- 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

SVG有大量的内置图形。例如：轴对齐矩形和圆形。在D3中结合D3的路径数据生成器和SVG的路径（path）元素能更加灵活地绘制图形。

形状生成器，例如d3.svg.arc返回的，既是一个对象又是一个函数。也就是说，你可以像其他函数一样调用形状生成器。形状生成器对象遵循链式语法。

SVG 元素

所有的SVG形状都可以使用transform属性。你可以将转换直接应用在SVG形状上，或者应用在g元素上。这样你可以旋转或者转换形状。形状可使用 fill 和 stroke 样式填充或者描边（你也可以使用同名称的属性，但为了兼容外部样式时推荐使用样式）。

svg:**rect** x="0" y="0" width="0" height="0" rx="0" ry="0"

矩形元素 rect 定义一个轴对齐的矩形。使用*x*，*y*属性定位矩形的左上角，使用*width*和*height*指定尺寸。使用可选参数 *rx* 和 *ry*可以生成圆角矩形。

svg:**circle** cx="0" cy="0" r="0"

圆形元素circle定义了一个基于中心点和半径的圆形。使用*cx*和*cy*属性指定圆的中心。使用*r*属性指定圆的半径。

svg:**ellipse** cx="0" cy="0" rx="0" ry="0"

椭圆元素ellipse定义了一个轴对齐的椭圆，基于一个中心点和两个半径。中心点使用*cx*和*cy*属性定位。而半径使用*rx*和*ry*指定。

svg:**line** x1="0" y1="0" x2="0" y2="0"

直线元素line定义线段始于一个点而终止于另一个点。第一个点使用*x1*和*y1*属性指定，第二个点使用*x2*和*y2*属性指定。线元素常用来画规则，参考线，轴和刻度线。

svg:**polyline** points=""

折线元素polyline定义一组相连的直线段。通常折线元素定义的是开放的形状。使用*points*属性指定构成折线的点。在D3中推荐将d3.svg.line路径生成器和path元素一起使用，这样会更方便灵活。

svg:**polygon** points=""

多边形元素polygon定义一个由一组相连的直线段构成的闭合图形。使用points属性指定构成多边形的点。注意：在D3中通常将d3.svg.line 路径生成器和path元素一起使用。线可以使用路径关闭命令(closepath)"Z"闭合。

svg:**text** x="0" y="0" dx="0" dy="0" text-anchor="start"

文本元素text定义由文本组成的图形元素。使用*x*，*y*属性来控制文本元素的锚点。另外，可以使用*dx*，*dy*属性控制文本元素的偏移。由text-anchor属性控制横向文本对齐方式，下面是一些例子：

```
<svg:text text-anchor="start">left-align, bottom-baseline</svg:text>
<svg:text text-anchor="middle">center-align, bottom-baseline</svg:text>
<svg:text text-anchor="end">right-align, bottom-baseline</svg:text>
<svg:text dy=".35em" text-anchor="start">left-align, middle-baseline</svg:text>
```

```
<svg:text dy=".35em" text-anchor="middle">center-align, middle-baseline</svg:text>
<svg:text dy=".35em" text-anchor="end">right-align, middle-baseline</svg:text>
<svg:text dy=".71em" text-anchor="start">left-align, top-baseline</svg:text>
<svg:text dy=".71em" text-anchor="middle">center-align, top-baseline</svg:text>
<svg:text dy=".71em" text-anchor="end">right-align, top-baseline</svg:text>
```

可以用SVG的基线对齐属性(baseline alignment properties)指定文字的基线。字体的颜色通常使用*fill*样式指定（你也可以使用*stroke*），字体使用*font*，*font-family*，*font-size*和相关的样式控制。一些浏览器也支持CSS3属性，例如*text-shadow*。

svg:**path** d="" transform=""

路径元素path代表了形状的轮廓可以被填充，描边，用做剪裁路径，或者这三者的任意组合。*d*属性定义了路径数据，这是路径命令的一个迷你语言(mini-language)，例如*moveto* (M)，*lineto* (L)和*closepath* (Z)。路径元素是一个SVG中所有其他形状的概括，几乎可以用来画出任何东西!

路径数据生成器

为了简化路径元素的*d*属性的构造，D3包含很多辅助类用来生成路径数据。如果你的数据是*xy*坐标序列，你可以定义访问器函数，然后用路径生成器制造路径数据。例如：

```
var line = d3.svg.line()
    .x(function(d) { return d.x; })
    .y(function(d) { return d.y; })
    .interpolate("basis");
```

然后，你可以使用下面函数设置*d*属性：

```
g.append("path")
    .attr("d", line);
```

不论数据是否绑定到 `g` 元素（在这个例子中）都将传递给 `line` 实例。这样，数据必须指定为一个数组。对于数据数组中的每个元素，*x*和*y*访问器函数用来抽出控制点坐标。

