--

[动画 1](#_Toc2832)

[requestAnimationFrame 1](#_Toc21494)

[探究运动曲线方程 2](#_Toc13375)

[曲线相关的网站 3](#_Toc19049)

[sass实现复杂动画 3](#_Toc27404)

[demo 4](#_Toc1835)

[抛物线 4](#_Toc618)

[zhangxinxu tween.js 4](#_Toc1224)

[velocity 4](#_Toc14474)

[velocity 4](#_Toc26400)

[Basics 基础用法 4](#_Toc20089)

[参数 5](#_Toc3344)

[链式调用 5](#_Toc18843)

[loop-重复 6](#_Toc26744)

[easing-弹性动画 6](#_Toc5653)

[feature:color 6](#_Toc6031)

[command:scroll 6](#_Toc14689)

[command:reverse 6](#_Toc20888)

[Command:Stop 6](#_Toc32416)

[command:finish 6](#_Toc2674)

[transform 6](#_Toc4713)

[Feature: Promises 6](#_Toc2986)

[Feature: Mock  7](#_Toc29222)

[tween.js可生成平滑动画效果的js动画库 7](#_Toc30438)

Libscore scans the top million sites on the web to collect   
stats on JavaScript library usage. Explore this month's data.

动画

|  |  |
| --- | --- |
|  | H5动效的常见制作手法  <https://isux.tencent.com/h5active.html?variant=zh-hans>  H5玩法知多少  <https://isux.tencent.com/h5-play-ways.html>  H5动画开发快车道AnimateCC与createjs开发实践  <https://isux.tencent.com/h5-animation-highway.html>  使用缓动函数制作更自然的动画  <https://zhuanlan.zhihu.com/p/26535740>  自由分享jQuery、html5和css3的插件库  <http://www.htmleaf.com/>  10个值得前端收藏的CSS3动效库（工具）  <http://www.uisdc.com/best-css3-animation-library>  实用的 CSS — 动画性能对比  <http://joveyzheng.com/2017/07/18/css-animations-performance/> |
|  | **[SVGI](https://angelmmiguel.github.io/svgi/" \t "http://www.uisdc.com/_blank)**  IMG_256  SVGI 是一款用来查找SVG元素和文件的命令行工具，安装它之后，你可以采用svgi 命令来检索 SVG 文档的大小、元素、层次结构和节点列表。 |
|  | Modernizr 是一个 JavaScript 库，用于检测用户浏览器的 HTML5 与 CSS3 特性。  为什么使用 Modernizr  使用炫酷新颖的 web 技术是一件很有趣的事情，但你不得不面对落后的浏览器。Modernizr 使你可以方便地为各种情况编写 JavaScript 和 CSS，无论浏览器是否支持这些特性。这是处理渐进增强的完美方案。  工作原理  Modernizr 会在页面加载后立即检测特性；然后创建一个包含检测结果的 JavaScript 对象，同时在 html 元素加入方便你调整 CSS 的 class 名。  查看 Modernizr 检测的[特性列表](http://modernizr.cn/docs/" \l "features-css)。 |
|  |  |

