教程一：[js数据结构和算法 二叉树](https://segmentfault.com/a/1190000000740261)

二叉树

二叉树中每一个节点都是一个对象，每一个数据节点都有三个指针，分别是指向父母、左孩子和右孩子的指针。每一个节点都是通过指针相互连接的。相连指针的关系都是父子关系。

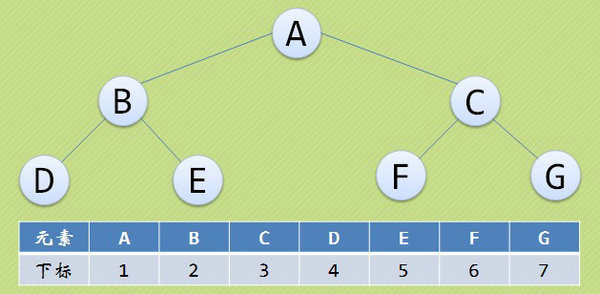
二叉树的性质

二叉树的性质一：在二叉树的第i层上至多有2^(i-1)个结点(i>=1)

二叉树的性质二：深度为k的二叉树至多有2^k-1个结点(k>=1)

二叉树的顺序存储结构

二叉树的顺序存储结构就是用一维数组存储二叉树中的各个结点，并且结点的存储位置能体现结点之间的逻辑关系。



## 二叉链表

二叉树的存储按照国际惯例来说一般也是采用链式存储结构的。

二叉树每个结点最多有两个孩子，所以为它设计一个数据域和两个指针域是比较自然的想法，我们称这样的链表叫做二叉链表。



二叉树的遍历

二叉树的遍历(traversing binary tree)是指从根结点出发，按照某种次序依次访问二叉树中所有结点，使得每个结点被访问一次且仅被访问一次。

二叉树的遍历有三种方式，如下：

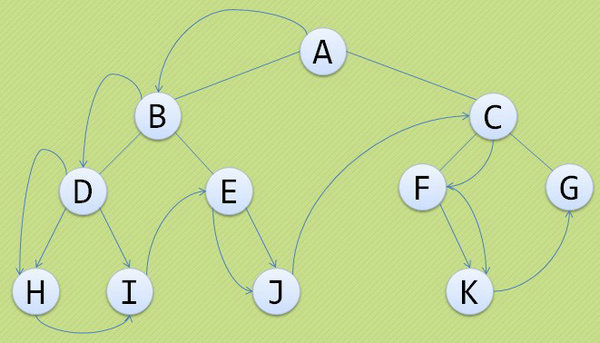
（1）前序遍历（DLR），首先访问根结点，然后遍历左子树，最后遍历右子树。简记根-左-右。

（2）中序遍历（LDR），首先遍历左子树，然后访问根结点，最后遍历右子树。简记左-根-右。

（3）后序遍历（LRD），首先遍历左子树，然后遍历右子树，最后访问根结点。简记左-右-根。

前序遍历：

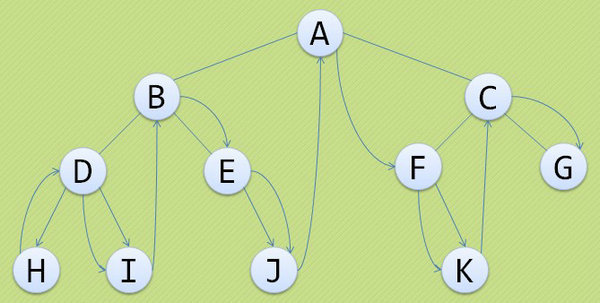
若二叉树为空，则空操作返回，否则先访问根结点，然后前序遍历左子树，再前序遍历右子树。



