

Raphaël 中文参考——by txp

1. 动画

1.1 Animation.delay(delay) ➡

创建现有的动画对象的副本，并指定延迟时间。

参数		
delay	数字	动画开始和实际动画之间的毫秒数
返回：	对象	新的动画对象

```
var anim = Raphael.animation({cx: 10, cy: 20}, 2e3);  
  
circle1.animate(anim); // run the given animation immediately  
  
circle2.animate(anim.delay(500)); // run the given animation after 500 ms
```

1.2 Animation.repeat(repeat) ➡

创建现有动画对象的副本，并指定重复频率。

参数		
repeat	数字	动画迭代次数。0 为无限
返回：	对象	新的动画对象

2. 元素

2.1 Element.animate(...) ➡

为指定元素创建动画，并播放。

参数		
params	对象	元素的最终属性，见 Element.attr
ms	数字	动画持续时间（毫秒）
easing	字符串	曲线类型。接受 Raphael.easing_formulas 其中之一或 CSS 格式：cubic-bezier(XX, XX, XX, XX)
callback	函数	回调函数。动画结束时将被调用。
或		
animation	对象	动画对象，见 Raphael.animation
返回：	对象	原始元素

2.2 Element.animateWith(...) ➡

作用与 [Element.animate](#) 类似，但保证动画与另一元素的同步播放。

参数		
el	对象	要同步的元素
anim	对象	要同步的动画
params	对象	元素的最终属性，见 Element.attr

ms	数字	动画持续时间（毫秒）
easing	字符串	曲线类型。接受 Raphael.easing_formulas 或 CSS 格式： cubic-bezier(XX, XX, XX, XX)
callback	函数	回调函数。动画结束时将被调用。
或		
element	对象	要同步的元素
anim	对象	要同步的动画
animation	对象	动画对象，见 Raphael.animation

返回：	对象	原始元素
-----	----	------

2.3 Element.attr(…) ➞

设置元素的属性。

参数		
attrName	字符串	属性名称
value	字符串	值
或		
params	对象	名称/值对
或		
attrName	字符串	属性的名称
或		

attrNames	数组	在这种情况下，方法返回指定属性名的当前值组成的数组
-----------	----	---------------------------

返回： 对象 Element ，如果传入 attrsName 和 value 或者 params。

返回： ...属性的值，如果只有 attrsName 被传入。

返回： 数组属性值的数组，如果 attrNames 被传入。

返回： 对象属性对象，如果没有传入任何参数。

可能的参数

请参阅解释这些参数的 [SVG 规范](#) 。

- arrow-end **字符串** 路径的末尾显示箭头。字符串格式是 <type>[-<width>[-<length>]]。可能的类型 :classic、block、open、oval、diamond、none , 宽 : wide、narrow、midium , 长 : long、short、midium。
- clip-rect **字符串** 剪贴矩形。逗号或空格分隔的值 : x , y , 宽度和高度
- cursor **字符串** 光标的 CSS 类型
- cx **数字** 圆或椭圆的圆心的 x 轴坐标
- cy **数字** 圆或椭圆的圆心的 y 轴坐标
- fill **字符串** 填充。颜色、渐变或图像
- fill-opacity **数字** 填充不透明度
- font **字符串** 文本特性
- font-family **字符串** 字体
- font-size **数字** 字体大小 (像素)
- font-weight **字符串** 字体粗细

height 数字 高度

href 字符串 URL。指定时，元素表现为超链接

opacity 数字 透明度

path 字符串 SVG 的路径字符串格式

r 数字 圆、椭圆或圆角矩形的半径

rx 数字 椭圆的横向半径

ry 数字 椭圆的垂直半径

src 字符串 图像的 URL，只适用于 [Element.image](#) 元素

stroke 字符串 笔触颜色

stroke-dasharray 字符串 ["", "-", ".", "-.", "-..", ".", "- ", "--",
"-.", "--.", "--."]

stroke-linecap 字符串 ["butt", "square", "round"]

stroke-linejoin 字符串 ["bevel", "round", "miter"]

stroke-miterlimit 数字

stroke-opacity 数字

stroke-width 数字 笔触宽度（像素，默认为 1）

target 字符串 与 href 一起使用

text 字符串 文本元素的内容。使用\n 换行

text-anchor 字符串 ["start", "middle", "end"], 默认为 "middle"

title 字符串 工具提示内容

transform 字符串 见 [Element.transform](#)

width 数字

x 数字

y 数字

线性渐变

线性渐变的格式：“<angle>-<colour>[-<colour>[:<offset>]]*-<colour>”，例如：

“90-#fff-#000” 90°线性渐变从白色到黑色；“0-#fff-#f00:20-#000” 0°从白色通过红色（20%位置）渐变为黑色。

径向渐变：“r[(<fx>, <fy>)]<colour>[-<colour>[:<offset>]]*-<colour>”，例如：“r#fff-#000”从白色到黑色；“r(0.25, 0.75)#fff-#000”从白色渐变到黑色，焦点在 0.25，0.75。焦点坐标的外围是 0 .. 1。径向渐变只适用于圆形和椭圆形。

路径字符串

请参考 [SVG 文件路径字符串](#)。Raphael 完全支持它。

颜色解析

颜色的名称（“red”，“green”，“cornflowerblue”等）

#...——缩写 HTML 颜色：（“#000”，“#fc0”等）

#.....——HTML 颜色：（“#000000”，“#bd2300”）

rgb(..., ..., ...)——红色、绿色和蓝色通道值：（“rgb(200, 100, 0)”）

rgb(...%, ...%, ...%)——和上面一样，但是用百分比：（“rgb(100%, 175%, 0%)”）

rgba(..., ..., ..., ...)——红色、绿色、蓝色通道值：（“rgba(200, 100, 0, .5)”）

rgba(•••%, •••%, •••%, •••%)——和上面一样，但是用百分比：("rgba(100%, 175%, 0%, 50%)")

hsb(•••, •••, •••)——色相、饱和度和亮度值：("hsb(0.5, 0.25, 1)")

hsb(•••%, •••%, •••%) ——和上面一样，但是用百分比

hsba(•••, •••, •••, •••)——和上面一样，但是有透明度

hsl(•••, •••, •••)——基本和 HSB 相同，见 [Wikipedia page](#)

hsl(•••%, •••%, •••%)——和上面一样，但是用百分比

hsla(•••, •••, •••, •••)——和上面一样，但是有透明度

对于 HSB 及 HSL，你可以指定色相度数：“hsl(240deg, 1, .5)” ，或者 “hsl(240°, 1, .5)”

2.4 Element.click(handler) ➡

为元素添加 click 事件的处理程序。

参数		
handler	函数	事件处理函数
返回： 对象元素		

2.5 Element.clone() ➡

返回：	对象	给定的元素的一个克隆元素
-----	----	--------------

2.6 Element.data(key, [value]) ➡

添加或检索与给定的键关联的值。见 [Element.removeData](#)

参数

key

字符串

数据的键

value

任何

数据的值

返回： 对象 [元素](#)