requestAnimationFrame

|  |  |
| --- | --- |
|  | CSS3动画那么强，requestAnimationFrame还有毛线用？  <http://www.zhangxinxu.com/wordpress/2013/09/css3-animation-requestanimationframe-tween-%E5%8A%A8%E7%94%BB%E7%AE%97%E6%B3%95/>  window.requestAnimationFrame  <https://developer.mozilla.org/zh-CN/docs/Web/API/Window/requestAnimationFrame>  浅析 requestAnimationFrame  <http://taobaofed.org/blog/2017/03/02/thinking-in-request-animation-frame/>  **[requestAnimationFrame，Web中写动画的另一种选择](http://www.cnblogs.com/Wayou/p/requestAnimationFrame.html)**  http://www.cnblogs.com/Wayou/p/requestAnimationFrame.html  如果你想做逐帧动画的时候，你应该用这个方法。这就要求你的动画函数执行会先于浏览器重绘动作。通常来说，被调用的频率是每秒60次，但是一般会遵循W3C标准规定的频率。如果是后台标签页面，重绘频率则会大大降低。  回调函数只会被传入一个[DOMHighResTimeStamp](https://developer.mozilla.org/zh-CN/docs/Web/API/DOMHighResTimeStamp" \o "DOMHighResTimeStamp 是一个double类型，用于存储时间值。该值可以是离散的时间点或两个离散时间点之间的时间差。T单位为毫秒 ms (milliseconds) ，应准确至5微秒 µs (microseconds)。但是，如果浏览器无法提供准确到5微秒的时间值(例如,由于硬件或软件的限制), 浏览器可以以毫秒为单位的精确到毫秒的时间表示该值。)参数，这个参数指示当前被 requestAnimationFrame 序列化的函数队列被触发的时间。因为很多个函数在这一帧被执行，所以每个函数都将被传入一个相同的时间戳，尽管经过了之前很多的计算工作。这个数值是一个小数，单位毫秒，精确度在 10 µs。  语法  requestID = window.requestAnimationFrame(callback); // Firefox 23 / IE10 / Chrome / Safari 7 (incl. iOS)  requestID = window.mozRequestAnimationFrame(callback); // Firefox < 23  requestID = window.webkitRequestAnimationFrame(callback); // Older versions Chrome/Webkit  参数  callback  在每次需要重新绘制动画时,会调用这个参数所指定的函数。这个回调函数会收到一个参数，这个 [DOMHighResTimeStamp](https://developer.mozilla.org/zh-CN/docs/Web/API/DOMHighResTimeStamp" \o "DOMHighResTimeStamp 是一个double类型，用于存储时间值。该值可以是离散的时间点或两个离散时间点之间的时间差。T单位为毫秒 ms (milliseconds) ，应准确至5微秒 µs (microseconds)。但是，如果浏览器无法提供准确到5微秒的时间值(例如,由于硬件或软件的限制), 浏览器可以以毫秒为单位的精确到毫秒的时间表示该值。) 类型的参数指示当前时间距离开始触发 requestAnimationFrame 的回调的时间。  返回值  requestID 是一个长整型非零值,作为一个唯一的标识符.你可以将该值作为参数传给 [window.cancelAnimationFrame()](https://developer.mozilla.org/zh-CN/docs/Web/API/Window/cancelAnimationFrame" \o "取消一个先前通过调用window.requestAnimationFrame()方法添加到计划中的动画帧请求.) 来取消这个回调函数。  例子  window.requestAnimationFrame = window.requestAnimationFrame || window.mozRequestAnimationFrame || window.webkitRequestAnimationFrame || window.msRequestAnimationFrame;  var start = null;  var d = document.getElementById('SomeElementYouWantToAnimate');  function step(timestamp) {  if (start === null) start = timestamp;  var progress = timestamp - start;  d.style.left = Math.min(progress/10, 200) + "px";  if (progress < 2000) {  requestAnimationFrame(step);  }  }  requestAnimationFrame(step);  requestAnimationFrame就是为了这个而出现的。我所做的事情很简单，跟着浏览器的绘制走，如果浏览设备绘制间隔是16.7ms，那我就这个间隔绘制；如果浏览设备绘制间隔是10ms, 我就10ms绘制。这样就不会存在过度绘制的问题，动画不会掉帧，自然流畅  IMG_256    使用方式：var handle = requestAnimationFrame(renderLoop);  没调用一次只会执行一次绘制，因此，如果想要实现联系动画，就使用renderLoop  想要简单的兼容，可以这样子：  window.requestAnimFrame = (function(){  return window.requestAnimationFrame ||  window.webkitRequestAnimationFrame ||  window.mozRequestAnimationFrame ||  function( callback ){  window.setTimeout(callback, 1000 / 60);  };  })();  但是呢，并不是所有设备的绘制时间间隔是1000/60 ms, 以及上面并木有cancel相关方法，所以，就有下面这份更全面的兼容方法：  (function() {  var lastTime = 0;  var vendors = ['webkit', 'moz'];  for(var x = 0; x < vendors.length && !window.requestAnimationFrame; ++x) {  window.requestAnimationFrame = window[vendors[x] + 'RequestAnimationFrame'];  window.cancelAnimationFrame = window[vendors[x] + 'CancelAnimationFrame'] || // Webkit中此取消方法的名字变了  window[vendors[x] + 'CancelRequestAnimationFrame'];  }  if (!window.requestAnimationFrame) {  window.requestAnimationFrame = function(callback, element) {  var currTime = new Date().getTime();  var timeToCall = Math.max(0, 16.7 - (currTime - lastTime));  var id = window.setTimeout(function() {  callback(currTime + timeToCall);  }, timeToCall);  lastTime = currTime + timeToCall;  return id;  };  }  if (!window.cancelAnimationFrame) {  window.cancelAnimationFrame = function(id) {  clearTimeout(id);  };  }  }()); |