遍历的顺序为：A B D H I E J C F K G

中序遍历：

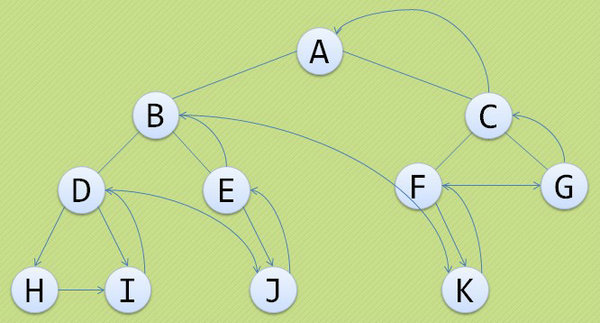
若树为空，则空操作返回，否则从根结点开始（注意并不是先访问根结点），中序遍历根结点的左子树，然后是访问根结点，最后中序遍历右子树。



遍历的顺序为：H D I B E J A F K C G

后序遍历：

若树为空，则空操作返回，否则从左到右先叶子后结点的方式遍历访问左右子树，最后访问根结点。



遍历的顺序为：H I D J E B K F G C A

教程二：[Data Structures With JavaScript: Tree](http://code.tutsplus.com/articles/data-structures-with-javascript-tree--cms-23393)

 two different methods of tree traversal: Depth-First Search (DFS) and Breadth-First Search (BFS)

#### Node

* data stores a value.
* parent points to a node's parent.
* children points to the next node in the list.

#### Tree

* \_root points to the root node of a tree.
* traverseDF(callback) traverses nodes of a tree with DFS.
* traverseBF(callback) traverses nodes of a tree with BFS.
* contains(data, traversal) searches for a node in a tree.
* add(data, toData, traverse) adds a node to a tree.
* remove(child, parent) removes a node in a tree.

#### Properties of a Node

For our implementation, we will first define a function named Node and then a constructor named Tree.

|  |
| --- |
| * function Node(data) { * this.data = data; * this.parent = null; * this.children = []; * } |

Every instance of Node contains three properties: data, parent, and children. The first property holds data associated with a node. The second property points to one node. The third property points to many children nodes.

#### Properties of a Tree

Now, let's define our constructor for Tree, which includes the Node constructor in its definition:

|  |  |
| --- | --- |
| 1  2  3  4 | function Tree(data) {      var node = new Node(data);      this.\_root = node;  } |

Tree contains two lines of code. The first line creates a new instance of Node; the second line assigns node as the root of a tree.

var tree = new Tree('CEO');

// {data: 'CEO', parent: null, children: []}

tree.\_root;

#### Methods of a Tree

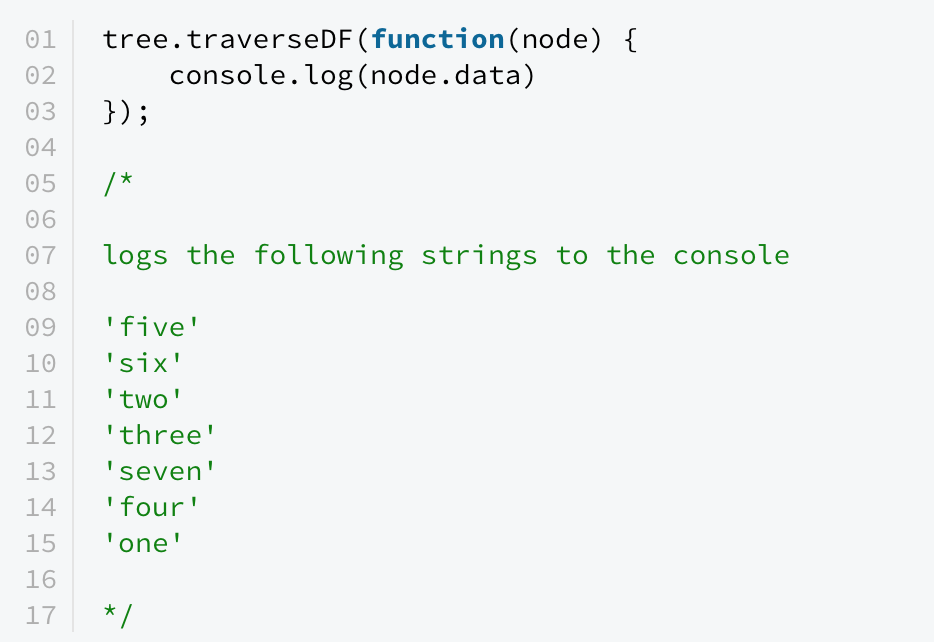
Next, we will create the following five methods:

