- 本文档是D3官方文档中文翻译，并保持与最新版同步。
- 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

过渡是一种特殊类型的选择器([[selection|选择器]])，这种操作符的应用随时间平滑而不是瞬间变换。你可以使用过渡[[transition()|选择器#transition]] 操作符从选择得到一个过渡。但是，过渡通常支持和选择器（例如attr和style）一样的操作符，不是所有的操作符都支持。例如，你必须在过渡开始前添加元素。移除remove操作符用来在过渡结束后方便地移除元素。

过渡可能有每个元素的延迟和持续时间，使用类似于其他操作符的数据计算功能。这可以很容易地给不同元素切换过渡，不论基于数据还是索引。例如，你可以给元素排序然后在变换中交错过渡得到更好的元素重排序效果。想了解更多关于此技术的内容请参考Heer和Robertson的Animated Transitions in Statistical Data Graphics。

D3有许多内置插值来简化任意值的过渡。例如，你可以从字体字符串"500 12px sans-serif"到"300 42px sans-serif"过渡，D3将发现嵌入在字符串内的数字自动地插入字体尺寸（size）和重量（weight）。您甚至可以插入任意的嵌套对象和数组或SVG路径数据。如果你找到内置插值的不足，D3允许使用attrTween和styleTween操作符自定义的内插器。D3的插值器提供[[比例尺|比例尺]]的基础并且可以用在过渡的外部。内插器是一个函数将定义域[0,1]中的而一个值t映射为一个颜色（color），数字或者任意的值。

更多关于过渡的内容可以参考Working with Transitions 教程。

对给定的元素在给定的时间只有一个过渡被激活。然而，在相同的元素可能安排多个过渡；只要是错开的时间，每一个过渡将按顺序运行。如果有新转变给定元素上运行，它隐含取消所有旧的转变，包括任何被安排但尚未运行。这允许一个新的过渡，例如那些将对一个新的用户事件响应的过渡，将会取代老的过渡，即使那些旧的过渡已经上演过了或者延迟上演了。多级过渡（过渡是在较早一个过渡的end事件中创建的过渡）被认为是和原始的过渡相同的"年龄"的；这是内部通过在单调增加的唯一ID跟踪的，其中ID是创建多级过渡时继承的。

开始过渡

d3.**transition**([*selection*], [*name*])

创建动画过渡。这等价于 `d3.select(document).transition()` 。这个方法很少使用，因为它通常更容易从现有选择导出的转变，而不是从现有的过渡导出的选择。当使用可选的选择调用，这个方法通常返回指定选择；例如它是一个空操作。然而，transition.each的上下文中，这个方法将为指定的选择（继承了延迟，持续和其他父过渡中的属性）创建一个新的过渡。这用来实现可以不论在选择器或者在过渡中都能被调用的可重用组件是非常有用的。在后者的情况下支持导出并行过渡。相关的一个实例可以参考D3的axis component。

transition.**delay**([*delay*])

指定过渡*延迟（delay）*以毫秒为单位。如果*延迟*是一个常量，则所有的元素将被赋予相同的延迟；如果*延迟*是一个函数，则这个函数将被每个选中的元素（按顺序）计算，将被传递当前的数据d和当前的索引i作为函数的变量，使用this上下文作为当前DOM元素。这个函数的返回值将被用来为每个元素的延迟设置值。默认的延迟是0。如果*[[延迟(duration)|过渡#duration]]*没有被指定，就返回过渡中第一个非空元素绑定的delay值。 设定延迟为索引i的倍数是一种方便方式错开元素过渡。例如，如果你使用duration 中固定的持续，并且当前选择有n个元素，你可以通过2 * *duration* 错开过渡：

```
.delay(function(d, i) { return i / n * duration; })
```

你也可以计算出延迟作为数据的一个功能，从而产生一个数据驱动的动画。

注意: 延迟总是相对一系列过渡中的*第一个*而言.

transition.**duration**([*duration*])

指定每个元素的持续时间（duration ），单位为毫秒。如果持续时间是常量的，那么所有的元素被给予相同的持续时间；否则，如果持续时间是函数，则该函数为每个选定的元件（按顺序）计算持续时间，被传递的当前数据d与当前索引i，并用this表示当前的DOM元素。该函数的返回值被用于设置每个元素的持续时间。默认持续时间为250毫秒。如果没有指定时间，则返回绑定过渡（transition）中的第一个非空元素的持续时间。

transition.**ease**([*value*[, *arguments*]])

