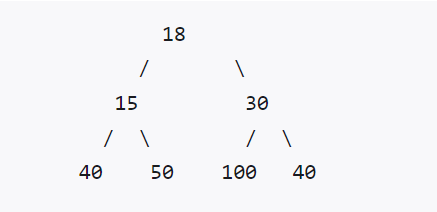
二叉搜索树是二叉树的一种，但是只允许你在左侧节点插入比父节点小的值，在右侧节点插入比父节点大的值。



在Js中我们可以借助Es6之后给我们提供的class类实现，Es6之后，给我们提供的class，可以快速生成一个对象”{}”（我们知道在程序中对象其实就是资料和方法的集合）。

构建一个基础的节点类



构建一个基础的节点类，用于保存节点的值和保存左右两侧节点的引用。

构建一个binarySearchTree

构建一个binarySearchTree，我们要考虑的是：

1. 如何向二叉搜索树中插入节点。
2. 二叉树的遍历（中序、先序遍历、后续遍历）。
3. 如何从二叉搜索树中找到指定节点。
4. 如何从二叉搜索树中找到并删除指定节点。
5. 搜索二叉搜索树中的最小值和最大值。

只有完成这些基础的功能才算是完构建出了一个二叉搜索树，后续我们在构建AVL自平衡树的时候就是以这个二叉搜索树为基础的。包括后续的红黑树。接下来我将一一介绍如何实现上面提到的5个部分。值得注意的是，有这5个部分的功能，有时候是互相以来的。

向二叉搜索树中插入一个key



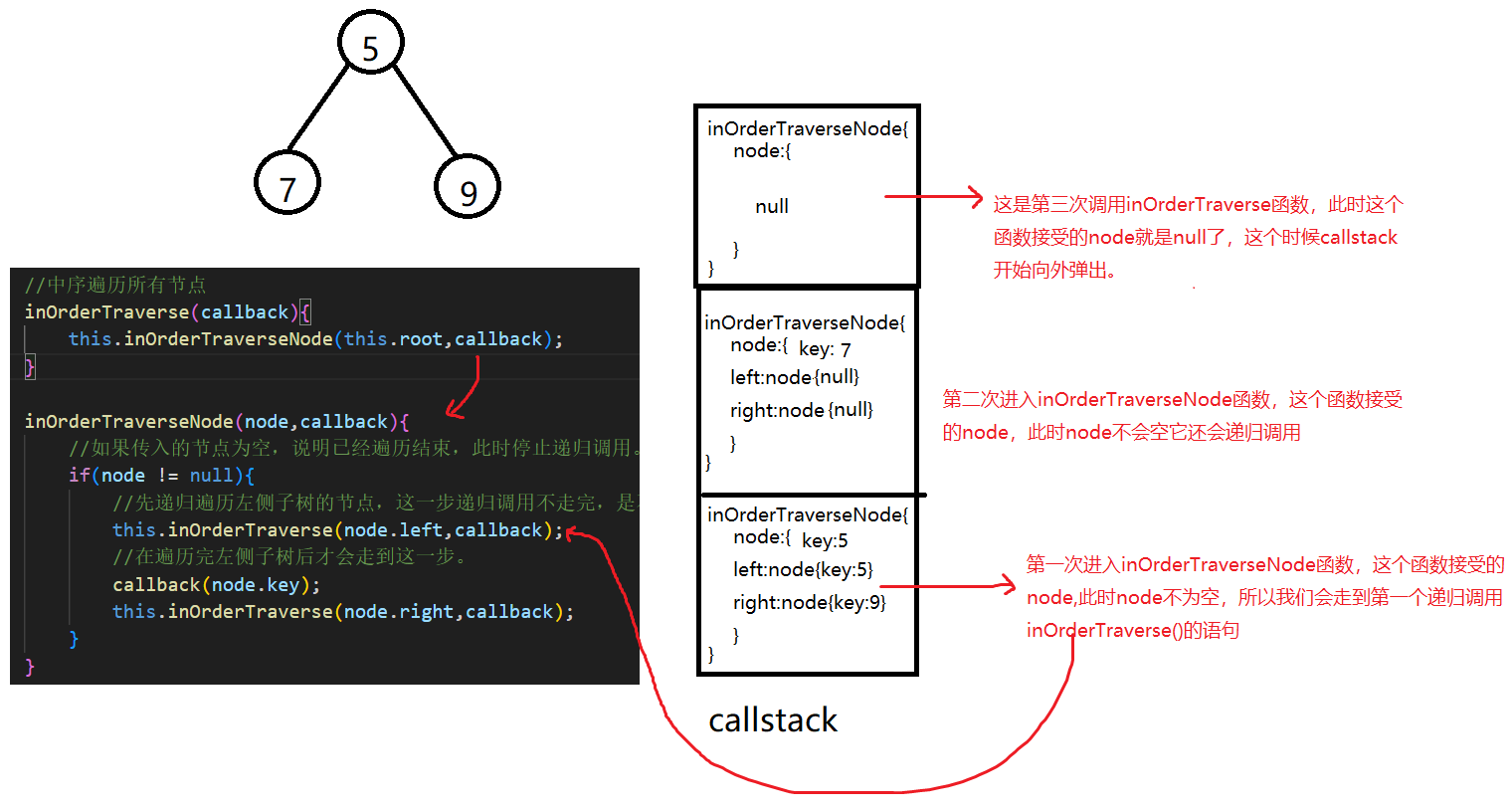
在根节点不为空的时候，我们就要判断这个key要插入到根节点的左侧还是右侧。但有可能根节点的左侧或右侧此时已经有节点了，所以我们要递归的调用insertNode方法直到找到要插入的位置并把节点插入进去位置。所以这个递归算法的停止条件就是node.left或者node.right为null的时，我们插入了新节点之后。

