```
/*
给定一个二叉树，判断它是否是高度平衡的二叉树。
本题中，一棵高度平衡二叉树定义为：
    一个二叉树每个节点 的左右两个子树的高度差的绝对值不超过1。
示例 1:
给定二叉树 [3,9,20,null,null,15,7]
    3
   / \
  9  20
    /  \
   15   7
返回 true 。
示例 2:
给定二叉树 [1,2,2,3,3,null,null,4,4]

       1
      / \
     2   2
    / \
   3   3
  / \
 4   4
返回 false 。
来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/balanced-binary-tree
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
*/
```

分析:

- 方法一:维护一个对象的成员变量(ret_val),初始化时都赋值$true$,在先序遍历各个节点获取两个子树的深度的递归过程中如果发现两个子树的深度不符合高度平衡二叉树的定义,就把这个对象的成员变量赋值为$false$.

方法一:C++_递归先序遍历

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution
{
    private:

        bool ret_val    = true  ;
        int __subtreeDepth(TreeNode* node, int cur_depth)
        {
            int left_depth  = 0     ;
```

```
            int right_depth = 0     ;

            if(node==NULL)
            {
                return cur_depth;
            }
            /*如果已经判断出子树存在不平衡的情况就直接返回,不必再递归遍历了*/
            if(ret_val == false)
            {
                return -1;
            }

            /*获取左右子树深度*/
            left_depth  =   __subtreeDepth(node->left,cur_depth+1);
            right_depth =   __subtreeDepth(node->right,cur_depth+1);

            if(abs(left_depth - right_depth) > 1)
            {
                ret_val = false;
            }
            /*如果已经判断出子树存在不平衡的情况就直接返回,不必再递归遍历了*/
            if(ret_val == false)
            {
                return -1;
            }
            return left_depth>=right_depth?left_depth:right_depth;
        }
    public:
        bool isBalanced(TreeNode* root)
        {
            ret_val = true; /*每次被调用需要进行初始化*/
            __subtreeDepth(root,0);
            return ret_val;
        }
};

/*
执行结果：(https://leetcode-cn.com/submissions/detail/27313132/)
通过
显示详情
执行用时 :20 ms, 在所有 C++ 提交中击败了70.73% 的用户
内存消耗 :16.3 MB, 在所有 C++ 提交中击败了98.08%的用户
*/

```

AlimyBreak
2019.08.28