|  |
| --- |
| CSS动画的原理 Firefox和Webkit系列的浏览器都已经支持CSS 动画，是时候动手学习这个新的技术了。 不管动画以何种技术载体呈现，无论是传统的手绘动画，还是电脑生成的3D动画，也无论是使用Flash技术还是CSS3， 制作动画的基本原理和设计准则都未曾改变。 在这篇文章中，我们会先解释CSS动画的基本概念，并且介绍使用CSS创建动画的要点。我们将以一个网球下落的例子贯穿始终，利用传统动画的准则来逐步构建这个CSS动画。最后我们会看一些真实世界里的CSS动画用法。  CSS 动画属性  在开始前，我们需要了解CSS 动画的基础知识。  动画是一种新的CSS 特性，它可以在不借助JavaScript和Flash的情况下使绝大多数HTML元素动起来（例如div，h1，或者Span等等） 。  现在它已经被Webkit家族的浏览器（包括Safari 4+， Safari for iOS(iOS2+),Chrome1+ ） 以及Firefox 5 所支持。不支持这些属性的浏览器则会直接忽略它们，所以使用它们时最好确定你的页面不会完全依赖这些动画属性（译者：或者你可以使用一些代码来检查用户的浏览器是否支持css动画特性 https://developer.mozilla.org/en/CSS/CSS\_animations/Detecting\_CSS\_animation\_support）  目前这一属性还比较新，所以各家浏览器厂商在自己支持的属性前都加上了前缀 如-moz- （firefox）, -webkit- （Safari 和Chrome）在本文的示例代码中为简洁起见，我们只使用-webkit- 前缀。  CSS动画的声明就是将一段描述动画的样式附加到目标元素上，   1. /\* 这是一段动画代码\*/ 2. @-webkit-keyframes example { 3. from { transform: scale(2.0); } 4. to   { transform: scale(1.0); } 5. } 7. /\* 这是动画即将应用的目标元素 \*/ 8. div { 9. -webkit-animation-name: example; 10. -webkit-animation-duration: 1s; 11. -webkit-animation-timing-function: ease; /\* ease is the default \*/ 12. -webkit-animation-delay: 1s;             /\* 0 is the default \*/ 13. -webkit-animation-iteration-count: 2;    /\* 1 is the default \*/ 14. -webkit-animation-direction: alternate;  /\* normal is the default \*/ 15. }   这段代码中，第一部分是描述动画的样式，名字叫“example"，它可以出现在样式表中的任何位置，或前或后。在描述元素的类中可以使用animation-name 属性来指定要使用哪个动画效果。  跟其他的样式写法类似的，上面例子中的几个动画属性都可以简写成如下格式：   1. div { 2. -webkit-animation: example 1s ease 1s 2 alternate; 3. }   甚至你也可以把所有的值都省略掉，这时浏览器会直接使用默认值。 好了，这些就是基本概念了，应该挺简单的吧？  传统动画的基本准则  迪士尼 - 传动动画的鼻祖和大师- 很早就提出了制作动画的12条准则，并出版了著作：The Illusion of life 来阐述这些准则。 这些基本的准则可以应用到所有的动画中去，实际上你不需要是个动画专家也可以学着遵循这些准则。 我们会试着遵循这12条“动画军规”，并使用CSS来实现更逼真可信的动画效果。  挤压与拉伸（Squash and Stretch)  如果物体在运动过程中它本身的形体不发生任何变化，那会显得的非常僵硬。上图演示是一个夸张的弹跳的网球的草图。当网球从高处落下并撞击地面时，你会看到球被压扁了，而后它再次离地弹起时，又被拉长了。   球被压扁和拉伸的过程能够让人感觉到网球的弹性，想象一下如果是一个又重又硬的保龄球掉落下来，肯定不会有这样的效果， 那肯定要把地板砸出个窟窿。 实现上面草图中网球球落地压扁再拉伸的效果，我们只要应用CSS3中的transform属性就能实现：   1. @-webkit-keyframes example { 2. 0% { -webkit-transform: scaleY(1.0); } 3. 50% { -webkit-transform: scaleY(1.2); } 4. 100% { -webkit-transform: scaleY(1.0); } 5. }   上述代码会在动画的不同阶段把页面对象按照纵向（ｙ轴）　拉伸1.2倍再变回原样。  这段代码中我们使用了百分比的方式来指定了变形在动画过程中发生具体时间（也叫关键帧）。对于简单的动画，你可以直接使用from和to来指定开始和结束的动画状态。使用百分比，则可以更加精细的控制动画播放的关键过程。   接下来我们可以使用translate 属性来让物体动起来。我们可以把transform和translate合并使用：   1. 