posti

Web Workers, Comlink

Maksim Riazanov

29.05.2023

HelsinkiJS

Agenda

- **Web Worker**
  - API
  - Applications
  - Benefits and Limitations
- **Comlink**
  - API
  - Benefits of ”enchancing” Web Worker
- Live Example

Web Workers

posti

What is Web Worker

Web Workers makes it possible to run a script operation in a background thread separate from the main execution thread of a web application. The advantage of this is that laborious processing can be performed in a separate thread, allowing the main (usually the UI) thread to run without being blocked/slowed down.

- MDN Docs Definition

posti

What Web Worker is NOT...

Service Worker

"That thing you need for Progressive Web App"

- Enables "offline-first" behaviour
- Intercepts network requests, facilitates complex and custom caching behaviour
- Persistent: remains active when page is close
- More complex lifecycle
- Runs in special `ServiceWorkerGlobalScope`
  Special kind of worker context, running off the main script execution thread, with no DOM access

Worklet

"Lightweight version of Web Workers. Gives developers access to low-level parts of the rendering pipeline."

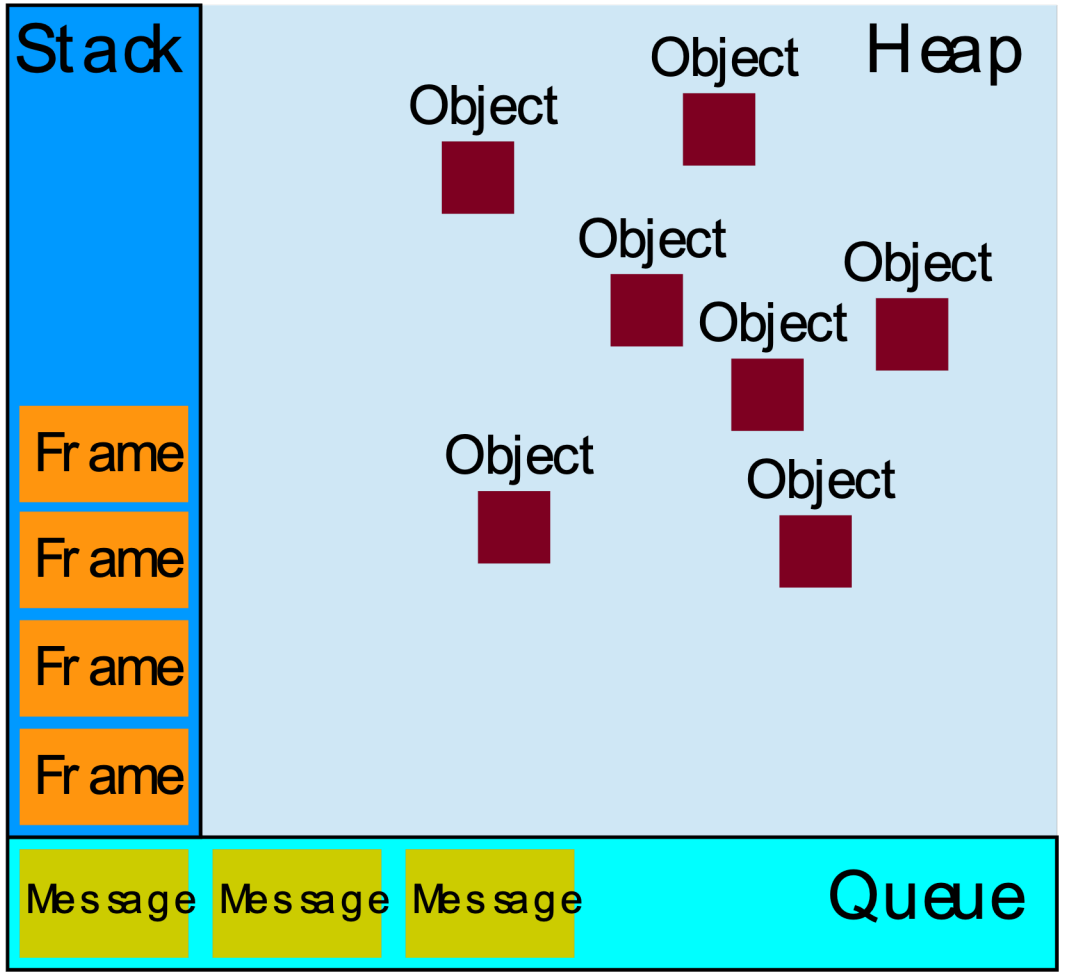
- These are for low-level dark arts practitioners
- Variants
  - `AudioWorklet`: audio processing with custom AudioNodes.
  - `AnimationWorklet`: creating scroll-linked and other high performance procedural animations.
  - `LayoutWorklet`: defining the positioning and dimensions of custom elements.

posti

and just in case...

