# Transitions

[Wiki](https://github.com/mbostock/d3/wiki) ▸ API Reference ▸ Core ▸ Transitions

## Transitions(过渡效果)

 [interpolate](https://github.com/mbostock/d3/wiki/Transitions#wiki-_interpolate) - 插值函数。输入参数在[0, 1]之间。

 [d3.interpolateNumber](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolateNumber) - 在两个数字间插值。

 [d3.interpolateRound](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolateRound) - 在两个数字间插值，返回值会四舍五入取整。

 [d3.interpolateString](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolateString) - 在两个字符串间插值。解析字符串中的数字，对应的数字会插值。

 [d3.interpolateRgb](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolateRgb) - 在两个RGB颜色间插值。

 [d3.interpolateHsl](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolateHsl) - 在两个HSL颜色间插值。

 [d3.interpolateLab](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolateLab) - 在两个L\*a\*b\*颜色间插值。

 [d3.interpolateHcl](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolateHcl) - 在两个HCL颜色间插值。

 [d3.interpolateArray](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolateArray) - 在两个数列间插值。d3.interpolateArray( [0, 1], [1, 10, 100] )(0.5); // returns [0.5, 5.5, 100]

 [d3.interpolateObject](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolateObject) - 在两个object间插值。d3.interpolateArray( {x: 0, y: 1}, {x: 1, y: 10, z: 100} )(0.5); // returns {x: 0.5, y: 5.5, z: 100}

 [d3.interpolateTransform](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolateTransform) - 在两个2D仿射变换间插值。

 [d3.interpolateZoom](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolateZoom) - 在两个点之间平滑地缩放平移。[示例](http://bl.ocks.org/mbostock/3828981)

 [d3.interpolators](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolators) - 添加一个自定义的插值函数.

## Interpolation

D3有很多内置interpolators来简化任意值的过渡;插值器是一个函数,用来映射参数值t的值域[0,1]中的一种颜色,号码或任意值。

#### [#](https://github.com/mbostock/d3/wiki/Transitions" \l "wiki-d3_interpolate) d3.interpolate(a, b)

返回两个值a和b之间的默认插值器。插值器的类型是基于最终值b的类型,使用以下算法:

1.如果b是一个颜色,interpolateRgb将被使用。

2.如果b是一个字符串,interpolateString将被使用。

3.如果b是一个数组,interpolateArray将被使用。

4.如果b是一个对象,而不是可强迫的一个数字,interpolateObject将被使用。

5.否则,interpolateNumber将被使用。

基于选择插值器,a将被强迫为一个适当的对应类型。颜色检查适用于两个实例， [d3.rgb](https://github.com/mbostock/d3/wiki/Colors#wiki-d3_rgb)和其他颜色空间以及/^(#|rgb\(|hsl\()/形式的颜色字符串或[CSS named colors](http://www.w3.org/TR/SVG/types.html#ColorKeywords)（CSS指定颜色）。

这个默认插值器的行为可能会通过推动定制interpolators到[d3.interpolators](https://github.com/mbostock/d3/wiki/Transitions" \l "wiki-d3_interpolators)数组从而扩展至支持其他的类型。

#### [#](https://github.com/mbostock/d3/wiki/Transitions#wiki-_interpolate) interpolate(t)

通常给定一个在区间[0,1]的参数t,返回相关的插入值。插值器通常与映射一个输入域(如定量维度)的尺度到一个输出范围(如一系列颜色或像素位置)的方式结合使用。

#### [#](https://github.com/mbostock/d3/wiki/Transitions" \l "wiki-d3_interpolateNumber) d3.interpolateNumber(*a*, *b*)

返回一个a和b两个数字之间的数字插值器。返回的插值器相当于:

function interpolate(t) {

return a \* (1 - t) + b \* t;

}

警告:当插值器用于生成一个字符串时(例如[attr](https://github.com/mbostock/d3/wiki/Transitions" \l "wiki-attr)) ，应该避免插入值或数字零（翻译不好，不理解：avoid interpolating to or from the number zero）。当使用字符串转化时，非常小的值可能转化为科学记数法格式从而产生一个临时的无效属性或样式属性值。例如,数字0.0000001便转换为字符串“1 e-7”。当插入不透明值时（翻译不好：when interpolating opacity values），这是特别显而易见的现象。为了避免科学记数法这种现象,以1e-6来开始或结束过渡,此值是最小的值不是用指数表示法方式来生硬的转化为字符串（翻译不好：To avoid scientific notation, start or end the transition at 1e-6, which is the smallest value that is not stringified in exponential notation）。

#### [#](https://github.com/mbostock/d3/wiki/Transitions" \l "wiki-d3_interpolateRound) d3.interpolateRound(*a*, *b*)

返回一个a和b两个数字之间的数字插值器;插值器类似于[interpolateNumber](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolateNumber),除了它不能将结果值设置为最近的整数。

#### [#](https://github.com/mbostock/d3/wiki/Transitions" \l "wiki-d3_interpolateString) d3.interpolateString(*a*, *b*)

返回一个a和b两个字符串之间的字符串插值器。字符串插值器寻找数字嵌入在a和b里,其中每个数字的形式如下:

/[-+]?(?:\d+\.?\d\*|\.?\d+)(?:[eE][-+]?\d+)?/g

对于嵌入到b的每个数字,插值器将尝试从a中找到一个相应的数字。如果找到相应的数,一个数字插值器便通过使用[interpolateNumber](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolateNumber)创建出来。字符串b的其余部分将作为一个模板:b字符串的静态部分对于插值器将保持值不变,插入的数字值将嵌入到模板。例如,如果a是“300 12 px sans-serif”,b是“500 36 px Comic-Sans”,两个嵌入的数字被发现。字符串剩下的静态部分是两个数字之间的空间(" "),和后缀(“px Comic-Sans”)。当传入值t = 0.5时， 插值器的结果是“400 24px Comic-Sans”。

#### [#](https://github.com/mbostock/d3/wiki/Transitions" \l "wiki-d3_interpolateRgb) d3.interpolateRgb(*a*, *b*)

返回一个a和b两种颜色值之间的RGB颜色空间插值器。颜色a和b不需要在RGB里,但他们将通过[d3.rgb](https://github.com/mbostock/d3/wiki/Colors#wiki-d3_rgb)转换为RGB值。红、绿、蓝通道是线性的来插入值，在某种程度上相当于[interpolateRound](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolateRound),即小数部分的通道值是不允许返回的。插值器的返回值是一个十六进制RGB字符串。

#### [#](https://github.com/mbostock/d3/wiki/Transitions" \l "wiki-d3_interpolateHsl) d3.interpolateHsl(*a*, *b*)

返回一个a和b两种颜色之间的[HSL](#_HSL)颜色空间插值器。颜色之间的a和b不需要在HSL范围中,但他们会通过[d3.hsl](https://github.com/mbostock/d3/wiki/Colors#wiki-d3_hsl)被转换成HSL值。色相、饱和度和明度是线性的来插入值，在某种程度上相当于[interpolateNumber](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolateNumber)。(在开始和结束色调之间的最短路径是常被习惯使用的)。插值器的返回值是一个十六进制RGB字符串。

#### [#](https://github.com/mbostock/d3/wiki/Transitions" \l "wiki-d3_interpolateLab) d3.interpolateLab(*a*, *b*)

返回一个a和b两种颜色之间的[Lab](#_LAB)颜色空间插值器。颜色a和b将会通过[d3.lab](https://github.com/mbostock/d3/wiki/Colors#wiki-d3_lab)（如果需要的话）被转换成Lab值。然后颜色通道线性的来插入值，在某种程度上相当于[interpolateNumber](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolateNumber)。插入器的返回值是一个十六进制RGB字符串。

#### [#](https://github.com/mbostock/d3/wiki/Transitions" \l "wiki-d3_interpolateHcl) d3.interpolateHcl(*a*, *b*)

返回一个a和b颜色之间的[HCL](#_HCL)颜色空间插值器。颜色a和b将会通过[d3.hcl](https://github.com/mbostock/d3/wiki/Colors#wiki-d3_hcl)（如果需要的话）被转换成HCL值。然后颜色通道线性的来插入值，在某种程度上相当于[interpolateNumber](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolateNumber)。(在开始和结束色调之间的最短路径是常被习惯使用的)。插值器的返回值是一个十六进制RGB字符串。

#### [#](https://github.com/mbostock/d3/wiki/Transitions" \l "wiki-d3_interpolateArray) d3.interpolateArray(*a*, *b*)

返回一个在两个数组a和b之间的数组插值器。一个拥有与b相同长度的数组模板将被创建。对于b中的每一个元素，如果在a中存在一个相应的元素,那么对于这两个元素就会有一个通用的插值器将通过[interpolate](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolate)被创建。如果没有这样的元素,那么来源于b的静态值将在这个模板中被使用。然后,对于给定的参数t,模板的嵌入的插值器将被进行求值。然后返回更新后的数组模板。例如,如果a数组为[0,1],b数组为[1,10,100] ,然后当参数t =0.5时，插值器的结果便是数组[0.5,5.5,100]。

注意:防御性复制（不理解）的模板数组是不会被创建的,返回数组的修改可能会不利地影响插值器的后续求值。不能复制,因为插值器应该保证快速,即它们是动画内部循环的一部分。

#### [#](https://github.com/mbostock/d3/wiki/Transitions" \l "wiki-d3_interpolateObject) d3.interpolateObject(*a*, *b*)

返回一个于a和b两个对象之间的插值器。一个拥有与b相同属性的对象模板将被创建。如果在a中存在一个对应的属性, 那么对于这两个元素就会有一个通用的插值器将通过[interpolate](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolate)被创建。如果没有这样的属性,来源于对象b的静态值将被使用在模板里。然后, 对于给定的参数t,模板的嵌入的插值器将被进行求值。然后返回更新后的数组模板。例如,如果a对象是{ x:0,y:1 }和b对象是{ x:1,y:10 z:100 },那么当参数t =0.5时插值器的结果便是对象{ x: 5,y:5.5,z:100 }。