路径生成器，例如d3.svg.line返回的既是一个函数又是一个对象。这样，你就可以像其他函数一样调用生成器了。并且生成器还有额外的方法来改变它的行为。像D3中的其他类一样，路径生成器遵循方法链模式其中的setter方法返回生成器自身。允许在一个简单的声明中调用多个setter方法。

d3.svg.**line**()

构造一个新的线生成器使用默认的*x*和*y*访问器函数（假设输入数据是一个两元素数字数组；详见下文），和线性插值器。返回的函数生成路径数组为开口分段线性曲线，折线或者，如折线图：


line

通过改变插值器，你可以生成样条线以及步长函数。另外，不要害怕最后粘上其他路径命令。例如，如果你想生成一个封闭的路径，追加closepath (Z)命令：

```
g.append("path")
    .attr("d", function(d) { return line(d) + "Z"; });
```

线生成器设计来和面积生成器(area)一起使用。例如，当生成面积图时，你可能使用带有填充属性的面积生成器，以及带有描边属性的线生成器来突出面积的上边缘。当线生成器只是用来设置*d*属性，你可以使用SVG样式和属性控制线的展示，例如*fill*，*stroke*和*stroke-width*。

line(*data*)

为指定的*data*元素数组返回路径数据字符串，如果路径是空则返回 `null` 。

line.**x**([*x*])

如果指定了*x*，为指定的函数或常量设置*x*访问器。如果*x*没有指定，返回当前的*x*访问器。这个访问器为传递给*x*线生成器的数据数组中的每个元素。默认的访问器假设每个输入的元素是一个二元素数字数组：

```
function x(d) {
  return d[0];
}
```

通常，一个*x*访问器是指定的，因为输入数据是不同格式的，或者因为你想使用比例尺(数值比例尺)。例如，如果你的数据指定为一个含有 `x` ， `y` 属性的的对象，而不是一个元组，你可以反引用这些属性同时应用比例尺：

```
var x = d3.scale.linear().range([0, w]),
    y = d3.scale.linear().range([h, 0]);

var line = d3.svg.line()
    .x(function(d) { return x(d.x); })
    .y(function(d) { return y(d.y); });
```

*x*访问器像D3中其他值函数一样的方式调用。函数的*this*上下文就是选择中的当前元素（通常，相同的*this*上下文调用线函数；然而，在通常情况下线生成器传递给attr操作符，*this*上下文将关联DOM元素）。函数传入两个入参当前的数据d和当前的索引i。*this*上下文，索引就是控制点中的索引，而不是选择中当前元素的索引。*x*访问器按照数据数组中指定的顺序，每个元素恰好调用一次。这样，就可能指定不确定性访问器，例如随机数生成器。也可以指定x访问器为一个常量而不是函数，在所有点都有相同的*x*坐标情况下。

line.**y**([*y*])

如果指定了*y*，为指定的函数或常量设置*y*访问器。如果*y*没有指定，返回当前的*y*访问器。这个访问器为传递给*y*线生成器的数据数组中的每个元素。默认的访问器假设每个输入的元素是一个二元素数字数组：

```
function y(d) {
  return d[1];
}
```

怎样指定一个*y*访问器，参见相似的x访问器。 注意：像所有的其他图形库，SVG使用左上角作为原点。这样在屏幕中较高的*y*值就**更低**。对于可视化我们常常想要原点是在左下角。一个简单的方法实现这个可以使用 `range([h, 0])` 替代 `range([0, h])` 反转*y*比例尺的范围。

line.**interpolate**([*interpolate*])

如果指定了*interpolate* 参数，就会设置插值器模式为指定的字符串或者函数。如果没有指定，就返回当前的插值器模式。支持下面的命名插值器模式：

- linear -分段线性片段，如折线。

- linear-closed –闭合直线段，以形成一个多边形。
- step - 水平和垂直段之间交替，如在step函数中。
- step-before -垂直和水平段之间交替，如在step函数中。
- step-after -水平和垂直段之间交替，如在step函数中。
- basis - B样条曲线(B-spline)，在两端的控制点的重复。
- basis-open – 开放B样条曲线，首尾不会相交。
- basis-closed -封闭B样条曲线，如在一个循环。
- bundle – 等价于basis, 除了使用tension参数拉直样条曲线。
- cardinal – 基本样条曲线(Cardinal spline)，在末端控制点的重复。
- cardinal-open –开放的基本样条曲线，首尾不会相交，但和其他控制点相交。
- cardinal-closed -封闭基本样条曲线，如在一个循环。
- monotone - 三次插值(cubic interpolation)，可以保留y的单调性。

其中一些插值模式的行为可能通过指定的张力tension进行进一步定制。

如果*interpolate* 是一个函数，然后这个函数将被调用来反转一个形如 `[[x0, y0], [x1, y1], …]` 的点数组，返回一个SVG路径数据字符串(SVG path data string)，用来展示线。字符串开始处的"M" 隐含的，不应该被返回。例如，线性插值被实现为：

```
function interpolateLinear(points) {
  return points.join("L");
}
```

这相当于（并且比这更有效）：

```
function interpolateLinear(points) {
  var path = "";
  for (var i = 0; i < points.length; i++) {
    if (i) path += "L";
    path += points[i][0] + "," + points[i][1];
  }
  return path;
}
```

