Web Animations - element.animate() is now in Chrome 36

By Brendan Kenny
Lighthouse Bulb Installer

Animation on the web was once the province of JavaScript, but as the world moved to mobile, animations moved to CSS for the declarative syntax and the optimizations browsers were able to make with it. With 60fps on mobile always your goal, it makes sense to never step outside of what browsers know how to efficiently display.

More tools are appearing to make JavaScript-driven animations more efficient, but the holy grail is a unification of declarative and imperative animations , where the decision of how to write your animations is based on what's the clearest code, not what is possible in one form and not in the other.

Web Animations stand to answer that call, and the first part of it has landed in Chrome 36 in the form of `element.animate()`. This new function lets you create an animation purely in JavaScript and have it run as efficiently as any CSS Animation or Transition (in fact, as of Chrome 34, the exact same Web Animations engine drives all of these methods).

The syntax is simple, and its parts should be familiar to you if you've ever written a CSS Transition or Animation:

```
element.animate([
  {cssProperty: value0},
  {cssProperty: value1},
  {cssProperty: value2},
  //...
], {
    duration: timeInMs,
    iterations: iterationCount,
    delay: delayValue
});
```

The biggest advantage of this new function is the elimination of a lot of awkward hoops we formerly had to jump through to get a smooth, jank-free animation.

As an example, for Santa Tracker last year, we wanted to have snow falling continuously, and we decided to animate it via CSS so that it could be done so efficiently.

However, we wanted to pick the snow's horizontal position dynamically based on screen and events going on in the scene itself, and of course the height of the snow's fall (the height of the user's browser window) wouldn't be known until we were actually running. This meant we really had to use CSS Transitions, as authoring a CSS Animation at runtime gets complex quickly (and hundreds of snowflakes means hundreds of new styling rules).

So we took the following approach, which should be familiar:

```
snowFlake.style.transform = 'translate(' + snowLeft + 'px, -100%)';
// wait a frame
snowFlake.offsetWidth;
snowFlake.style.transitionProperty = 'transform';
snowFlake.style.transitionDuration = '1500ms';
snowFlake.style.transform = 'translate(' + snowLeft + 'px, ' + window.innerHeight
```

The key is in that 'wait a frame' comment. In order to successfully start a transition, the browser has to acknowledge that the element is in the starting position. There are a few ways to do this. One of the most common ways is to read from one of the element properties that forces the browser to compute layout, thereby ensuring it knows that the element has a starting position before transitioning to the ending position. Using this method allows you to congratulate yourself on your superior knowledge of browser internals while still feeling dirty with every keystroke.

In contrast, the equivalent `element.animate()` call couldn't be more clear, saying exactly what is intended:

```
snowFlake.animate([
  {transform: 'translate(' + snowLeft + 'px, -100%)'},
  {transform: 'translate(' + snowLeft + 'px, ' + window.innerHeight + 'px)'}
], 1500);
```

There are many more options. Just like with its CSS counterparts, Web Animations can be delayed and iterated:

```
snowFlake.animate([
  {transform: 'translate(' + snowLeft + 'px, -100%)'},
  {transform: 'translate(' + snowLeft + 'px, ' + window.innerHeight + 'px)'}
], {
  duration: 1500,
  iterations: 10,
  delay: 300
});
```

AnimationPlayer

`element.animate()` actually returns an AnimationPlayer object, which will become increasingly important as more of the Web Animations spec is launched. Both JavaScript- and CSS-created animations will have associated AnimationPlayers, allowing them to be seamlessly combined in useful and interesting ways.

For now, though, AnimationPlayer only has two pieces of functionality, both very useful. You can cancel an animation at any time by using **`AnimationPlayer.cancel()`**:

```
var player = snowFlake.animate([
  {transform: 'translate(' + snowLeft + 'px, -100%)'},
  {transform: 'translate(' + snowLeft + 'px, ' + window.innerHeight + 'px)'}
], 1500);
// less than 1500ms later...changed my mind
player.cancel();
```

And, to the relief of everyone who has attempted to build an animation system around CSS Animations or Transitions in the past, Web Animations always fire an event when they're finished:

```
var player = snowFlake.animate([
  {transform: 'translate(' + snowLeft + 'px, -100%)'},
  {transform: 'translate(' + snowLeft + 'px, ' + window.innerHeight + 'px)'}
], 1500);
player.onfinish = function(e) {
  console.log('per aspera ad terra!');
}
```

Try it out

This is all shipping in Chrome 36, moving to beta today! If you'd like to try it, try working with the native implementation in Chrome 36. However, there is a Web Animations polyfill, which brings a significantly larger part of the full Web Animations specification to any of the modern, evergreen browsers.

A demo of the snow effect is available for you to try using both the native version of **`element.animate()`** and the polyfill.

Let us know what you think

Really, though, this is a preview of what's to come, and is being released specifically to get developer feedback right away. We're not sure yet if we've hit every use case, or sanded down every rough edge of the current APIs for animation. The only way for us to know and to really get this right is for developers to try it out and let us know what they think.

Comments on this post are of course valuable, and comments on the standard itself can be addressed to the CSS and SVG Working Groups via the public-fx mailing list.

Update, October 2014: Chrome 39 adds support for several additional methods related to controlling playback, such as `play()`, `pause()`, and `reverse()`. It also supports jumping to a specific point in an animation's timeline via the `currentTime` property. You can see this functionality in action in this new demo.

Thanks to Addy Osmani and Max Heinritz for their assistance with this post.