指定过渡的缓动函数(easing function。如果*value*参数是一个函数，它被用于缓解当前参数定时值*t*，*t*通常在范围[0,1]内。 （在过渡结束时，*t*可以是稍大于1）否则，*value*参数被假定为一个字符串，这个参数被传递给[[d3.ease|Transitions#d3_ease]]方法来产生一个缓动函数。默认缓动函数是“cubic-in-out”。需要注意的是它不能用来定制每个元件或每个属性的缓动函数；但是，如果你使用的“线性（linear）”缓和函数，可以通过使用attrTween或styleTween插值器中内置的定制缓动功能。如果未指定缓动，则返回绑定到此过渡（transition）中的第一个非空元素的缓动函数。

过渡中的操作

内容

transition.**attr**(*name*, *value*)

通过指定的*name*和*value*过渡属性的值。过渡的初始值是当前属性值（一定要事先设定一个初始值，如果你不想坏的意外），结束值是指定的值。如果value 是常量的，那么所有的元素被给予相同的属性值； 否则，如果value 是函数，则该函数为每个选定的元件（按顺序）计算属性值，被传递的参数是当前数据d与当前索引i，用this表示当前的DOM元素。该函数的返回值被用于设置每个元素的属性。不支持null值因为插值器将变成未定义的（undefined）；如果你想在过渡结束后移除属性值可以使用remove函数。 内插器基于终值自动选择。如果终值是一个数字，开始值被强制转换为一个数字，并使用interpolateNumber内插器。如果结束值是一个字符串，执行一个检查，以查看该字符串是否代表一个颜色形如/^(#|rgb(|hsl()/或CSS命名颜色中的一个；如果是，则初始值被强制为RGB颜色，使用interpolateRgb内插器。否则，interpolateString内插器被使用，在字符串中插入数字。

transition.**attrTween**(*name*, *tween*)

根据指定的补间（tween）函数，通过指定的名称（name）过渡属性值。过渡的开始和结束值由补间函数决定。补间函数在每个元素的过渡开始的时候被调用，传入参数有当前数据d，当前索引i，this上下文表示当前DOM元素。补间函数的返回值必须是一个插值器：函数将定义域[0,1]映射为颜色、数字或者属性值。例如，attr操作符是建立在attrTween操作符之上的。补间函数被attr操作符使用取决于终值是一个函数还是常量。如果终值是一个函数则：

```
function tween(d, i, a) {
    return d3.interpolate(a, String(value.call(this, d, i)));
}
```

如果终值是一个常量则：

```
function tween(d, i, a) {
    return d3.interpolate(a, String(value));
}
```

这个attrTween操作符在你需要一个自定义的内插器的时候用到，例如：理解SVG路径数据语义。一种常见的技术是数据空间插值：其中interpolateObject用来插入两个数据值，这个插值的结果（也就是用一个shap）用来计算新属性值。使用attr操作符处理简单常见的情况，即内插器可以自动地从当前属性值转到期望的终值。

transition.**style**(*name*, *value*[, *priority*])

通过指定的名称（name）和值（value）过渡CSS样式属性值。可选参数优先级（priority）也可以指定为null或者字符串"important"（不带感叹号）。过渡的起始值是当前计算的样式属性值，终值是指定的value。如果value是一个常量那么所有的元素都被指定为相同的样式属性值；否则，如果value是一个函数那么函数将为每个选定的元素（按顺序）计算值，被传递的参数有当前数据d，当前索引i，this上下文代表当前DOM元素。函数的返回值之后被用来转换每个元素的样式属性值。null是不支持的，因为插值器需要是undefined；如果你想在转换之后移除样式属性，参考end事件。 内插器基于终值自动选择。如果终值是一个数字，开始值被强制转换为一个数字，并使用interpolateNumber内插器。如果结束值是一个字符串，执行一个检查，以查看该字符串是否代表一个颜色形如/^(#|rgb(|hsl()/或CSS命名颜色中的一个；如果是，则初始值被强制为RGB颜色，使用interpolateRgb内插器。否则，interpolateString内插器被使用，在字符串中插入数字。 需要注意的是所计算的初始值也可以和以前设定的值不同，尤其是当样式属性使用的缩写属性（如“font”样式，简写为“font-size”，“font-face”，等等）。此外，计算尺寸，如“font-size”和“line-height”是以像素为单位，所以你也应该适当地指定以像素为单位的终值。

transition.**styleTween**(*name*, *tween*[, *priority*])