| 探究运动曲线方程 | CSS3支持的动画效果有限  由于CSS3动画的贝塞尔曲线是一个标准3次方曲线（详见：[贝塞尔曲线与CSS3动画、SVG和canvas的基情](http://www.zhangxinxu.com/wordpress/?p=3626" \t "http://www.zhangxinxu.com/wordpress/2013/09/css3-animation-requestanimationframe-tween-%E5%8A%A8%E7%94%BB%E7%AE%97%E6%B3%95/_blank)），因此，只能是：Linear, Sine, Quad, Cubic, Expo等，但对于Back, Bounce等缓动则只可观望而不可亵玩焉。  IMG_256 IMG_257  缓动(Tween)：  Linear：无缓动效果  Quadratic：二次方的缓动（t^2）  Cubic：三次方的缓动（t^3）  Quartic：四次方的缓动（t^4）  Quintic：五次方的缓动（t^5）  Sinusoidal：正弦曲线的缓动（sin(t)）  Exponential：指数曲线的缓动（2^t）是一个开始非常慢，中后期非常快的曲线  Circular：圆形曲线的缓动（sqrt(1-t^2)）顾名思义就是弧（1/4圆，如果选择了InOut就是两个外切的1/4圆）  Elastic：指数衰减的正弦曲线缓动[这是个模拟弹簧运动的曲线]  超过范围的三次方缓动（(s+1)\*t^3 – s\*t^2）  指数衰减的反弹缓动  每个效果都分三个缓动方式，分别是（可采用后面的邪恶记忆法帮助记忆）：  easeIn：从0开始加速的缓动，想象OOXX进去，探路要花时间，因此肯定是先慢后快的；  easeOut：减速到0的缓动，想象OOXX出来，肯定定先快后慢的，以防掉出来；  easeInOut：前半段从0开始加速，后半段减速到0的缓动，想象OOXX进进出出，先慢后快然后再慢。 |
|  | 用得比较多的应该是其中的几个：  1. Quad – x^2：这条二次方曲线，就是匀变速直线运动曲线，大家应该还记得初中背得滚瓜烂熟的s=0.5 \* a \* t²吧。 有了匀变速运动曲线，很多现实中的运动都可以模拟了，如[匀加速运动](http://code.w3ctech.com/detail/2591)、[摩擦力匀减速运动](http://code.w3ctech.com/detail/2592)。如果再组合使用曲线，就能模拟出更多运动了，例如y轴使用二次曲线，x轴使用线性曲线，就模拟出一个[平抛动画](http://code.w3ctech.com/detail/2594)了。 月影大神分享过一个ppt，里面列举了一些匀加/减速时的二维运动的动画曲线及实现：[戳我戳我](http://matrix.h5jun.com/slide/show?id=117" \l "/)  2. Cubic – x^3:这是条三次方曲线，大家还记得初中物理哪儿用到这条曲线吗？。。。。对了，就是变加速直线运动，如下图：  IMG_256  3.elastic曲线：这个就是前面在研究的弹簧曲线。实现了和ios的spring动画相似的效果。  4.Bounce曲线：模拟小球落地效果的曲线。  5.通过用sin曲线设置物体的透明度，可以实现呼吸灯效果。  IMG_260 |
|  |  |
|  |  |
|  | 如何让你的动画更自然运动曲线探究与应用-大熊  <https://isux.tencent.com/native-animation.html> |
|  | 1.为什么需要探究更自然的动画  自css animation推出后，强大的功能使得我们通过css也能制作出媲美flash的动画效果。然而在制作动画的时候，我们也许会常常纠结怎么设置timing-function。一般情况下，我们会直接使用自带的五个动画函数（linear、ease、ease-in、ease-out、ease-in-out），或是在[cubic-bezier.com](http://cubic-bezier.com/)创建一些自定义的动画函数(cubic-bezier(n,n,n,n))。但往往这一切都只是局限于使用，而不知道其原理究竟是什么，以及没有背后的物理原理支撑，使得做出来的动画可能会变得有点形而上学。例如用ease-in来做小球从高处掉下的效果，这个加速效果没有遵循相关物理原理，使得出来的动画效果不太自然。  自然的动画效果应该是和我们在现实生活中看到的物体运动轨迹相似的。这样的效果往往与背后的运动曲线函数紧密联系在一起。如上面提到的小球从高处掉下效果，对应的是匀加速运动函数s1=0.5\*g\*t²。若再探讨之后受到空气阻力及接触面材质影响，回弹的高度s2=s1\*n(0<n<1，可以假定n=0.64)，如此循环下去，直至小球最后停在地上，这样就可以模拟出整个[小球掉下效果](http://code.w3ctech.com/detail/2606)。  现实生活中的运动效果丰富多样，只靠css3提供的几个基本动画函数是不足以模拟的，例如弹簧动画效果等。要模拟这些真实的效果，就要学会如何获得这些效果背后的动画函数了。 |
| 曲线相关的网站 | <http://cubic-bezier.com/#.17,.67,.83,.67>  贝塞尔曲线绘制    <http://easings.net/zh-cn>  缓动函数速查表，并使用方式    <http://zh.numberempire.com/graphingcalculator.php?>  函数图像绘制    <https://greensock.com/ease-visualizer>  GreenSock库中，还有[一些动画曲线](http://greensock.com/ease-visualizer)可供使用：  GreenSock还提供了一个[在线调节参数预览效果的页面](http://greensock.com/ease-visualizer)。  IMG_256 |
|  |  |
|  |  |
|  |  |
|  |  |
| sass实现复杂动画 | 使用css实现曲线动画效果  我们也可以把这些运动曲线运用到CSS Animation的@keyframes中。以下还是以让目标通过弹簧效果从x轴上400像素位置移动到0像素位置为例，使用Sass来做：  //引入函数库 https://github.com/terkel/mathsass，实现sin，cos等数学函数  @import "node\_modules/mathsass/dist/math";    //编写弹簧曲线函数  @function elasticAniFn($t) {  @return -0.5 \* pow(exp(1), (-6 \* $t)) \* (-2 \* pow(exp(1), (6 \* $t)) + sin(12 \* $t) + 2 \* cos(12 \* $t))  }    //编写物体位移随时间变化的函数  //$b: 初始值  //$c: 变化量  //$p: 当前运动的进度百分比  //可以理解为物体从$b运动到$c,$p用来表示当前运动了 $p%  @function aniFn($b, $c, $p) {  @return $b + $p \* ($c - $b);  }    //声明动画  //由此生成的css：  //@keyframes moveAni {  //0% {  // transform: translateX(400px);  //}  //1% {  // transform: translateX(396.54493px);  //}  //2% {  // transform: translateX(386.76446px);  //}  //3% {  // transform: translateX(371.53953px);  //}  @keyframes moveAni {  @for $i from 0 through 100 {  {$i}% {  transform:translateX(aniFn(400px, 0, elasticAniFn($i / 100)));  }  }  }    //使用动画  .box {  animation: 1s moveAni linear;  transform: rotate  } |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

