# Hierarchy Layout

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [API Reference](https://github.com/mbostock/d3/wiki/API-Reference) ▸ [Layouts](https://github.com/mbostock/d3/wiki/Layouts) ▸ **Hierarchy Layout**

### [层级布局(Hierarchy)](https://github.com/mbostock/d3/wiki/Hierarchy-Layout)

* [d3.layout.hierarchy](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-hierarchy) - 获得一个自定义的层级布局的实现.
* [hierarchy.sort](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-sort) - 获取或设置一个函数, 用来给兄弟节点(同一父结点的子结点)的排序.
* [hierarchy.children](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-children) - 获取或设置子结点的访问器.
* [hierarchy.nodes](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-nodes) - 计算并返回指定结点的子结点信息.
* [hierarchy.links](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-links) - 指定一个子结点数组(通常是**nodes**函数返回值), 计算它们与父结点的连接信息.
* [hierarchy.value](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-value) - 获取或设置结点的**值**访问器.
* [hierarchy.revalue](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-revalue) - 重新计算层级布局.

层次结构布局是一个抽象的布局，不是直接使用，而是允许代码共享多个层次之间的布局。下边的每一个都是一个实现：

* [Cluster](https://github.com/mbostock/d3/wiki/Cluster-Layout) – 集群实体的树图。
* [Pack](https://github.com/mbostock/d3/wiki/Pack-Layout) – 使用递归圆填充方式生成一个层次分级的布局。

* [Partition](https://github.com/mbostock/d3/wiki/Partition-Layout) – 递归划分一个节点树的阳光状或冰柱状。
* [Tree](https://github.com/mbostock/d3/wiki/Tree-Layout) – 整齐的安置为树节点。
* [Treemap](https://github.com/mbostock/d3/wiki/Treemap-Layout) – 使用递归细分来显示一个树节点。

虽然[bundle layout](https://github.com/mbostock/d3/wiki/Bundle-Layout)（包布局）不是一个层级布局，但包布局也被设计为与层次结合使用。

#### [#](https://github.com/mbostock/d3/wiki/Hierarchy-Layout" \l "wiki-hierarchy) d3.layout.hierarchy()

创建一个新的层级布局，默认的设置为：默认排序顺序为值递降；默认的值访问器中每个输入值为一个带有一个数值属性的对象；默认的子访问器中每个输入值是一个带有一个子数组的对象。

#### [#](https://github.com/mbostock/d3/wiki/Hierarchy-Layout" \l "wiki-_hierarchy) hierarchy(*root*)

运用此层级布局，将返回关联于指定root节点的数组。布局的输入参数是层级的根节点，并且输出域返回的值是一个代表所有节点的计算位置值的数组。在每个节点中有几个属性：

* parent – 父节点，对于根节点来说为null。
* children – 子节点的数组，对于末尾的叶子节点为null。
* value – 节点值，即值访问器的返回值。
* depth – 节点深度，根节点的深度为0。

此外，大多数的层级布局也会计算x和y作为节点的位置；有关详细内容，请参阅实现类。

#### [#](https://github.com/mbostock/d3/wiki/Hierarchy-Layout" \l "wiki-links) hierarchy.links(*nodes*)

鉴于指定的节点数组返回一个对象数组，这些对象代表着每个节点从父节点到子节点的链接关系。叶子节点将不会再有这种链接关系。每个链接关系是一个拥有两个属性的对象：

* source – 父节点（如上所述）。
* target – 子节点。

该方法用于取回一组适合显示的链接关系，通常与[diagonal](https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-diagonal)形状生成器协同使用。例如：

svg.selectAll("path")

.data(partition.links(nodes))

.enter().append("path")

.attr("d", d3.svg.diagonal());

#### [#](https://github.com/mbostock/d3/wiki/Hierarchy-Layout" \l "wiki-children) hierarchy.children([*accessor*])

如果参数accessor被指定，将设置此指定的子访问器方法。如果参数accessor没有被指定，则返回当前子访问器方法，默认情况下输入数据是一个带有一个子数组的对象：

function children(d) {

return d.children;

}

通常，通过使用[d3.json](https://github.com/mbostock/d3/wiki/Requests" \l "wiki-d3_json)来装载节点层级是非常方便的，并且其还代表输入层级作为一个嵌套的[JSON](http://json.org)对象。例如：

