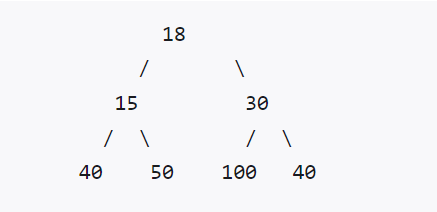
二叉搜索树是二叉树的一种，但是只允许你在左侧节点插入比父节点小的值，在右侧节点插入比父节点大的值。



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。



我们来一步步分析这个callstack弹出时做了那些操作。

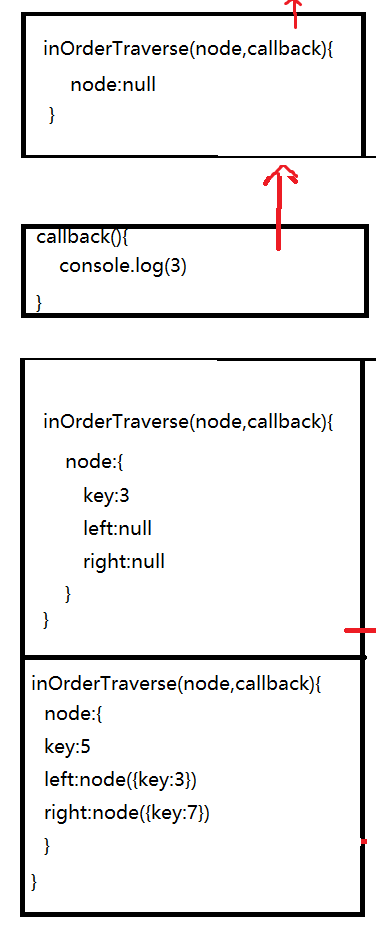


当第二个inOrder压入callstack的时候，node的key为3，不为null，此时，它就又会走到

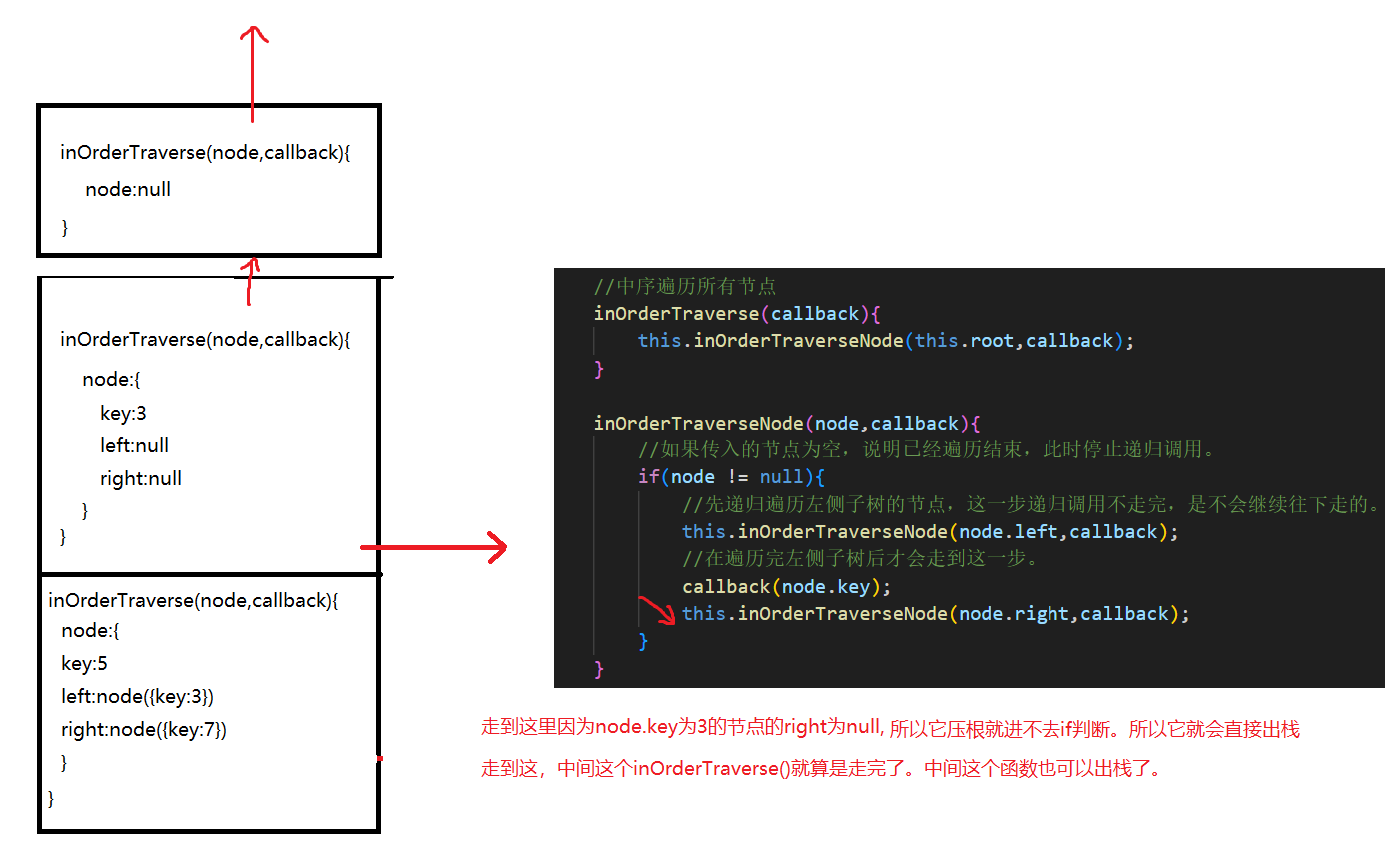
第一个this.inOrderTraverseNode(node.left,callback)语句。所以callstack又会压入一个inOrderTracerseNode()方法，此时callstack的状况应该如下图所示。