或者，如果该值未指定：

返回： 任何值

用法

```
for (var i = 0, i < 5, i++) {  
  
    paper.circle(10 + 15 * i, 10, 10)  
  
        .attr({fill: "#000"})  
  
        .data("i", i)  
  
        .click(function () {  
  
            alert(this.data("i"));  
  
        });  
  
}
```

2.7 Element.dblclick(handler) ➞

为元素添加 dblclick 事件的处理程序。

参数

handler

函数

事件处理函数

返回：**对象**元素

2.8 Element.drag(onmove, onstart, onend, [mcontext], [scontext], [econtext]) ➞

为元素添加 drag 事件的处理程序。

参数

onmove

函数

移动处理函数

onstart

函数

拖拽开始的处理函数

onend

函数

拖拽结束处理函数

mcontext

对象

移动处理函数环境（上下文）

scontext

对象

拖拽开始处理函数环境

econtext

对象

拖拽结束处理函数环境

下面额外的拖拽事件会被触发：开始时 `drag.start.<id>`；结束时 `drag.end.<id>`；每次移动时 `drag.move.<id>`。当元素被拖入了另一个元素 `drag.over.<id>` 将被触发。

拖拽开始事件和拖拽开始处理函数将在指定的环境内调用 ,或在包含以下元素的环境内调用：

`x` 数字 鼠标 x 位置
`y` 数字 鼠标 y 位置
`event` 对象 DOM 事件对象

移动事件和移动处理函数将在指定的环境内调用 ,或在包含以下元素的环境内调用：

`dx` 数字 相对起始点的 x 位移
`dy` 数字 相对起始点的 y 位移
`x` 数字 鼠标 x 位置
`y` 数字 鼠标 y 位置
`event` 对象 DOM 事件对象

拖拽结束事件和拖拽结束处理函数将在指定的环境内调用 ,或在包含以下元素的环境内调用：

`event` 对象 DOM 事件对象

返回：`元素`

2.9 Element.getBBox(isWithoutTransform) ➡

返回指定元素的边界框

参数

isWithoutTransform

布尔

如果你想得到变换之前的边界，使用 *true*。默认值为 *false*。

返回： 对象 边界框对象：

```
{
```

x: 数字 左上角 x

y: 数字 左上角 y

x2: 数字 右下角 x

y2: 数字 右下角 y

width: 数字 宽

height: 数字 高

```
}
```

2.10 Element.getPointAtLength(length) ➡

返回在指定路径上指定长度的坐标点。只适用于“路径”类型的元素。

参数

length

数字

返回：对象点：

{

x:数字 x 坐标

y:数字 y 坐标

alpha:数字 导数（切线）的角度

}

2.11 Element.getSubpath(from, to) ➞

返回在指定路径上指定长度定子路径。只适用于“路径”类型的元素。

参数

from

数字

片段开始位置

to

数字

片段结束的位置

返回：字符串段的路径字符串

2.12 Element.getTotalLength() ➞

返回路径的长度，以像素为单位。只适用于“路径”类型的元素。

返回：数字长度。

2.13 Element.glow([glow])

为指定元素添加类似光晕的效果，返回创建的光晕元素组。见 [Paper.set](#)。

注：光晕不与元素关联。如果你改变元素的属性，它并不会自我调整。

参数

glow

对象

参数对象（所有属性可选）：

{

width 数字 光晕大小，默认是 10

fill 布尔 是否填充，默认是 false

opacity 数字 透明度，默认是 0.5

offsetx 数字 横向偏移，默认为 0

offsety 数字 垂直偏移，默认为 0

color 字符串 发光颜色，默认为 black

}

返回：对象 表示光晕的元素组，见 [Paper.set](#)

2.14 Element.hide()

隐藏指定元素。见 [Element.show](#)。

返回： 对象 [元素](#)

2.15 `Element.hover(f_in, f_out, [icontext], [ocontext])` ➞

为元素增加 hover 事件的处理程序。

参数

`f_in`

函数

悬停进入处理句柄

`f_out`

函数

悬停离开处理句柄

`icontext`

对象

悬停进入处理句柄环境

`ocontext`

对象

悬停离开处理句柄环境

返回： 对象 [元素](#)

2.16 `Element.id` ➞

数字

元素的唯一的 ID。当你要监听元素的事件时尤其有用。因为所有的事件被下面的格式触发：`<module>.<action>.<id>`。对 [Paper.getById](#) 方法也有用。

2.17 Element.insertAfter() ➞

在指定对象后插入当前对象。

返回： 对象元素

2.18 Element.insertBefore() ➞

在指定对象前插入当前对象。

返回： 对象元素

2.19 Element.isPointInside(x, y) ➞

确定指定点在当前元素的形状内

参数

x

数字

点的 x 坐标

y

数字

点的 y 坐标

返回： 布尔 *true* 如果点在形状内

2.20 Element.matrix ➞

对象

[矩阵](#)对象，它代表元素的变换

2.21 Element.mousedown(handler) ➞

为元素添加 mousedown 事件的处理程序。

参数

handler

[函数](#)

事件处理函数

返回：[对象](#) [元素](#)

2.22 Element.mousemove(handler) ➞

为元素添加 mousemove 事件的处理程序。

参数

handler

[函数](#)

事件处理函数

返回：[对象](#) [元素](#)

2.23 Element.mouseout(handler) ➞

为元素添加 mouseout 的事件的处理程序。

参数

handler

函数

事件处理函数

返回：对象元素

2.24 Element.mouseover(handler) ➞

为元素添加 mouseover 事件的处理程序。

参数

handler

函数

事件处理函数

返回：对象元素

2.25 Element.mouseup(handler) ➞

为元素添加 mouseup 事件的处理程序。

参数

handler

函数

事件处理函数

返回：对象元素

2.26 Element.next ➞

对象

层次结构中的下一个元素的引用。

2.27 Element.node ➞

对象

DOM 对象的引用。

用法

```
// draw a circle at coordinate 10,10 with radius of 10var c = paper.circle(10, 10, 10);

c.node.onclick = function () {

    c.attr("fill", "red");

};
```

2.28 Element.onDragOver(f) ➞

为元素分配 `drag.over.<id>` 事件处理程序的快捷方式，其中 `id` 是元素的 `id` (见 [Element.id](#))。

参数

f

函数

事件的处理程序，第一个参数是你拖拽的元素

2.29 Element.paper ➞

对象

绘制对象的“画布”的内部引用。主要用于插件和元素扩展。

用法

```
Raphael.el.cross = function () {  
  
    this.attr({fill: "red"});  
  
    this.paper.path("M10,10L50,50M50,10L10,50")  
  
        .attr({stroke: "red"});  
  
}
```

2.30 Element.pause([anim]) ➞

暂停播放动画，稍后可以恢复播放。

参数

anim

对象

动画对象

返回：对象 原始元素

2.31 Element.prev ➞

对象

层次结构中前一个元素的引用。

2.32 Element.raphael ➞

对象

[Raphael](#) 对象的内部引用。即使它目前不可用。

用法

```
Raphael.el.red = function () {  
  
    var hsb = this.paper.raphael.rgb2hsb(this.attr("fill"));  
  
    hsb.h = 1;  
  
    this.attr({fill: this.paper.raphael.hsb2rgb(hsb).hex});  
  
}
```