{

"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}

]

}

]

}

]

}

层级布局的根节点首先将调用子访问器。如果访问器返回null，那么该节点便是一个叶子节点并且此布局遍历终止。否则，访问器将返回一个代表子节点数据元素的数组。此访问器调用时有node和depth参数。

#### [#](https://github.com/mbostock/d3/wiki/Hierarchy-Layout" \l "wiki-sort) hierarchy.sort([*comparator*])

如果参数比较器comparator被指定，将通过指定的比较器方法来设置此布局中同级节点的排序顺序。如果比较器没有被指定，将返回当前组排序顺序，默认为相关输入值的数值属性的降序排序。

function comparator(a, b) {

return b.value - a.value;

}

此比较器方法将被一对节点调用，通过传递每个节点的输入数据。一个null比较器禁用排序，并使用树遍历顺序。比较器可能也会通过使用[d3.ascending](https://github.com/mbostock/d3/wiki/Arrays" \l "wiki-d3_ascending)或[d3.descending](https://github.com/mbostock/d3/wiki/Arrays" \l "wiki-d3_descending)来实现。

#### [#](https://github.com/mbostock/d3/wiki/Hierarchy-Layout" \l "wiki-value) hierarchy.value([*value*])

如果参数value被指定，将设置值访问器为指定的方法。如果参数没有被指定，返回当前的值访问器。默认的访问器采取为输入数据将为一个带有一个数值属性的对象。

function value(d) {

return d.value;

}

此值访问器将被每个输入数据元素调用，并且一定会返回一个代表此节点数值的数字。对于像treemaps 的area-proportional（区域-成比例）布局来说，这个值被用来设置每一个成比例节点区域到此值上；对于其他层级布局，这个值将对布局没有任何影响。

#### [#](https://github.com/mbostock/d3/wiki/Hierarchy-Layout" \l "wiki-revalue) hierarchy.revalue(*root*)

将从root（根节点）开始的指定树中每个节点重新评估求值，不会重新排序或重新计算子节点。该方法被用来再计算每个节点的值，不会改变层级的任何结构。从根本上来说，它支持[sticky treemaps](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-sticky)。

# Histogram Layout

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [API Reference](https://github.com/mbostock/d3/wiki/API-Reference) ▸ [Layouts](https://github.com/mbostock/d3/wiki/Layouts) ▸ **Histogram Layout**

### [直方图(Histogram)](https://github.com/mbostock/d3/wiki/Histogram-Layout)

* [d3.layout.histogram](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-histogram) - 构建一个默认直方图(用来表示一组离散数字的分布,横轴表示区间,纵轴表示区间内样本数量或样本百分比).
* [histogram.value](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-value) - 获取或设置值访问器.
* [histogram.range](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-range) - 获取或设置合法值范围.
* [histogram.bins](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-bins) - 指定如何将数据分组到不同的区间(bin)里, 返回一个[构造函数](https://github.com/mbostock/d3/wiki/Histogram-Layout" \l "wiki-_histogram) .
* [histogram](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-_histogram) - 根据已设置的区间将数据分组,返回已分组的二维数组(compute the distribution of data using quantized bins).
* [histogram.frequency](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-frequency) - 设置直方图Y轴值是区间内数据的总量还是百分比(compute the distribution as counts or probabilities).

直方图布局展示了数据的分配，通过分组离散数据点到容器中。实例使用请看[bl.ock 3048450](http://bl.ocks.org/mbostock/3048450)。

#### [#](https://github.com/mbostock/d3/wiki/Histogram-Layout" \l "wiki-histogram) d3.layout.histogram()

构造一个新的直方图方法，默认的值访问器，取值范围方法和容器方法。默认情况下，直方图方法返回频率。这个返回的布局对象是一个对象和一个方法。也就是说：你可以像调用任何其他方法一样调用布局方法，并且该布局有额外的改变自己行为的方法。就像d3中其他类一样，布局遵循这个链式格局方法，setter方法将会返回布局本身，允许通过简洁语句多次调用setter方法。