50% { 2. -webkit-transform: translateY(-300px) scaleY(1.2); 3. }   使用translate属性可以在不改变对象其他属性（例如位置，高度或宽度）的前提下操纵对象的位置，因此该属性常用于CSS动画制作。上述例子代码，指定了球在动画播放中点（50%）处的样子，使它看起来像是正在反弹离开地面。    注意：要看到我们的例子运行时的效果，你需要使用最新的FF，Chrome或Safari， 在本文写作时，Safari提供了最好的动画体验）  [查看示例](http://dl.dropbox.com/u/12132497/12-principles/1.html)  好吧，我得承认到目前为止，这个动画还是挺垃圾的。但是我们已经向“逼真”这个目标迈进了第一步！  预期（Aniticipation）  预期，就是在人物出场或动作开始前的先兆，它可以增加悬念。例如，当一个人屈腿下蹲，就会让观众联想到他马上将要起跳了。在我们的掉落小球的动画里，如果我们在球出现之前，加上一个球的阴影，那么无疑是在为观众构造了一个悬念：马上要有一个球掉下来了！  [查看示例](http://dl.dropbox.com/u/12132497/12-principles/2.html)  我们使用了一个新的DIV元素来表示阴影，这样我们就可以在动画中分别控制阴影和小球。 为了制造这种球即将下落的悬念，我们故意在动画开始时增加一点延时，让小球不是立即掉下来。 只要稍稍调整一下我们动画设置的百分比的位置即可：  **[html]** [view plain](http://blog.csdn.net/dojotoolkit/article/details/6859754" \o "view plain)[copy](http://blog.csdn.net/dojotoolkit/article/details/6859754" \o "copy)   1. @-webkit-keyframes example { 2. 0% { -webkit-transform: translateY(-300px) scaleY(1.2); } 3. 35% { -webkit-transform: translateY(-300px) scaleY(1.2); } /\* 和开始时同样的位置，即让球等待一会儿再下落 0% \*/ 4. 65% { -webkit-transform: translateY(0px) scaleY(1.2); }    /\* 当动画播放到65%时，球再开始下落 \*/ 5. 67% { -webkit-transform: translateY(10px) scaleY(0.8); } 6. 85% { -webkit-transform: translateY(-100px) scaleY(1.2); } 7. 100% { -webkit-transform: translateY(0px); } 8. }   在动画35%的关键帧设置中，球的位置没有变化，而到65%时，球才完全落地。 除了使用这种方法，你还可以用 animate-delay属性来设置动画延迟以达到增加悬念的效果   1. div { 2. -webkit-animation-delay: 1s; 3. }   舞台布局（Staging）  迪士尼动画电影里绚丽多彩的背景是表现人物的重要手段！因此我们也需要为我们的动画创造一个舞台。  舞台同时也是帮助动画观众集中注意力的方法，就像在一个剧院中舞台上的聚光灯，会牵引观众的视线。 在我们的小球动画中，我已经加入了一个简单的背景，这个背景使得观众能够知道动画即将发生在页面的中央。  顺序动画和关键帧填充（Straight-Ahead vs. Pose to Pose）  传统动画中，有两种构建动画的手段。第一种是顺序动画，即从动画的第一帧开始绘制按照时间顺序一帧一帧的完成整个动画。第二种方法称为"Pose to Pose", 直译为从姿势到姿势。即先描绘出动画过程中若干关键帧的内容，然后再去填充关键帧之间的部分从而逐步完成整个动画。 填充关键帧之间内容的过程称为”in-betweening,” or “tweening,” 如果你制作过Flash动画，应该对这些术语不会陌生。   对CSS动画而言，我们一般使用关键帧填充的方式来完成动画。我们先描述动画过程中的关键帧，而后浏览器会自动帮我们完成填充。 不过，有时候我们也需要借助顺序动画的方法来达到精细控制动画效果的目的，浏览器能够自动实现的效果总是有限的。  动作跟随和重叠（Follow-Through and Overlapping）  动作跟随的本质是物理学，要让动画逼真就得让其中的物体和人物的行为符合物理定律。动作跟随和动作重叠常常会用在人物动画中，尤其是肢体的运动，例如当人物把手臂或长发放下时，手臂或头发会在放下后继续前后摇摆一阵。再设想一下一个有啤酒肚的人物（比如史莱克）快速转身时的情形： 他的身体会先转过来，而他突出的肚子则会比身体转动的要慢一些。   对我们的例子而言，就是要让球下落时更符合物理定律。而我们之前展示的动画的小球下落时还不够自然。我们希望看到球可以自由落体式的下落然后再弹起来。 不过，且让我们继续看下一条军规。  慢进慢出 （Slow  in and Slow Out）  这条规则讲得是物体的运动都是渐变的，加速和减速都需要过程。设想一辆高速行驶的汽车需要停下，如果画面中的汽车突然停下，那肯定让观众觉得不真实。