demo

|  |  |
| --- | --- |
| 抛物线 | 小折腾：JavaScript与元素间的抛物线轨迹运动  [www.zhangxinxu.com/wordpress/2013/12/javascript-js-元素-抛物线-运动-动画/](http://www.zhangxinxu.com/wordpress/2013/12/javascript-js-元素-抛物线-运动-动画/) |
| 抛物线曲线是：y = a\*x\*x + b\*x + c  求解：a, b, c三个参数。  3个参数需要3个条件才能完全求解。  由于我们要实现元素A到元素B的抛物线运动，因此，我们可以指定两个点的坐标位置，也就是知道了2个条件，那第3个条件呢？  了解抛物线函数的同学应该知道，a可以近似理解为弧度，曲率。在我们实现效果的时候，这个是应该要可控的。——你总不可能把元素抛到天宫号上再掉下来吧。在网页上，一般都是起伏不大的抛物线（否则会出屏幕之外）。  因此，我们可以把a作为一个参数常量。  于是，我们任务就变成了，已知参数a，以及两个点的坐标，求该抛物线函数。  哈，事情就简单多了。  y = a\*x\*x + b\*x + c  ↓  y1 = a \* x1\*x1 + b\*x1 + c;  y2 = a \* x2\*x2 + b\*x2 + c;  ↓  a已知，求b, c  为了简化我们的计算求解，以及与我们高中时候的函数坐标匹配，我们可以以移动元素的初始位置作为坐标轴的中心（网页默认的坐标系左上角是中心，x轴向右，y轴向下，与高中的坐标轴不一样）。如下链接对应demo所示。  您可以狠狠地点击这里：[元素抛物线运动demo](http://www.zhangxinxu.com/study/201312/js-parabola.html" \t "http://www.zhangxinxu.com/wordpress/2013/12/javascript-js-%E5%85%83%E7%B4%A0-%E6%8A%9B%E7%89%A9%E7%BA%BF-%E8%BF%90%E5%8A%A8-%E5%8A%A8%E7%94%BB/_blank)  也就是，我们限定了抛物线经过中心点(0, 0), 代入y1 = a \* x1\*x1 + b\*x1 + c可以得到c = 0, 于是b = (y2+ a\*x2\*x2) / x2, 带入大便池元素的坐标，就可以计算出b的大小，于是，抛物线函数就出来了。  注意，这里“大便池元素的坐标”是相对于小球球的相对坐标，而不是页面左上角。  有了抛物线函数，我们就有了小球球运动的轨迹了。 |
| zhangxinxu tween.js | 每个动画效果都有其自身的算法。我们都知道jQuery UI中就有缓动，As脚本也内置了缓动，其中的运动算法都是一致的；  [GitHub访问](https://github.com/zhangxinxu/Tween/blob/master/tween.js" \t "http://www.zhangxinxu.com/wordpress/2013/09/css3-animation-requestanimationframe-tween-%E5%8A%A8%E7%94%BB%E7%AE%97%E6%B3%95/_blank)：  <https://github.com/zhangxinxu/Tween/blob/master/tween.js>  本地：    使用requestAnimationFrame与动画算法实现弹簧运动效果：  <https://code.w3ctech.com/detail/3611>  <http://www.zhangxinxu.com/study/201309/requestanimationframe-tween-easeoutbounce.html>  使用分析  <http://www.qdfuns.com/notes/27225/983a65df303c523ea1ccbb8a4e55c363.html> |
| 核心动画部分的脚本是：  funFall = function() {  var start = 0, during = 100;  var \_run = function() {  start++;  var top = Tween.Bounce.easeOut(start, objBall.top, 500 - objBall.top, during);  ball.css("top", top);  shadowWithBall(top); // 投影跟随小球的动  if (start < during) requestAnimationFrame(\_run);  };  \_run();  };  大部分的曲线动画都包含4个入参：  \* t:当前时间  \* b:初始位置  \* c: 结束位置  \* d:运动时间  我们主要关心的就是b、c、d，可以理解为物体用了d毫秒从b变成c。这是不是很像设置css动画时要关心的东西呢。而t是给程序获得当前时间，计算出此时间下对应的值。  有些动画函数，例如弹簧动画函数Elastic，还有a和p参数。经试验，a:影响振幅，p影响来回次数，按这儿关于ios弹簧动画的描述，a的设置相当于质量，而p相当于阻尼系数。 |
| <**style**>  #ball {  position: absolute;  width: 20px;  height: 20px;  border-radius: 20px;  background-color: red;  } </**style**> <**div** id="ball"></**div**> <**script** type="text/javascript">  **var Tween** = {  "Elastic": {  easeOut: **function** (t, b, c, d, a, p) {  **var** s;  **if** (t == 0) **return** b;  **if** ((t /= d) == 1) **return** b + c;  **if** (**typeof** p == "undefined") p = d \* .3;  **if** (!a || a < Math.abs(c)) {  a = c;  s = p / 4;  } **else** {  s = p / (2 \* Math.PI) \* Math.asin(c / a);  }  **return** (a \* Math.pow(2, -10 \* t) \* Math.sin((t \* d - s) \* (2 \* Math.PI) / p) + c + b);  }  }  };   **var ball** = document.getElementById("ball");  **var elasticFall** = **function** () {  **var** start = 0,  beginingValue = 0,  changeValue = 400,  during = 100;  **var \_run** = **function** () {  start++;  **var** top = **Tween**.Elastic.easeOut(start, beginingValue, changeValue, during);  **ball**.style.webkitTransform = "translateY(" + top + "px)";  **if** (start < during) requestAnimationFrame(**\_run**);  };  **\_run**();  };  **elasticFall**(); </**script**> |

