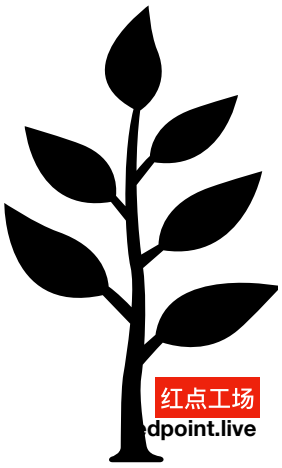
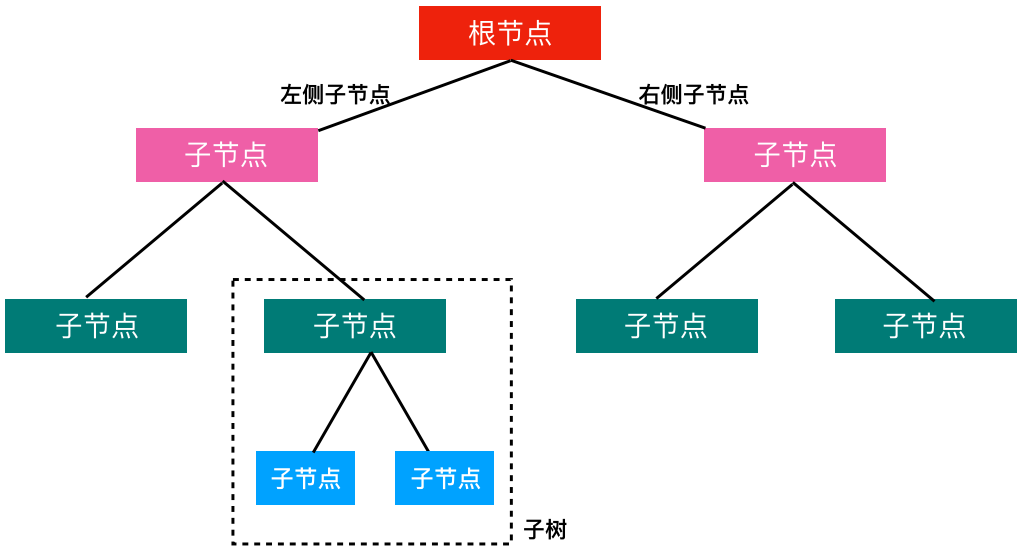
Javascript数据结构