大家都知道汽车停下来会经过减速的过程。因此要做到逼真，就必须模拟汽车先刹车减速，并最终慢慢停下的过程。  这条规则和重力的效果也有关系。设想一个正在荡秋千的小孩子。当他从低荡到最高点时，他在逐渐减速，由高处再荡回来的过程又是逐渐加速的过程，他的将在秋千的最低点达到最快的速度。 然后他再此经历减速过程荡向反方向的高点。  **[html]** [view plain](http://blog.csdn.net/dojotoolkit/article/details/6859754" \o "view plain)[copy](http://blog.csdn.net/dojotoolkit/article/details/6859754" \o "copy)   1. **<img** src="http://hi.csdn.net/attachment/201110/10/0\_13182499848iN3.gif" alt=""**>**   把这一准则应用到我们的例子中，通过调整小球进入画面和弹出画面的速度，我们可以让观众感到更加逼真的效果：当小球落到地面，并反弹起来的过程中，反弹的越接近高点它的速度就越慢。  在css中我们可以使用animation-timing-function 属性来配置动画随时间变化的速率：  **[html]** [view plain](http://blog.csdn.net/dojotoolkit/article/details/6859754" \o "view plain)[copy](http://blog.csdn.net/dojotoolkit/article/details/6859754" \o "copy)   1. -webkit-animation-timing-function: ease-out;   这一属性有４个可选值：  ease-in ：开始时慢，并逐渐加速 ease-out： 开始时快，并逐渐减速停下 ease-in-out： 开始慢并加速，到中间时最快，然后又逐渐减速并停下 linear　：匀速播放  你还可以使用bezier-curve 函数来创建自己的速率函数（easing speed）  弧形运行(Arc) 和跟随准则一样，弧度准则也是基于对基本物理定律的观察：”弹起来的物体最终总会落下“， 考虑物体运行轨迹时，恰当运用弧线。  设想我们把小球从画面的左侧掷向舞台。　逼真的动画应该能够模拟出它沿着弧线（抛物线）落下，并弹起，再次沿弧线落下的过程。   使用CSS制作这样一个动画需要一些更加精细的控制。我们需要同时控制小球在纵向和横向两个方向上的运行。　因此我们实际上需要让小球一面平滑的自左向右运动，再结合上之前我们已经制作出的自由下落的动画。与其用一个复杂的动画来描述这两个方向的运动，我们可以直接使用２个动画来实现，更加简单。为了达到这一目的，我们需要在我们的小球之外再添加一个div并为它单独配置动画效果（横向运动）  HTML 代码：  **[html]** [view plain](http://blog.csdn.net/dojotoolkit/article/details/6859754" \o "view plain)[copy](http://blog.csdn.net/dojotoolkit/article/details/6859754" \o "copy)   1. **<div** class="ball-arc"**>** 2. **<div** class="ball"**></div>** 3. **</div>**   CSS代码：  **[html]** [view plain](http://blog.csdn.net/dojotoolkit/article/details/6859754" \o "view plain)[copy](http://blog.csdn.net/dojotoolkit/article/details/6859754" \o "copy)   1. .ball-arc { 2. -webkit-animation: ball-x 2.5s cubic-bezier(0, 0, 0.35, 1); 3. } 4. /\* cubic-bezier 函数的使用是用来调节动画运行的速度，这里我们使球慢慢减速\*/ 6. @-webkit-keyframes ball-x { 7. 0% { -webkit-transform: translateX(-275px); } 8. 100% { -webkit-transform: translateX(0px); } 9. }   上述代码可以看到，我们新加了一个动画ball-x 它作用在外层div上，负责小球的横向运动，而ball-y则作用在内层div上负责纵向运动。   当然，这样将动作拆分为横向和纵向的写法有它的缺点，当你真的要创作一些非常复杂的动画时，这些代码在语义上比较难以理解。  [查看示例](http://dl.dropbox.com/u/12132497/12-principles/7.html)  次要动作（Secondary Action)  要使动画逼真必须关注细节，除了描绘动画角色的主要动作外，需要通过对次要动作的描绘进一步增加动画的真实感。正如谚语所说“恶魔藏在细节里” 。例如，如果要刻画一个长发飘飘的姑娘在走路时，人物行走是主要动作，而次要动作则是她那跟随步伐摇摆弹跳的长发，又或是她随风摆动的裙角。 