requestAnimationFrame API: now with sub-millisecond precision

By Paul Irish

Paul is a contributor to Web**Fundamentals**

If you've been using `requestAnimationFrame` you've enjoyed seeing your paints synchronized to the refresh rate of the screen, resulting in the most high-fidelity animations possible. Plus, you're saving your users CPU fan noise and battery-power when they switch to another tab.

There is about to be a change to part of the API, however. The Timestamp that is passed into your callback function is changing from a typical `Date.now()`-like timestamp to a high-resolution measurement of floating point milliseconds since the page was opened. **If you use this value, you will need to update your code**, based on the explanation below.

Just to be clear, here is what I'm talking about:

```
// assuming requestAnimationFrame method has been normalized for all vendor
requestAnimationFrame(function(timestamp){
    // the value of timestamp is changing
});
```

If you're using the common `requestAnimFrame` shim provided here, then you're not using the timestamp value. You're off the hook. :)

Why

Why? Well rAF helps you get the ultimate 60 fps that is ideal, and 60 fps translates to 16.7ms per frame. But measuring with integer milliseconds means we have a precision of 1/16 for everything we want to observe and target.

As you can see above, the blue bar represents the maximum amount of time you have to do all your work before you paint a new frame (at 60fps). You're probably doing more than 16 things, but with integer milliseconds you only have the ability to schedule and measure in those very chunky increments. That's not good enough.

The High Resolution Timer solves this by providing a far more precise figure:

```
Date.now()          //  1337376068250
performance.now()   //  20303.427000007
```

The high resolution timer is currently available in Chrome as **`window.performance.webkitNow()`**, and this value is generally equal to the new argument value passed into the rAF callback. Once the spec progresses through standards further, the method will drop the prefix and be available through **`performance.now()`**.

You'll also notice the two above values are many orders of magnitude different. **`performance.now()`** is a measurement of floating point milliseconds since that particular page started to load (the **`performance.navigationStart`** to be specific).

In use

The key issue that crops is animation libraries that use this design pattern:

```
function MyAnimation(duration) {
   this.startTime = Date.now();
   this.duration = duration;
   requestAnimFrame(this.tick.bind(this));
}
MyAnimation.prototype.tick = function(time) {
   var now = Date.now();
   if (time > now) {
     this.dispatchEvent("ended");
     return;
   }
    ...
  requestAnimFrame(this.tick.bind(this));
}
```

An edit to fix this is pretty easy... augment the **`startTime`** and **`now`** to use **`window.performance.now()`**.

```
this.startTime = window.performance.now ?
                 (performance.now() + performance.timing.navigationStart) :
                 Date.now();
```

This is a fairly naive implementation, it doesn't use a prefixed **`now()`** method and also assumes **`Date.now()`** support, which isn't in IE8.

Feature detection

If you're not using the pattern above and just want to identify which sort of callback value you're getting you can use this technique:

```
requestAnimationFrame(function(timestamp){

  if (timestamp < 1e12){
      // .. high resolution timer
  } else {
      // integer milliseconds since unix epoch
  }

  // ...
```

Checking **`if (timestamp < 1e12)`** is a quick duck test to see how big of a number we're dealing with. Technically it could false positive but only if a webpage is open continuously for 30 years. But we're not able to test if it's a floating point number (rather than floored to an integer). Ask for enough high resolution timers and you're bound to get integer values at some point.

We plan on pushing this change out in Chrome 21, so if you're already taking advantage of this callback parameter, be sure to update your code!

Except as otherwise noted, the content of this page is licensed under the Creative Commons Attribution 3.0 License, and code samples are licensed under the Apache 2.0 License. For details, see our Site Policies. Java is a registered trademark of Oracle and/or its affiliates.

Last updated July 2, 2018.