树

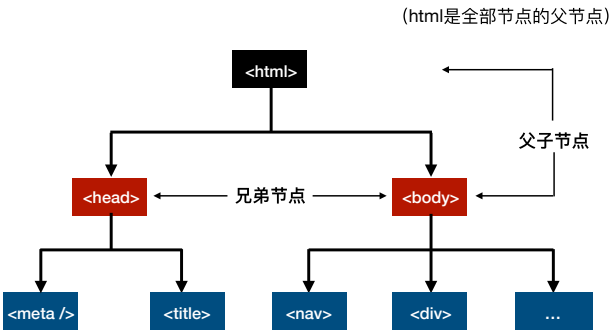


Skipper

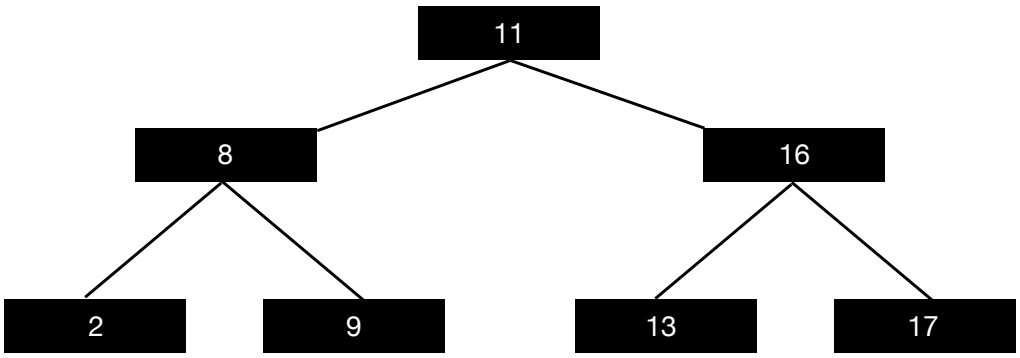
树结构示意图



HTML结构就是典型的树结构



经典实例：二叉搜索树

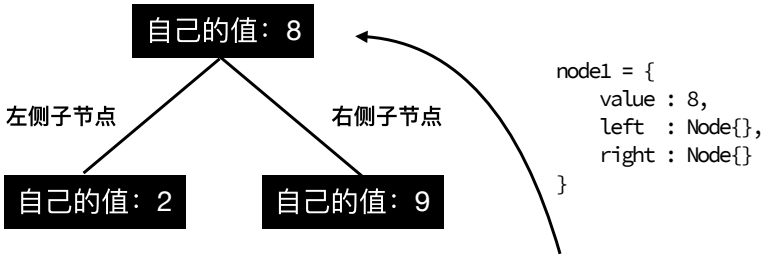


在左侧存储比父节点小的值

在右侧存储比父节点大的值

实现分析

树结构的核心是一个个节点（Node）




```
node1 = {
    value : 8,
    left  : Node{},
    right : Node{}
}
```

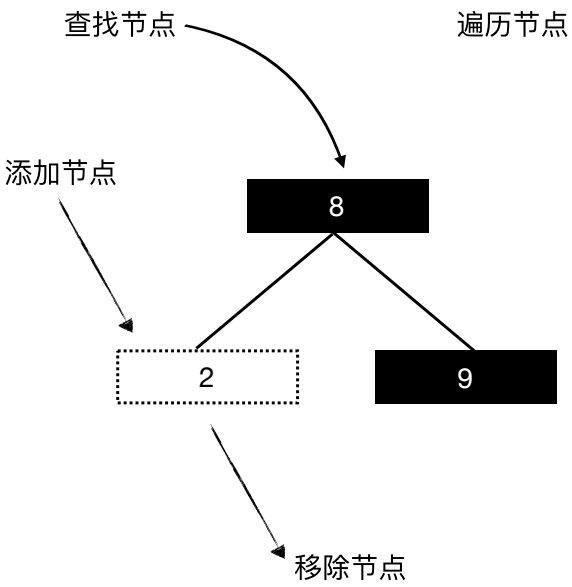
每个节点都包含了：

- **左侧子节点**
- **右侧子节点**
- **自己的值**

```
var Node = function(value){
    this.value = value
    this.left  = null
    this.right = null
}
```

不对外暴露创建Node实例的方法

外部只管操作 树（Tree）



树（Tree）有

- 根节点
- 内部的Node类
- 查找
- 删除
- 插入
- 遍历
- ……

```
var Tree = function(){

    var Node = function(value){
        this.value = value
        this.left  = null
        this.right = null
    }

    var root = null       //设置为私有变量
    this.insert = function(value){}
    this.search = function(value){}
    this.remove = function(value){}
    this.traverse = function(value){}
}
```

插入数据 insert(value)

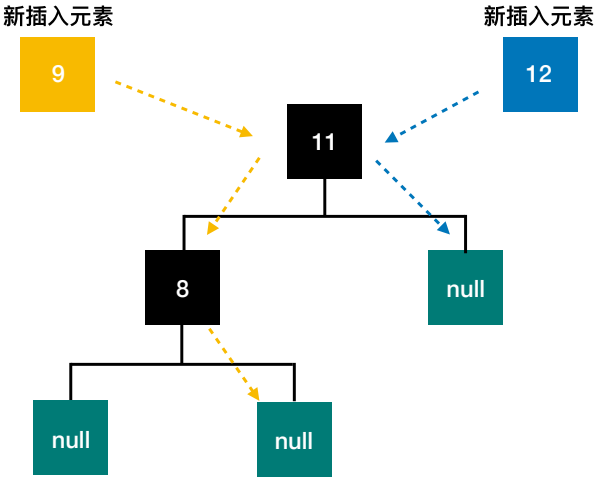
考虑两种情况：

1. 第一次插入数据，设置为根节点
2. 其余插入数据，比较大小

```
this.insert = function(value){

    var newNode = new Node(value)

    if(root == null){
        root = newNode
    } else {
        //实现一个insertNode方法用于实现插入逻辑
        insertNode(root, newNode)
    }
}
```