通过描绘这些细节（次要动作），能够让观众进一步感到动画的逼真。  当然在我们的小球动画中，要比这简单多了。要给小球的下落增加一些细节，我们可以选择让小球在运动中旋转。那会创造一种小球被人投掷到舞台上的感觉。   这次我们不用额外添加一个div来实现这一效果，我们可以把这个旋转的动画直接应用到用来显示小球纹理的img元素上  **[html]** [view plain](http://blog.csdn.net/dojotoolkit/article/details/6865222" \o "view plain)[copy](http://blog.csdn.net/dojotoolkit/article/details/6865222" \o "copy)   1. .ball img { 2. -webkit-animation: spin 2.5s; 3. } 5. @-webkit-keyframes spin { 6. 0% { -webkit-transform: rotate(-180deg); } 7. 100% { -webkit-transform: rotate(360deg); } 8. }   作者： Tom Waterhouse 译者： feijia  (tiimfei@gmail.com)  恰当的节奏（Timing ）  这一点很好理解，动画中的物体的运动的速度，节奏，直接关系到他们看起来是不是逼真。  我们的小球动画就是一个生动的例子。 我们目前设定的小球运动的节奏使它看起来比较符合一个质量很轻的网球被抛掷的速度，试想如果我们是在掷一个沉重的保龄球，那么球下落的一定会比较快。 （译者：作者原话，纠结于“两个铁球同时落地“的同学，我想他的意思是，如果是一个弹性很小的保龄球，它落地后再反弹的高度会短很多）同样，如果我们让动画再慢一些，那同样会让它看起来不真实，好似是在太空中打网球了。    通过调整animation-duration属性，你可以调整动画运行的整体时间从而调整动画运动的节奏，同样你也可以通过设置百分比的方法来进行微调。  夸张（Exaggeration） 虽然前面我们说了种种通过模拟物理规律来让动画逼真的方法，但是卡通形象当然也不必处处遵循物理规律，恰到好处的夸张可以增加效果。 比如卡通形象可以变形成任何形状，并最后再变回来。夸张的效果可以使一个动作看起来不那么平淡，也可以起到强调的作用。  掌握何时使用夸张的火候很有技巧，迪斯尼动画有一条规则说：在动画中大部分时候尽量模拟真实的基础上再并稍稍使它超出一点。好像一个动画人物跑向一堵墙，他的身体会被挤压到墙上（当然现实世界不会发生这样的情形），这种挤压就是那部分微微夸张的效果。   我们使用了夸张这一技巧（挤压和拉伸）来强调球在落地时的效果。我还额外添加了一个微小的颤动效果。最后，我们还加入了球在弹起后加入了拉伸效果来表达它的速度感。  向前几次一样，为了加入颤动，我又加了一个新的div，它将会随着球落地而抖动。示例代码如下：  **[html]** [view plain](http://blog.csdn.net/dojotoolkit/article/details/6865222" \o "view plain)[copy](http://blog.csdn.net/dojotoolkit/article/details/6865222" \o "copy)   1. @-webkit-keyframes wobble { 3. 0%, 24%, 54%, 74%, 86%, 96%, 100% { 4. -webkit-transform: scaleX(1.0); 5. /\* 在这些位置，球将会回到正常形状 \*/ 6. } 8. 25%, 55%, 75% { 9. -webkit-transform: scaleX(1.3) scaleY(0.8) translateY(10px); 10. /\* 球落地的位置：压缩效果（宽度增加，高度变小） \*/ 11. } 13. 30%, 60%, 80% { 14. -webkit-transform: scaleX(0.8) scaleY(1.2); 15. /\* 球落地后的几个收缩的点 \*/ 16. } 18. 75%, 87% { 19. -webkit-transform: scaleX(1.2); 20. /\* Subtler squash for the last few bounces \*/ 21. } 23. 97% -webkit-transform: scaleX(1.1); 24. /\* Even subtler squash for last bounce \*/ 25. } 27. }   代码看起来有些复杂，不过不用担心，通过不断调整直到你觉得达到了你设想的效果。　 作者： Tom Waterhouse 译者： feijia  (tiimfei@gmail.com) 扎实的绘画和人物的魅力（Solid Drawing and Appeal）  我能教你的就这么多了，至少没有更多的示例代码了。最后两条准则没办法CSS代码演示。　可它们是你必须掌握的使动画臻于完美的技巧。　  当迪斯尼在制作白雪公主动画时，它把动画师们派回重新去上绘画课，重新理解人类的动作。这种对细节的关注直接提现在它的作品中，优秀的动画必须有扎实的绘画功底，以及对你所描绘事物的细致理解（也许是一个网球，也许是一个人物）  虽然大多数ＣＳＳ动画可能不会像一个人物动画那么复杂，不过这一原则仍然有效。