参见bl.ocks.org/3310323是另外一个自定义线性插值器。

line.**tension**([*tension*])

如果指定*tension* 参数，设置基本样条曲线插值器拉伸为指定的范围[0, 1]内的数字。如果没有指定*tension* 放回当前的拉伸。拉伸只在基本样条曲线模块有效：样条曲线、开放样条曲线和闭合样条曲线。默认拉伸值是0.7。在某种意义上，这可以解释为切线的长度； 1将产生全为零的切线，0产生Catmull-Rom样条曲线(Catmull-Rom spline)。

注意：拉伸必须指定为一个常量，而不是函数，因为它是整个线的常数。但是，也可能使用同一个生成器生成的不同拉伸生成多条线。例如：

```
svg.selectAll("path")
    .data([0, 0.2, 0.4, 0.6, 0.8, 1])
  .enter().append("path")
    .attr("d", function(d) { return line.tension(d)(data); });
```

在这个例子中(见实例live version)，拉伸是在线生成器调用之前设置的，这样就可以在线中使用相同的数据生成不同的路径了。

line.**defined**([*defined*])

取得或者设置访问器函数控制线的生成。如果指定了*defined* 参数，就设置新的访问器函数并返回线。如果没有指定*defined*，就返回当前生成器默认就是 `function() { return true; }`。定义的访问器可以用来定义是否定义线，通常在和缺失数据一起使用时很有用。生成的数据将自动被分成多个不同的子路径，跳过未定义数据，例如，如果你想忽略非数字（或者未定）的*y*值，可以这样写：

```
line.defined(function(d) { return !isNaN(d[1]); });
```

若一组数据有定义却被置于未定义数据中， 该数据不可视。

d3.svg.line.**radial**()

构造一个新的径向线生成器使用默认的半径和角度访问器函数（假设输入数据是一个两元素的数字数组；详见下文），和一个线性插值器。返回的函数为开放的分段线性曲线，或折线与笛卡尔线生成器(line)生成路径数据。

line(*data*)

为指定的*data*元素数组返回路径数据字符串

line.**radius**([*radius*])

如果指定了*radius* 参数，则设置半径访问器为指定的函数或常量。如果*radius* 没有指定，返回当前的半径访问器。这个访问器为传递给线生成器的数据数组中的每个元素调用。默认的访问器假设每个输入元素都是一个两元素的数字数组：

```
function radius(d) {
  return d[0];
}
```

这个方法是笛卡尔line.x方法的变形。

line.**angle**([*angle*])

如果指定了*angle*参数，设置角度访问器为指定的函数或者弧度为单位的常量。如果没有指定*angle* 参数，返回角度访问器。访问器为传递给线生成器的数据数组中的每个元素调用。默认的访问器假设每个输入元素是一个两元素的数字数组。

```
function angle(d) {
  return d[1];
}
```

这个方法就是笛卡尔line.y方法的变形。

line.**interpolate**([*interpolate*])

参见笛卡尔line.interpolate方法， 投影到笛卡尔空间之后惨发生插值。

line.**tension**([*tension*])

参见笛卡尔 line.tension方法， 投影到笛卡尔空间之后惨发生插值。

line.**defined**([*defined*])

参见笛卡尔line.defined方法

d3.svg.**area**()

构造一个新的面积生成器，使用默认的*x*（横坐标），*y0*（基线），*y1*（顶线）访问器函数（假定输入数据是一个双元素数字数组，详见下文）和线性插值器。对于分段线性曲线，或多边形，返回的函数生成一个封闭的path数据，如在面积图中:


area

从概念上讲，多边形的形成是通过使用两条线(lines)：顶线的形成是通过*x*（横坐标）和*y1*（顶线）访问器方法，从左至右生成。底线是添加到这一条线上，使用x （横坐标）和*y0*（基线）访问器方法，从右到左去生成。通过将转换属性(transform)设置为旋转path元素90度，您还可以生成垂直的面积图。通过改变插值，您还可以生成splines和台step函数。

区域生成器被设计为与线生成器(line)一起使用的。例如，当生成区域图表时，你可能用一个带有填充样式的区域生成器和一个带有描边样式的线生成器，此描边样式可以使区域的顶部边缘更加突出显示。由于区域生成器只可以用于设置*d*属性，所以你可以通过使用标准的SVG样式和属性去控制区域的外观，例如填充样式*fill*。

创建streamgraphs (堆叠区域图)，使用stack（叠层）布局。此布局可以为一个系列中的每个值去设置*y0*属性，这个系列值可从*y0*（基线），*y1*（顶线）访问器中使用。注意，每个系列中必须有相同数量的值，并且每个值必须有相同的*x*横坐标；如果你在每系列里有缺失数据或不一致的*x*横坐标，那么在计算叠层布局之前，你必须重新取样和插入你的数据。

area(*data*)