velocity

|  |  |
| --- | --- |
| velocity | <http://blog.csdn.net/mlin_123/article/details/51814346>  <http://www.cnblogs.com/guandekuan/p/6643988.html> |
|  | 基础用法  Velocity专为动画而设计 ，简单易用，功能强大，广泛地被一些主流公司所使用（包括[Tumblr](https://www.tumblr.com/" \t "http://blog.csdn.net/mlin_123/article/details/_blank), [Microsoft](http://windows.com/" \t "http://blog.csdn.net/mlin_123/article/details/_blank) and [WhatsApp](https://www.whatsapp.com/" \t "http://blog.csdn.net/mlin_123/article/details/_blank)）。并且Velocity 是基于[MIT](http://en.wikipedia.org/wiki/MIT_License" \t "http://blog.csdn.net/mlin_123/article/details/_blank)许可协议的开源库。  Velocity模仿了jQuery的语法，可以完美地同jQuery协作，当然也能独立地使用，所以对你来说应该很容易学。由于jQuery使用普遍，我也将向你展示Velocity如何同它协作。 |
|  | <**script** src="./js/jquery-1.8.3.js"></**script**> <**script** src="./js/velocity.min.js"></**script**> <**style**>  .element {  width: 100px;  height: 100px;  border: 1px solid red;  } </**style**> <**div** class="element"></**div**> <**script**>  // 标准写法  // (为了简洁，后面的示例代码中 将直接使用 $element 来代替 jQuery 选择器)  $(".element").velocity({  width: "200px"  }, {  duration: 450,  delay: 1000  });   // $.animate() 的写法，效果同上  // $(".element").delay(1000).velocity({width: "200px"}, 450); </**script**> |
| Basics 基础用法 | 1. Dependencies 模块依赖  1-1. Velocity 不依赖 jQuery  Velocity.js 可以在不引入 jQuery 的情况下单独使用。如果 你需要大部分动画效果能兼容 IE8，就必须引入 jQuery 1×。 它也可以和 Zepto 一起使用，写法和 jQuery 一样：  JavaScript  // 无 jQuery 或 Zepto 时，Velocity()方法挂载在 window 对象上 (window.velocity)// ( 第一个参数为原生js的dom选择器 )  Velocity(document.getElementById("dummy"), {  opacity: 0.5  }, {  duration: 1000  });  // 使用 jQuery 或 Zepto 时  $("#dummy").velocity({  opacity: 0.5  }, {  duration: 1000  });  1-2. 模块加载器：Webpack, Browserify  使用 jQuery 时，必须在 Velocity 之前加载 jQuery：  JavaScript  window.jQuery = window.$ = require("path/to/jquery-x.x.x.js");require("path/to/velocity.js");require("path/to/velocity.ui.js"); // velocity.ui 在 velocity 之后加载  /\* Your app code here. \*/  $("body").velocity({ opacity: 0.5 });  不使用 jQuery 就直接加载 Velocity：  JavaScript  var Velocity = require("path/to/velocity.js");require("path/to/velocity.ui.js");  /\* Your app code here. \*/  Velocity(document.body, { opacity: 0.5 });  1-3. 模块加载器：RequireJS  JavaScript  require.config({  paths: {  "jquery": "/path/to/jquery-x.x.x",  "velocity": "path/to/velocity",  // 如果你使用了 velocity.ui  "velocity-ui": "path/to/velocity.ui"  },  shim: {  "velocity": {  deps: [ "jquery" ]  },  // velocity.ui 依赖 velocity  "velocity-ui": {  deps: [ "velocity" ]  }  }  });  require([ "jquery", "velocity", "velocity-ui" ], function ($, Velocity) {  /\* Your app code here. \*/  $("body").velocity({ opacity: 0.5 });  }); |
| 参数 | 2-1. 参数概述和基础写法：  第一个参数是是强制要求的,它可以是一个命令(就像上面例子中的"scroll"),或者是由CSS属性值组成的对象(这些值是动画的目标值)：  第二个参数是由动画附加选项组成的对象,比如过渡时长,张弛度,延时还有回调函数。  $element.velocity({ opacity: "50%", left: "500px"});  $element.velocity({ opacity: "50%" }, { delay: 200, duration :200 });  下面为参数的写法：  $element.velocity({  width: "500px", // 动画属性 宽度到 "500px" 的动画  property2: value2 // 属性示例  }, {  /\* Velocity 动画配置项的默认值 \*/  duration: 400, // 动画执行时间  easing: "swing", // 缓动效果  queue: "", // 队列  begin: undefined, // 动画开始时的回调函数  progress: undefined, // 动画执行中的回调函数（该函数会随着动画执行被不断触发）  complete: undefined, // 动画结束时的回调函数  display: undefined, // 动画结束时设置元素的 css display 属性  visibility: undefined, // 动画结束时设置元素的 css visibility 属性  loop: false, // 循环  delay: false, // 延迟  mobileHA: true // 移动端硬件加速（默认开启）  });  2-2. 单一对象的参数写法：  Velocity 也支持 single-argument 的语法，看下面示例：  JavaScript  $element.velocity({  properties: { opacity: 1 },  options: { duration: 500 }  });  // 或者：  $element.velocity({  p: { opacity: 1 }, // 可以将 properties 简写成 p  o: { duration: 500 }  });  2-3. 逗号分割的参数写法（类似 $.animate）：  另外作为一种可选的快捷方式,你可以以任何顺序以逗号隔开的形式传递时长,张弛度（easing）和回调函数中的一个或多个参数,然而如果你想再传入其他动画选项的话必须使用数据对象的形式,而且这俩种方式不能混合使用。对于这里小编举几个例子说明：  $element.velocity({ top: 50 }, 1000);  $element.velocity({ top: 50 }, 1000, "swing");  $element.velocity({ top: 50 }, "swing");  $element.velocity({ top: 50 }, 1000, function() { alert("Hi"); });  $element.velocity({left:"1000px"},200,"swing",function(){console.log("ok")});  //等价于  $element.velocity({left:"1000px"},{"duration":200,"easing":"swing","complete":function(){console.log("ok")})  //下面这个用法是错的,俩种风格不能混用  $element.velocity({ left: "50px" }, 500, "ease-in-out", function () { alert("done") }, { delay: 200 });  Note: 第二类参数是可选择的(原文是第二个参数,但从上面的例子可以看出可以是不止俩个参数,从参数分类上却恰好是俩类,所以小编这里将其译为第二类（properties和options俩类,在其官方文档中可以看出来）),因为对于那些重要的属性都有默认值,比如easing的默认值是swing ,duration的默认值是400ms。所有其他的动画选项都是可选的。  同时你只能为每一个CSS属性设置一个值,所以 "padding:'10px 15px'"这样的设置是不合法的。替代做法是单独设置每一个分属性：{paddingLeft:"10px",paddingTop:"15px",….}。这样不仅表达清晰,而且意味着你可以特别指定每一个css分属性的值,而不是只设置一个整体的,给予了你足够的空间定制动画。  Note: 有多个单词描述的CSS属性（比如font-size和padding-left）必须使用骆驼拼写法表示（fontSize && paddingLeft）. |
|  | Property values  如果CSS属性值没有给定确切的单位,那么时间默认是：ms,长度默认是 px,角度默认是 deg。为了表达清晰最好还是显示注明单位,那样以后再看代码时也容易理解。如果有一个值不仅仅由数字表示,那么必须加引号。比如：duration:500（默认是500ms）是合法的,但是duration:500ms就不合法,必须加上引号：duration:"500ms".  JS动画运许传入简单的表达式作为CSS的属性值,但这些表达式只限于：+=,-=,\*=,/=,表示目标值在其本来的值的基础上加多少,减多少,乘多少,除多少,任何其他的表达式是不允许的。实例：  $element.velocity({  width: "+= 50px", // Adds 50px to the current width value  eight: "/= 2" // divides the current height value by two  });  表示动画完成后width比原值大50px,height变为原来的二分之一。  在Velocity动画引擎里使用这些快速功能保证了动画的逻辑可读性,使代码简洁清晰（消除了值的手工计算）,而且向Velocity引擎提供更多有关于你动画意图的信息将有助于提高动画的性能。动画逻辑包含的越多,Velocity就能更好地提高代码的性能。 |
|  | Properties Map 动画属性  如果不写属性值的单位, Velocity 会将像素(px)作为默认单位  // 等同 padding: 1px  $element.velocity({ padding: 1 });  // 等同 padding-left: 1px, padding-right: 1px  $element.velocity({  paddingLeft: 1,  paddingRight: 1  });  // 但你不能这样写！因为这样相当于为 padding 赋了多个值  $element.velocity({ padding: "1 1 1 1" }); // error  Velocity 在 1.2.0+ 版本里增加了对 "px, em, rem, %, deg, vw/vh" 这些单位的支持， 如果不填写属性单位 默认单位还是"px"，但 "deg" 用于 rotateZ 属性时可以省略不写。 Velocity 还支持动态计算属性值，包括 "+, -, \*, /"，还可以设置元素在动画执行前的初始值，看下面示例： ( 注意："rem" 只支持 IE9+，"vh/vw" 只支持 IE9+ 和 Android 4.4+ )  $(".element").velocity({  top: 50, // 等同于 "50px"  left: "50%",  width: "+=5rem", // 每次在当前值上叠加 5rem  height: "\*=2", // 每次在当前值上叠乘 2  background: ["#888", "#000"], // 每次动画执行前，color 的初始值都为"#000"（从"#000"过渡成"#888"）  borderRadius:["10px","100px"] }); |
| 链式调用 | /\*\*  \* Chaining 链式动画  \* 当一个元素连续应用多个velocity()时，动画将以队列的方式执行  \*  \*  \* \*/ $element.velocity({width: 75})// 先执行宽度变为75px的动画  .velocity({height: 0});//等前面的宽度动画结束后，再执行高度变为0的动画 |
|  |  |

