==null&&pRootOfTree.righ
    t==null)
29              return pRootOfTree;
30          // 1.将左子树构造成双链表，并返回链表头节
    点
31          TreeNode left =
    Convert(pRootOfTree.left);
```

```
        TreeNode p = left;
        // 2.定位至左子树双链表最后一个节点
        while(p!=null&&p.right!=null){
            p = p.right;
        }
        // 3.如果左子树链表不为空的话，将当前root追加到左子树链表
        if(left!=null){
            p.right = pRootOfTree;
            pRootOfTree.left = p;
        }
        // 4.将右子树构造成双链表，并返回链表头节点
        TreeNode right = Convert(pRootOfTree.right);
        // 5.如果右子树链表不为空的话，将该链表追加到root节点之后
        if(right!=null){
            right.left = pRootOfTree;
            pRootOfTree.right = right;
        }
        return left!=null?left:pRootOfTree;
    }
}
```

Python 2.7.3

```
# -*- coding:utf-8 -*-
```

```python
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    def Convert(self, pRootOfTree):
        # write code here
        #先中序遍历，将所有的节点保存到一个列表中。
        #对这个list[:-1]进行遍历，每个节点的right设为下一个节点，下一个节点的left设为上一个节点。
        if not pRootOfTree:return
        self.arr = []
        self.midTraversal(pRootOfTree)
        for i,v in enumerate(self.arr[:-1]):
            v.right = self.arr[i + 1]
            self.arr[i + 1].left = v
        return self.arr[0]
    def midTraversal(self,root):
        if not root:return
        self.midTraversal(root.left)
        self.arr.append(root)
        self.midTraversal(root.right)
```