<https://www.chromestatus.com/features/5144752345317376>

<https://kangax.github.io/compat-table/es6/>  
<https://developers.google.com/web/updates/2017/04/devtools-release-notes#coverage>

<https://moz.com/learn/seo/page-authority>

To create a web worker, pass a file to the worker constructor, which starts running that file in a separate thread:

Communicate with the web worker by sending messages via the postMessage API. Pass the message value as a parameter in the postMessage call and then add a message event listener to the worker:

main.js #

const worker = new Worker("./worker.js");

worker.postMessage([40, 2]);

worker.js #

addEventListener("message", event => {

const [a, b] = event.data;

// Do stuff with the message

});

To send a message back to the main thread, use the same postMessage API in the web worker and set up an event listener on the main thread:

main.js #

const worker = new Worker("./worker.js");

worker.postMessage([40, 2]);

worker.addEventListener("message", event => {

console.log(event.data);

});

worker.js #

addEventListener("message", event => {

const [a, b] = event.data;

// Do stuff with the message

postMessage(a+b);

});

a library whose goal is to let you use web workers without having to think about the details of postMessage

Don't confuse web workers with service workers or worklets. While the names are similar, the functionality and uses are different.

Web workers don't have access to the DOM and many APIs like WebUSB, WebRTC, or Web Audio, so you can't put pieces of your app that rely on such access in a worker.

It moves JavaScript execution costs to a separate thread. · It moves parsing costs, meaning UI might boot up faster. That might reduce First Contentful Paint or even Time to Interactive, which can in turn increase your Lighthouse score.

The Worklet interface is a lightweight version of Web Workers and gives developers access to low-level parts of the rendering pipeline. With Worklets, you can run JavaScript and WebAssembly code to do graphics rendering or audio processing where high performance is required.

AudioWorklet audio processing with custom AudioNodes

The Cache Storage API proves very useful when called from your service worker's JavaScript code. https://web.dev/service-workers-cache-storage/

A service worker has some special powers—among other duties, it patiently waits for your web app to make an outgoing request, and then springs into action by intercepting it. What the service worker does with this intercepted request is up to you!

For some requests, the best course of action might be just to allow the request to continue on to the network, just like what would happen if there were no service worker at all.

For other requests, though, you can take advantage of something more flexible than the browser's HTTP cache, and return a reliably fast response without having to worry about the network. That entails using the other piece of the puzzle: the Cache Storage API.