2.33 Element.remove() ➞

从画布上移除元素。

2.34 Element.removeData([key]) ➞

删除元素与指定键相关联的值。如果不提供关键，则删除元素中的所有数据。

参数

key

字符串

键

返回：对象 [元素](#)

2.35 Element.resume([anim]) ➞

恢复动画，如果它被 [Element.pause](#) 方法暂停。

参数

anim

对象

动画对象

返回：对象原始元素

2.36 Element.rotate(deg, [cx], [cy]) ➞

建议不要使用！使用 [Element.transform](#) 代替。围绕指定的点旋转指定角度。

参数

deg

数字

角度

cx

数字

旋转中心的 x 坐标

cy

数字

旋转中心的 y 坐标

如果 cx 和 cy 没有指定，形状的中心将作为旋转中心。

返回：对象元素

2.37 Element.scale(sx, sy, [cx], [cy]) ➞

建议不要使用。使用 [Element.transform](#) 代替。为元素添加缩放。

参数

sx

数字

水平缩放值

sy

数字

垂直缩放值

cx

数字

缩放中心的 x 坐标

cy

数字

缩放中心的 y 坐标

如果 cx 和 cy 没有指定，则使用形状的中心。

返回：对象元素

2.38 Element.setTime(anim, value) ⇨

以毫秒为单位给元素设置动画的状态。类似 [Element.status](#) 方法。

参数

anim

对象

动画对象

value

数字

动画持续的毫秒数

返回：对象如果 value 被指定则返回原始元素

请注意，在动画过程中会触发下列事件：

在每个动画帧 `anim.frame.<id>`，在开始时 `anim.start.<id>` 结束时

`anim.finish.<id>`。

2.39 Element.show() ⇨

使元素可见。见 [Element.hide](#) 。

返回： 对象 [元素](#)

2.40 Element.status([anim], [value]) ➞

获取或设置元素的动画状态。

参数

anim
对象
动画对象
值
数字
0 - 1。如果指定，方法就像一个 setter，并设置指定动画的状态为指定值。这将导致动画跳转到指定位置。

返回： 数字 状态

或

返回： 数组 状态，如果 *anim* 未指定。对象数组：

```
. {  
  .  
  . anim:对象 动画对象  
  .  
  . status:数字 状态  
  .  
}
```

或

返回： 对象 如果 *value* 被指定则返回原始元素

2.41 Element.stop([anim]) ➞

停止播放动画。

参数

anim

对象

动画对象

返回： 对象 原始元素

2.42 Element.toBack() ➞

将元素向下移动。

返回： 对象 元素

2.43 Element.toFront() ➞

将元素向上移动。

返回： 对象 元素

2.44 Element.touchcancel(handler) ➞

为元素添加 touchcancel 事件的处理程序。

参数

handler

函数

事件处理函数

返回：对象元素

2.45 Element.touchend(handler) ➞

为元素添加 touchend 事件的处理程序。

参数

handler

函数

事件处理函数

返回：对象元素

2.46 Element.touchmove(handler) ➞

为元素添加 touchmove 事件的处理程序。

参数

handler

函数

事件处理函数

返回：对象元素

2.47 Element.touchstart(handler) ➞

为元素添加 touchstart 事件的处理程序。

参数

handler

函数

事件处理函数

返回：对象元素

2.48 Element.transform([tstr]) ➞

```
r-30, 50, 10t10, 20s1.5
```

为元素增加变换，这是独立于其他属性的变换，即变换不改变矩形的 x 或 y 。变换字符串跟路径字符串的语法类似：

```
"t100,100r30,100,100s2,2,100,100r45s1.5"
```

每个字母是一个命令。有四个命令： t 是平移， r 是旋转， s 是缩放， m 是矩阵。

也有另类的“绝对”平移、旋转和缩放： T 、 R 和 S 。他们不会考虑到以前的变换。例如： $\dots T100,0$ 总会横向移动元素 100px，而 $\dots t100,0$ 会移动垂直移动

如果之前有 $r90$ 。比较以下 $r90t100,0$ 和 $r90T100,0$ 的结果。

所以，上面的例子可以读作“平移 100,100 围绕 100,100 旋转 30° 围绕 100,100 缩放两倍；围绕中心旋转 45°；相对中心缩放 1.5 倍”。正如你可以看到旋转和缩放命令的原点坐标为可选参数，默认的是该元素的中心点。矩阵接受六个参数。

用法

```

var el = paper.rect(10, 20, 300, 200); // translate 100, 100, rotate 45°,
translate -100, 0

el.transform("t100,100r45t-100,0"); // if you want you can append or prepend
transformations

el.transform("...t50,50");

el.transform("s2..."); // or even wrap

el.transform("t50,50...t-50-50"); // to reset transformation call method with
empty string

el.transform(""); // to get current value call it without parameters

console.log(el.transform());

```

参数

tstr

字符串

变换字符串

如果未指定 tstr

返回： 字符串 转换字符串

else

返回： 对象 元素

2.49 Element.translate(dx, dy) ➞

建议不要使用。使用 [Element.transform](#) 代替。为元素增加平移变换。

参数

dx

数字

水平位移

dy

数字

垂直位移

返回：对象元素

2.50 Element.unclick(handler) ➞

移除元素的单击事件处理程序。

参数

handler

函数

事件处理函数

返回：对象元素

2.51 Element.undblclick(handler) ➞

移除元素的 dblclick 事件处理程序。

参数

handler

函数

事件处理函数

返回：对象元素

2.52 Element.undrag() ➞

移除元素的 drag 事件处理程序。

2.53 Element.unhover(f_in, f_out) ➞

移除元素的 hover 事件处理程序。

参数

f_in

函数

开始悬停的处理程序

f_out

函数

结束悬停的处理程序

返回：对象元素

2.54 Element.unmousedown(handler) ➞

删除元素的 mousedown 事件处理程序。

参数

handler

函数

事件处理函数

返回：对象元素

2.55 Element.unmousemove(handler) ➞

删除元素的 mousemove 事件处理程序。

参数

handler

函数

事件处理函数

返回：对象元素

2.56 Element.unmouseout(handler) ➞

删除元素的 mouseout 事件处理程序。

参数

handler

函数

事件处理函数

返回：对象元素

2.57 Element.unmouseover(handler) ➞

移除元素的 mouseover 事件处理程序。

参数

handler

函数

事件处理函数

返回：对象元素

2.58 Element.unmouseup(handler) ➞

删除元素的 mouseup 事件处理程序。

参数

handler

函数

事件处理函数

返回：对象元素

2.59 Element.untouchcancel(handler) ➞

移除元素的 touchcancel 事件处理程序。

参数

handler

函数

事件处理函数

返回：对象元素

2.60 Element.untouchend(handler) ➞

移除元素的 touchend 事件处理程序。

参数

handler

函数

事件处理函数

返回： [元素](#)

2.61 Element.untouchmove(handler)

移除元素的 touchmove 事件处理程序。

参数

handler

[函数](#)