Promises are not forked threads

- Operates on the main thread
  - Queued as a micro-task
  - Have access to global context
  - No restrictions on DOM or shared objects manipulation
- Non-blocking low-level I/O operations
  - ”When the application is waiting for an IndexedDB query to return or an XHR request to return, it can still process other things like user input.”



- MDN Web Docs, “The event loop”

Simple usage example

```
// main.js

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

// send payload
myWorker.postMessage([5, 22]);

// receive result
myWorker.onmessage = (e) => {
  result.textContent = e.data;
  console.log("Message received from worker");
};

// clean up
myWorker.terminate();
```

```
// worker.js

onmessage = (e) => {
  const workerResult = `Result: ${e.data[0] * e.data[1]}`;
  console.log("Posting message back to main script");
  postMessage(workerResult);
};
```


posti

Web Worker Benefits

- Keep UI thread responsive
  - No need to chunk blocking tasks
  - Heavier tasks could be done on the client
  - Better experience for low-end devices
- Simple lifecycle, easy to create/terminate

posti

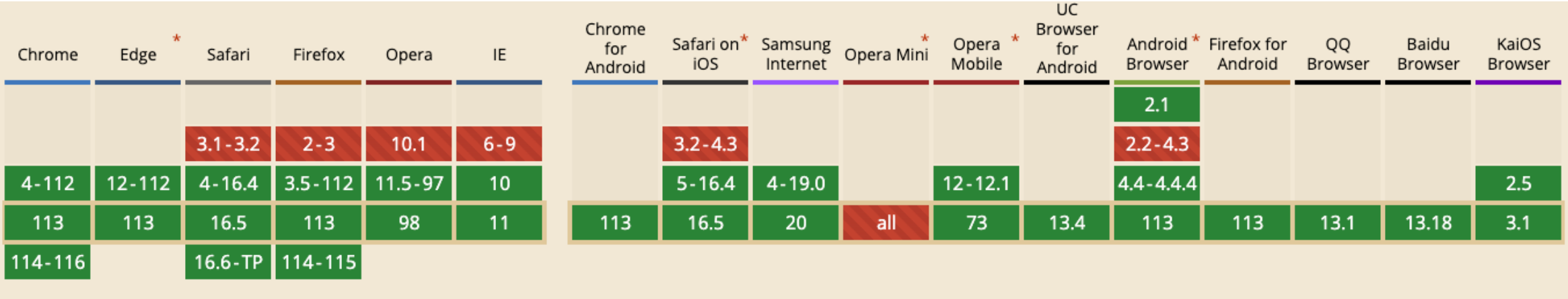
Web Worker Limitations

- Truly isolated thread (context)
  - Can’t access shared memory
  - Severely limited special `DedicatedWorkerGlobalScope`
- Communication through `postMessage`
  - Restrictive and clunky API
  - Everything goes through structuredClone
    `Transferrables` are exempt, ownership is passed to other context instead
- Difficulties with debugging
- TypeScript typings are not great

posti

caniuse? Web Workers

Yes! Supported large majority of browsers



Chrome	Edge *	Safari	Firefox	Opera	IE	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
												2.1				
		3.1-3.2	2-3	10.1	6-9		3.2-4.3					2.2-4.3				
4-112	12-112	4-16.4	3.5-112	11.5-97	10		5-16.4	4-19.0		12-12.1		4.4-4.4.4				2.5
113	113	16.5	113	98	11	113	16.5	20	all	73	13.4	113	113	13.1	13.18	3.1
114-116		16.6-TP	114-115													

Usecases

- Image compression | https://github.com/GoogleChromeLabs/squoosh
- Game Logic | https://github.com/GoogleChromeLabs/proxx
- Redux off the main thread | https://www.npmjs.com/package/use-workerized-reducer
- Training RNN with Tensorflow | https://www.tensorflow.org/js/tutorials/training/web_worker
- Client-bound workloads: validations, number crunching etc
- Your next amazing project?

