New in Chrome 66

By Pete LePage
Pete is a Developer Advocate

- CSS manipulation becomes easier with the new CSS Typed Model Object.
- Access to the clipboard is now asynchronous.
- There's a new rendering context for canvas elements.

And there's plenty more!

I'm Pete LePage. Let's dive in and see what's new for developers in Chrome 66!

Note: Want the full list of changes? Check out the Chromium source repository change list.

CSS Typed Object Model

If you've ever updated a CSS property via JavaScript, you've used the CSS object model. But it returns everything as a string.

```
el.style.opacity = 0.3;
console.log(typeof el.style.opacity);
> 'string' // A string!?
```

To animate the `opacity` property, I'd have to cast the string to a number, then increment the value and apply my changes. Not exactly ideal.

```
function step(timestamp) {
  const currentOpacity = parseFloat(el.style.opacity);
  const newOpacity = currentOpacity + 0.01;
  element.style.opacity = newOpacity;
  if (newOpacity <= 1) {
    window.requestAnimationFrame(step);
  }
}
```

With the new CSS Typed Object Model, CSS values are exposed as typed JavaScript objects, eliminating a lot of the type manipulation, and providing a more sane way of working with CSS.

Instead of using **`element.style`**, you access styles through the **`.attributeStyleMap`** property or **`.styleMap`**. They return a map-like object that makes it easy to read or update.

```
el.attributeStyleMap.set('opacity', 0.3);
const oType = typeof el.attributeStyleMap.get('opacity').value;
console.log(oType);
> 'number' // Yay!
```

Compared to the old CSS Object Model, early benchmarks show about a 30% improvement in operations per second - something that's especially important for JavaScript animations.

```
el.attributeStyleMap.set('opacity', 0.3);
el.attributeStyleMap.has('opacity'); // true
el.attributeStyleMap.delete('opacity');
el.attributeStyleMap.clear(); // remove all styles
```

It also helps to eliminate bugs caused by forgetting to cast the value from a string to a number, and it automatically handles rounding and clamping of values. Plus, there's some pretty neat new methods for dealing with unit conversions, arithmetic and equality.

```
el.style.opacity = 3;
const opacity = el.computedStyleMap().get('opacity').value;
console.log(opacity);
> 1
```

Eric has a great post with several demos and examples in his explainer.

Async Clipboard API

```
const successful = document.execCommand('copy');
```

Synchronous copy & paste using **`document.execCommand`** can be OK for small bits of text, but for anything else, there's a good chance it's synchronous nature will block the page, causing a poor experience for the user. And the permission model between browsers is inconsistent.

The new Async Clipboard API is a replacement that works asynchronously, and integrates with the permission API to provide a better experience for users.

Text can be copied to the clipboard by calling **`writeText()`**.

```
navigator.clipboard.writeText('Copy me!')
  .then(() => {
    console.log('Text is on the clipboard.');
  });
```

Since this API is asynchronous, the **`writeText()`** function returns a Promise that will be resolved or rejected depending on whether the text we passed is copied successfully.

Similarly, text can be read from the clipboard by calling **`getText()`** and waiting for the returned Promise to resolve with the text.

```
navigator.clipboard.getText()
  .then((text) => {
    console.log('Clipboard: ', text);
  });
```

Check out Jason's post and demos in the explainer. He's also got examples that use **async** functions.

New Canvas Context **`BitmapRenderer`**

The **canvas** element lets you manipulate graphics at the pixel level, you can draw graphs, manipulate photos, or even do real time video processing. But, unless you're starting with a blank **canvas**, you need a way to render an image on the **canvas**.

Historically, that's meant creating an **`image`** tag, then rendering it's contents on to the **canvas**. Unfortunately that means the browser needs to store multiple copies of the image in memory.

```
const context = el.getContext('2d');
const img = new Image();
```

```
img.onload = function () {
  context.drawImage(img, 0, 0);
}
img.src = 'llama.png';
```

Starting in Chrome 66, there's a new asynchronous rendering context that's streamlined the display of `ImageBitmap` objects. They now render more efficiently and with less jank by working asynchronously and avoiding memory duplication.

To use it:

1. Call `createImageBitmap` and hand it an image blob, to create the image.
2. Grab the `bitmaprenderer` context from the `canvas`.
3. Then transfer the image in.

```
const image = await createImageBitmap(imageBlob);
const context = el.getContext('bitmaprenderer');
context.transferFromImageBitmap(image);
```

Done, I've rendered the image!

AudioWorklet

Worklets are in! PaintWorklet shipped in Chrome 65, and now we're enabling AudioWorklet by default in Chrome 66. This new type of Worklet can be used to process audio in the dedicated audio thread, replacing the legacy ScriptProcessorNode which ran on the main thread. Each AudioWorklet runs in its own global scope, reducing latency and increasing throughput stability.

There are some interesting examples of AudioWorklet over on Google Chrome Labs.

And more!

These are just a few of the changes in Chrome 66 for developers, of course, there's plenty more.

- `TextArea` and `Select` now support the `autocomplete` attribute.
- Setting `autocapitalize` on a `form` element will apply to any child form fields, improving compatibility with Safari's implementation of `autocapitalize`.

- `trimStart()` and `trimEnd()` are now available as the standards-based way of trimming whitespace from strings.

Be sure to check out New in Chrome DevTools, to learn what's new in for DevTools in Chrome 66. And, if you're interested in Progressive Web Apps, check out the new PWA Roadshow video series. Then, click the subscribe button on our YouTube channel, and you'll get an email notification whenever we launch a new video, or add our RSS feed to your feed reader.

I'm Pete LePage, and as soon as Chrome 67 is released, I'll be right here to tell you – what's new in Chrome!

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.