事件处理函数

返回： [元素](#)

2.62 Element.untouchstart(handler)


移除元素 touchstart 的事件处理程序。

参数

handler

[函数](#)

事件处理函数

返回： [元素](#)

3. 矩阵

3.1 Matrix.add(a, b, c, d, e, f, matrix)

为现有矩阵增加一个矩阵。

参数

a

数字

b

数字

c

数字

d

数字

e

数字

f

数字

matrix

对象

矩阵

3.2 Matrix.clone() ➞

返回矩阵的副本

返回：对象 矩阵

3.3 Matrix.invert() ➞

返回矩阵的反转版本

返回：对象 矩阵

3.4 Matrix.rotate(a, x, y) ➡

旋转矩阵

参数

a

数字

x

数字

y

数字

3.5 Matrix.scale(x, [y], [cx], [cy]) ➡

缩放矩阵

参数

x

数字

y

数字

cx

数字

cy

数字

3.6 Matrix.split()



将矩阵分割成简单的变换

返回：**对象** 格式：

dx **数字** 水平平移

dy **数字** 垂直平移 y

scalex **数字** 水平缩放

scaley **数字** 垂直缩放

shear **数字** 剪切

rotate **数字** 旋转角度

isSimple **布尔** 是否可以简单变换代替

3.7 Matrix.toTransformString()



返回转换字符串

返回：**字符串** 转换字符串

3.8 Matrix.translate(x, y)



平移矩阵

参数

数字

y

数字

3.9 Matrix.x(x, y) ➞

返回给定点在经过变换之后的 x 坐标。见 [Matrix.y](#)

参数

x

数字

y

数字

返回：数字 x

3.10 Matrix.y(x, y) ➞

返回给定点在经过变换之后的 y 坐标。见 [Matrix.x](#)

参数

x

数字

y

数字

返回：数字 y

4. 画布

4.1 Paper.add(json) ➡

导入 JSON 格式的元素。格式：`{type: type, <attributes>}`

参数

json

数组

返回 **对象** 导入的元素集合

用法

```
paper.add([
  {
    type: "circle",
    cx: 10,
    cy: 10,
    r: 5
  },
  {
    type: "rect",
    x: 10,
    y: 10,
    width: 10,
```

```
    height: 10,  
    fill: "#fc0"  
}  
]);
```

4.2 Paper.bottom ➞

指向画布上的底层元素

4.3 Paper.ca ➞

对象

[Paper.customAttributes](#) 的引用

4.4 Paper.circle(x, y, r) ➞

绘制一个圆。

参数

x

数字

圆心的 x 坐标

y

数字

圆心的 y 坐标

r

数字

半径,半径距离

返回：对象 Raphael 元素对象，类型为“圆”

用法

```
var c = paper.circle(50, 50, 40);
```

4.5 Paper.clear() ➞

清空画布，即删除所有画布上的元素。

4.6 Paper.customAttributes ➞

对象

你可以添加自定义属性来方便设置一批属性：

用法

```
paper.customAttributes.hue = function (num) {  
    num = num % 1;  
    return {fill: "hsb(" + num + ", 0.75, 1)"};  
};  
  
// Custom attribute "hue" will change fill to be given hue with fixed  
saturation and brightness. // Now you can use it like this.  
var c = paper.circle(10, 10, 10).attr({hue: .45}); // or even like this:  
  
c.animate({hue: 1}, 1e3);  
  
// You could also create custom attribute with multiple parameters:  
  
paper.customAttributes.hsb = function (h, s, b) {  
    return {fill: "hsb(" + [h, s, b].join(",") + ")"};  
};
```

```
};  
  
c.attr({hsb: "0.5 .8 1"});  
  
c.animate({hsb: [1, 0, 0.5]}, 1e3);
```

4.7 Paper.ellipse(x, y, rx, ry) ➞

绘制一个椭圆。

参数

x

数字

该中心的 x 坐标

y

数字

y 坐标的中心

rx

数字

水平半径

ry

数字

垂直半径

返回： 对象 Raphael 元素对象，类型为“椭圆”

用法

```
var c = paper.ellipse(50, 50, 40, 20);
```

4.8 Paper.forEach(callback, thisArg) ➞

为画布上每个元素执行指定函数

如果回调函数返回 *false* 则会停止循环。

参数

callback

函数

函数运行

thisArg

对象

回调函数的环境

返回：对象 画布对象

用法

```
paper.forEach(function (el) {  
  
    el.attr({ stroke: "blue" });  
  
});
```

4.9 Paper.getById(id) ➞

返回匹配指定 ID 的元素。

参数

id


数字

ID

返回：对象 Raphael 元素对象

4.10 Paper.getElementByPoint(x, y) ➞

返回位于指定位置的最上层元素

返回：  Raphael 元素对象

参数

x

数字

相对窗口左上角的 x 坐标

y

数字

相对窗口左上角的 y 坐标

用法

```
paper.getElementByPoint(mouseX, mouseY).attr({stroke: "#f00"});
```

4.11 Paper.getElementsByPoint(x, y) ➞

返回位于指定位置的所有元素集合

参数

x


数字

点的 x 坐标

y

数字

点的 y 坐标

返回：  集合

4.12 Paper.getFont(family, [weight], [style], [stretch]) ➞

根据指定的参数从注册字体中获取字体对象。您可以只指定字体名称中的一部分，像 “Myriad” ， “Myriad Pro” 。

参数

family	
字符串	
	字体名称或其中的任何单词
weight	
字符串	
	字体粗细
style	
字符串	
	字体样式
stretch	
字符串	
	字体变形

返回： 对象 字体对象

用法

```
paper.print(100, 100, "Test string", paper.getFont("Times", 800), 30);
```

4.13 Paper.image(src, x, y, width, height) ➡

嵌入一张图像。

参数

src	
字符串	
	源图像的 URI
x	
数字	
	x 坐标

y

数字

y 坐标

width


数字

图像的宽度

height

数字

图像的高度

返回：  Raphael 元素对象，类型为“图像”

用法

```
var c = paper.image("apple.png", 10, 10, 80, 80);
```

4.14 Paper.path([pathString])

由指定的路径字符串创建一个路径元素。

参数

pathString

字符串

SVG 格式的路径字符串。

路径字符串由一个或多个命令组成。每个命令以一个字母开始，随后是逗号(“,”)

分隔的参数。例如：

```
"M10,20L30,40"
```

我们看到两个命令：“M”与参数(10, 20)和“L”与参数(30, 40)大写字母的意思是命令是绝对的，小写是相对的。

这里是可用命令的简表，详细内容见 [SVG 路径字符串格式](#)。

命令	名称	参数
M	移动到 (moveTo)	(x y)+
Z	闭合路径 (closepath)	(none)
L	直线 (lineTo)	(x y)+
H	水平直线	x+
V	垂直直线	y+
C	曲线 (curveto)	(x1 y1 x2 y2 x y)+
S	平滑曲线	(x2 y2 x y)+
Q	二次贝赛尔曲线	(x1 y1 x y)+
T	平滑二次贝塞尔曲线	(x y)+
A	椭圆弧	(rx ry x-axis-rotation large-arc-flag sweep-flag x y)+
R	Catmull-Rom 曲线 *	x1 y1 (x y)+