posti

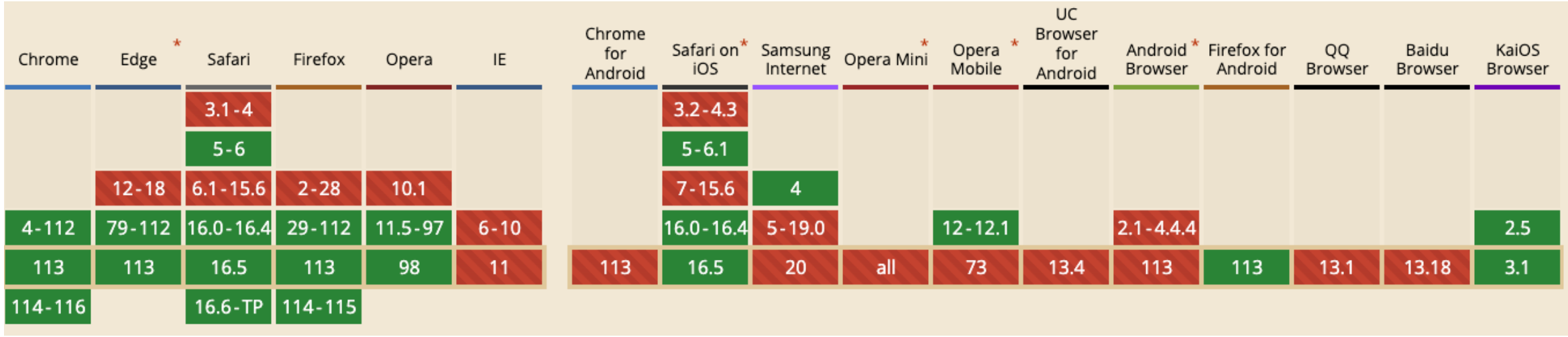
Shared Web Worker

- Allows to create a worker that is shared across multiple contexts (windows, iframes, other workers)
- Different API
  - Uses `onconnect` instead of `onmessage`
  - Worker initialisation requires “handshake”
  - `SharedWorkerGlobalScope`
- `SharedWebWorker` is not terminated immeditaly upon the page exit

posti

caniuse? Shared Web Worker

Maybe? Support is missing for mobile browsers



Chrome	Edge *	Safari	Firefox	Opera	IE
		3.1-4			
		5-6			
	12-18	6.1-15.6	2-28	10.1	
4-112	79-112	16.0-16.4	29-112	11.5-97	6-10
113	113	16.5	113	98	11
114-116		16.6-TP	114-115		

Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
	3.2-4.3									
	5-6.1									
	7-15.6	4								
	16.0-16.4	5-19.0		12-12.1		2.1-4.4.4				2.5
113	16.5	20	all	73	13.4	113	113	13.1	13.18	3.1

Comlink

What is Comlink

“Comlink makes **WebWorkers** enjoyable.

Turns message-based API into a something more developer-friendly by providing an RPC implementation: Values from one thread can be used within the other thread (and vice versa) just like local values.”

Largely made by this person **Surma**

Advocating for “always off the main thread” for a long time, loads of amazing libraries, experiments and write-ups on the topic.

posti

Simple example

```
// main.js
import { wrap } from "comlink";

// create worker
const myComlinkWorker = new Worker("comlink-worker.js");

// wrap worker
wrap(myComlinkWorker);

// use worker
result.textContent = await wrap.myFunction(5, 22);
console.log("Message received from worker");
```

```
// comlink-worker.js
import { expose } from "comlink";

expose({
  myFunction: (arg1, arg2, ...args) => {
    const workerResult = () => `Result: ${arg1 * arg2}`;
    console.log("Posting message back to main script");
    return workerResult;
  },
});
```


posti

Comlink benefits

- No more `postMessage`
  - Arguments provided directly without unpacking `data`
  - Return results directly
  - Catch errors directly with normal `try/catch`
  - TS types applied nicely
- Flexibility in worker definition
  - No need to comply with single `onmessage` per-file
- Able to pass callback functions using `Comlink.proxy`
- Also works with `SharedWebWorker` and Node's `worker_threads`

posti

Show me the code!

posti

Bundling

Vite

- Native support
- Additional import “query suffix” to simplify initialisation

```
import MyWorker from './worker?worker'
const worker = new MyWorker()
```

Webpack

- Native suppport since Webpack 5
- Some syntax restrictions on worker initialisation