**Tree**

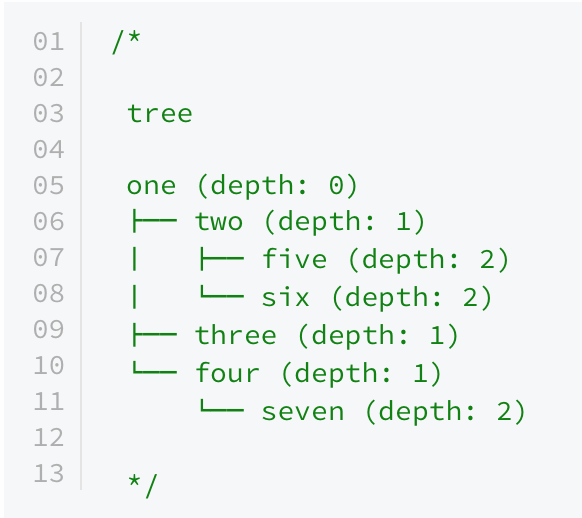
1. traverseDF(callback)
2. traverseBF(callback)
3. contains(data, traversal)
4. add(child, parent)
5. remove(node, parent)

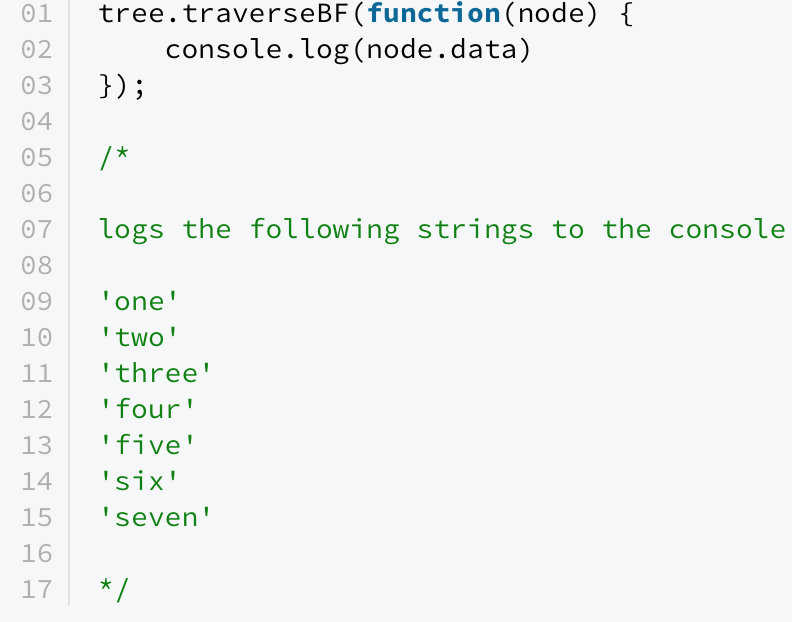
**1 of 5: traverseDF(callback)**





**2 of 5: traverseBF(callback)**

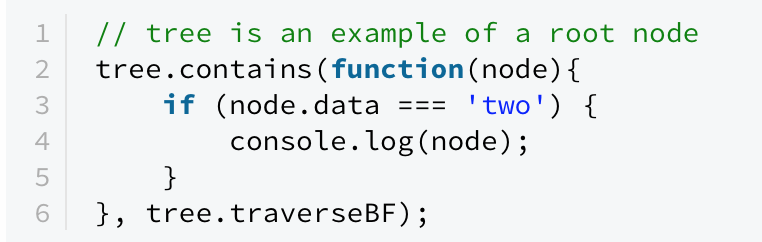