* “Catmull-Rom 曲线” 不是标准 SVG 命令，在 2.0 版时被加入。注：有种特殊情况，路径只包含三个命令：“M10,10R...z”。在这种情况下，路径将平滑连接到它的起点。

用法

```
var c = paper.path("M10 10L90 90");// draw a diagonal line:// move to 10,10,  
line to 90,90
```

路径字符串的例子可以查看这些图标：<http://raphaeljs.com/icons/>

4.15 Paper.print(x, y, string, font, [size], [origin], [letter_spacing]) ➞

创建路径，表示指定字体，指定大小，在指定位置的指定文本。该方法的结果是路径元素，它包含整个文本作为一个单独的路径。

参数

x

数字

x 坐标

y

数字

y 坐标

string

字符串

显示文本

font

对象

字体对象，见 [Paper.getFont](#)

size

数字

字体大小，默认是 16

origin

字符串

文本垂直对齐方式 "baseline" 或 "middle"，默认为 "middle"

letter_spacing

数字

范围 -1..1，默认为 0

返回：对象产生的路径元素，其中包括所有字母

用法

```
var txt = r.print(10, 50, "print", r.getFont("Museo"), 30).attr({fill: "#fff"});
```

4.16 Paper.raphael ➞

指向 [Raphael](#) 对象

4.17 Paper.rect(x, y, width, height, [r]) ➞

绘制一个矩形。

参数

x

数字

左上角的 x 坐标

y

数字

左上角的 y 坐标

width

数字

宽度

height

数字

高度

r

数字

圆角半径，默认为 0

返回： 对象 Raphael 元素对象，类型为“矩形”

用法

```
// regular rectanglevar c = paper.rect(10, 10, 50, 50); // rectangle with
rounded cornersvar c = paper.rect(40, 40, 50, 50, 10);
```

4. 18 Paper.remove() ➞

从 DOM 中删除的画布。

4. 19 Paper.renderfix() ➞

修复 Firefox 和 IE9 的像素渲染的问题。如果画布依赖于其他元素，换行后，它可能会移动半像素导致线条失去细节。这个方法解决此问题。

4.20 Paper.safari() ➞

Safari (WebKit) 有一个不方便的渲染错误：有时需要强制渲染。此方法对于处理此错误应该有帮助。

4.21 Paper.set() ➞

创建类数组对象，保存和一次操作其中的多个元素。警告：它不创建任何元素，只是分组已存在的元素。“集合”充当伪元素 - 所有元素可用的方法，都可以在集合上使用。

返回： 对象 类数组对象表示元素集合

用法

```
var st = paper.set();  
  
st.push(  
    paper.circle(10, 10, 5),  
    paper.circle(30, 10, 5)  
);  
  
st.attr({fill: "red"}); // changes the fill of both circles
```

4.22 Paper.setFinish() ➞

见 [Paper.setStart](#) 。此方法结束捕获并返回元素集合。

返回： 对象 集合

4.23 Paper.setSize(width, height) ➞

如果需要更改尺寸的画布，调用此方法

参数

width

数字

新的画布宽度

height

数字

新的画布高度

4.24 Paper.setStart() ➞

创建 [Paper.set](#) 。调用此方法之后到调用 [Paper.setFinish](#) 方法之前创建的所有元素都会被添加到集合内。

用法

```
paper.setStart();

paper.circle(10, 10, 5),

paper.circle(30, 10, 5)var st = paper.setFinish();

st.attr({fill: "red"}); // changes the fill of both circles
```

4.25 Paper.setViewBox(x, y, w, h, fit) ➞

设置画布的视框。实际上，你可以通过指定新的边界来缩放和平移整个画布。

参数

x

数字

新 x 坐标，默认为 0

y

数字

新 y 坐标，默认为 0

w

数字

新的画布宽度

h

数字

新的画布高度

适合

布尔

设置 `true`，如果你想要图形适应新的边界框

4.26 Paper.text(x, y, text) ↗

Raphaëlkicksbutt!

绘制一个文本字符串。如果你需要换行，使用 “\n”。

参数

x

数字

x 坐标

y

数字

y 坐标

text

字符串

文本字符串

返回： **对象** Raphael 元素对象，类型为 “文本”

用法

```
var t = paper.text(50, 50, "Raphaël\nkicks\nbutt!");
```

4.27 Paper.top ➞

指向画布上的顶端元素

5. Raphael (...) ➞

创建一个画布对象用来绘制。你必须第一步就这么做，该实例将来调用的所有绘图方法都被绑定到这个画布。

参数

container

HTML 元素字符串

DOM 元素或它的 ID，这将是绘图表面的父级节点

width

数字

height

数字

callback

函数

回调函数，新创建的画布将成为该函数的环境

或

x

数字

y

数字

width

数字

height

数字

callback

函数

回调函数，新创建的画布将成为该函数的环境

或

all

数组

（前 3 个或 4 个数组元素对应 [containerID,width,height]或[x,y,width,height]。其余数组元素是元素描述{type:类型,<属性>}）。见 [Paper.add](#) 。

callback

函数

回调函数，新创建的画布将成为该函数的环境

或

onReadyCallback

函数

DOM ready 事件回调。你也可以使用 eve 的“DOMLoad”事件。在这种情况下该方法将返回 *undefined*。

返回：对象 [画布](#)

用法

```
// Each of the following examples create a canvas that is 320px wide by 200px high. // Canvas is created at the viewport's 10,50 coordinate.  
var paper
```

```

= Raphael(10, 50, 320, 200); // Canvas is createa at the top left corner of the
#notepaa element // (or its top right corner in dir="rtl" elements) var paper =
Raphael(document.getElementById("notepad"), 320, 200); // Same as above var paper
= Raphael("notepad", 320, 200); // Image dump var set = Raphael(["notepad", 320,
200, {
    type: "rect",
    x: 10,
    y: 10,
    width: 25,
    height: 25,
    stroke: "#f00"
}, {
    type: "text",
    x: 30,
    y: 40,
    text: "Dump"
}]);

```

5.1 Raphael.angle(x1, y1, x2, y2, [x3], [y3]) ⇨

返回两个或三个点之间的角度

参数

x1

数字

第一点的 x 坐标

y1

数字

第一点的 y 坐标

x2

数字

第二点的 x 坐标

y2

数字

第二点的 y 坐标

x3

数字

第三点的 x 坐标

y3

数字

第三点的 y 坐标

返回：数字度角

5.2 Raphael.animation(params, ms, [easing], [callback]) ➞

创建一个动画对象可以传递给 [Element.animate](#) 或 [Element.animateWith](#) 方法。见 [Animation.delay](#) 和 [Animation.repeat](#)。

参数

params

对象

元素的最终属性，见 [Element.attr](#)

ms

数字

动画持续时间（毫秒）

easing

字符串

曲线类型。接受 [Raphael.easing_formulas](#) 其中之一或 CSS 格式: `cubic-bezier(XX, XX, XX, XX)`

callback

函数

回调函数。动画结束时将被调用。

返回： 对象 [动画](#)