insertNode函数逻辑

- 比较新插入的值和目前节点的大小
- 小则向左继续检查
- 大则向右继续检查



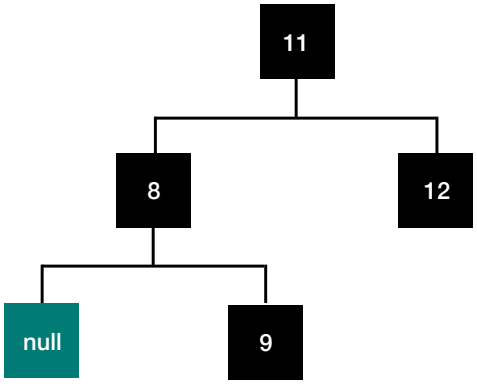
insertNode函数逻辑

- 比较新插入的值和目前节点的大小
- 小则向左继续检查
- 大则向右继续检查

```
var insertNode = function(node, newNode){
    if(newNode.value > node.value){
        //插入值大于节点值
        if(node.right == null){
            node.right = newNode
        } else {
            insertNode(node.right, newNode)
        }
    } else {
        //插入值小于节点值
        if(node.left == null){
            node.left = newNode
        } else {
            insertNode(node.left, newNode)
        }
    }
}
```

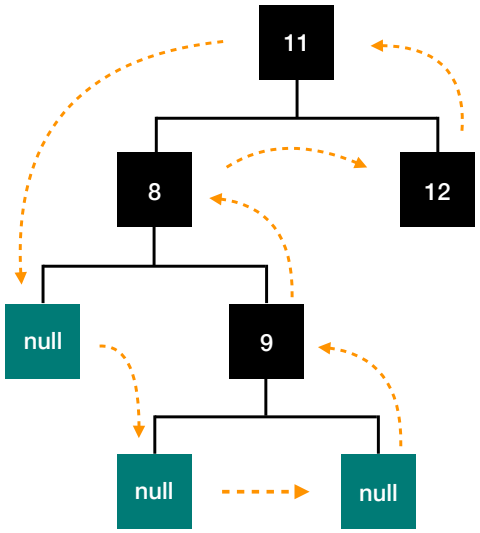
遍历树 traverse(cb)