对于指定的数据元素数组*data*，返回路径（path）数据字符串；当路径是空的时候返回null。

area.**x**([*x*])

如果指定了*x*参数，设置*x*（横坐标）访问器为指定的方法或常量。如果没有指定*x*，返回当前*x*访问器。为传递给区域生成器的数据数组中的每个元素，调用该访问器。默认访问器中假定每个输入元素是双元素数字数组:

```
function x(d) {
  return d[0];
}
```

通常情况下，*x*访问器被指定是因为输入数据是不同格式的，或者因为你想应用一个比例尺。例如，如果你的数据被指定为一个带有 `x` 和 `y` 属性的对象，而不是一个元组，那么你可能会引用这些属性同时应用比例尺：

```
var x = d3.scale.linear().range([0, w]),
    y = d3.scale.linear().range([h, 0]);

var area = d3.svg.area()
    .x(function(d) { return x(d.x); })
    .y0(h)
    .y1(function(d) { return y(d.y); });
```

*x*访问器会和D3中其他值函数一样的方式被调用。函数中的*this*上下文就是选择中的当前元素。（从技术上讲，相同的*this*上下文调用区域函数；然而，通常情况下，区域生成器被传递到attr操作符，*this*上下文将会被关联到DOM元素上）。函数被传递两个参数，当前的数据（d）和当前的索引值（i）。在*this*上下文中，索引值就是控制数据点的数组

中的索引，而不是当前选择元素的索引。*x*访问器在每一个数据中按照数据数组指定的顺序被恰好调用一次。因此，可以指定一个不确定的访问器，例如随机数生成器。也可以指定*x*访问器为一个常量，而不是一个函数，在这种情况下，所有的点将有相同的横坐标值。

area.**x0**([*x0*])

…

area.**x1**([*x1*])

…

area.**y**([*y*])

…

area.**y0**([*y0*])

如果*y0*被指定，*y0*（基线）访问器为指定的方法或常量。如果没有指定*y0*，返回当前*y0*（基线）访问器。将为传递给区域生成器数据数组中的每个元素，调用该访问器函数。默认的访问器是常量0，也就是使用一个固定的基线*y* = 0。对于如何指定一个*y0*（基线）访问器的例子，请看类似的x访问器。

area.**y1**([*y1*])

如果指定参数*y1*，*y1*访问器为指定的方法或常量。如果没有指定*y1*，就会返回当前的*y1*访问器。将为传递给区域生成器数据数组中的每个元素，调用该访问器函数。默认的访问器假定每个输入元素是一个双元素的数字数组:

```
function y1(d) {
  return d[1];
}
```

关于如何指定一个*y1*（顶线）访问器的一个例子，请看类似的x访问器。注意，像大多数其他的图形库，SVG使用顶部-左侧作为原点，因此更高的数值*y*将会在屏幕**更低**的位置。对于可视化而言，我们通常希望原点是位于底部-左侧的位置。可以做到这一点的一个简单的方法便是转化*y*比例尺的范围，即通过使用范围([h,0])而不是范围([0,h])。

area.**interpolate**([*interpolate*])

如果指定插值器*interpolate*，便设置插值器模式为指定的字符串或函数。如果没有指定插值器*interpolate*，返回当前插值器模式。插值器模式支持以下命名:

- linear（线性）：分段的线性片段，如折线。
- step（步）：水平和垂直片段之间交替，如台阶函数。
- step-before：垂直和水平片段之间交替，如台阶函数。
- step-after： 水平和垂直片段之间交替，如台阶函数。
- basis：一个B-spline，在末尾控制点的重复。
- basis-open：一个开放的B-spline；首尾不相交。
- cardinal：一个Cardinal spline，在末尾控制点的重复。
- cardinal-open：一个开放的Cardinal spline；首尾不相交，但是会和其他控制点相交。
- monotone -立方插值(cubic interpolation)保存*y*值得单调性。

其中一些插值模式的行为通过指定的张力(tension)可能被进一步自定义。从技术上讲,同样也会支持basis-closed和cardinal-closed的插值模式，但这些模式应用在一条线上比应用在一个区域上更有意义。

如果参数*interpolate*是一个函数，那么这个函数将被调用去转换一个形式为[[x0, y0], [x1, y1], …]的数组，返回一个SVG路径数据字符串(SVG path data string)，它将用于显示区域。字符串起始位置的“M”是隐含的，不会被返回。例如，线性插值的实现为:

```
function interpolateLinear(points) {
  return points.join("L");
}
```

这等效于（并更加高效）：

```
function interpolateLinear(points) {
  var path = "";
  for (var i = 0; i < points.length; i++) {
    if (i) path += "L";
    path += points[i][0] + "," + points[i][1];
  }
  return path;
}
```

请查看另一个自定义插值的样例：bl.ocks.org/3310323

area.**tension**([*tension*])