根据指定的补间函数指定的名称过渡CSS样式属性的值。可选参数priority 可以被指定为null或者字符串“important”（没有感叹号）。过渡的起始和结束值由补间函数决定；补间函数在没给元素过渡开始的时候调用，被传递的参数是当前数据d，当前索引I，当前属性值a，this上下文代表当前DOM元素。补间函数的返回值必须是一个插值器：一个函数映射定义域[0,1]中的值t为一个颜色、数字或者属性值。 例如，style操作符是建立在styleTween操作符之上的。补间函数被style操作符使用取决于终值是一个函数还是常量。终值是函数时：

```
function tween(d, i, a) {
    return d3.interpolate(a, String(value.call(this, d, i)));
}
```

终值是常量时：

```
function tween(d, i, a) {
    return d3.interpolate(a, String(value));
}
```

styleTween操作符在需要定制插值器的使用使用到，例如：理解CSS3转换的语义。对简单常用的情况使用style操作符，即插值器可以从当前计算的样式属性自动驱动到期望的终值。

transition.**text**(*value*)

文本（text）操作符是基于textContent 属性：设置文本内容将取代任何现有的子元素。在转换开始时设置所有选定元素的文本内容为指定值。如果值是一个常数，那么所有的元素被给予相同的文本内容；否则，如果值是一个函数，则该函数被每个选定的元件（按顺序）计算，被传递当前数据d与当前索引i，this上下文作为当前的DOM元素。该函数的返回值被用于设置每个元素的文本内容。null值会清除内容。

transition.**tween**(*name*, *factory*)

注册一个自定义补间指定的名称（name）。当在过渡开始时，指定的工厂函数将被过渡中每个选定的元素调用，传递该元素的数据（d）和索引（i）作为参数，元素作为上下文（this）。工厂应反回补间函数，在整个过渡过程中被调用。补间函数之后被反复调用，传递当前的归一化时间t在[0，1]范围内。如果工厂返回null，则补间不会在选定元素上运行。 补间函数是用来内部实现attr和style补间，并可以用来对其它文档内容进行内插。例如，内插从0到100的文本内容：

```
selection.transition().tween("text", function() {
    var i = d3.interpolateRound(0, 100);
```

```
    return function(t) {
        this.textContent = i(t);
    };
});
```

补间常使用闭包捕捉过渡开始时创建的状态。上例中，i在过渡开始的时候初始化，随后用在过渡过程中（虽然注意到，在上面的例子中，转变的开始值是硬编码为0，而过渡更常用的初始值是基于DOM中的当前状态）。 # transition.**remove**()

在过渡结束时删除选定的元素。如果在任何选定的元素有新的过渡计划，这些元素不会被删除；然而，“end”事件仍会被调度。

子过渡

过渡可以从现有的过渡派生，类似子选择的方式。 子过渡继承缓动，持续和延迟父过渡。

transition.**select**(*selector*)

当前过渡的每个元素中，选择指定的选择字符串匹配的第一个后代元素。如果当前元素没有元素匹配指定的选择器字符串，那么返回的选择中当前索引的元素将是null。操作符（除数据）自动的跳过空值，从而保持现有选择的索引。如果当前元素具有相关联的数据，该数据是由返回的部分选定继承，并自动绑定到新选定的元素。如果多个元素匹配选择器，只在文件遍历顺序的第一个匹配的元素将被选中。 这个方法大约相当于： 其中，selection 是当前过渡的隐含选择器。另外，返回的新过渡从当前的过渡继承缓动，持续和延迟。

transition.**selectAll**(*selector*)

对当前过渡的每个元素，选择匹配指定选择字符串后代元素。返回的选择按当前选择的祖先节点进行分组。如果当前元素没有元素匹配指定选择器，返回的选择中当前索引的组将是空的。部分选取不从当前选择继承数据。然而，如果数据之前绑定到选定的元素上，那个数据可用于操作符。 这个方法大约相当于：

```
selection.selectAll(selector).transition()
```

其中，selection 是当前过渡的隐含选择器。另外，返回的新过渡从当前的过渡继承缓动，持续和延迟。当前过渡中每个子元素的持续和延迟都是继承自父元素。

transition.**filter**(*selector*)