**3 of 5: contains(callback, traversal)**

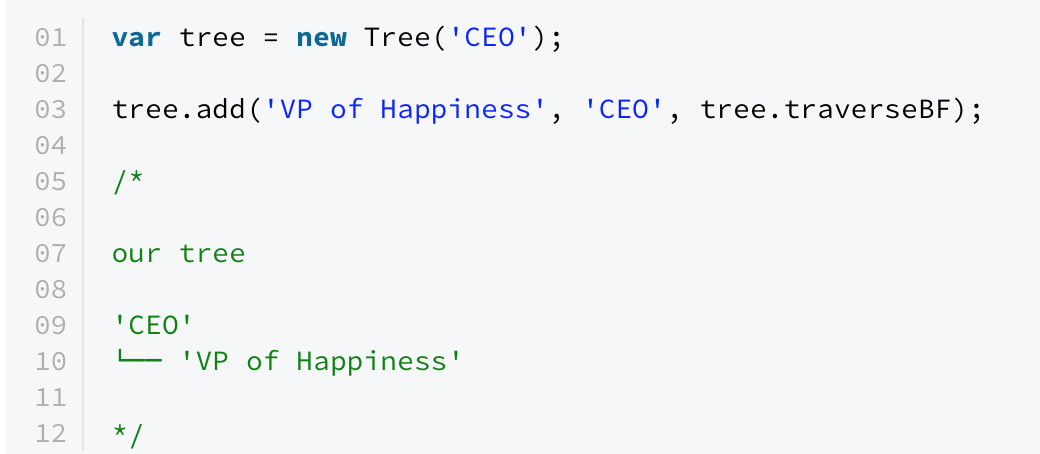
a method that will allow us to search for a particular value in our tree

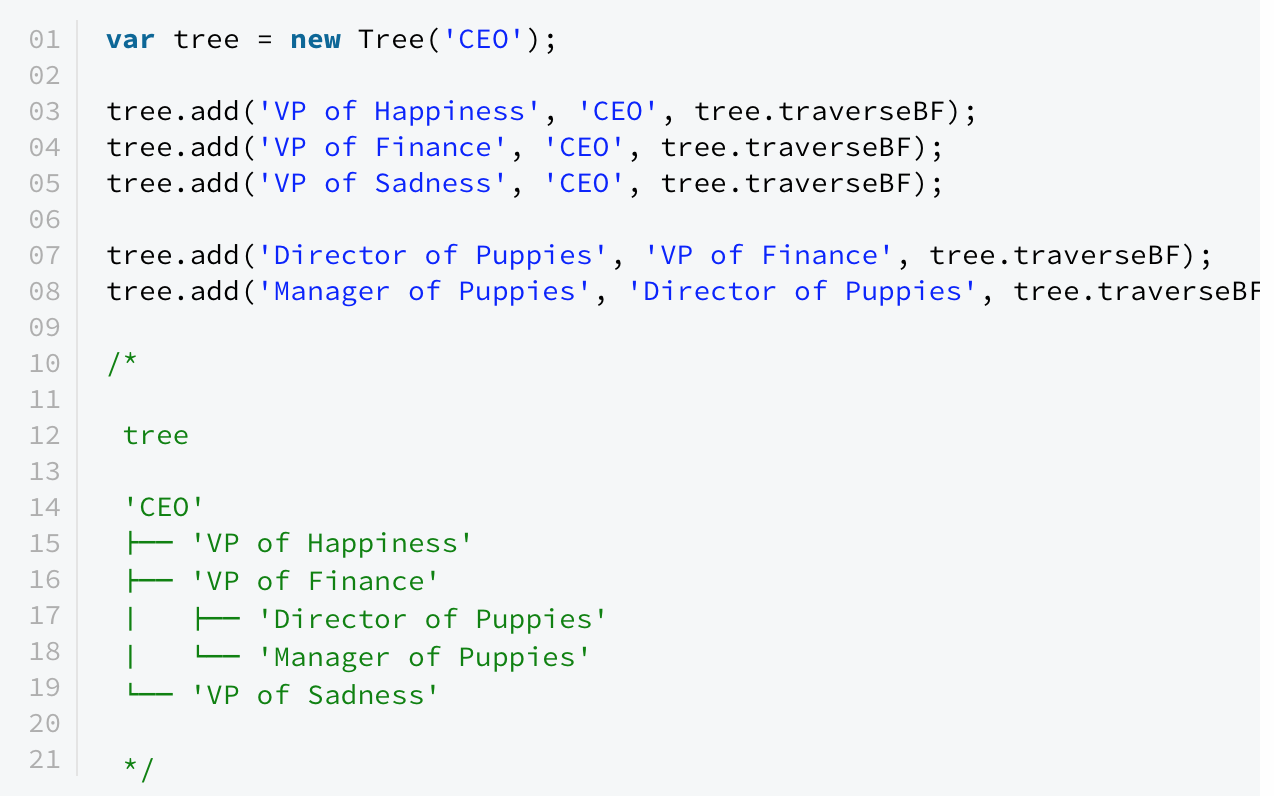
two arguments: the data to search and the type of traversal.