如果指定张力*tension*，便将基数样条(Cardinal spline)插值张力（tension）设置为指定区间[0,1]内的数字。如果没有指定张力*tension*，返回当前的张力值。张力值只会影响基本插值模式：cardinal，cardinal-open 和cardinal-closed。默认的张力值是0.7。在某种意义上，这可以解释为切线长度；值1将产生零切线，值0将产生一个Catmull-Rom spline。注意，张力值必须指定为一个常数，而不是一个函数，因为它是整个区域的常数。

area.**defined**([*defined*])

获取或设置访问器函数，此方法用来控制已经明确定义的区域。如果指定参数 `defined` （定义），便设置新的取值函数（访问器）并返回该区域。如果没有指定 `defined` （定义），则返回当前访问器的默认函数，即 `function() {return true;}` 。定义访问器可用于定义哪些区域是定义的，哪些是未定义的，通常对于有数据缺失情况是很有用的；生成的路径数据将自动被分为多个不同的子路径，同时跳过未定义的数据。例如，如果您想要忽略那些不是一个数字的（或未定义的）*y*值，你可以这样写:

```
area.defined(function(d) { return !isNaN(d[1]); });
```

d3.svg.area.**radial**()

...

area(*data*)

对于指定的数组中的数据元素*data*，返回路径数据字符串。

area.**radius**([*radius*])

...

area.**innerRadius**([*radius*])

...

area.**outerRadius**([*radius*])

...

area.**angle**([*angle*])

...

area.**startAngle**([*angle*])

...

area.**endAngle**([*angle*])

...

d3.svg.**arc**()

构造一个新的弧生成器，使用默认的内半径、外半径、开始弧度和结束弧度访问器（假定输入数据是包含匹配于访问器的命名属性的对象，详见下文）；而默认的访问器假定弧的尺寸是动态指定的，当然设置一个或多个的尺寸为常量也是很常见的，例如饼图的内半径设为0；返回的函数为封闭的实心弧度生成路径数据，如饼图或环图：

事实上，有四种可能性：圆disk（内半径为0，角度跨度大于等于2π），扇形circular sector（内半径为0，角度跨度小于2π），环形 annulus（内半径大于0，角度跨度为2π），以及环形扇区(annular sector)（内半径大于0，并且角度跨度小于2π）。

arc(*datum*[, *index*])

为指定的*datum*参数返回路径数据字符串。可选参数*index* 可能会指定，传递给弧度访问器函数。

arc.**innerRadius**([*radius*])

通常情况下，指定内半径访问器是由于：输入数据是不同的格式、或你想要应用比例尺，亦或你想为圆环指定个恒定的内半径。内半径访问器会像D3的其他函数一样被调用；函数中的this代表选择中的当前元素（技术上来说，this上下文调用弧函数；然而一般情况下，弧生成器传递给attr操作符， this上下文将关联DOM元素）；函数传递两个参数：当前数据d和索引i；当然，也可以指定其为一个常数而不是函数。默认访问器假定输入数据是带有适当命名属性的对象：

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

新增内容

```
The arc generator arguments (typically `d` and `i`) and context (`this`) are passed
through to the accessor function. An inner radius accessor function is useful for
handling data in a different format or for applying a [quantitative scale]
(Quantitative Scales) to encode data. A constant inner radius may be used to create a
standard pie or donut chart.
```

arc.**outerRadius**([*radius*])

通常情况下，指定外半径访问器是由于：输入数据是不同的格式、或你想要应用比例尺，亦或你想为圆环指定个恒定的外半径。外半径访问器会像D3的其他函数一样被调用；函数中的this代表选择中的当前元素（技术上来说，this上下文调用弧函数；然而一般情况下，弧生成器传递给attr操作符， this上下文将关联DOM元素）；函数传递两个参数：

当前数据d和索引i；当然，也可以指定其为一个常数而不是函数；默认访问器假定输入数据是带有适当命名属性的对象：

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

New Content 新增内容

```
The arc generator arguments (typically `d` and `i`) and context (`this`) are passed
through to the accessor function. An outer radius accessor function is useful for
handling data in a different format or for applying a [quantitative scale]
(Quantitative Scales) to encode data. A constant outer radius may be used to create a
standard pie or donut chart.

<a name="arc_cornerRadius" href="#arc_cornerRadius">#</a> arc.**cornerRadius**
([*radius*])

If *radius* is specified, sets the corner radius accessor to the specified function or
constant. If *radius* is not specified, returns the current outer radius accessor,
which defaults to zero. Although a constant corner radius is typically used, the
corner radius may also be specified as a function. The arc generator arguments
(typically `d` and `i`) and context (`this`) are passed through to the accessor
function.

<a name="arc_padRadius" href="#arc_padRadius">#</a> arc.**padRadius**([*radius*])

If *radius* is specified, sets the pad radius accessor to the specified function or
constant. If *radius* is not specified, returns the current pad radius accessor, which
defaults to “auto”. The “auto” pad radius method computes the pad radius based on the
previously-computed [inner](#arc_innerRadius) and [outer](#arc_outerRadius) as:

javascript
function padRadius(innerRadius, outerRadius) {
  return Math.sqrt(innerRadius * innerRadius + outerRadius * outerRadius);
}


This implementation is designed to preserve the approximate relative area of arcs in
conjunction with [pie.padAngle](Pie-Layout#padAngle).

The pad radius is the radius at which the [pad angle](#arc_padAngle) is applied: the
nominal padding distance between parallel edges of adjacent arcs is defined as
padRadius * padAngle. (The padding distance may be smaller if the inner radius is
small relative to the pad angle.) The pad radius typically does not need to be changed
from “auto”, but can be useful to ensure parallel edges between arcs with differing
inner or outer radii, as when [extending an arc on hover]
(http://bl.ocks.org/mbostock/32bd93b1cc0fbccc9bf9).

If the pad radius is specified as an accessor function, the arc generator arguments
```

```
(typically `d` and `i`) and context (`this`) are passed through to the accessor
function.
```

