> *Wiki* ‣ *[[API--中文手册]]* ‣ *[[布局]]* ‣ ***树布局***

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3.js:437278817，大数据可视化：436442115
- Github小组：VisualCrew

树布局使用Reingold–Tilford "tidy" algorithm算法生成整齐的链接点模型树状图。一个典型的例子：在软件的架构中，树布局可以用来组织软件的包及类的层级结构。

像其他大多数布局一样，`var tree = d3.layout.tree()` 得到的即是一个对象也是一个函数；也就是说：可以像调用函数一样调用布局（`tree()`），也可以直接调用 *tree* 提供的一些 API 接口来改变其行为，如：
```tree.children(Function)，同样支持级联调用。

# d3.layout.**tree**()

创建一个树布局实例，使用默认的设置：

- `tree.sort = null`;
- `tree.children = function(d) { return d.children; }`, d.children 是一个数组;
- `tree.separation = function(a, b) { return a.parent == b.parent ? 1 : 2; }`, 即：兄弟节点间是 **1** 倍距离、表亲兄弟间是 **2** 倍距离；(注：该 **距离** 是树布局基于当前 *depth* 层级计算出来基础间距；)

# **tree**(*root*)
# tree.**nodes**(*root*)

运行树布局，返回一个基于根结点 root 的所有结点的数组；树布局是 D3 层布局家族中的一部分，这些布局都遵循类似的基本结构：布局的输入参数是层次结构的根节点 root，输出值是一个经过计算的全部结点的位置的数组；每个节点中包含以下属性：

- *parent* - 父节点引用，根节点为 `null`;
- *children* - 所有子节点，叶子节点为 `null`;
- *depth* - 当前节点所处层级（也叫深度）；根节点为 0;
- *x* - 计算得到的当前节点的计 *x* 坐标;
- *y* - 计算得到的当前节点的计 *y* 坐标;

尽管树布局计算得到的 x 和 y 有 size 大小概念的味道，但是，这里的 x 和 y 依然可以理解为任意坐标系；举个例子说，把 x 看成是角度，把 y 看做是半径，这时 `tree.size([360, 960])` 的设定就意味着，半径的取值范围是 0-960，角度的取值范围是 0-360，所以，树布局计算出来的每个节点的 x 和 y 实际表示的是径向的树结构（平面中的点可以是：直角坐标系的 [x, y] 或 极坐标系的 { θ, r }）。

# tree.**links**(*nodes*)

根据给定的结点数组 *nodes*（比如：通过 `tree.nodes(root)` 返回的一组节点集），生成一组表示从父节点到子节点关系对象，叶子节点不包含该种"输出"关系；每个关系对象有两个属性：

- source - 父节点的引用;
- target - 子节点的引用;

该方法的设定目的是，用于检索一组有连接关系的节点集，并输出这种连接关系；通常会配合如diagonal（对角线形状生成器器）一起使用；如下：
```

```
svg.selectAll("path")
    .data(tree.links(nodes))
  .enter().append("path")
    .attr("d", d3.svg.diagonal());
```

tree.**children**([*children*])

如果指定了 *children* 参数，则设置子节点访问器函数为 *children*；如果未指定，则返回当前的子节点访问器函数；默认的子节点访问器假定输入数据是一个有 *children* 的 KEY 的对象，形式如下：

```
function children(d) {
  return d.children;
}
```

【略过】Often, it is convenient to load the node hierarchy using d3.json, and represent the input hierarchy as a nested JSON object. For example:

```
{
 "name": "flare",
 "children": [
  {
   "name": "analytics",
   "children": [
    {
     "name": "cluster",
     "children": [
      {"name": "AgglomerativeCluster", "size": 3938},
      {"name": "CommunityStructure", "size": 3812},
      {"name": "MergeEdge", "size": 743}
     ]
    },
    {
     "name": "graph",
     "children": [
      {"name": "BetweennessCentrality", "size": 3534},
      {"name": "LinkDistance", "size": 5731}
     ]
    }
   ]
  }
 ]
}
```

