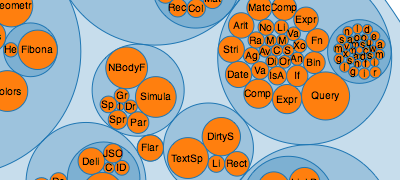
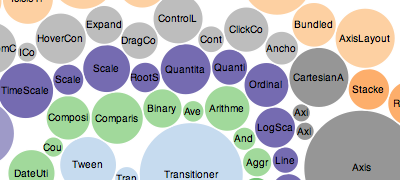
# [d3.layout.pack](https://github.com/mbostock/d3/wiki/Pack-Layout" \l "wiki-pack) -API释义

pack部分官方API :<https://github.com/mbostock/d3/wiki/Pack-Layout>。

下图采用嵌套结构来呈现出层级结构。每一个叶子节点的尺寸都是由该节点的数据量化而来。如图，大圈套小圈，通过小圈的数量累积形成了大圈的尺寸，毕竟是由数值计算而来，在这个计算过程中，难免有小数产生的精度问题，所以圈与圈之间有些失真，但无伤大雅，因为叶子节点不是通过累加计算出的尺寸，所以对“我”（叶子节点）又有什么影响呢？相对treemap而言，尽管pack貌似浪费了空间（圈间间隙），但是结构上的分层效果却更加的清晰，你觉得呢？



你不说我浪费空间吗？OK!为了节省下空间，pack可以通过压缩，将尽可能的计算出有效利用面积。不过，你不觉得变味了吗？看起来有点像bubble charts。



和D3中其他类相似，layout中的各种布局（如Pie、Force、Hierarchy等）都支持方法链式调用。下面将对你可能调用的方法做个介绍。Follow me!

#d3.laoout.**pack**()

创建pack布局，布局中节点默认按照升序排列，子节点和其叶子节点都通过对象表示，就好比子节点是一个电影院，其叶子节点为电影院里的观众，观众座位号码按升序排列。

#**pack**(root)

#pack.**nodes**(root)

使用pack布局，返回根节点直接子节点（仅与根节点相关联），集群布局当属分层布局的一部分。这些布局大同小异，都是从根节点切入，然后找出子节点以及叶子节点，每一个节点都是个对象，是对象就有属性，这些属性也就决定了节点的性质。以下就是你可能用到的几个属性。

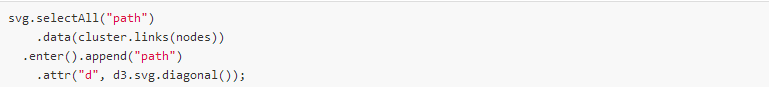
* parent – 根节点喽，要么为null，要么为根节点。
* children – 子节点。（可能是个集合）。
* value –节点value属性，可保存你觉得有意义的值(例如可以放个对排序有意义的值)。
* depth – 层次深度（你应该了解SVG绘图层次关系）。
* x – 节点横坐标。
* y – 节点纵坐标（温馨提示：y值的计算通用h-y，你懂得）。
* r – 节点半径。

#pack.**links**(nodes)

它表示一种关系，比如nodes为root，它就返回root的直接子节点，直接子节点拥有两个属性：

* source – 谁是它的父节点
* target –谁是他的子节点

这种承上启下的衔接，用于点到点画线再好不过了，画个斜线试试。



类似这种效果：



#pack.**children**([children])

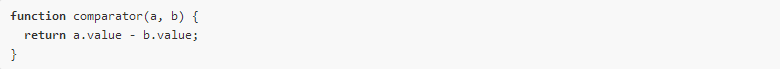
这里允许你望文生义，它就是获取子节点。传来节点，返回它的子节点，只不过这个方法很执着，它会不断的返回子节点的子节点，知道找到叶子节点才罢休。例如下面这段信息：



如果你传的是flare节点，那么返回的就是一个数组对象（子节点集合以及子子节点集合），这种方法对json数据的解析非常便利。上面不就是段json数据吗？

#pack.**sort**([comparator])

同级关系节点排序（也就是兄弟节点），父节点调用sort后，可以自定义比较器，按照我们的意愿来对直接子节点进行排序，也可以不定义比较器，d3默认通过获取节点的value值进行升序排列。



#pack.**value**([value])

每个节点都有个value，但跟这个value不是一个含义，可以理解为这个value是每个节点的区分度，官方文档为了语言的严谨性，说了一大堆东西绕来绕去，无非就是要告诉你，你所见到的圈的大小，就是从这计算出来的。D3的强大之处在于，它可以凭借本身一套计算体系，分析出每个节点的大小。

#pack.**size**([size])

你的pack布局范围在这里设置，也就是使用pack作图占用的面积。

#pack.**radius**([radius])

指定画圈叶子节点半径，这样指定之后，叶子节点面积大小的计算，将不再由value计算得来。

#pack.**padding**([padding])

相邻节点（同级）间的距离，默认为0。