```
/*
给定一个单链表，其中的元素按升序排序，将其转换为高度平衡的二叉搜索树。

本题中，一个高度平衡二叉树是指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:

给定的有序链表： [-10, -3, 0, 5, 9],

一个可能的答案是：[0, -3, 9, -10, null, 5], 它可以表示下面这个高度平衡二叉搜索树：

      0
     / \
   -3   9
   /   /
 -10  5

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/convert-sorted-list-to-binary-search-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:
  - 链表原本有序,可以先遍历链表将链表元素全部存储到容器类型中去.
  - 然后每次构架搜索树时都把容器类型分为三部分,即left-mid-right三部分,mid部分只有一个元素,即容器中间的元素,用这个元素构建要返回的根节点,用left部分构建根节点的左子树,用right构建根节点的右子树.
  - 由于原始链表已经有序,递归用中间节点构造BST的方法能够保证任意节点的子树节点数总和相差不差过1.
- 方法二:
  - 根据方法一的思路,其实我们只要获取有序链表的中间节点即可,用中间节点构造要返回的树的根节点,用原始head到中间节点的前一个节点继续递归构造根节点的左子树,用中间节点的下一个节点到尾巴节点构造根节点的右子树.
  - 寻找链表的中间节点,可以使用快慢指法.
  - 在递归构造前,应当注意将中间节点的上一个节点的$next$域设置为$NULL$,以保证在递归调用时,链表已经被合理分割.
  - 缺点:分割链表会破坏原有的链表结构.

方法一:C++_容器法

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
```

```
 */
/**
 * 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:
        TreeNode* __sortedListToBST(vector<int>& vi , int left , int right)
        {
            if(left > right)
            {
                return NULL;
            }
            if(left == right)
            {
                return new TreeNode(vi[left]);
            }

            int mid = left + ( right - left) / 2;   /*防止假加法溢出*/
            TreeNode* tnp = new TreeNode(vi[mid]);
            tnp->left  = __sortedListToBST(vi,left,mid-1);
            tnp->right = __sortedListToBST(vi,mid+1,right);
            return tnp;
        }
    public:
        TreeNode* sortedListToBST(ListNode* head)
        {
            vector<int> vi;
            while(head)
            {
                vi.push_back(head->val);
                head = head->next;
            }

            return __sortedListToBST(vi,0,vi.size()-1);
        }
};
/*
执行结果:
通过
显示详情
执行用时 :36 ms, 在所有 C++ 提交中击败了92.29% 的用户
内存消耗 :24.7 MB, 在所有 C++ 提交中击败了20.10%的用户
*/
```

方法二:C++_快慢指针法

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
```

```
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
/**
 * 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
{

    TreeNode* __sortedListToBST(ListNode* head)
    {
        /*极端情况,输入没有节点或者只有一个节点*/
        if(head==NULL)
        {
            return NULL;
        }
        else if(head->next == NULL)
        {
            return new TreeNode(head->val);
        }

        ListNode* pre   =   head;
        ListNode* slow  =   pre->next   ;/*慢指针*/
        ListNode* fast  =   slow->next  ;/*快指针*/

        while(  (fast!=NULL)
              &&(fast->next!=NULL)
        )
        {
            pre     = slow; /*保存中间节点的前一个节点,用于链表截断*/
            slow    = slow->next;
            fast    = fast->next->next;
        }

        /*
            head...pre-->slow-->slow_next-->...
            截断到pre
            slow用来构造根节点返回
            head和slow_next分别用构造左子树和右子树.
        */
        pre->next = NULL;   /*截断链表*/
        TreeNode* tn = new TreeNode(slow->val);   /*用中间节点构造根节点*/
        tn->left  =   __sortedListToBST(head);
        tn->right =   __sortedListToBST(slow->next);
        return tn;
    }
    public:
        TreeNode* sortedListToBST(ListNode* head)
        {
            return __sortedListToBST(head);
```

```
        }
};
/*
执行结果：
通过
显示详情
执行用时 :56 ms, 在所有 C++ 提交中击败了43.52% 的用户
内存消耗 :24.3 MB, 在所有 C++ 提交中击败了89.95%的用户
缺点:破坏了原有链表结构.
*/
```

AlimyBreak
2019.09.30