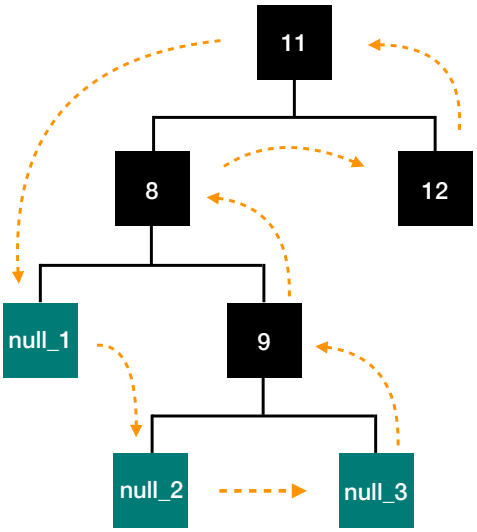
*cb为function类型回调函数
*在回调函数中操作遍历到的元素



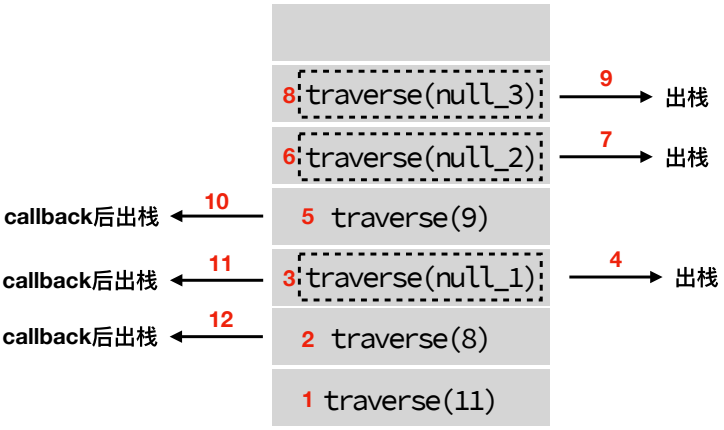

```
this.traverse = function(callback){
    //实现私有traverse方法进行递归遍历
    traverse(root, callback)
}
```

```
var traverse = function(node, callback){
    if(node == null) return

    traverse(node.left, callback)
    traverse(node.right, callback)
    callback(node.value)
}
```





递归栈



红点工场
redpoint.live

三种遍历方式

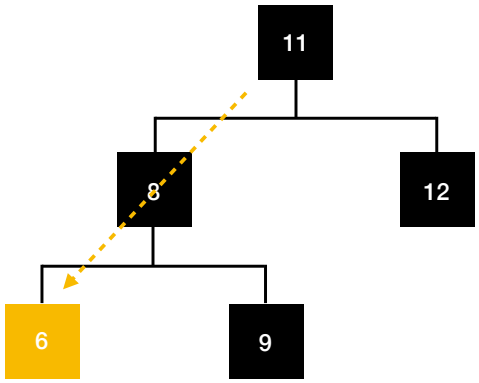
前序遍历

```
var traverse = function(node, callback){
    if(node == null) return

    callback(node.value)
    traverse(node.left, callback)
    traverse(node.right, callback)
}
```

中序遍历

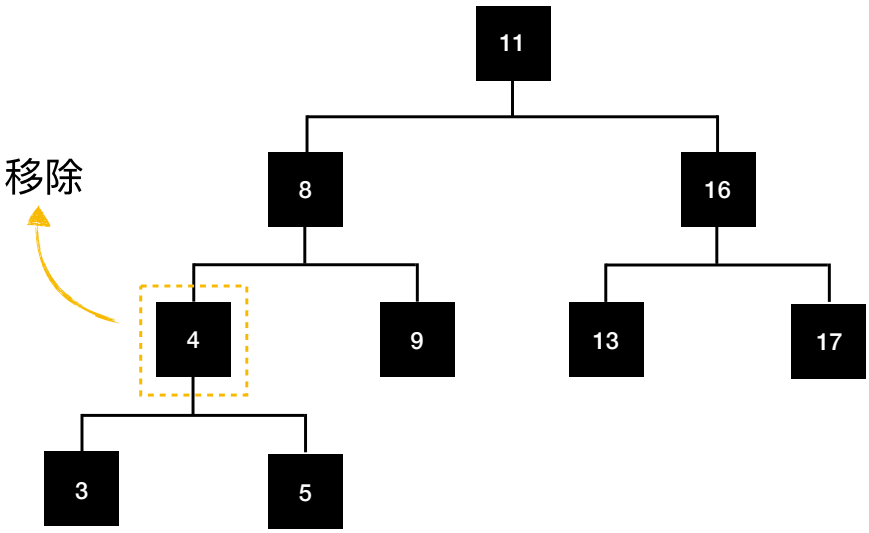
```
var traverse = function(node, callback){
    if(node == null) return

    traverse(node.left, callback)
    callback(node.value)
    traverse(node.right, callback)
}
```

获取最小值 min()