5.3 Raphael.bezierBBox(…) ➞

实用方法返回指定三次贝赛尔曲线的边界框

参数

p1x

数字

第一点的 x 坐标

p1y

数字

第一点的 y 坐标

c1x

数字

第一个锚点的 x 坐标

c1y

数字

第一个锚点的 y 坐标

c2x

数字

第二个锚点的 x 坐标

c2y

数字

第二个锚点的 y 坐标

p2x

数字

第二点的 x 坐标

p2y

数字

第二点的 y 坐标

或

bez

数组

贝赛尔曲线的 6 个点组成的数组

返回：对象点，格式：

```
{
```

```
  min: {
```

```
    x: 数字最左边点的 x 坐标
```

```
    y: 数字最上边点的 y 坐标
```

```
  }
```

```
  max: {
```

```
    x: 数字最右边点的 x 坐标
```

```
    y: 数字最下边点的 y 坐标
```

```
  }
```

```
}
```

5.4 Raphael.color(clr) ➞

解析指定颜色字符串并返回包含全部值的颜色对象。

参数

CLR

字符串

颜色字符串，支持的格式见 [Raphael.getRGB](#)

返回： 对象 RGB 和 HSB 组合对象：

```
{
  r 数字 红色,
  g 数字 绿色,
  b 数字 蓝色,
  hex 字符串 颜色的 HTML/CSS 格式：#.....,
  error 布尔 true , 如果无法解析字符串,
  h 字符串 色相,
  s 字符串 饱和度,
  v 字符串 色调,
  l 字符串 亮度
}
```

5.5 Raphael.createUUID() ➞

返回 RFC4122 , 第 4 版 ID

5.6 Raphael.deg(deg) ➞

弧度转角度

参数

deg

数字

弧度值

返回：数字角度值

5.7 Raphael.easing_formulas ↗

对象，其中包含动画过渡公式。你可以自己扩展。默认情况下，它含有以下过渡：

- “linear”（线性）
- “<” 或 “easeIn” 或 “ease-in”（由慢到快）
- “>” 或 “easeOut” 或 “ease-out”（又快到慢）
- “<>” 或 “easeInOut” 或 “ease-in-out”（由慢到快再到慢）
- “backIn” 或 “back-in”（开始时回弹）
- “backOut” 或 “back-out”（结束时回弹）
- “elastic”（橡皮筋）
- “bounce”（弹跳）

见[过渡示例](#)。

5.8 Raphael.el ↗

对象

你可以给元素添加自己的方法。当你要修改默认功能或要在一个方法中包装一些常见的变换或属性，这很有用。与画布方法不同，你可以随时重新定义元素的方法。扩展元素的方法，不会影响元素集合。

用法

```
Raphael.el.red = function () {  
    this.attr({fill: "#f00"});  
}; // then use it  
paper.circle(100, 100, 20).red();
```

5.9 Raphael.findDotsAtSegment(p1x, ply, clx, cly, c2x, c2y, p2x, p2y, t) ↗

实用方法在给定的三次贝赛尔曲线上查找给定位置的点的坐标。

参数

p1x

数字

第一点的 x 坐标

ply

数字

第一点的 y 坐标

clx

数字

第一个锚点的 x 坐标

cly

数字

第一个锚点的 y 坐标

c2x

数字

第二个锚点的 x 坐标
c2y
数字
第二个锚点的 y 坐标
p2x
数字
第二点的 x 坐标
p2y
数字
第二点的 y 坐标
t
数字
曲线上的位置（0 .. 1）

返回：对象点，格式：

```
{
  .
  .
  .
  x: 数字点的 x 坐标
  y: 数字点的 y 坐标
  m: {
    .
    .
    .
    x: 数字左锚的 x 坐标
    y: 数字左锚的 y 坐标
  }
  n: {
    .
    .
    .
    x: 数字右锚的 x 坐标
    y: 数字右锚的 y 坐标
  }
}
```

```

start: {
    x: 数字 曲线起点的 x 坐标
    y: 数字 曲线起点的 y 坐标
}
end: {
    x: 数字 曲线终点的 x 坐标
    y: 数字 曲线终点的 y 坐标
}
alpha: 数字 曲线上该点导数（切线）的角度
}

```

5.10 Raphael.fn

对象

你可以为画布添加你自己的方法。例如，如果你想画一个饼图，您可以创建自己的饼图函数然后作为 Raphael 插件。要这么做，你需要扩展 *Raphael.fn* 对象。您应该在 Raphael 实例被创建之前修改 *fn* 对象，否则将没有任何效果。请注意：Raphael2.0 移除了为插件提供命名空间的功能。应该由插件确保命名空间负责保证适当的环境的安全。

用法

```

Raphael.fn.arrow = function (x1, y1, x2, y2, size) {
    return this.path( ... );
}; // or create namespace

Raphael.fn.mystuff = {
    arrow: function () {...},
    star: function () {...},
    // etc...
};

var paper = Raphael(10, 10, 630, 480); // then use it

paper.arrow(10, 10, 30, 30, 5).attr({fill: "#f00"});

paper.mystuff.arrow();

paper.mystuff.star();

```

5.11 Raphael.format(token, ...)

简单的格式话功能。将 " {<number>} " 结构替换为相应的参数。

参数

token

字符串

格式字符串

...

字符串

参数的其余部分将被视为替换参数

返回： 字符串 经过格式化的字符串

用法

```

var x = 10,

    y = 20,

    width = 40,

    height = 50; // this will draw a rectangular shape equivalent to
                  "M10,20h40v50h-40z"

paper.path(Raphael.format("M{0},{1}h{2}v{3}h{4}z", x, y, width, height, -width));

```

5.12 Raphael.fullfill(token, json) ↗

比 [Raphael.format](#) 更先进一点点的格式化功能。将 " {<name>} " 结构替换为相应的参数。

参数

token

字符串

格式

json

对象

对象，其属性将被用来做替换

返回： 字符串 经过格式化的字符串

用法

```

// this will draw a rectangular shape equivalent to "M10,20h40v50h-40z"

paper.path(Raphael.fullfill("M{x},{y}h{dim.width}v{dim.height}h{dim['negative width']}z", {

    x: 10,

    y: 20,

```

```
dim: {  
  
  width: 40,  
  
  height: 50,  
  
  "negative width": -40  
  
}  
  
});
```

5.13 Raphael.getColor([value]) ➞

每次调用返回色谱中的下一个颜色。要重置回红色，调用

[Raphael.getColor.reset](#)

参数

值

数字

亮度，默认为 0.75

返回： 字符串 颜色的十六进制表示。

5.14 Raphael.getColor.reset() ➞

复位色谱位置 [Raphael.getColor](#) 回到红色。

5.15 Raphael.getPointAtLength(path, length) ➞

返回位于路径上给定长度的点的坐标。

参数

path

字符串

SVG 路径字符串

length

数字

返回：对象点：

{

x:数字 x 坐标

y:数字 y 坐标

alpha:数字 导数 (切线) 的角度

}

5.16 Raphael.getRGB(colour) ⇨

将颜色字符串解析为 RGB 对象

参数

colour

字符串

颜色字符串，格式：

- 颜色的名称 ("red " , "green " , "cornflowerblue "等)
- #...——HTML 颜色缩写 : ("#000" , "#fc0" 等)
- #.....——HTML 颜色 : ("#000000" , "#bd2300")