访问器首先会在在层次结构的根节点被调用；如果访问器返回 null，则该节点会被认为是叶子节点，从而终止向下遍历；否则，访问器应返回一个包含所有子节点数据元素的数组。

tree.**separation**([*separation*])

如果指定了 *separation* 参数，则设置相邻节点的间距计算器为 *separation*；如果未指定，则返回当前的间距计算器函数；默认的间距计算器函数形式如下：

```
function separation(a, b) {
  return a.parent == b.parent ? 1 : 2;
}
```

对于一些特殊的，如径向树图，差异化的间距值可以避免叶子节点间的锯齿感，因此，使用如下的间距计算器可以优化该问题：

```
function separation(a, b) {
  return (a.parent == b.parent ? 1 : 2) / a.depth;
}
```

间距计算器函数有两个入参：*a* 和 *b*，该函数需要返回在这两个节点间的距离期望**倍数**；*a*、*b* 节点通常是相邻的兄弟；当然，也不排除树布局有意将两个表亲兄弟节点或更远的节点放置在"相邻"的位置。

注：上文提到的间距计算器函数的返回值被称做**倍数**，并不是来自原英文，而是译者结合测试、使用经验自译的；请参考原文。

tree.**size**([*size*])

如果指定了 *size* 参数，则设置树布局可用的空间范围为指定的两元素数组 *size*；如果未指定，则返回当前的空间范围；默认是：`[1, 1]`；*size* 的值可能被认为是：*size[0]* => *x*, *size[1]* => *y*；但是，这并不局限于平面直角坐标系，而适用于任意的坐标系，*x* 和 *y* 只是一个标志位的作用；

举个例子说，把 x 看成是角度，把 y 看做是半径，这时 `tree.size([360, 960])` 的设定就意味着，半径的取值范围是 0-960，角度的取值范围是 0-360，所以，树布局计算出来的每个节点的 x 和 y 实际表示的是径向的树结构（平面中的点可以是：直角坐标系的 [x, y] 或 极坐标系的 { θ, r }）。

size 属性与 tree.nodeSize 不同时共存，设置了 *size* 就必须设置 *nodeSize* 为 null；

tree.**nodeSize**([*nodeSize*])

如果指定了 *nodeSize* 参数，则为每个节点设置一个固定 *size* 的表示为 *x* 和 *y* 的两元素数组；如果未指定，则返回当前的配置；默认值为 null，表示树布局使用整体配置的 *tree.size* 值来计算布局；同样，这里的所对应的 *x* 和 *y* 也不局限于特定的坐标系。

nodeSize 属性与 tree.size 不同时共存，设置了 *nodeSize* 就必须设置 *size* 为 null；

tree.**sort**([*comparator*])

如果指定了 *comparator* 参数，则启用排序算法并设置排序的比较器为 *comparator* 函数；如果未指定，则返回当前的比较器；默认的比较器为 null，表示不对原数据进行排序；和数组的原生 **sort** 函数一样，比较器也有类似的入参及返回值；一个降序排序的参考示例如下：

```
function comparator(a, b) {
  return b.value - a.value;
}
```

当然，还有更好的选择，参考：d3.ascending、d3.descending；

tree.**value**([*value*])

如果指定了 *value* 参数，则设置取值器为指定的 *value* 函数；如果未指定，则返回当前的取值器；默认值为 null；获取节点的 *value* 值当前并不会影响到树布局的计算中，只是层布局的一个通用 API 接口。

---

changelog

- 【阿呆不呆】译于 2014-11-28
- 【大傻】校于 2014-12-07 10:20:53
- 【二傻】校于 2016-07-27
  - 诸多解释性描述的校对；
  - 译文布局统一的校对；
  - 链接/锚点的校对；
  - 页头、页尾统一化；