Web Animations Playback Control in Chrome 39

By Sam Thorogood

Evangelises Chrome and the mobile web in the Developer Relations team at Google.

Earlier this year, Chrome 36 landed the element.animate method as a part of the broader Web Animations spec. This allows for efficient, native animations written imperatively - giving developers the choice to build their animations and transitions with the most suitable approach for them.

For a quick refresher, here's how you might animate a cloud across the screen, with a callback when done:

```
var player = cloud.animate([
  {transform: 'translateX(' + start + 'px)'},
  {transform: 'translateX(' + end + 'px)'}
], 5000);
player.onfinish = function() {
  console.info('Cloud moved across the screen!');
  startRaining(cloud);
};
```

This alone is incredibly easy and is well worth considering as part of your toolbox when building animations or transitions imperatively. However, in Chrome 39, playback control features have been added to the `AnimationPlayer` object returned by `element.animate`. Previously, once an animation was created, you could only call `cancel()` or listen to the finish event.

These playback additions open up the possibilities of what Web Animations can do - turning animations into a general-purpose tool, rather than being prescriptive about transitions, i.e., 'fixed' or predefined animations.

Pause, rewind or change playback rate

Let's start by updating the above example to pause the animation if the cloud is clicked:

```
cloud.addEventListener('mousedown', function() {
  player.pause();
});
```

You could also modify the `playbackRate` property:

```
function changeWindSpeed() {
  player.playbackRate *= (Math.random() * 2.0);
}
```

You can also call the `reverse()` method, which is normally equivalent to inverting the current `playbackRate` (mutiply by -1). There are a couple of special cases, however:

- If the change caused by the `reverse()` method would cause the running animation to effectively end, the `currentTime` is also inverted - e.g., if a brand new animation is reversed, the whole animation will play backwards
- If the player is paused, the animation will start playing.

Scrubbing the player

An `AnimationPlayer` now allows its `currentTime` to be modified while an animation is running. Normally, this value will increase over time (or decrease, if the `playbackRate` is negative). This might allow an animation's position to be externally controlled, perhaps through user interaction. This is commonly referred to as scrubbing.

For example, if your HTML page represented the sky, and you'd like a drag gesture to change the position of a currently playing cloud, you could add some handlers to the document:

```
var startEvent, startEventTime;
document.addEventListener('touchstart', function(event) {
  startEvent = event;
  startEventTime = players.currentTime;
  player.pause();
});
document.addEventListener('touchmove', function(event) {
  if (!startEvent) return;
  var delta = startEvent.touches[0].screenX -
      event.changedTouches[0].screenX;
  player.currentTime = startEventTime + delta;
});
```

As you drag over the document, the `currentTime` will be changed to reflect the distance from your original event. You might also like to resume playing the animation when the gesture ends:

```
document.addEventListener('touchend', function(event) {
  startEvent = null;
  player.play();
});
```

This could even be combined with reversing behavior, depending on where the mouse was lifted from the page (combined demo).

Instead of scrubbing an `AnimationPlayer` in response to a user interaction, its `currentTime` could also be used to show progress or status: for example, to show the status of a download.

The utility here is that an `AnimationPlayer` allows a value to be set and have the underlying native implementation take care of its progress visualization. In the download case, an animation's duration could be set to the total download size, and the `currentTime` set to the currently downloaded size (demo).

UI transitions and gestures

Mobile platforms have long been the realm of common gestures: dragging, sliding, flinging and the like. These gestures tend to have a common theme: a draggable UI component, such as a list view's "pull to refresh" or a sidebar being wrought from the left side of the screen.

With Web Animations, a similar effect is very easy to replicate here on the web - on desktop or on mobile. For example, when a gesture controlling `currentTime` completes:

```
var steps = [ /* animation steps */ ];
var duration = 1000;
var player = target.animate(steps, duration);
player.pause();
configureStartMoveListeners(player);

var setpoints = [0, 500, 1000];
document.addEventListener('touchend', function(event) {
  var srcTime = player.currentTime;
  var dstTime = findNearest(setpoints, srcTime);
  var driftDuration = dstTime - srcTime;

  if (!driftDuration) {
    runCallback(dstTime);
    return;
  }

  var driftPlayer = target.animate(steps, {
    duration: duration,
    iterationStart: Math.min(srcTime, dstTime) / duration,
    iterations: Math.abs(driftDuration) / duration,
    playbackRate: Math.sign(driftDuration)
  });
  driftPlayer.onfinish = function() { runCallback(dstTime); };
  player.currentTime = dstTime;
});
```

This creates an additional animation that performs a 'drift'. This plays between where the gesture was completed, through to our known good target.

This works as animations have a priority based on their creation order: in this case, `driftPlayer` will take precedence over player. When `driftPlayer` completes, it and its effects will disappear. However, its final time will match the underlying player's currentTime, so your UI will remain consistent.

Finally, if you like kittens, there's a demo web application which shows off these gestures. It's mobile-friendly and uses the polyfill for backwards-compatability, so try loading it on your mobile device!

Go forth and element.animate

The `element.animate` method totally rocks *right now* - whether you're using it for simple animations, or leveraging its returned `AnimationPlayer` in other ways.

These two features are also fully supported in other modern browsers via a lightweight polyfill. This polyfill also performs feature detection, so as browser vendors implement the spec, this feature will only get faster and better over time.

The Web Animations spec will also continue to evolve. If you're interested in playing around with upcoming features, they're also available now in a more detailed polyfill: web-animations-next.