但是在执行最上面的inOrderTraverseNode方法的时候，传进来的node为null，无法通过if条件判断，所以此时它就会直接出栈，紧接着中间的inOrderTraverseNode()方法就会走到会callback()，所以它又会向callstack中压入一个callback()方法。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3，然后这个callback()就执行完了，紧接着就会出栈。



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key为3的节点的node.right为null,所以压入的inOrderTraverse()压根什么都不做。就直接出栈，然后中间的inOrderTraverse()也就执行完了，紧接着也会出栈。到此，第一个压入栈的inOrderTraverse()方法终于走到了callback()这一步。所以控制台会打印5。



当走到this.inOrderTraverseNode(node.right,callback)的时候，就又会出现递归调用。但是本质上套路跟我们之前讨论过得一样，这里我们就不重复的分析了。

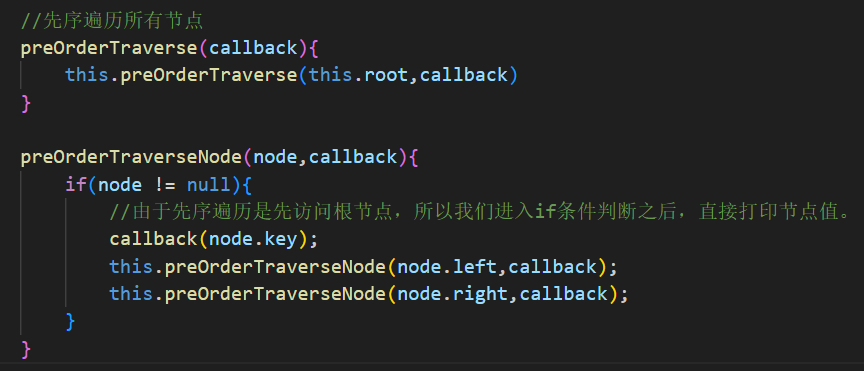
所以最终控制台会以此打印出：3,5,7。

这样我们就算是实现了二叉搜索树的中序遍历。

实现二叉搜索树的先序遍历

二叉搜索树的先序遍历就是先访问根节点，再依次访问左节点和右节点。

由于我们之前详细分析过二叉搜索树的中序遍历思路，所以对于前序遍历我们只需要简单的调整一下我们的思路就可以了。

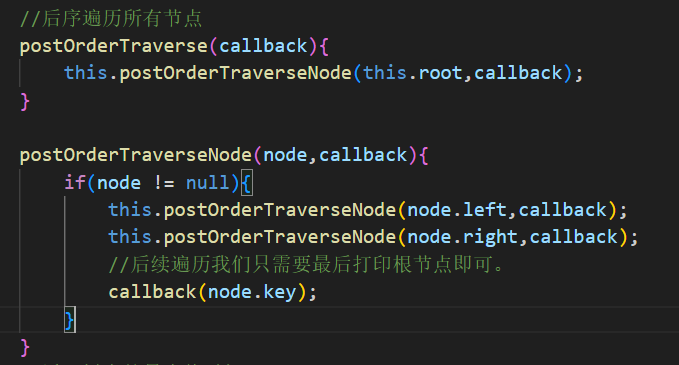


我们只需要将打印节点值得语句放到if判断的第一句执行就可以了。

实现二叉搜索树的后续遍历

读到这里相信聪明的你应该已经知道，后序遍历应该怎么实现了。不过这里我还是要介绍一下，什么是二叉搜索树的后续遍历：后边遍历，就是先依次访问左右子节点，再访问根节点的遍历顺序。