过滤过渡，返回一个新的过渡只包含其指定的选择是正确的元素。选择器可以被指定为一个函数或选择字符串，如“foo”。和其他操作符一样，该函数被传递当前数据d和当前索引i，this上下文作为当前的DOM元素。类似内置的数组过滤函数，返回选择不保留原选择的索引；返回移除元素的副本。如果您想保留索引，使用select代替。例如，选择所有其他元素：

```
var odds = transition.select(function(d, i) { return i & 1 ? this : null; });
```

等价于使用过滤函数：

```
var odds = transition.filter(function(d, i) { return i & 1; });
```

或者一个过滤选择器：

```
var odds = transition.filter(":nth-child(odd)");
```

这样你就可以使用select和过滤器在元素子集上应用补间了。

transition.**transition**()

在同样的选中元素上创建一个新的选择，在这个过渡结束后启动。新的过渡继承当前过渡的持续和缓动。这可以用来定义链式过渡，而无需监听“end”事件。

// **revision needed 需要重新翻译**

```
Creates a new transition on the same selected elements that starts when this
transition ends. The new transition inherits a delay equal to this transition's delay
+ duration to control this timing. The new transition also inherits this transition’s
name, duration, and easing. This can be used to define [chained transitions]
(http://bl.ocks.org/mbostock/4341417) without needing to listen for "end" events.

As described above, the delay state of chained transitions is used as a scaffold for
the chain.  This means that the delay method will have unexpected behaviour in this
context, however, if you want to add a delay between chained transitions, an easy way
to do this is to have a no-op transition in between:


selection.transition() // this transition runs from t=1s to t=3s
    .delay(1000)
    .duration(2000)
  .transition() // then a delay from t=3s to t=4s
    .duration(1000)
  .transition() // then lastly another transition from t=4s to t=5s


Though, note that the last transition in this chain inherits the 1s duration from the
interstitial pause, and not the 2s duration from the first transition.
```

控制

transition.**each**([*type*,]*listener*)

如果指定了*类型（type）*，就为过渡事件增加一个侦听器(listener)，同时支持“start”, “end”和"interrupt"件。侦听器会被过渡中每个单独的元素调用。

*start*事件在过渡的第一个异步回调（tick）中被调用，于任何补间(tween)被调用之前。对于0延迟过渡，通常在过渡之后的17ms被调用。状态函数对触发每个元素的瞬间变化是很有用的，例如改变不能被打断的属性。

*end*事件在过渡的持续和延迟结束之后最后一个异步回调（tick）中被调用，在所有的补间都使用t=1调用之后。注意，如果对于一个给定的元素过渡被之后调度的过渡取代了， end事件就不会分配到这个元素上；中断过渡不会触发结束事件。例如，[[transition.remove|#remove]]调度的每个元素在过渡结束时被除去，但如果过渡被中断了，元素将不被删除。end事件可以被用作替代[[transition.transition|#transition]],通过选择当前元素this，并导出新过渡来创建过渡链(chained transitions)。

*interrupt*事件会在同一元素上的同一种过渡打算正在运行的过渡时触发。 **translation needed: 需要翻译:**

```
The interrupt event is dispatched on the interrupted transition immediately prior to
the start event on the interrupting transition. Note that if a transition is cancelled
before it starts (such as when a later-scheduled transition starts before a delayed
transition), no interrupt event is dispatched.
```

如果没有指定type，行为类似于[[selection.each|Selections#each]]：直接为当前过渡的每个元素调用指定的函数，通过在当前的数据d和索引i，与当前的DOM元素的this上下文。 transition.each将从父过渡继承过渡参数，包括id，延迟和缓动。因此，transition.each内创建过渡不会打断父过渡，类似subtransitions。

transition.each方法可用于链接过渡并在一组过渡上分享定时（timing）。例如：

```
d3.transition()
    .duration(750)
    .ease("linear")
    .each(function() {
        d3.selectAll(".foo").transition()
            .style("opacity", 0)
            .remove();
    })
    .transition()
    .each(function() {
        d3.selectAll(".bar").transition()
            .style("opacity", 0)
            .remove();
    });
```

通过使用transition.each中的 `d3.select(this)` ，你甚至可以继承一组选定元素的交错延迟。这个技术在数轴组件中用来支持自动过渡。此方法被用于[[SVG 轴|SVG-轴]]来支持自动过渡(automatic transitions).

transition.**call**(*function*[, *arguments…*])

