98. Validate Binary Search Tree

这个是我写的递归, 但是递归也可以多传两个值进去.

```
    public boolean isValidBST(TreeNode root) {
        if(root == null) return true;
        List<Integer> res = new ArrayList();
        validBst(root, res);
        for(int i = 1; i < res.size(); ++i){
            if(res.get(i - 1) >= res.get(i)) return false;
        }
        return true;

    }
    public void validBst(TreeNode node, List list){
        if(node != null){
            validBst(node.left,list);
            list.add(node.val);
            validBst(node.right,list);
        }
    }
```

我改的递归的inorder写法

```
    public boolean isValidBST(TreeNode root) {
        if(root == null) return true;
        List<Integer> res = new ArrayList();
        return validBst(root, res);


    }
    public boolean validBst(TreeNode node, List list){
        if(node != null){
            if(!validBst(node.left,list))
                return false;
            if(list.size() > 0){
                if ((int)list.get(list.size() - 1) >= node.val)
                    return false;
            }
            list.add(node.val);
            if(!validBst(node.right,list))
                return false;;
        }
        return true;
    }
//试试穿入上一次的值而不是用List存储
```

好的递归写法

```
public boolean isValidBST(TreeNode root) {
     return helper(root, null, null);
 }

 private boolean helper(TreeNode root, TreeNode min, TreeNode max) {
     if (root == null)
         return true;
     if ((min != null && root.val <= min.val) || (max != null && root.val >=
 max.val))
         return false;

     return helper(root.left, min, root) && helper(root.right, root, max);
 }
```

iterator版本

```
 public boolean iteratorHelper(TreeNode node){
     Stack<TreeNode> stack = new Stack();
     TreeNode previous = null;
     while(node != null || !stack.isEmpty()){
         while(node != null){
             stack.push(node);
             node = node.left;
         }
         node = stack.pop();
         if(previous != null && previous.val >= node.val)
             return false;
         previous = node;
         node = node.right;

     }
     return true;
 }
```