Chrome Supports createImageBitmap() in Chrome 50

By Paul Lewis
Paul is a Design and Perf Advocate

Decoding images for use with a canvas is pretty common, whether it's to allow users to customize an avatar, crop an image, or just zoom in on a picture. The problem with decoding images is that it can be CPU intensive, and that can sometimes mean jank or checkerboarding. As of Chrome 50 (and in Firefox 42+) you now have another option: `createImageBitmap()`. It allows you to decode an image in the background, and get access to a new `ImageBitmap` primitive, which you can draw into a canvas in the same way you would an `<img>` element, another canvas, or a video.

Drawing blobs with createImageBitmap()

Let's say you download a blob image with `fetch()` (or XHR), and you want to draw it into a canvas. Without `createImageBitmap()` you would have to create an image element and a Blob URL to get the image into a format you could use. With it you get a much more direct route to painting:

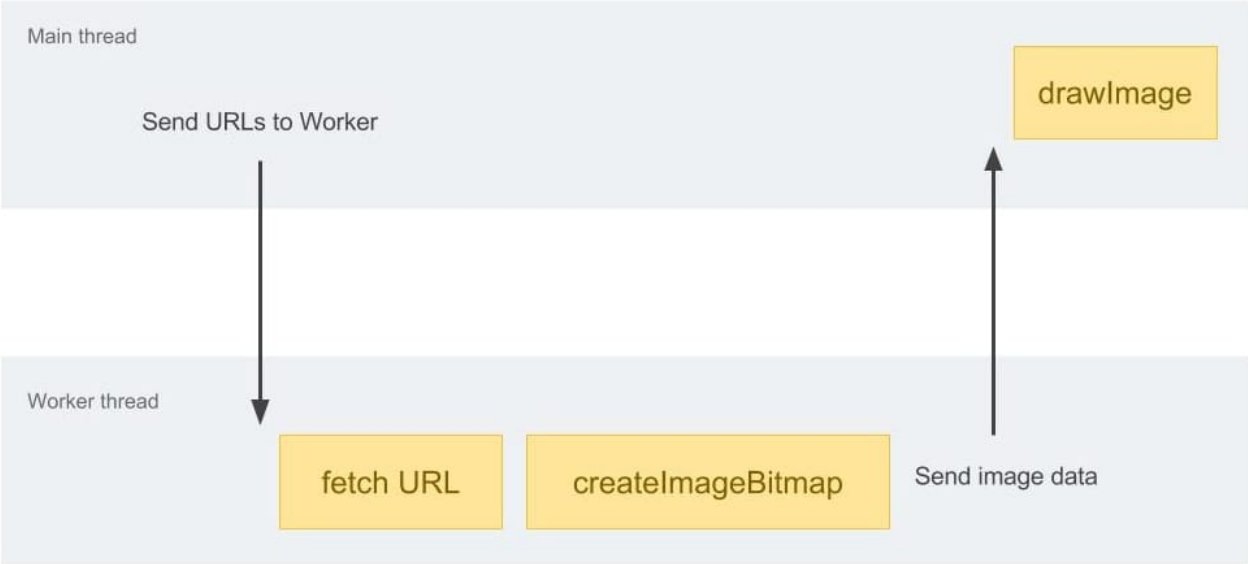
```
fetch(url)
  .then(response => response.blob())
  .then(blob => createImageBitmap(blob))
  .then(imageBitmap => ctx.drawImage(imageBitmap, 0, 0));
```

This approach will also work with images stored as blobs in IndexedDB, making blobs something of a convenient intermediate format. As it happens Chrome 50 also supports the `.toBlob()` method on canvas elements, which means you can – for example – generate blobs from canvas elements.

Using createImageBitmap() in web workers

One of the nicest features of `createImageBitmap()` is that it's also available in workers, meaning that you can now decode images wherever you want to. If you have a lot of images

to decode that you consider non-essential you would ship their URLs to a Web Worker, which would download and decode them as time allows. It would then transfer them back to the main thread for drawing into a canvas.



The code for doing this may look something like:

```
// In the worker.
fetch(imageURL)
  .then(response => response.blob())
  .then(blob => createImageBitmap(blob))
  .then(imageBitmap => {
    // Transfer the imageBitmap back to main thread.
    self.postMessage({ imageBitmap }, [imageBitmap]);
  }, err => {
    self.postMessage({ err });
  });

// In the main thread.
worker.onmessage = (evt) => {
  if (evt.data.err)
    throw new Error(evt.data.err);

  canvasContext.drawImage(evt.data.imageBitmap, 0, 0);
}
```

Today if you call **`createImageBitmap()`** on the main thread, that's exactly where the decoding will be done. The plans are, however, to have Chrome automatically do the decoding in another thread, helping to keep the main thread workload down. In the

meantime, however, you should be mindful of doing the decoding on the main thread, as it is intensive work that could block other essential tasks, like JavaScript, style calculations, layout, painting, or compositing.

A helper library

To make life a little simpler, I have created a helper library that handles the decoding on a worker, and sends back the decoded image to the main thread, and draws it into a canvas. You should, of course, feel free to reverse engineer it and apply the model to your own apps. The major benefit is more control, but that (as usual) comes with more code, more to debug, and more edge cases to be considered than using an `<img>` element.

If you need more control with image decoding, `createImageBitmap()` is your new best friend. Check it out in Chrome 50, and let us know how you get on!

Last updated July 2, 2018.