#### [#](https://github.com/mbostock/d3/wiki/Histogram-Layout" \l "wiki-_histogram) histogram(*values*[, *index*])

在指定的值（values）数组中求直方图方法的值。一个可选的参数索引（index）也可被指定，此索引将沿着取值范围和容器方法来传递值。此返回值是一个二维数组：外层数组中的每一个元素代表一个容器，并且每一个容器包含相关于输入值的元素。此外，每个容器对象中有3个属性：

* *x* – 容器的下限边界（包含）。
* *dx* – 容器的宽度；x+dx是容器的上限边界（包含）。
* *y* – 计数（如果频率为真），或者概率（如果频率为假）。

注意，在频率模式下y属性与长度属性相同。

#### [#](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-value) histogram.value([*accessor*])

指定如何从相关数据中提取一个值；访问器是一个被每个传递给直方图（[histogram](https://github.com/mbostock/d3/wiki/Histogram-Layout" \l "wiki-_histogram)）的输入值调用的方法，相当于在计算直方图前调用*values.map(accessor)*方法。默认值方法是内置的[Number](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Number)，类似于身份特性方法（which is similar to the identity function.）。如果参数accessor没有被指定，返回当前的值访问器。

#### [#](https://github.com/mbostock/d3/wiki/Histogram-Layout" \l "wiki-range) histogram.range([*range*])

指定直方图的范围。指定范围值之外的值将被忽略。此方位可能被指定为范围最小值与最大值的双元素数组，或指定为一个方法返回数组范围值和当前传递给直方图[histogram](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-_histogram)的索引值。默认的范围（最小值和最大值）是值的范围。如果range没有被指定，则返回当前范围。

#### [#](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-bins) histogram.bins() [#](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-bins) histogram.bins(*count*) [#](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-bins) histogram.bins(*thresholds*) [#](https://github.com/mbostock/d3/wiki/Histogram-Layout" \l "wiki-bins) histogram.bins(*function*)

指定如何将直方图中的值放在容器中。如果没有指定参数，返回当前的放值方法，默认为一个[Sturges' formula](http://en.wikipedia.org/wiki/Histogram)的实现类，将值间隔均匀地分成容器。如果指定参数count（计数），值范围（[range](https://github.com/mbostock/d3/wiki/Histogram-Layout" \l "wiki-range)）将均匀地划分成指定数量的容器。

如果指定一个阈值（*thresholds*）的数组，它定义此阈值应用于容器，开始于最左边（最低）值和结束于最右边（最高）值。n+1阈值指定n个容器。任何比thresholds[1]小的值将被放置在第一个容器中；同样任何大于或等于thresholds[thresholds.length - 2]的值将被放在最后一个容器中。因此，尽管第一个和最后一个阈值不会用于容器中，但它们任然必须去分别定义第一个容器的x属性和最后一个容器的dx属性。

最后，如果指定参数function，即指定一个值装箱方法。当该布局传递数据（传递当前范围（[range](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-range)），传递值数组和当前传递给直方图（[histogram](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-_histogram)）的索引值）的时候此方法被调用。然后该方法必须返回一个前边内容提到的阈值数组。

#### [#](https://github.com/mbostock/d3/wiki/Histogram-Layout" \l "wiki-frequency) histogram.frequency([*frequency*])

指定直方图的y值是否是一个数值（频率）或一个概率（密度）；默认的是一个频率。如果没有指定参数频率frequency，则返回当前是否为频率的布尔值。

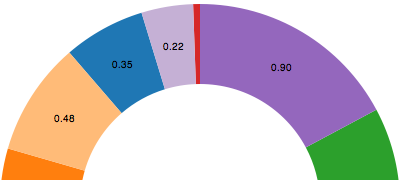
# Pie Layout

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [API Reference](https://github.com/mbostock/d3/wiki/API-Reference) ▸ [Layouts](https://github.com/mbostock/d3/wiki/Layouts) ▸ **Pie Layout**

### [Pie](https://github.com/mbostock/d3/wiki/Pie-Layout)

* [d3.layout.pie](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-pie) – 构造一个新的默认饼图布局。
* [pie](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-_pie) – 计算每一个饼图或圆环图的每个弧的起始和结束的角度。
* [pie.value](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-value) – 得到或设置值访问器。
* [pie.sort](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-sort) – 控制设置饼图切片的顺时针排序。
* [pie.startAngle](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-startAngle) – 得到或设置饼图的起始角度。
* [pie.endAngle](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-endAngle) -得到或设置饼图的结束角度。