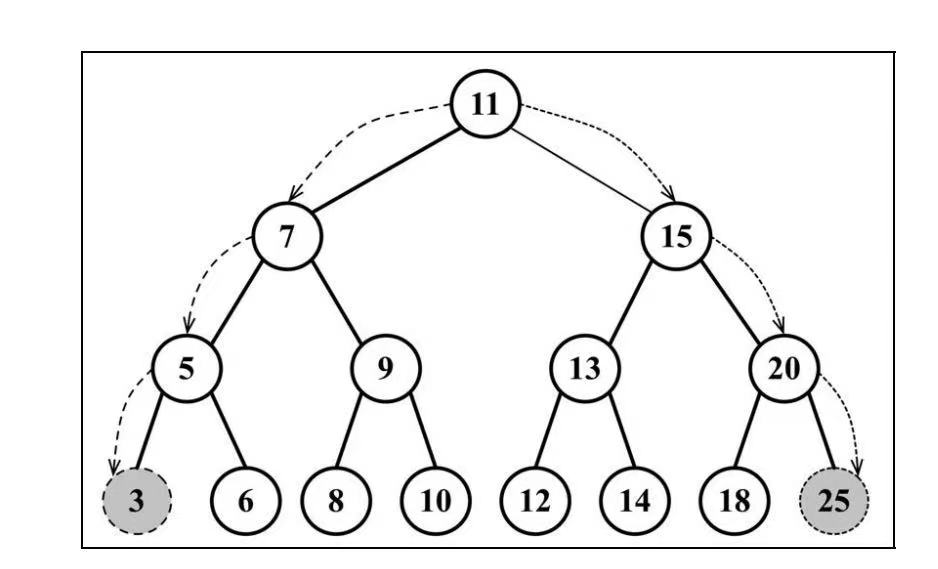
我们来看一下具体是如何实现的。



依据二叉搜索树的特点实现搜索树中的值

在实现搜索二叉搜索树之前，我需要再次向你说明一下二叉搜索树的特点。

二叉搜索树只允许在根节点的左侧存取节点值比根节点值小的值，在右侧存取比根节点值大的节点。所以二叉搜索树的节点值最大的节点肯定是在右侧，最小的节点值的节点一定是在左侧。以下图为例：

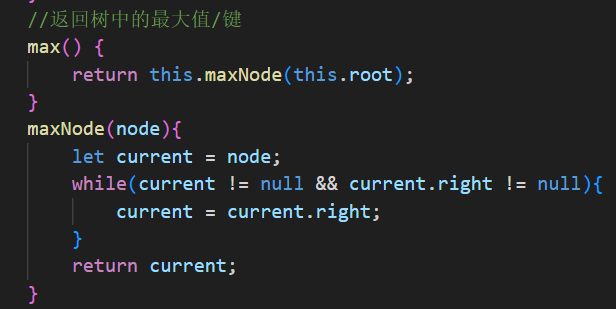


接下来我们就利用二叉搜索树的这个特性来完成搜寻二叉搜索树中最小值和最大值方法的书写。

找最小值得算法



找最大值的算法



这两个其实特别简单，就是不停的递归的找树的最左或最右侧节点。

搜寻二叉搜索树上的一个特定的节点

接下来我们将迎接一个最重要的算法，那就是找二叉搜索树中特定值得节点。因为只有找了节点，我们才能完成后续的删除操作。



我们需要做的就是，现将传进来的key与跟节点比较，确定它在树的左侧还是右侧。

确定了之后，我们递归的调用这个方法就可以。同时你需要注意停止递归的两个条件：

1. 传入的节点为空，如果发生在一开始，说明这个树就是一个空树，如果是空树压根也就不存在所谓的找特定值得节点了，直接return false就行。另外，如果是递归传递进来的节点为空，说明我们找不到这个节点，那自然也是return false。
2. 如果这个key的值即不小于也不大于传入的node.key说明我们已经找到了，此时直接return true就行。

从二叉搜索树中删除一个特定的节点。

接下来我们将完成二叉搜索树中最复杂的操作：“删除一个特定的节点”。我们需要考虑的是，如果删除的节点，只有有左侧节点，或只有右侧节点。或者两侧都有节点，或两侧都没有节点。

要移除一个特定值的节点，首先我们就要找到这个要移除的节点。先前虽然我们写过一个搜寻二叉搜索树中指定的节点的算法，但是我们不能在那个方法中直接将节点return出来。因为我们不想让用户能够直接获取到节点对象，所以我们这里需要重写一遍寻找指定值得算法。