- `rgb(•••, •••, •••)`——红色、绿色和蓝色通道值：(`"rgb(200, 100, 0)"`) RGB (••••••••) -红色，绿色和蓝色通道值 (`rgb(200, 100, 0)`)
- `rgb(•••%, •••%, •••%)`——和上面一样，但是用百分比：(`"rgb(100%, 175%, 0%)"`)
- `hsb(•••, •••, •••)`——色相、饱和度和亮度值：(`"hsb(0.5, 0.25, 1)"`)
- `hsb(•••%, •••%, •••%)` ——和上面一样，但是用百分比
- `hsl(•••, •••, •••)` ——和 HSB 类似
- `hsl(•••%, •••%, •••%)` ——和 HSB 类似

返回： 对象 RGB 对象，格式：

```
{
```

```
  r 数字 红色,
```

```
  g 数字 绿色,
```

```
  b 数字 蓝色
```

```
  hex 字符串 颜色的 HTML/CSS 格式：#••••••,
```

```
  error 布尔 true，如果不能解析
```

```
}
```

5.17 Raphael.getSubpath(path, from, to) ➡

从指定路径获取一段子路径，指定起始和结束位置。

参数

path

字符串

SVG 路径字符串

from

数字

片段开始位置

to

数字

片段结束的位置

返回：字符串段的路径字符串

5.18 Raphael.getTotalLength(path) ➞

返回在指定路径的长度（像素）。

参数

path

字符串

SVG 的路径字符串。

返回：数字长度

5.19 Raphael.hsb(h, s, b) ➞

HSB 值转换为十六进制颜色。

参数

h

数字

色相

s

数字

饱和度

b

数字

色调或明度

返回： 字符串 颜色的十六进制表示。

5.20 Raphael.hsb2rgb(h, s, v) ↗

HSB 值转换为 RGB 对象。

参数

h

数字

色相

s

数字

饱和度

v

数字

色调或明度

返回： 对象 RGB 对象，格式：

```
{
```

```
  r 数字 红色,
```

```
  g 数字 绿色,
```

```
  b 数字 蓝色,
```

```
  hex 字符串 HTML/CSS 颜色格式： #••••••
```

```
}
```

5.21 Raphael.hsl(h, s, l)

HSL 值转换为十六进制颜色。

参数

h

数字

色相

s

数字

饱和度

l

数字

亮度

返回： 字符串颜色的十六进制表示。

5.22 Raphael.hsl2rgb(h, s, l)

HSL 值转换为 RGB 对象。

参数

h

数字

色相

s

数字

饱和度

l

数字

亮度

返回：对象 RGB 对象，格式：

```
{
```

```
  r 数字红色,
```

```
  g 数字绿色,
```

```
  b 数字蓝色,
```

```
  hex 字符串 HTML/CSS 颜色格式： #.....
```

```
}
```

5.23 Raphael.is(o, type) ➞

代替 *typeof* 操作符。

参数

o
...

任何对象或基本类型变量

type

字符串

类型，即“string”，“function”，“number”等名称

返回：布尔 指定值是指定的类型

5.24 Raphael.isBBoxIntersect(bbox1, bbox2) ➞

实用方法返回 *true* ，如果两个边界框相交

参数

bbox1

字符串

第一个边界框

bbox2

字符串

第二个边界框

返回：布尔 *true*，如果它们相交

5.25 Raphael.isPointInsideBBox(bbox, x, y) ↗

实用方法返回 *true*，如果给定的点在边界框内。

参数

bbox

字符串

边界框

x

字符串

点的 x 坐标

y

字符串

点的 y 坐标

返回：布尔 *true*，如果点在边界框内

5.26 Raphael.isPointInsidePath(path, x, y) ↗

实用方法返回 *true*，如果指定的点在给定的封闭路径内。

参数

路径

字符串

路径字符串

x

数字

点的 x 坐标

y

数字

点的 y 坐标

Returns: 布尔 true，如果点在闭合路径内

5.27 Raphael.mapPath(path, matrix) ➞

用指定矩阵变换路径

参数

path

字符串

路径字符串

matrix

对象

见 矩阵

返回： 字符串 经过变换的路径字符串

5.28 Raphael.matrix(a, b, c, d, e, f) ➞

实用方法用指定的参数生成矩阵

参数

a

数字

b

数字

c

数字

d

数字

e

数字

f

数字

返回：对象 矩阵

5.29 Raphael.ninja() ➞

如果你不想保留 Raphael 的痕迹（好吧，Raphael 只创建一个全局变量 *Raphael*，但那又怎样。）你可以使用 *ninja* 方法。请注意，在这种情况下，插件可能会停止工作，因为它们依赖全局变量的存在。

返回：对象 Raphael 对象

用法

```
(function (local_raphael) {  
  
    var paper = local_raphael(10, 10, 320, 200);  
  
    ...  
  
})(Raphael.ninja());
```

5.30 Raphael.parsePathString(pathString) ➡

实用方法将指定路径字符串解析成一个数组，其中的元素是路径段数组。

参数

pathString

字符串数组

路径字符串或路径段数组（在后一种情况下，将马上返回）

返回： 数组 路径段数组。

5.31 Raphael.parseTransformString(TString) ➡

实用方法解析指定路径字符串为变换数组。

参数

TString

字符串数组

变换字符串或变换数组（在后一种情况下，将马上返回）

返回： 数组 变换数组。

5.32 Raphael.path2curve(pathString) ➡

实用方法将指定路径转换为新路径，新路径的所有片段都是三次贝赛尔曲线。

参数

pathString

字符串数组

路径字符串或路径段数组

返回： 数组 路径段数组

5.33 Raphael.pathBBox(path) ➞

实用方法返回指定路径的边界框

参数

path

字符串

路径字符串

返回： 对象 边界框

```
{
```

x: 数字 框左上角的 x 坐标

y: 数字 框左上角的 y 坐标

x2: 数字 框右下角的 x 坐标

y2: 数字 框右下角的 y 坐标

width: 数字 框的宽度

height: 数字 框的高度

```
}
```

5.34 Raphael.pathIntersection(path1, path2) ➞

实用方法查找两条路径的交点

参数

path1

字符串

路径字符串

path2

字符串

路径字符串

Returns: 数组 交点

[

{

x: 数字 点的 x 坐标

y: 数字 点的 y 坐标

t1: 数字 path1 部分路径段的 t 值

t2: 数字 path2 部分路径段的 t 值

segment1: 数字 path1 部分路径段的顺序

segment2: 数字 path2 部分路径段的顺序

bez1: 数组 path1 部分路径段的贝赛尔曲线，表示为 8 个坐标

bez2: 数组 path2 部分路径段的贝赛尔曲线，表示为 8 个坐标

}

]