|  |  |  |
| --- | --- | --- |
| loop-重复 | //将宽度在500px与100px之间来回切换 $element.velocity({width: "500px"}, {  duration: 1000,  loop: true //1 2 }); | |
| progress-过程 | **$element**.velocity({  opacity: 0,  tween: 1000 // Optional }, {  progress: **function** (elements, complete, remaining, start, tweenValue) {  console.log((complete \* 100) + "%");  console.log(remaining + "ms remaining!");  console.log("The current tween value is " + tweenValue)  } }); | |
| queue-是否加入队列，或者直接执行 | **$element**.velocity({  width: 50 }, {  duration: 3000 }); **$element**.velocity({  height: 50 }, {  duration: 1500 }); | **$element**.velocity({  width: 50 }, {  duration: 3000 }); **$element**.velocity({  height: 50 }, {  queue: **false**,  duration: 1500 }); |
| 动画1执行完毕后，才会执行动画2 | 动画2的执行会紧接着动画1 |
| easing-弹性动画 | Physics-Based Motion  现实世界的运动模型既不是线性的也不是平滑的。它有快速的部分，有慢的部分，可能存在摩擦，甚至需要考虑重量。使用JS可以很好地模拟这些运动，并且看起来非常的自然。而线性的运动效果看起来则非常机械，不怎么中看。  为了方便模拟现实世界的运动，Velocity提供弹性动画功能（[spring](http://lib.csdn.net/base/javaee" \o "Java EE知识库" \t "http://blog.csdn.net/mlin_123/article/details/_blank)），以一个张弛度（默认500）和一个摩擦系数（默认值20）作为他的参数（参考<http://velocityjs.org/>获取更多信息）  Example:  $element.velocity({ width: "100px"}, [ 1000, 40 ]);  $element.velocity({width: "1000px"}, {  duration:"1000",  easing: [1000, 40] });  较高的张弛度将提高整体速度和动画的反弹力度，较小的摩擦系数将提高动画结束时的速度。（较大的摩擦系数将使动画快速减速）。调整这些值会让每个动画独一无二，逼真并且有趣。（具体操作一下感觉更直观）  上面那个示例运行起来应该非常快，非常灵动（因为设置了较大的张弛度），并且将快速减速，在接近动画末尾的时候速度将变得很慢。（因为设置了较大的摩擦系数）。  PS:小编在这里解释一下，spring是动画选项中easing的一个可选值，至于easing是什么在后面有介绍。这里仅仅解释一下spring到底是 什么玩意儿。准确的讲spring对应一个JS函数，而这个函数描述的是一个运动模型，什么模型呢？答案是弹簧振子模型 ，想必大家初中的时候就做过这种物理题了。spring的俩个参数---张弛度和摩擦系数，其实就对应弹簧的初始弹性势能和地面的摩擦系数。 大家自己都可以把这个模型写出来。 | |
|  | /\* Use one of the jQuery UI easings. \*/ $element.velocity({ width: 50 }, "easeInSine");  /\* Use a custom bezier curve. \*/ $element.velocity({ width: 50 }, [ 0.17, 0.67, 0.83, 0.67 ]);  /\* Use spring physics. \*/ $element.velocity({ width: 50 }, [ 250, 15 ]);  /\* Use step easing. \*/ $element.velocity({ width: 50 }, [ 8 ]); | |
| Begin & Complete |  | |