调用指定的*函数（function）*一次，通过一些可选的*参数（arguments）*传递当前的过渡。call操作符总是返回当前的过渡，与指定函数的返回值无关。call操作符和手动执行一个函数是一样的。但是，它更容易使用方法链。例如，假设我们要在许多不同的地方以同样的方式设置一些属性。我们编写代码，把它封装为一个可重复使用的函数：

```
function foo(transition) {
    transition
        .attr("name1", "value1")
        .attr("name2", "value2");
}
```

现在我们可以这样使用 `foo()` ：

```
foo(d3.selectAll("div").transition());
```

或者, 等价地：

```
d3.selectAll("div").transition().call(foo);
```

在很多情况下，是可以在过渡和选择上调用同一个函数foo的，因为在过渡和选择上是一样的方法。被调函数的 `this` 就是当前过渡。第一个用法略显多余，这我们可能在未来解决。

transition.**empty**()

返回true如果当前过渡是空的；过渡是空的，是指它不包含任何非null元素。

transition.**node**()

返回在当前过渡的第一个非空元素。如果转换为空，则返回null。

transition.**size**()

返回在当前的过渡元素的总数。

Easing

d3.**ease**(*type*[, *arguments*…])

返回一个指定类型（type），带有任何可选参数（arguments）的内置缓动函数。一个缓动函数将当前参数化的时间值t从定义域[0,1]映射到一个相似返回的其他值；这通常用来设置过渡的缓动。D3支持以下的缓动类型：

- linear -标识函数, t.
- poly(k) – t 的 k次方 (例如3).
- quad – 等价于poly(2).
- cubic - 等价于poly(3).
- sin – 使用三角函数 sin.
- exp – 2的t次方
- circle - 四分之一圈
- elastic(a, p) - 模拟一个橡皮筋；略有延长可能会超出0,1。
- back(s) - 模拟备份到一个停车位。
- bounce - 模拟一个有弹性碰撞。 这些内置的类型可以采用各种方式进行扩展：
- in - 标识的功能。
- out -逆转缓动的方向为[1,0]。
- in-out -从[0,.5]和[.5,1]复制和镜像缓动函数。
- out-in -从[1,.5]和[.5,0]复制和镜像缓动函数。 默认的缓动函数是"cubic-in-out" 它提供了适合慢入慢出动画。

ease(*t*)

给定的参数时间t，通常在范围[0,1]内，返回的缓动的时间。返回的值通常是在范围[0,1]为好，但对于某些缓动的函数也可以超出这个范围，比如"弹性（elastic）"就可稍微延伸。

Timers

D3在内部维护一个高效的定时器队列，使成千上万的定时器可以用最小的开销并发地处理；此外，该定时器队列保证动画的一致的定时时同时或分阶段转换被调度。如果您的浏览器支持它，定时器队列将使用requestAnimationFrame流体高效的动画。定时器队列也是聪明的使用setTimeout的时候有一个调度事件之前，出现长时间的延迟。

d3.**timer**(*function*[, *delay*[, *time*]])

启动一个自定义动画计时器，重复地调用指定的函数（function ），直到它返回true。计时器启动后没有办法把它取消，所以一定要确保完成时，你的计时器函数返回true！ 当给定函数将在一段延迟之后被调用时，一个以毫秒为单位的可选数字delay可能被指定。延时是相对于指定时间从UNIX纪元以毫秒为单位;如果没有指定时间，则默认为Date.now。 您可以使用延迟（delay ）和时间（time ）以指定function应该开始被调用的相对和绝对时刻。例如，一个日历通知可能被编码为： d3.timer(notify, -4 * 1000 * 60 * 60, +new Date(2012, 09, 29)); // four hours before midnight October 29 (months are zero-based)

d3.timer.**flush**()

立即执行（调用一次）任何活动的计时器。通常，在瞬时延迟（<10毫秒）之后执行零延迟过渡。如果浏览器两次呈现页面，这可能会导致一个简短的闪烁。一旦在第一事件循环的结束，然后再次紧接在第一定时器的回调。通过在第一事件循环结束时刷新定时器队列，可以立即执行任何零延迟过渡，避免了闪烁。

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，用来将值域[0,1]中参数值t映射为一种颜色，数字或任意值。

d3.**interpolate**(*a*, *b*)