arc.**startAngle**([*angle*])

如果指定*angle*，则设置开始角度访问器startAngle-accessor为指定的函数或常数，如果未指定，则返回当前的访问器；默认访问器假定输入数据是带有适当命名属性的对象：

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

角度使用弧度radians表示，尽管SVG中使用的是角度；角度为0对应12点钟的指针方向（负数*y*）并且顺时钟方向继续旋转2π；访问器参数(通常为 `d` 和 `i`)和 `this` 指针传递给弧生成器时调用。

为了构建饼图和环图，需要计算每个弧的起始角度和上个弧的结束角度；使用饼布局 pie layout会非常方便的，即类似于堆叠布局stack；给定一组输入数据，饼布局会调用弧对象来生成开始弧度和结束弧度属性(`startAngle` and `endAngle`)，你也可以使用默认的弧访问器。 开始角度访问器startAngle-accessor会像D3的其他函数一样被调用；函数传递两个参数：当前数据d和索引i；当然，也可以指定其为一个常数。

arc.**endAngle**([*angle*])

如果指定*angle*，则设置开始角度访问器endAngle-accessor为指定的函数或常数，如果未指定，则返回当前的访问器；角度使用弧度radians表示，尽管SVG中使用的是角度；访问器会以传递给弧度生成器的参数形式被调用；默认访问器假定输入数据是包含适当命名属性的对象：

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

为了构建饼图和环图，需要计算每个弧的起始角度和上个弧的结束角度；这保证了你会非常方便的使用饼布局 pie layout，即类似于堆叠布局stack；给定一组数据，饼布局会调用弧对象来生成开始弧度和结束弧度属性，你也可以使用默认的弧访问器。 开始角度访问器endAngle-accessor会像D3的其他函数一样被调用；函数传递两个参数：当前数据d和索引i；当然，也可以指定其为一个常数。

New Content 新增内容

```
<a name="arc_padAngle" href="#arc_padAngle">#</a> arc.**padAngle**([*angle*])

If *angle* is specified, sets the pad angle accessor to the specified function or
constant. If *angle* is not specified, returns the current pad angle accessor, which
defaults to:

javascript
function padAngle(d) {
  return d.padAngle;
}


Angles are specified in radians. If the pad angle is specified as an accessor
function, the arc generator arguments (typically `d` and `i`) and context (`this`) are
passed through to the accessor function.
```

```
Although the pad angle can be specified as a constant, it is preferable to use the
default pad angle accessor and instead use [pie.padAngle](Pie-Layout#padAngle) to
compute the appropriate pad angle, and to recompute the start and end angles of each
arc so as to preserve approximate relative areas.
```

arc.**centroid**(*arguments...*)

计算以指定输入参数*arguments*产生的弧的圆心，通常情况下，参数为：当前数据d和索引i；圆心被定义为内外半径和开始结束角度的极坐标系的中心点；这为圆弧的标签提供了方便的位置信息，例如：

```
arcs.append("text")
    .attr("transform", function(d) { return "translate(" + arc.centroid(d) + ")"; })
    .attr("dy", ".35em")
    .attr("text-anchor", "middle")
    .text(function(d) { return d.value; });
```

或者，你可以使用SVG的变换transform 属性来旋转文本的位置，即使你可能需要转换弧度为角度；另一种可能性是使用 textPath 元素来依照弧的path路径来弯曲文本标签显示。

待翻译

```
<a name="symbol" href="SVG-Shapes#symbol">#</a> d3.svg.**symbol**()

Constructs a new symbol generator with the default type- and size-accessor functions
(that make no assumptions about input data, and produce a circle sized 64 square
pixels; see below for details). While the default accessors generate static symbols,
it is common to set one or more of the accessors using a function, such as setting the
size proportional to a dimension of data for a scatterplot. The returned function
generates path data for various symbols, as in a dot plot:

Note that the symbol does not include accessors for x and y. Instead, you can use the
path element's transform attribute to position the symbols, as in:

javascript
vis.selectAll("path")
    .data(data)
  .enter().append("path")
    .attr("transform", function(d) { return "translate(" + x(d.x) + "," + y(d.y) +
")"; })
    .attr("d", d3.svg.symbol());


In the future, we may add x- and y-accessors for parity with the line and area
generators. The symbol will be centered at the origin (0,0) of the local coordinate
system. You can also use SVG's built-in basic shapes to produce many of these symbol
types, though D3's symbol generator is useful in conjunction with path elements
because you can easily change the symbol type and size as a function of data.

<a name="_symbol" href="SVG-Shapes#_symbol">#</a> **symbol**(*datum*[, *index*])

Returns the path data string for the specified datum. An optional index may be
```

specified, which is passed through to the symbol's accessor functions.