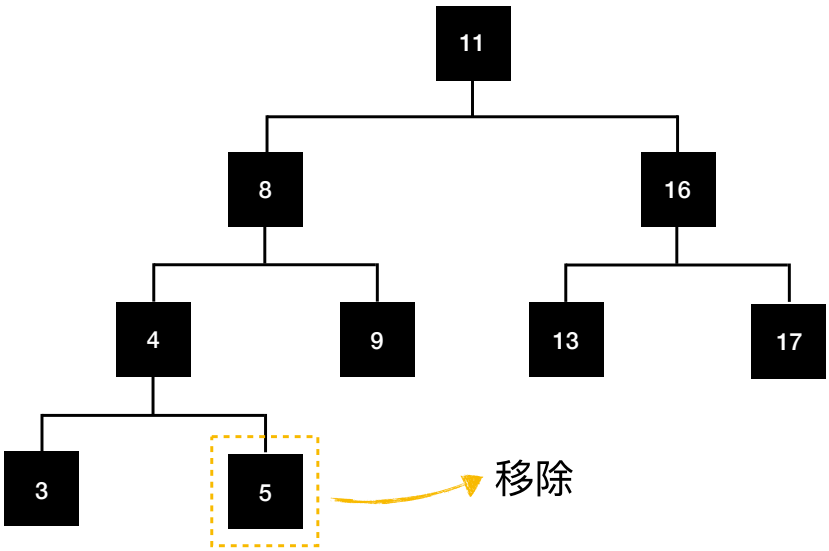

```
this.min = function(){
    return min(root)
}

var min = function(node){
    if(!node) return null

    while(node && node.left){
        node = node.left
    }

    return node.value
}
```

移除节点 remove(value)