命令

|  |  |
| --- | --- |
| command:scroll | 现在有一个流行的Web设计，即在一个长页面中有序放置所有内容，只需轻微滑动一下鼠标，窗口就会平滑地滚动到恰当的页面区域，而不是像传统一样导航到另一个不同的页面。这是JS动画最流行的一种应用，并且无法用CSS实现。  如早些说过的一样 ，scroll命令将促使页面滚动到指定元素的上边缘，你可以向其他动画一样传递同样的附加选项，并且同样支持链式调用。  只需在目标元素上调用velocity()方法并传入"scroll"命令和过渡时间：  这个方法将使浏览器滚动到选定的元素（jQuery对象,$element）,过渡时间为1000ms。仅使用jQuery实现这个功能也会很复杂，而且将用到多个函数操作。  $element.velocity("scroll", 1000);  // 链式调用：  $element  .velocity("scroll", { duration: 1000 })  .velocity({ opacity: 1 });  // 默认的滚动沿y轴方向，想改到x轴方向的话可以使用axis选项：  $element.velocity("scroll", { axis: "x" });  Velocity允许你设置窗口滚动后元素上边缘距窗口上边缘的距离，你只需设置 offset选项即可完成：  // 滚动之后窗口上边缘将位于元素上边缘之上100px的地方  $element.velocity("scroll", { duration: 1000, offset: "-100px" });  // 滚动之后窗口上边缘将位于元素上边缘之下100px的地方  $element.velocity("scroll", { duration: 1000, offset: "100px" }); |
|  | <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3** id="test">test</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**h3**>111</**h3**> <**script**>  $("#test").velocity("scroll",{  container: $("body"),  duration: 800,  delay: 500  }) </**script**>  <**script**>  **/\* jquery.js \*/  /\* jquery.velocity.js \*/** // Use scroll with the container option.  // Note: When you want to scroll the browser window itself, do not pass in a container.  $("#element3").velocity("scroll", {  container: $("#container"),  duration: 800,  delay: 500  }); </**script**> |
|  |  |
| command:reverse | 在jQuery中，想要使元素恢复成动画运行之前的状态，你必须手动的设置元素原来的属性值：  $element.animate({ width: "100px" }, 400);  $element.animate({ width: "50px" }, 400);// 又使元素在400ms内恢复成原来50px的宽度  相较地，在Velocity里只是一个运行"reversr"命令:  $element.velocity({ width: "100px" }, 400);  $element.velocity("reverse");//这条命令将使$element回到运行前的状态，而你不用手动的去设置目标 width; |
| **$element**.velocity({  width: "1000px",  backgroundColor: "#ff0000" }, {  duration: "1000",  easing: [1000, 40] }); **$element**.on("click", **function** () {  **$element**.velocity("reverse", {duration: 2000}); }); |
| Command:Stop | **var $element** = $(".element"); **$element**.velocity({  width: "500px" }, {  duration: 2000 }); setTimeout(**function** () {  **$element**.velocity("stop")  .velocity("reverse"); }, 1000); |
| command:finish | **var $element** = $(".element"); **$element**.velocity({  width: "500px" }, {  duration: 2000 }); setTimeout(**function** () {  **$element**.velocity("finish"); }, 1000); |
| feature:transform | Transforms  想要结合CSS和JS设计动画？ 设置一些CSS变换规则，允许你做一些2D或3D的动画，比如平移，扩大，旋转。注意这些变换不会影响元素在网页中的位置，也不会影响该元素周围 的元素在页面中的位置。 Velocity支持下面的变换：  translateX: 从左向右沿x轴移动元素  translateY: 从上到下沿y轴移动元素  rotateZ: 关于z轴旋转元素  rotateX: 关于x轴旋转元素（看起来由里向外）  rotateY: 关于y轴旋转元素（由左到右）  scaleX: 成倍数改变元素宽度  scaleY: 成倍数改变元素高度  $element.velocity({  rotateZ: "90deg", // rotate clockwise 90 degrees  scaleX: 2.0 // double the width  }); |
| feature:color | Velocity 允许制作基于颜色变换的动画，比如:color, backgroundColor, borderColor还有outlineColor.所有这些颜色的值只允许是16进制的字符串(黑色：#000000 , 脸谱蓝：#3b5998(for Facebook blue).也许你想使用rgb格式的颜色设置,甚至指定颜色透明度(值介于0到1)，只需简单地在CSS颜色名后面加上“Red”，“Green”,”Blue” 或者“Alpha”即可：  $element.velocity({  borderColor: "#f06d06",  backgroundColorAlpha: 0.8,//背景颜色透明度变到80%  colorBlue: 200//rgb 方式  }); |
| feature: Promise | $.Velocity.animate(**$element**, {  width: "1000px",  backgroundColor: "#ff0000" }, {  duration: "1000",  easing: [1000, 40]- }).then(**function** (ele) {  console.log(ele); }).catch(**function** (err) {  console.log(err); }); |
| Feature: Mock |  |