symbol.**type**([*type*])

If type is specified, sets the type-accessor to the specified function or constant. If type is not specified, returns the current type-accessor. The default accessor is the constant "circle", and the following types are supported:

- circle - a circle.
- cross - a Greek cross or plus sign.
- diamond - a rhombus.
- square - an axis-aligned square.
- triangle-down - a downward-pointing equilateral triangle.
- triangle-up - an upward-pointing equilateral triangle.

Types are normalized to have the same area in square pixels, according to the specifiedsize. However, note that different types' sizes may be affected by the stroke and stroke width in different ways. All of the types are designed to be visible when only a fill style is used (unlike the Protovis cross), although they generally look better when both a fill and stroke is used.
The type-accessor is invoked in the same manner as other value functions in D3. The thiscontext of the function is the current element in the selection. (Technically, the same thiscontext that invokes the arc function; however, in the common case that the symbol generator is passed to the attr operator, the this context will be the associated DOM element.) The function is passed two arguments, the current datum (d) and the current index (i). It is also possible to specify the type-accessor as a constant rather than a function.

symbol.**size**([*size*])

If size is specified, sets the size-accessor to the specified function or constant in square pixels. If size is not specified, returns the current size-accessor. The default is 64. This accessor is invoked on the argument passed to the symbol generator. Typically, a size-accessor is specified as a function when you want the size of the symbol to encode aquantitative dimension of data, or a constant it you simply want to make all the dots bigger or smaller. If you want to specify a radius rather than the size, you must do so indirectly, for example using a pow scale with exponent 2.

d3.svg.**symbolTypes**

The array of supported symbol types.

d3.svg.**chord**()

Constructs a new chord generator with the default accessor functions (that assume the input data is an object with named attributes matching the accessors; see below for details). While the default accessors assume that the chord dimensions are all specified dynamically, it is very common to set one or more of the dimensions as a constant, such as the radius. The returned function generates path data for a closed shape connecting two arcs with quadratic Bézier curves, as in a chord diagram:

A chord generator is often used in conjunction with an arc generator, so as to draw annular segments at the start and end of the chords. In addition, the chord layout is

useful for generating objects that describe a set of grouped chords from a matrix, compatible with the default accessors.

**chord**(*datum*[, *index*])

Returns the path data string for the specified datum. An optional index may be specified, which is passed through to the chord's accessor functions.

chord.**source**
([*source*])

If source is specified, sets the source-accessor to the specified function or constant. If sourceis not specified, returns the current source-accessor. The purpose of the source accessor is to return an object that describes the starting arc of the chord. The returned object is subsequently passed to the radius, startAngle and endAngle accessors. This allows these other accessors to be reused for both the source and target arc descriptions. The default accessor assumes that the input data is an object with suitably-named attributes:

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

The source-accessor is invoked in the same manner as other value functions in D3. The thiscontext of the function is the current element in the selection. (Technically, the same thiscontext that invokes the arc function; however, in the common case that the symbol generator is passed to the attr operator, the this context will be the associated DOM element.) The function is passed two arguments, the current datum (d) and the current index (i). It is also possible to specify the source-accessor as a constant rather than a function.

chord.**target**
([*target*])

If target is specified, sets the target-accessor to the specified function or constant. If target is not specified, returns the current target-accessor. The purpose of the target accessor is to return an object that describes the ending arc of the chord. The returned object is subsequently passed to the radius, startAngle and endAngle accessors. This allows these other accessors to be reused for both the source and target arc descriptions. The default accessor assumes that the input data is an object with suitably-named attributes:

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

The target-accessor is invoked in the same manner as other value functions in D3. The function is passed two arguments, the current datum (d) and the current index (i). It is also possible to specify the target-accessor as a constant rather than a function.

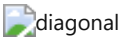
chord.**radius**
([*radius*])