**4 of 5: add(data, toData, traversal)**

 a method that will enable us to add a node to a specific node.





**5 of 5: remove(data, fromData, traversal)**

this method will remove a node and all of its children

# JavaScript prototype 属性

prototype 属性使您有能力向对象添加属性和方法。

语法：object.prototype.name=value



教程三：Computer science in javascript: Binary search tree

Binary search tree，BST，二叉搜索树

The unique part of a binary search tree, however, is that the nodes are ordered based on the value they contain: any values that are part of a node’s left subtree are always less than the node’s value and any values in the right subtree are always greater than the node’s value.

某个节点的左子树中的所有value一定比这个节点小，其右子树节点的所有value一定比这个节点大。

Flex布局

Flex 是 Flexible Box 的缩写，意为"弹性布局"，用来为盒状模型提供最大的灵活性。

任何一个容器都可以指定为 Flex 布局。

.box{

display: flex;

}

行内元素也可以使用 Flex 布局。

.box{

display: inline-flex;

}

二、基本概念

采用 Flex 布局的元素，称为 Flex 容器（flex container），简称"容器"。它的所有子元素自动成为容器成员，称为 Flex 项目（flex item），简称"项目"。



容器默认存在两根轴：水平的主轴（main axis）和垂直的交叉轴（cross axis）。主轴的开始位置（与边框的交叉点）叫做main start，结束位置叫做main end；交叉轴的开始位置叫做cross start，结束位置叫做cross end。

项目默认沿主轴排列。单个项目占据的主轴空间叫做main size，占据的交叉轴空间叫做cross size。

三、容器的属性

以下6个属性设置在容器上。

* flex-direction
* flex-wrap
* flex-flow
* justify-content
* align-items
* align-content

3.1 flex-direction属性

flex-direction属性决定主轴的方向（即项目的排列方向）。

.box {

flex-direction: row | row-reverse | column | column-reverse;

}

它可能有4个值。

* row（默认值）：主轴为水平方向，起点在左端。
* row-reverse：主轴为水平方向，起点在右端。
* column：主轴为垂直方向，起点在上沿。
* column-reverse：主轴为垂直方向，起点在下沿。

3.2 flex-wrap属性

默认情况下，项目都排在一条线（又称"轴线"）上。flex-wrap属性定义，如果一条轴线排不下，如何换行。

.box{

flex-wrap: nowrap | wrap | wrap-reverse;

}

它可能取三个值。

（1）nowrap（默认）：不换行。

（2）wrap：换行，第一行在上方。

（3）wrap-reverse：换行，第一行在下方。

3.3 flex-flow

flex-flow属性是flex-direction属性和flex-wrap属性的简写形式，默认值为row nowrap。

.box {

flex-flow: <flex-direction> || <flex-wrap>;

}

3.4 justify-content属性

justify-content属性定义了项目在主轴上的对齐方式。

.box {

justify-content: flex-start | flex-end | center | space-between | space-around;

}

它可能取5个值，具体对齐方式与轴的方向有关。下面假设主轴为从左到右。

* flex-start（默认值）：左对齐
* flex-end：右对齐
* center： 居中
* space-between：两端对齐，项目之间的间隔都相等。
* space-around：每个项目两侧的间隔相等。所以，项目之间的间隔比项目与边框的间隔大一倍。