返回一个介于a和b之间的默认插值器。插值器的类型是基于后面一个值b的类型,使用以下算法: 1.如果b是颜色（color）类型，则返回interpolateRgb插值器。 2.如果b是字符串（string）类型，则返回interpolateString插值器。 3.如果b是数组（array）类型，则返回interpolateArray插值器。 4.如果b是对象（object）类型，且不能强制转换为数字类型, 则返回interpolateObject插值器。 5.否则，返回interpolateNumber插值器。 基于选定的插值器,a将被强制转换为一个适当的对应类型。颜色检查适用于 d3.rgb和其他颜色空间以及/^(#|rgb(|hsl()/形式的颜色字符串或CSS指定的颜色。 这个默认插值器的行为可以扩展至支持其他的类型。只要添加用户自定义插值器到d3.interpolators数组中即可。

interpolate(*t*)

对在区间[0,1]中一个给定的参数t，返回相关的插入值。插值器通常用结合比例尺一起使用，映射一个输入域(如定量维度)到一个输出范围(如一系列颜色或像素位置)。

d3.**interpolateNumber**(*a*, *b*)

返回一个a，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;
}
```

注意:当插值器用于生成一个字符串时(例如attr)，应该避免插入0或从0返回插值器。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暂时无效的属性或样式属性值。例如,数字0.0000001便转换为字符串"1 e-7"。 当插入不透明度值时，这一点尤其明显。为了避免科学记数法导致过渡的开始和结束值都是1e-6这种现象，此值是不用指数表示法转化为字符串的最小的。

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,除了它会将结果值四舍五入为最近的整数。

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,便通过使用interpolateNumber创建一个数字插值器。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-seri"，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"。

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转换为RGB值。红、绿、蓝通道是线性地插入值，在某种程度上相当于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。

d3.**interpolateHsl**(*a*, *b*)

返回一个a和b两种颜色之间的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。(使用开始和结束色调之间的最短路径)。插值器的返回值是一个十六进制RGB字符串。

d3.**interpolateLab**(*a*, *b*)

返回一个a和b两种颜色之间的Lab颜色空间插值器。颜色a和b将会通过d3.lab（如果需要的话）被转换成Lab值。然后颜色通道是线性地插入值，在某种程度上相当于interpolateNumber。插入器的返回值是一个十六进制RGB字符串。

d3.**interpolateHcl**(*a*, *b*)

返回一个a和b颜色之间的HCL颜色空间插值器。颜色a和b将会通过d3.hcl（如果需要的话）被转换成HCL值。然后颜色通道线性地插入值，在某种程度上相当于interpolateNumber。(使用开始和结束色调之间的最短路径)。插值器的返回值是一个十六进制RGB字符串。 # 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:创建一个模板数组的非保护性拷贝；返回数组的修改将对内插器的随后的评价产生不利影响。不复制是为了保证插值器的快速，因为它们是动画内部循环的一部分。

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。对于b中的每个属性，如果在a中存在一个对应的属性，那么通过interpolate创建这两个元素通用的插值器。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来计算这个新的SVG路径数据。 注意:创建一个模板数组的非保护性拷贝；返回数组的修改将对内插器的随后的评价产生不利影响。不复制是为了保证插值器的快速，因为它们是动画内部循环的一部分。

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偏移和比例；然后插入这些转换分量。这种行为是由CSS规范的：详看matrix decomposition for animation。

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”（平滑有效的缩放和移动），返回一个于两个二维平面视图a和b之间的平滑插值器。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（duration ）属性，此属性推荐用以毫秒为单位的过渡时间来编码。这个持续时间是基于x，y空间弧形轨迹路径长度的。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*在原始论文中有描述)。

d3.geo.**interpolate**(*a*, *b*)

详见d3.geo.interpolate。

d3.**interpolators**

内置插值器工厂的数组，是d3.interpolate所使用的。额外的插值器工厂可能被添加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，工厂将返回一个假值并尝试返回其他的插

值器。 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)` ，返回 $16.67； `d3.interpolate("$20", "$10")(1)` ，返回 $10.00； `d3.interpolate("$20", "$10")(0)` ，返回 $20.00。

本文参与	人员	组织	时间
翻译1-16页	大傻	VisualCrew小组	2014-11-15 21:36:38
翻译17-26页	Harry	VisualCrew小组	2014-03-30
校对	大傻	VisualCrew小组	20141115
排版/校对	liang42hao	VisualCrew小组	2015-11-08T19:42:48Z