无论是模拟一扇打开的门，还是制作一个用于”联系我们“图表的打开信封的小动画，都应当尽量逼真。  每个人物的魅力（Ａｐｐｅａｌ）　都是独一无二的。迪斯尼的成功经验告诉我们，任何事物都可以有自己的性格：哪怕是一个茶壶，一棵树，或是一把调羹。使用ＣＳＳ的创造网页动画，我们也应该考虑这一动画如何帮助提升页面整体设计和体验的作用  feijia  (tiimfei@gmail.com) 动画是项了不起的CSS3新特性。就像其他CSS的新功能，人们很有可能会误用甚至滥用它，我们也许会重新看到过去那种Flash風格的花哨頁面。不过我相信今天的Web开发者们应该不会重蹈覆辙。   CSS动画可以让网站生动起来，我们的弹跳的网球球的示例代码也许不是最好的例子，但是它演示了要在页面中加入一个逼真的CSS动画的必要步骤。  有了CSS动画，你可以给你的页面元素加入许多互动性（而且是在不用Flash前提下！），配合JavaScript，它甚至可以用来制作网页游戏。 通过应用本文所介绍的12条准则，你可以为你的网站制作出更加逼真可信的动画，从而提升网站的整体用户体验。  作者： Tom Waterhouse 译者： feijia  (tiimfei@gmail.com)  CSS动画工具  ouse 译者： feijia  (tiimfei@gmail.com) 许多工具可以帮助你制作CSS动画。  [Sencha Animator](http://www.sencha.com/products/animator/)  [Adobe Edge](http://labs.adobe.com/technologies/edge/)  [Tumult(mac only)](http://tumultco.com/hype/)  真实的CSS动画案例  看看真实应用中的CSS动画会让你对它所能实现的效果有更深的影响。  [CSS Spider-man animation](http://www.optimum7.com/css3-man/)by Anthony Calzadilla  [CSS Tricks](https://developer.mozilla.org/en-US/demos/detail/css-tricks)(animated typography person),by [Mircea Piturca](https://developer.mozilla.org/en-US/demos/profile/Mircea Piturca)  [Walking man](http://andrew-hoyer.com/experiments/walking/),by [Andrew Hoyer](http://andrew-hoyer.com/)  [Learning CSS3: Useful References and Guidelines](http://www.smashingmagazine.com/learning-css3-useful-reference-guide/), on Smashing Magazine [Mastering CSS Principles: A Comprehensive Guide](http://www.smashingmagazine.com/mastering-css-principles-comprehensive-reference-guide/), on Smashing Magazine 作者： Tom Waterhouse 译者： feijia  (tiimfei@gmail.com)  ========================================imfei@gmail.com)  译者补充， 下面的代码可以检查用户浏览器是否支持CSS 动画  **[html]** [view plain](http://blog.csdn.net/dojotoolkit/article/details/6865222" \o "view plain)[copy](http://blog.csdn.net/dojotoolkit/article/details/6865222" \o "copy)   1. var animation = false, 2. animationstring = 'animation', 3. keyframeprefix = '', 4. domPrefixes = 'Webkit Moz O ms Khtml'.split(' '), 5. pfx  = ''; 7. if( elm.style.animationName ) { animation = true; } 9. if( animation === false ) { 10. for( var i = 0; i **<** **domPrefixes.length**; i++ ) { 11. if( elm.style[ domPrefixes[i] + 'AnimationName' ] !== undefined ) { 12. pfx = domPrefixes[ i ]; 13. animationstring = pfx + 'Animation'; 14. keyframeprefix = '-' + pfx.toLowerCase() + '-'; 15. animation = true; 16. break; 17. } |