3.5 align-items属性

align-items属性定义项目在交叉轴上如何对齐。

.box {

align-items: flex-start | flex-end | center | baseline | stretch;

}

它可能取5个值。具体的对齐方式与交叉轴的方向有关，下面假设交叉轴从上到下。

* flex-start：交叉轴的起点对齐。
* flex-end：交叉轴的终点对齐。
* center：交叉轴的中点对齐。
* baseline: 项目的第一行文字的基线对齐。
* stretch（默认值）：如果项目未设置高度或设为auto，将占满整个容器的高度。

3.6 align-content属性

align-content属性定义了多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用。

.box {

align-content: flex-start | flex-end | center | space-between | space-around | stretch;

}

该属性可能取6个值。

* flex-start：与交叉轴的起点对齐。
* flex-end：与交叉轴的终点对齐。
* center：与交叉轴的中点对齐。
* space-between：与交叉轴两端对齐，轴线之间的间隔平均分布。
* space-around：每根轴线两侧的间隔都相等。所以，轴线之间的间隔比轴线与边框的间隔大一倍。
* stretch（默认值）：轴线占满整个交叉轴。

四、项目的属性

以下6个属性设置在项目上。

* order
* flex-grow
* flex-shrink
* flex-basis
* flex
* align-self

4.1 order属性

order属性定义项目的排列顺序。数值越小，排列越靠前，默认为0。

.item {

order: <integer>;

}

4.2 flex-grow属性

flex-grow属性定义项目的放大比例，默认为0，即如果存在剩余空间，也不放大。

.item {

flex-grow: <number>; /\* default 0 \*/

}

如果所有项目的flex-grow属性都为1，则它们将等分剩余空间（如果有的话）。如果一个项目的flex-grow属性为2，其他项目都为1，则前者占据的剩余空间将比其他项多一倍。

4.3 flex-shrink属性

flex-shrink属性定义了项目的缩小比例，默认为1，即如果空间不足，该项目将缩小。

.item {

flex-shrink: <number>; /\* default 1 \*/

}

如果所有项目的flex-shrink属性都为1，当空间不足时，都将等比例缩小。如果一个项目的flex-shrink属性为0，其他项目都为1，则空间不足时，前者不缩小。

负值对该属性无效。

4.4 flex-basis属性

flex-basis属性定义了在分配多余空间之前，项目占据的主轴空间（main size）。浏览器根据这个属性，计算主轴是否有多余空间。它的默认值为auto，即项目的本来大小。

.item {

flex-basis: <length> | auto; /\* default auto \*/

}

它可以设为跟width或height属性一样的值（比如350px），则项目将占据固定空间。

4.5 flex属性

flex属性是flex-grow, flex-shrink 和 flex-basis的简写，默认值为0 1 auto。后两个属性可选。

.item {

flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]

}

该属性有两个快捷值：auto (1 1 auto) 和 none (0 0 auto)。

建议优先使用这个属性，而不是单独写三个分离的属性，因为浏览器会推算相关值。

4.6 align-self属性

align-self属性允许单个项目有与其他项目不一样的对齐方式，可覆盖align-items属性。默认值为auto，表示继承父元素的align-items属性，如果没有父元素，则等同于stretch。

.item {

align-self: auto | flex-start | flex-end | center | baseline | stretch;

}

该属性可能取6个值，除了auto，其他都与align-items属性完全一致。

setInterval()

## 定义和用法

setInterval() 方法可按照指定的周期（以毫秒计）来调用函数或计算表达式。

setInterval() 方法会不停地调用函数，直到 clearInterval() 被调用或窗口被关闭。由 setInterval() 返回的 ID 值可用作 clearInterval() 方法的参数。

### 语法

setInterval(code,millisec[,"lang"])

|  |  |
| --- | --- |
| **参数** | **描述** |
| code | 必需。要调用的函数或要执行的代码串。 |
| millisec | 必须。周期性执行或调用 code 之间的时间间隔，以毫秒计。 |

function show(){  
    trace("每隔一秒显示一次");  
}  
var sh;  
sh=setInterval(show,1000);  
clearInterval(sh);