If radius is specified, sets the radius-accessor to the specified function or constant. If radiusis not specified, returns the current radius-accessor. The default accessor assumes that the input source or target description is an object with

```
suitably-named attributes:
function radius(d) {
  return d.radius;
}
The radius-accessor is invoked in a similar manner as other value functions in D3. The
function is passed two arguments, the current source description (derived from the
current datum, d) and the current index (i). It is also possible to specify the
radius-accessor as a constant rather than a function.

<a name="chord_startAngle" href="SVG-Shapes#chord_startAngle">#</a>
chord.**startAngle**([*angle*])

If startAngle is specified, sets the startAngle-accessor to the specified function or
constant. IfstartAngle is not specified, returns the current startAngle-accessor.
Angles are specified inradians, even though SVG typically uses degrees. The default
accessor assumes that the input source or target description is an object with
suitably-named attributes:
function startAngle(d) {
  return d.startAngle;
}
The startAngle-accessor is invoked in a similar manner as other value functions in D3.
The function is passed two arguments, the current source or target description
(derived from the current datum, d) and the current index (i). It is also possible to
specify the startAngle-accessor as a constant rather than a function.

<a name="chord_endAngle" href="SVG-Shapes#chord_endAngle">#</a> chord.**endAngle**
([*angle*])

If endAngle is specified, sets the endAngle-accessor to the specified function or
constant. IfendAngle is not specified, returns the current endAngle-accessor. Angles
are specified inradians, even though SVG typically uses degrees. The default accessor
assumes that the input source or target description is an object with suitably-named
attributes:
function endAngle(d) {
  return d.endAngle;
}
The endAngle-accessor is invoked in a similar manner as other value functions in D3.
The function is passed two arguments, the current source or target description
(derived from the current datum, d) and the current index (i). It is also possible to
specify the endAngle-accessor as a constant rather than a function.
```

d3.svg.**diagonal**()

使用默认的配置构造一个对角线生成器，所有的默认配置请参考下面的 API 详细介绍；返回值可以用来当做函数使用来生成三次贝塞尔曲线数据，该曲线的若干条切线可以保证节点连接处会有平滑的介入效果；node-link diagram 效果图如下：


diagonal

虽然对角线生成器默认是笛卡尔（轴对齐）的定位方式，但依然可以自定义 projection 来生成径向或其他任意定位方式的路径数据。

diagonal(*datum*[, *index*])

根据指定的 *datum* 数据参数，生成 path 的路径数据字符串；一个可选的 *index* 参数，会传递给对角线生成器的函数。

diagonal.**source**([*source*])

如果指定了 *source* 参数，则设置起点存取器为 *source* 函数；如果未指定，则返回当前的起点存取器；该存取器函数的返回值会被直接传递给 projection 当做入参，存取器的目的是能够得到一个包含 *x*、*y* 坐标的对象，如 `{ x, y }`；*source* 也可以是常量；默认的起点存取器形如：

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

起点存取器的调用类似于 D3 中其他的大多数函数调用；函数的 *this* 对象指向了当前的 *DOM*，函数的两个入参：*d* 指向了当前 *DOM* 所绑定的数据、*i* 表示当前 *DOM* 所在的 *DOM* 树的索引。

diagonal.**target**([*target*])

如果指定了 *target* 参数，则设置末点存取器为 *target* 函数；如果未指定，则返回当前的末点存取器；该存取器函数的返回值会被直接传递给 projection 当做入参，存取器的目的是能够得到一个包含 *x*、*y* 坐标的对象，如 `{ x, y }`；*target* 也可以是常量；默认的末点存取器形如：

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

末点存取器的调用类似于 D3 中其他的大多数函数调用；函数的 *this* 对象指向了当前的 *DOM*，函数的两个入参：*d* 指向了当前 *DOM* 所绑定的数据、*i* 表示当前 *DOM* 所在的 *DOM* 树的索引。

diagonal.**projection**([*projection*])

如果指定了 *projection* 参数，则设置投影函数为 *projection*；如果未指定，则返回当前的投影函数；投影函数会转换形如 `{ x, y }` 的坐标系统为 `[x, y]`；默认的投影函数形如：

```
function projection(d) {
  return [d.x, d.y];
}
```

默认的投影函数兼容 D3 的大部分布局，包括：tree、partition、cluster；例如，一个生成径向对角线的方式如下：

```
function projection(d) {
  var r = d.y, a = (d.x - 90) / 180 * Math.PI;
  return [r * Math.cos(a), r * Math.sin(a)];
}
```

投影函数的调用类似于 D3 中其他的大多数函数调用；函数的两个入参：*d* 指向了当前 *DOM* 所绑定的数据（source 或者 target）、*i* 表示当前 *DOM* 所在的 *DOM* 树的索引。

d3.svg.diagonal.**radial**()

…

diagonal(*datum*[, *index*])

根据指定的 *datum* 数据参数，生成 path 的路径数据字符串；一个可选的 *index* 参数，会传递给对角线生成器的函数。

---

changelog

- SVG 元素部分
  - [大傻] 译于 2014-11-28 00:17:16
- Line 部分
  - [大傻] 译于 2014-11-29 04:34:09
- Symbol 部分
  - [大傻] 译于 2014-11-29 08:00
- Chord 部分
  - [大傻] 译于 2014-11-29 08:00
- Area部分
  - [Harry] 译于 2014-04-19
  - [大傻] 校于 2014-11-29 10:18:17
- Diagonal部分
  - [二傻] 译于 2014-04-18 17:14:18
  - [大傻] 校于 2014-11-29 11:06:40
  - [二傻] 校于 2016-07-28
- Arc部分
  - [二傻] 译于 2014-07-16 19:25
  - [大傻] 校于 2014-11-29 10:49:56