**JavaScript二叉树遍历**

1. 二叉树概述

二叉树（Binary tree）是每个节点最多存在左右两个分支的树形数据结构。通常分支被称作“左子树”或“右子树”，节点有根节点和子节点。二叉树的分支顺序不能随意颠倒。

1. 二叉树遍历有以下几种方式：
   1. 深度遍历
      * 先序优先（DLR）：根结点 > 左子树 > 右子树

**1 > 2 > 4 > 5 > 8 > 9 > 3 > 6 > 7**

* + - 中序优先（LDR）：左子树 > 根结点 > 右子树

**4 > 2 > 8 > 5 > 9 > 1 > 6 > 3 > 7**

* + - 后序优先（LRD）：左子树 > 右子树 > 根结点

**4 > 8 > 9 > 5 > 2 > 6 > 7 > 3 > 1**

* 1. 广度遍历(层级遍历)

**1 > 2 > 3 > 4 > 5 > 6 > 7 > 8 > 9**

1. 树结构代码

var tree = {

value: 1,

left: {

value: 2, left: { value: 4 },

right: { value: 5,

left: { value: 8 },

right: { value: 9 }

}

},

right: {

value: 3, left: { value: 6 },

right: { value: 7 }

}

}

1. 深度遍历实现
   1. 先序优先遍历DLR。

function preOrderTraverse(tree, result) {

result = result ? result : []

if (tree !== undefined) {

result.push(tree.value)

preOrderTraverse(tree.left, result)

preOrderTraverse(tree.right, result)

}

return result

}

preOrderTraverse(tree)

// [1, 2, 4, 5, 8, 9, 3, 6, 7]

* 1. 中序优先遍历LDR。

function inOrderTraverse(tree, result) {

result = result ? result : []

if (tree !== undefined) {

inOrderTraverse(tree.left, result)

result.push(tree.value)

inOrderTraverse(tree.right, result)

}

return result

}

inOrderTraverse(tree)

[4, 2, 8, 5, 9, 1, 6, 3, 7]

* 1. 后序优先遍历LRD。

function postOrderTraverse(tree, result) {

result = result ? result : []

if (tree !== undefined) {

postOrderTraverse(tree.left, result)

postOrderTraverse(tree.right, result)

result.push(tree.value)

}

return result

}

postOrderTraverse(tree)

[4, 8, 9, 5, 2, 6, 7, 3, 1]

1. 广度优先(层级遍历)。

function levelOrder(tree) {

var result = [], stack = []

if (tree !== undefined) {

stack.push(tree)

while (stack.length) {

tree = stack.shift()

result.push(tree.value)

if (tree.left !== undefined) {

stack.push(tree.left)

}

if (tree.right !== undefined) {

stack.push(tree.right)

}

}

}

return result

}

levelOrder(tree)

[1, 2, 3, 4, 5, 6, 7, 8, 9]

1. 深度遍历(先序优先非递归版)。

function preOrderUnRecursive(tree) {

var result = [], stack = []

if (tree !== undefined) {

while (stack.length || tree) {

if (tree) {

result.push(tree.value)

stack.push(tree)

tree = tree.left

} else {

tree = stack.pop()

tree = tree.right

}

}

}

return result

}

preOrderUnRecursive(tree)

[1, 2, 4, 5, 8, 9, 3, 6, 7]

源码：

function preOrderTraverse(tree, result) {

result = result ? result : []

if (tree !== undefined) {

result.push(tree.value)

preOrderTraverse(tree.left, result)

preOrderTraverse(tree.right, result)

}

return result

}

preOrderTraverse(tree)

// [1, 2, 4, 5, 8, 9, 3, 6, 7]

function inOrderTraverse(tree, result) {

result = result ? result : []

if (tree !== undefined) {

inOrderTraverse(tree.left, result)

result.push(tree.value)

inOrderTraverse(tree.right, result)

}

return result

}

inOrderTraverse(tree)

[4, 2, 8, 5, 9, 1, 6, 3, 7]

function postOrderTraverse(tree, result) {

result = result ? result : []

if (tree !== undefined) {

postOrderTraverse(tree.left, result)

postOrderTraverse(tree.right, result)

result.push(tree.value)

}

return result

}

postOrderTraverse(tree)

// [4, 8, 9, 5, 2, 6, 7, 3, 1]

function levelOrder(tree) {

var result = [], stack = []

if (tree !== undefined) {

stack.push(tree)

while (stack.length) {

tree = stack.shift()

result.push(tree.value)

if (tree.left !== undefined) {

stack.push(tree.left)

}

if (tree.right !== undefined) {

stack.push(tree.right)

}

}

}

return result

}

levelOrder(tree)

[1, 2, 3, 4, 5, 6, 7, 8, 9]

function preOrderUnRecursive(tree) {

var result = [], stack = []

if (tree !== undefined) {

while (stack.length || tree) {

if (tree) {

result.push(tree.value)

stack.push(tree)

tree = tree.left

} else {

tree = stack.pop()

tree = tree.right

}

}

}

return result

}

preOrderUnRecursive(tree)

[1, 2, 4, 5, 8, 9, 3, 6, 7]

function inOrderUnRecursive(tree) {

var result = [], stack = []

if (tree !== undefined) {

while (stack.length || tree) {

if (tree) {

stack.push(tree)

tree = tree.left

} else {

tree = stack.pop()

result.push(tree.value)

tree = tree.right

}

}

}

return result

}

inOrderUnRecursive(tree)

[4, 2, 8, 5, 9, 1, 6, 3, 7]

function postOrderUnRecursive(tree) {

var result = [], stack = [], tmp

if (tree !== undefined) {

stack.push(tree)

while (stack.length) {

// 先得到最后一项作为比较项

tmp = stack[stack.length - 1]

// 把左子树按深度全部追加到stack

if (tmp.left && tree !== tmp.left && tree !== tmp.right) {

stack.push(tmp.left)

// 把右子树追加到stack

} else if (tmp.right && tree !== tmp.right) {

stack.push(tmp.right)

} else {

// 左子树到最底层节点，开始打印left与right

result.push(stack.pop().value)

// 指向上一个节点，直到stack为空，也就是遍历至根节点

tree = tmp

}

}

}

return result

}

postOrderUnRecursive(tree)

[4, 8, 9, 5, 2, 6, 7, 3, 1]