tween.js可生成平滑动画效果的js动画库

|  |  |
| --- | --- |
|  |  |
|  | 使用案例  <http://www.htmleaf.com/jQuery/Layout-Interface/201501271284.html> |
|  | tween.js是一款可生成平滑动画效果的js动画库。  相关的动画库插件还有：[snabbt.js 强大的jQuery动画库插件](http://www.htmleaf.com/jQuery/Image-Effects/201501061112.html" \t "http://www.htmleaf.com/jQuery/Layout-Interface/_blank)和[Tweene-超级强大的jQuery动画代理插件](http://www.htmleaf.com/jQuery/Layout-Interface/201501081124.html" \t "http://www.htmleaf.com/jQuery/Layout-Interface/_blank)。  tween.js允许你以平滑的方式修改元素的属性值。你只需要告诉tween你想修改什么值，以及动画结束时它的最终值是什么，动画花费多少时间等信息，tween引擎就可以计算从开始动画点到结束动画点之间值，来产生平滑的动画效果。例如，假设你有一个对象position，它的坐标为 x 和 y：   |  | | --- | | var position = { x: 100, y: 0 } |   如果你想改变 x 的值从100到200，你只需要这样做：   |  | | --- | | // Create a tween for position first  var tween = new TWEEN.Tween(position);    // Then tell the tween we want to animate the x property over 1000 milliseconds  tween.to({ x: 200 }, 1000); |   到这里只是创建了tween对象，你需要激活它，让它开始动画：   |  | | --- | | // And set it to start  tween.start(); |   最后为了平滑动画效果，你需要在同一个循环动画中调用TWEEN.update方法。代码如下：   |  | | --- | | animate();    function animate() {      requestAnimationFrame(animate);      // [...]      TWEEN.update();      // [...]  } |   这个动作将会更新所有被激活的tweens，在1秒钟（例如1000ms）position.x 将变为200。  你也可以使用onUpdate回调函数将结果打印到控制台上。   |  | | --- | | tween.onUpdate(function() {      console.log(this.x);  }); |   这个函数在每次tweens被更新时都被调用。它的出现频次依赖于很多因素-例如：依赖于你的电脑或设备的运行速度。 |
|  | Tween.js本身不会运行，你需要通过update方法明确的告诉它什么时候开始运行。推荐在动画主循环中使用该方法。你可以通过调用requestAnimationFrame方法来获得良好的图像性能。  来看下面的例子：   |  | | --- | | animate();    function animate() {      requestAnimationFrame(animate);      // [...]      TWEEN.update();      // [...]  } |   这里使用无参数调用方式，update方法将明确当前时间，以便于获取上一次动画的执行时间。  你也可以为update方法明确一个时间：   |  | | --- | | TWEEN.update(100); |   上面语句的意思是说：update的时间=100毫秒。你可以使用这种方法来明确代码中所有随时间变化的函数。例如，动画已经开始，你想所有动画都同步进行，你的animate代码一改像这样：   |  | | --- | | var currentTime = player.currentTime;  TWEEN.update(currentTime); | |
|  | start和stop  Tween.start和Tween.stop分别用于控制tween动画的开始和结束。  对于已经结束和没有开始的动画，Tween.stop方法不起作用。Tween.start方法同样接收一个时间参数。如果你使用了该参数，tween动画将在延时该时间数后才开始动画。否则它将立刻开始动画。  update  可以通过TWEEN.update方法来执行动画的更新。  chain  如果你想制作多个多行，例如：一个动画在另一个动画结束后开始。可以通过chain方法来使实现。如下的代码，tweenB 在 tweenA 之后开始动画:   |  | | --- | | tweenA.chain(tweenB); |   可以像下面这样制作一个无限循环的动画：   |  | | --- | | tweenA.chain(tweenB);  tweenB.chain(tweenA); |   repeat  如果你想制作循环动画可以使用chain来实现，但是更好的方法是使用repeat方法。它接收一个用于描述你想循环多少次的参数：   |  | | --- | | tween.repeat(10); // repeats 10 times and stops  tween.repeat(Infinity); // repeats forever |   yoyo  这个函数只在你使用repeat方法是起作用。当它被激活时，tween 的效果类似yoyo效果。该效果是动画会在开始或结束处向反方向反弹。  delay  delay方法用于控制动画之间的延时。   |  | | --- | | tween.delay(1000);  tween.start(); | |
|  | 全局方法  以下的方法定义在 TWEEN 的全局对象中，其中大多数方法你都用不上，除了update方法：  TWEEN.update(time)  该方法用于所有被激活的tweens，如果time没有被指定，将使用当前时间。  TWEEN.getAll 和 TWEEN.removeAll  这两个方法用于胡获取被激活的tweens数组的一个引用，或从数组中删除所有tweens。  TWEEN.add(tween) 和 TWEEN.remove(tween)  用于向被激活的tweens中添加一个tween，或移除一个tween。  以上方法通常只是在内部使用，一般情况下你了解即可。  可用的easing函数：TWEEN.Easing  tween.js提供了一些可用的easing函数。可用函数有：Linear, Quadratic, Cubic, Quartic, Quintic, Sinusoidal, Exponential, Circular, Elastic, Back 和 Bounce。easing 类型分为: In, Out 和 InOut.  使用自定义的Easing函数  你不但可以使用tween.js提供的easing函数，还可以自定义easing函数。但必须遵守下面的规则：  它必须接收一个参数。  它必须基于输入参数返回一个值。  easing函数仅在每个tween每次被更新时调用，而不管有多少属性被改变。结果随后会被用于初始值：   |  | | --- | | easedElapsed = easing(k);  for each property:      newPropertyValue = initialPropertyValue + propertyDelta \* easedElapsed; |   下面是一个使用Math.floor来做easing效果的例子：   |  | | --- | | function tenStepEasing(k) {      return Math.floor(k \* 10) / 10;  } |   你可以在tween 这样使用它：   |  | | --- | | tween.easing(tenStepEasing); | |
|  | 回调函数  另外一个有用的特性是你可以在每次tween循环周期的指定时间点调用自定义的函数。  例如：假设你想使一些不能直接修改参数的对象执行动画，要访问该对象的参数只能通过setter方法，你可以通过update方法的回调函数来设置新的setter值。   |  | | --- | | var trickyObjTween = new TWEEN.Tween({      propertyA: trickyObj.getPropertyA(),      propertyB: trickyObj.getPropertyB()  })      .to({ propertyA: 100, propertyB: 200 })      .onUpdate(function() {          this.setA( this.propertyA );          this.setB( this.propertyB );      }); |   或者如果你想确认tween动画开始后某个对象指定状态下的某个值，你可以通过start回调来获取它：   |  | | --- | | var tween = new TWEEN.Tween(obj)      .to({ x: 100 })      .onStart(function() {          this.x = 0;      }); |   onStart  tween开始动画前的回调函数。  onStop  tween结束动画后的回调函数。  onUpdate  在tween每次被更新后执行。  onComplete  在tween动画全部结束后执行。  更多关于tween的资料请参考：[https://github.com/sole/tween.js/blob/master/docs/user\_guide.md](https://github.com/sole/tween.js/blob/master/docs/user_guide.md" \t "http://www.htmleaf.com/jQuery/Layout-Interface/_blank)  tween的相关资料：[Tween.js for Smooth Animation](http://learningthreejs.com/blog/2011/08/17/tweenjs-for-smooth-animation/" \t "http://www.htmleaf.com/jQuery/Layout-Interface/_blank) |

|  |  |
| --- | --- |
| Tweene | 现在已经有许多的javascript动画库插件，如[snabbt.js](http://www.htmleaf.com/jQuery/Image-Effects/201501061112.html" \t "http://www.htmleaf.com/jQuery/Layout-Interface/_blank)，它们每一个都有自己的优点和不足。每个程序员和每个项目都有其特定的要求，所以有时一个动画库可能不能满足我们的需要。我们并不建议你放弃那些已经存在的动画库而去自己编写一个新的动画库，即使你有这样的能力。  Tweene能够帮你解决这个难题。  Tweene是一个动画代理：它可以包装你选择的不同的动画库。它能够使你在这些动画库的基础上做更多的事情，并且他可以很容易的随时从一个动画库切换到另一个动画库。目前Tweene可以切换以下的动画库：[GSAP](http://www.greensock.com/gsap-js/" \t "http://www.htmleaf.com/jQuery/Layout-Interface/_blank)、[Velocity.js](http://julian.com/research/velocity/" \t "http://www.htmleaf.com/jQuery/Layout-Interface/_blank)、[Transit](http://ricostacruz.com/jquery.transit/" \t "http://www.htmleaf.com/jQuery/Layout-Interface/_blank)(CSS transitions)和[jQuery](http://jquery.com/" \t "http://www.htmleaf.com/jQuery/Layout-Interface/_blank)。  <http://tweene.com/html/libraries/>  [Tweene-超级强大的jQuery动画代理插件](http://www.htmleaf.com/jQuery/Layout-Interface/201501081124.html" \t "http://www.htmleaf.com/jQuery/Layout-Interface/_blank)  <http://www.htmleaf.com/jQuery/Layout-Interface/201501081124.html> |
|  |  |
|  |  |
|  |  |
|  |  |