饼图布局为一个方便计算构成饼图或圆环图的弧的起始和结束角度的布局。



你不需要使用饼图布局去创建一个饼图图表；如果你愿意的话，你可以直接使用弧形状（[arc shape](https://github.com/mbostock/d3/wiki/SVG-Shapes" \l "wiki-arc)）。该饼图布局会使一个数据数组简单便捷地转换为一个带有起始角度和结束角度属性的对象数组（角度范围为0到2π），然后将此传递给弧形状生成器。

#### [#](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-pie) d3.layout.pie()

构建一个新的饼图方法，默认值访问器（number类型），分类比较器（值递减），起始角度（0）和结束角度（2π）。该返回的布局对象是一个对象和一个方法。也就是说：你可以调用此布局想调用其他方法一样，并且该布局还有改变本身行为的方法。就像d3其他类一样，布局遵循链式方法，setter方法返回布局本身，允许使用简洁语句多次调用setters方法。

#### [#](https://github.com/mbostock/d3/wiki/Pie-Layout" \l "wiki-_pie) pie(*values*[, *index*])

在指定的值（values）数组上给饼图方法定值。一个可选的参数索引值（index）可能被指定，该索引值沿着起始和结束角度来传递。返回值是一个弧描述符的数组。

* value – 数据值，值访问器返回的值。
* startAngle – 弧的起始角度。
* endAngle – 弧的结束角度。
* data – 此弧形的原始数据。

该元素将以原始顺序返回，匹配值，即使应用一个分类([sort](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-sort))排序；这保留了原始值数组中每个元素的索引值，如果你想生成一个分类颜色或其他显示属性，通过使用索引值实现是很好的方式。

#### [#](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-value) pie.value([*accessor*])

指定如何从关联数据中提取一个值（例如，设置饼图布局访问器去应用）；参数访问器（*accessor*）是一个被每一个传递给[pie](https://github.com/mbostock/d3/wiki/Pie-Layout" \l "wiki-_pie)的输入值调用，相当于在计算饼图布局之前调用values.map(accessor)。此方法传递两个参数：当前数据和当前索引值。默认值方法是内置的[Number](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Number)方法，它类似于身份特性函数（which is similar to the identity function.）。如果访问器没有被指定，返回当前值访问器。

#### [#](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-sort) pie.sort([*comparator*])

如果指定参数comparator，通过指定的比较方法为布局设置数据的分类排序。若传递null则禁止使用排序。如果参数comparator没有被指定，则返回当前分类排序。此分类排序默认为值递减排序方式。分类保存输入值的索引值（和z-index），仅会影响角度的计算。此比较器方法将被传递给[pie](https://github.com/mbostock/d3/wiki/Pie-Layout" \l "wiki-_pie)的值数组中成对的数据元素调用。比较器方法也可以通过使用[d3.ascending](https://github.com/mbostock/d3/wiki/Arrays#wiki-d3_ascending)和[d3.descending](https://github.com/mbostock/d3/wiki/Arrays" \l "wiki-d3_descending)来实现。

#### [#](https://github.com/mbostock/d3/wiki/Pie-Layout" \l "wiki-startAngle) pie.startAngle([*angle*])

如果指定参数角度（angle），将饼图布局的起始角度设置为此指定的值。如果没有指定参数angle，则返回当前值，默认为0.该起始角度也可指定为一个常数或一个方法；如果是一个方法，当[pie](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-_pie)被调用时只求值一次，将传递当前数据和索引值。

#### [#](https://github.com/mbostock/d3/wiki/Pie-Layout" \l "wiki-endAngle) pie.endAngle([*angle*])

如果指定参数角度（angle），将饼图布局的结束角度设置为此指定的值。如果没有指定参数angle，则返回当前值，默认为2π。该结束角度也可指定为一个常数或一个方法；如果是一个方法，当[pie](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-_pie)被调用时只求值一次，将传递当前数据和索引值。

Harry

2014年5月4日 星期日