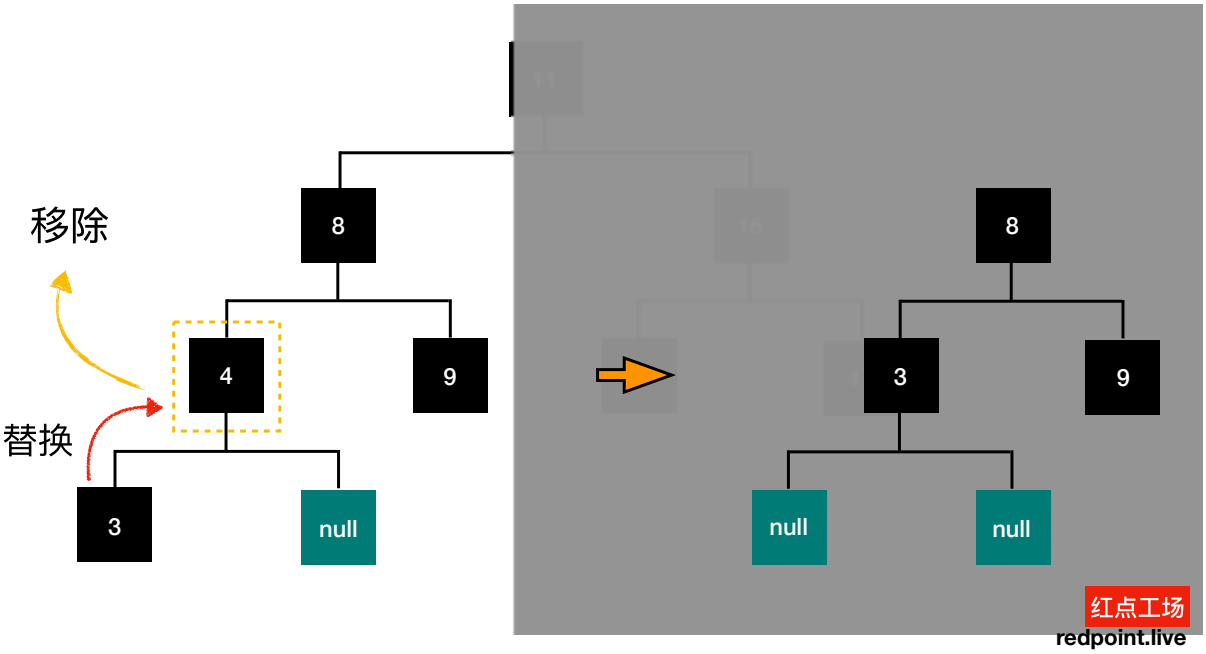


* “树结构最头疼的就是移除节点” — 头疼的一天

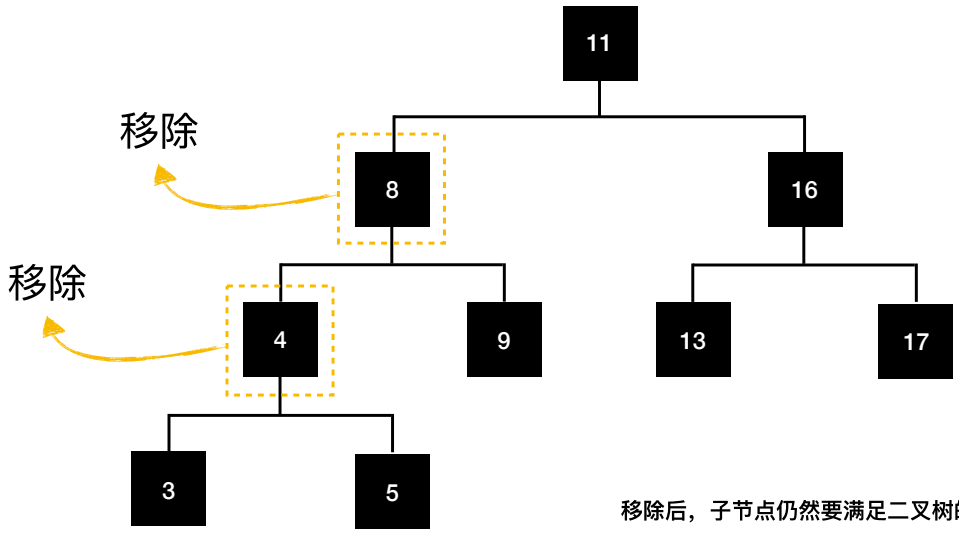
最简单的情况：移除叶节点



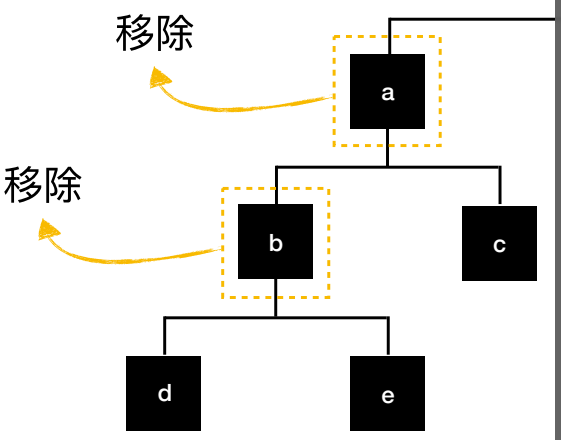
第二种情况：移除只有一个子节点的节点



第三种情况：移除有两个子节点的节点



移除后，子节点仍然要满足二叉树的条件



二叉树条件

b < a < c
d < b < e < a

一、 移除节点b后新的节点（B）需要满足的条件

d < B < e < a
1. 替换为节点d，即d->B
2. 替换节点e

二、 移除节点a后新的节点（A）需要满足的条件

b < A < c
e < d < b < A
最好的方式就是替换为 c

一句话总结：要替换为右侧子树的最小节点

移除代码实例

原理是重新构建树

```
//对外暴露的方法
this.remove = function(key){
    removeNode(root, key)
}
```

```
var findMinNode = function(node){
        while(node && node.left !== null){
            node = node.left
        }
        return node
    }
    var removeNode = function (node, key) {
        if (node === null) return null

        if (node.key < key) {
            node.right = removeNode(node.right, key)
            return node
        } else if(node.key > key) {
            node.left = removeNode(node.left, key)
            return node
        } else {
            if(node.left === null && node.right === null){
                node = null
                return node
            }
            if(node.left === null){
                node = node.right
                return node
            } else if(node.right === null){
                node = node.left
                return node
            }

            var aux = findMinNode(node.right)
            node.key = aux.key
            node.right = removeNode(node.right,aux.key)
            return node
        }
    }
```