5.35 Raphael.pathToRelative(pathString) ➡

实用方法将路径转换为相对形式

参数

pathString

字符串数组

路径字符串或路径段数组

返回： 数组 路径段数组组成的数组

5.36 Raphael.rad(deg) ➞

角度转弧度

参数

deg

数字

角度

Returns: 数字 弧度值。

5.37 Raphael.registerFont(font) ➞

给 Raphael 字体注册指定字体。应当作为 Cufón 字体文件的内部调用。返回原来的参数，所以它可以链式操作。


[更多关于 Cufón 和如何转换字体形式从 TTF , OTF 等等到 JavaScript 文件。](#)

参数

font

对象

注册的字体

返回：传入的字体

用法

```
Cufon.registerFont(Raphael.registerFont({...}));
```

5.38 Raphael.rgb(r, g, b)

将 RGB 值转换为十六进制颜色。

参数

r

数字

红色

g


数字

绿色

b

数字

蓝色

返回：颜色的十六进制表示。

5.39 Raphael.rgb2hsb(r, g, b)

将 RGB 值转换为 HSB 对象。

参数

r

数字

红色

g

数字

绿色

数字

蓝色

返回：对象 HSB 对象，格式：

h 数字色相

S 数字饱和度

b 数字 明度

5.40 Raphael.rgb2hsl(r, g, b) ➡

将 RGB 值转换为 HSL 对象。

参数

数字

红色

数字

绿色

数字

蓝色

返回对象 HSL 对象，格式：

```
{
```

```
h 数字色相
```

```
s 数字饱和度
```

```
l 数字亮度
```

```
}
```

5.41 Raphael.setWindow(newwin)

需要在<iframe>绘制时使用。切换窗口为 iframe。

参数

newwin

窗口

新的窗口对象

5.42 Raphael.snapTo(values, value, [tolerance])



将给定值对齐到指定网格。

参数

values

数组数字

指定值的数组或网格的梯级

value

数字

要调整的值

tolerance

数字

对齐的阈值默认值是 10 。

返回：数量经过调整的值。

5.43 Raphael.st ➞

对象

你可以为元素和集合添加自己的方法。为给元素添加的每个方法写一个集合方法是明智的，你就可以在集合上调用同样的方法。见 [Raphael.el](#) 。

用法

```
Raphael.el.red = function () {  
    this.attr({fill: "#f00"});  
};  
  
Raphael.st.red = function () {  
    this.forEach(function (el) {  
        el.red();  
    });  
};  
  
// then use it  
  
paper.set(paper.circle(100, 100, 20), paper.circle(110, 100, 20)).red();
```

5.44 Raphael.svg ➞

布尔

如果浏览器支持 SVG 则为 *true*。

5.45 Raphael.toMatrix(path, transform) ➞

实用方法返回指定路径所应用的变换矩阵

参数

path

字符串

路径字符串

transform

字符串数组

变换字符串

返回：对象 矩阵

5.46 Raphael.transformPath(path, transform) ➞

实用方法返回指定路径经过指定变换后的路径

参数

path

字符串

路径字符串

transform

字符串数组

变换字符串

返回：字符串 路径

5.47 Raphael.type ➞

字符串

可以是“SVG”，“VML” 或为空，依赖于浏览器的支持。

5.48 Raphael.vml ↗

布尔

true，如果浏览器支持 VML。

6. 集合

6.1 set.clear() ↗

移除集合中的所有元素

6.2 Set.exclude(element) ↗

从集合中删除指定元素

参数

element

对象

要删除的元素

Returns: 布尔 *true*，如果对象存在并且 已经被删除

6.3 Set.forEach(callback, thisArg) ↗

为集合中的每个元素执行指定函数。

如果函数返回 *false* 则停止循环运行。

参数

callback

函数

回调函数

thisArg

对象

回调函数的环境

返回： 对象集合对象

6.4 Set.pop() ➞

删除最后一个元素并返回它。

返回： 对象元素

6.5 Set.push() ➞

加入一个元素

返回： 对象原始元素

6.6 Set.splice(index, count, [insertion...]) ➞

从集合中删除指定元素

参数

index

数字

开始删除的位置

count

数字

删除个数

insertion...

对象

要在删除位置插入的元素

Returns: 对象 被删除的元素

7. eve(name, scope, varargs) ➞

使用指定的 *name*、作用域和参数触发事件。

参数

name

字符串

事件名称，点 (.) 或斜线 (/) 分隔

scope

对象

事件处理程序的环境

varargs

...

其余的参数将被发送到事件处理程序

Returns: 对象 监听器返回值组成的数组

7.1 eve.listeners(name) ➞

内部方法，返回监听指定 *name* 事件的回调函数组成的数组。

参数

name

字符串

事件名称，点（.）或斜线（/）分隔

返回：数组 事件处理程序的数组

7.2 eve.nt([subname]) ➞

可以在事件处理程序内使用来获取事件的实际名称。

参数

subname

字符串

事件的子名称

返回：字符串 名称的事件，如果没有指定 *subname*

或

返回：布尔 *true*，如果当前事件的名称中包含 *subname*

7.3 eve.off(name, f) ➞

从指定名称的事件监听器列表中删除指定的函数。

参数

name

字符串

事件名称，点（.）或斜线（/）分隔，可以用通配符

f

函数

事件处理函数

7.4 eve.on(name, f) ➡

为给定的名称绑定一个事件处理程序。可以在名称中使用通配符 “ * ”：

```
eve.on("*under.*", f);  
  
eve("mouse.under.floor"); // triggers f
```

使用 [eve](#) 触发监听。

参数

name

字符串

事件名称，点（.）或斜线（/）分隔，可以用通配符

f

函数

事件处理函数

返回： [函数](#) 返回的函数接受一个数字参数，表示处理程序的 z-index。这是一个可选功能，只有当你需要使用，以确保处理程序的一些子集被设置为特定的调用顺序，而不使用指派顺序。

例如：

```
eve.on("mouse", eat)(2);  
  
eve.on("mouse", scream);  
  
eve.on("mouse", catch)(1);
```

这将确保 *catch* 函数将被在 *eat* 之前调用。如果你想要把你的处理程序放在未索引的处理程序之前，指定一个负值。注：我假设大部分的时间你不需要担心的 z-index，但很高兴有此功能来 “以防万一”。

7.5 eve.once(name, f) ➞

为给定的名称绑定一个一次性事件处理程序。只运行一次，然后解除本身。

```
eve.once("login", f);  
  
eve("login"); // triggers f  
  
eve("login"); // no listeners
```

使用 [eve](#) 触发监听。

参数

name

字符串

事件名称，点（.）或斜线（/）分隔，可以用通配符

f

函数

事件处理函数

返回： [功能](#) 与 [eve.on](#) 返回的函数类似

7.6 eve.stop() ➞

用于在事件处理程序内停止事件，防止触发任何后续监听器。

7.7 eve.unbind() ➞

见 [eve.off](#)

7.8 eve.version ➞

字符串

当前库的版本。

txp