中序遍历所有节点

在正式介绍中序遍历如何实现之前，我们首先要知道什么是中序遍历。所谓中序遍历其实就是第先访问子节点，第二个访问根节点。

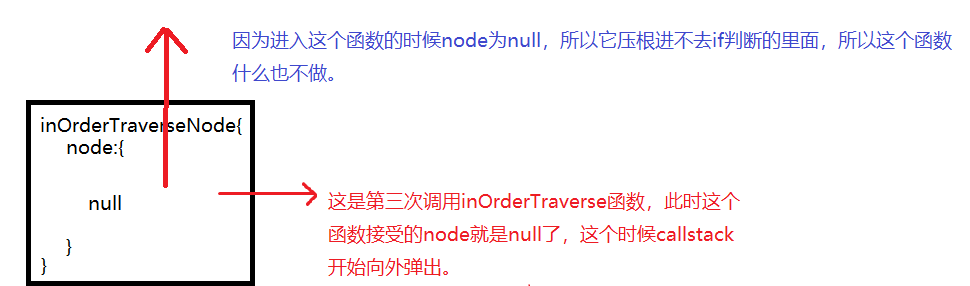


我们来找一个最简单的二叉搜索树详细分析这个递归算法。

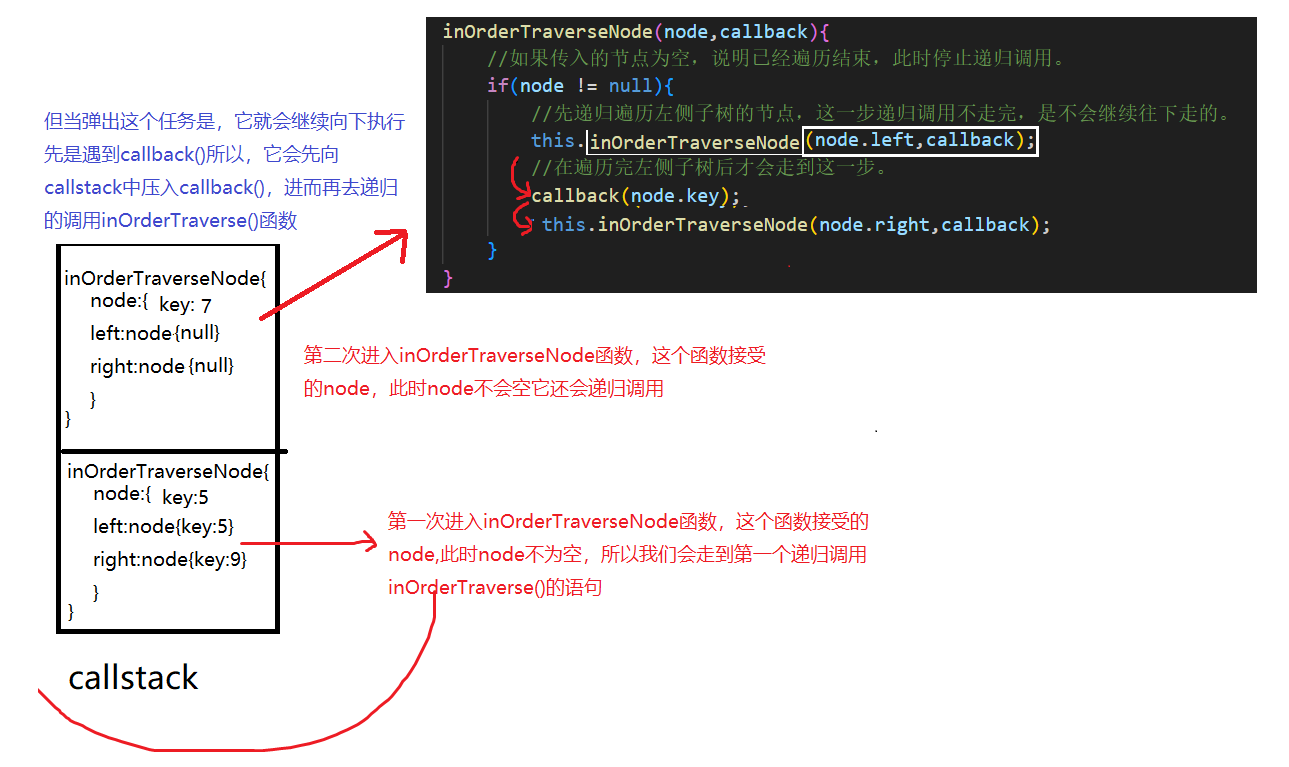


注意：上图中的递归调用的函数写错了，我在下图中已经改过来了，递归调用的应该是inOrderTraverseNode()方法。

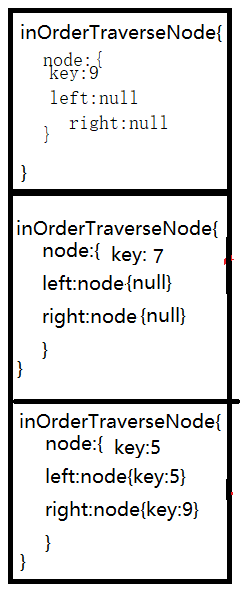
我们来一步步分析这个callstack弹出时做了那些操作。



此时也就是说它会直接弹出这个函数，然后调用栈中其实还剩两个inOrderTraverseNode方法待执行。



但是在执行中间这个inOrderTraverseNode方法的时候，会碰到一个callback()所以它又会向callstack中压入一个callback()方法，但我们之前提到过这个callback()其实就是一个打印node节点值的简单方法，所以此时控制台会log出7，然后这个callback()就执行完了，紧接着就会出栈完成。再紧着它就会递归的调用inOrderTraverseNode()方法，但此时接收的参数就是node.right和callback了。然后我们再来看看此时的callstack。



由于node.right不为null所以它又会进入到if判断力，紧接着它就又往callstack中压如一个inOrderTraverseNode()方法，不过此时，node.right为null所以这个函数也是什么都不做，然后直接出栈，出完栈之后，还是紧接着执行callback()回调函数，所以这里打印的