对于数据空间差值，当插入数据值而不是属性值时，对象插值器尤其有用。例如,您可以插入一个对象，用来描述一个饼图里的弧，然后使用[d3.svg.arc](https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-arc)来计算这个新的SVG路径数据。

注意: 防御性复制（不理解）的模板对象是不会被创建的,返回对象的修改可能会不利地影响插值器的后续求值。不能复制,因为插值器应该保证快速,即它们是动画内部循环的一部分。

#### [#](https://github.com/mbostock/d3/wiki/Transitions" \l "wiki-d3_interpolateTransform) d3.interpolateTransform(*a*, *b*)

返回一个由两个a和b表示的二维仿射变换的插值器。每个变换分解成一个标准的转换,旋转,x倾斜度（x-skew）和比例；然后插入这些组件转换。这种行为是由CSS来指定标准: 详看[matrix decomposition for animation](http://www.w3.org/TR/css3-2d-transforms/" \l "matrix-decomposition)。

#### [#](https://github.com/mbostock/d3/wiki/Transitions" \l "wiki-d3_interpolateZoom) d3.interpolateZoom(*a*, *b*)

基于Jarke J. van Wijk and Wim A.A. Nuij共同开发的[“Smooth and efficient zooming and panning”](https://www.google.com/search?q=Smooth+and+efficient+zooming+and+panning)（平稳有效的缩放和移动），返回一个于两个二维平面的视图a和b之间的平稳插值器。每个视图的定义是由三个数字构成的数组:cx,cy和width。前两个坐标cx,cy代表视窗的中心;最后width（宽度）代表视窗的大小。返回的插入器还有一个持续时间属性，此属性用推荐的以毫秒为单位的过渡时间来输入值。这个时间是基于通过x，y空间曲线轨迹的路径长度来设置的。如果你想更慢或更快的转换,那么就通过一个任意的比例因子与此相乘(V（在哪里？何意？）在原始论文中有描述)。

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-interpolate) d3.geo.interpolate(*a*, *b*)

详看 [d3.geo.interpolate](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-interpolate).

#### [#](https://github.com/mbostock/d3/wiki/Transitions" \l "wiki-d3_interpolators) d3.interpolators

内置插值器工厂的数组，是由[d3.interpolate](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolate)使用。额外的插值器工厂可能被推放到这个数组的末尾端。如果它支持可以插入两个指定的输入值，那么每个工厂可能会返回一个插值器；否则,工厂将返回一个虚假的值并且其他的插值器将被尝试返回。（翻译：otherwise, the factory should return a falsey value and other interpolators will be tried）

例如,注册一个自定义插值器用来格式化美元和美分,你可能会写:

d3.interpolators.push(function(a, b) {

var re = /^\$([0-9,.]+)$/, ma, mb, f = d3.format(",.02f");

if ((ma = re.exec(a)) && (mb = re.exec(b))) {

a = parseFloat(ma[1]);

b = parseFloat(mb[1]) - a;

return function(t) {

return "$" + f(a + b \* t);

};

}

});

然后, d3.interpolate("$20", "$10")(1/3) ，即返回 $16.67；

d3.interpolate("$20", "$10")(1) ，即返回 $10.00；

d3.interpolate("$20", "$10")(0) ，即返回 $20.00。

## 补充知识：

#### HSL

HSL色彩模式是工业界的一种颜色标准，是通过对色调(H)、饱和度(S)、亮度(L)三个颜色通道的变化以及它们相互之间的叠加来得到各式各样的颜色的，HSL即是代表色调，饱和度，亮度三个通道的颜色，这个标准几乎包括了人类视力所能感知的所有颜色，是目前运用最广的颜色系统之一。

　 H: hue,色调，

　　S：saturation 饱和度

　　L lum 亮度

　　概述

　　HSL色彩模式使用HSL模型为图像中每一个像素的HSL分量分配一个0~255范围内的强度值。HSL图像只使用三种通道，就可以使它们按照不同的比例混合，在屏幕上重现16777216种颜色。

在 HSL 模式下，每种 HSL 成分都可使用从 0到 255的值。（其中L是从黑（0）到白（255）渐变）。

#### LAB

Lab色彩模型是由亮度（L）和有关色彩的a, b三个要素组成。L表示亮度（Luminosity），a表示从红色至绿色的范围，b表示从黄色至蓝色的范围。L的值域由0到100，L=50时，就相当 于50%的黑；a和b的值域都是由+127至-128，其中+127 a就是红色，渐渐过渡到-128 a的时候就变成绿色；同样原理，+127 b是黄色，-128 b是蓝色。所有的颜色就以这三个值交互变化所组成。例如，一块色彩的Lab值是L = 100，a = 30, b = 0, 这块色彩就是粉红色。

#### HCL

HCL（Hue, Chroma, Luminance）的颜色空间是由3部分组成:色调、浓度、亮度;因此,HCL空间的每一个颜色用3位数字对应于这些组件。举例来说,一个强大的黄色将被表示为90，100，100,和亮粉红色为0,65，100。

Harry

2014年3月30日 星期日