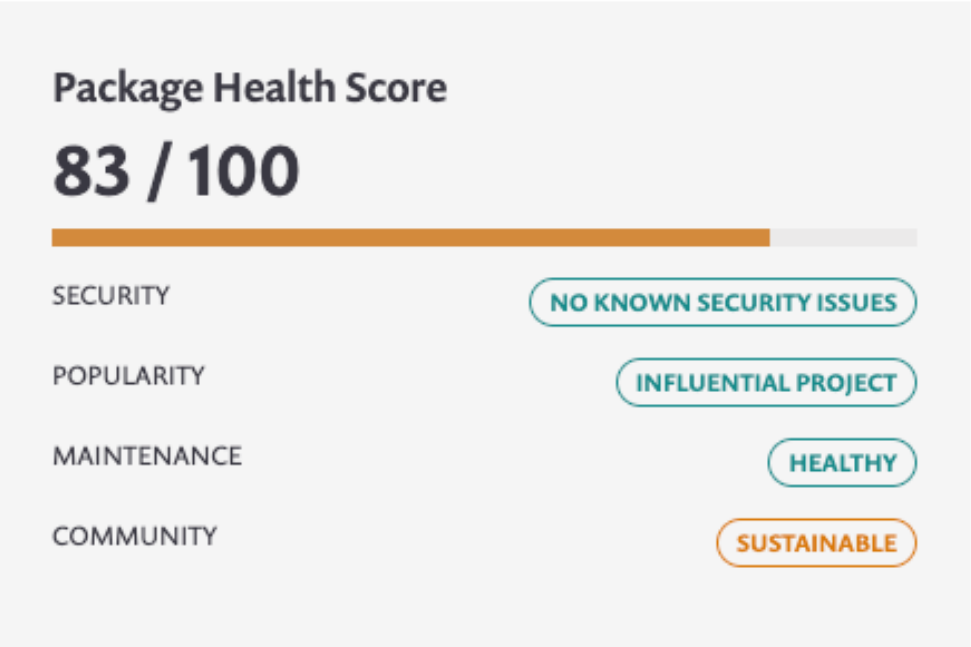
> **Warning**
>
> Using a variable in the `Worker` constructor is not supported by webpack. For example, the following code will not work: `const url = new URL('./path/to/worker.ts', import.meta.url); const worker = new Worker(url);`

posti

Project Health

Pretty good for niche case library

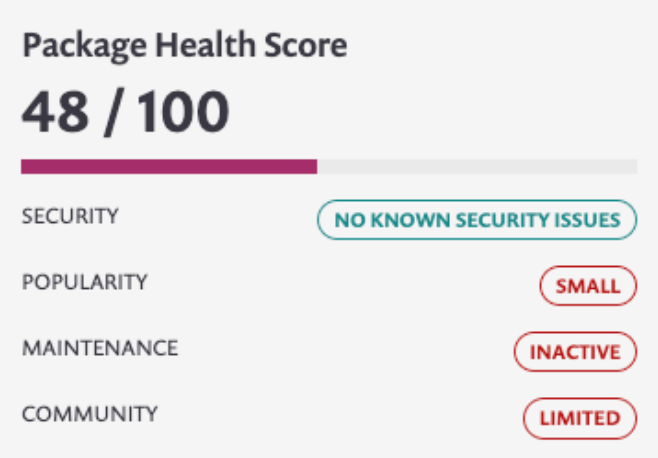
- 9.7k Stars, last commit in February 2023



Similar projects

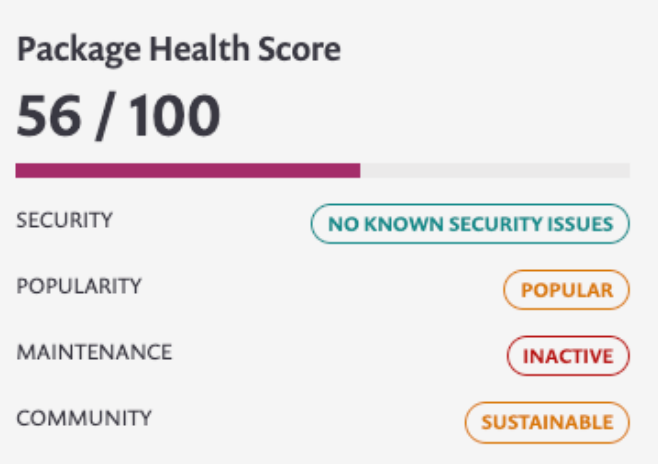
Workerize

- Moves a module into a Web Worker, automatically reflecting exported functions as asynchronous proxies.
- 4.3k Stars, last commit in 2018



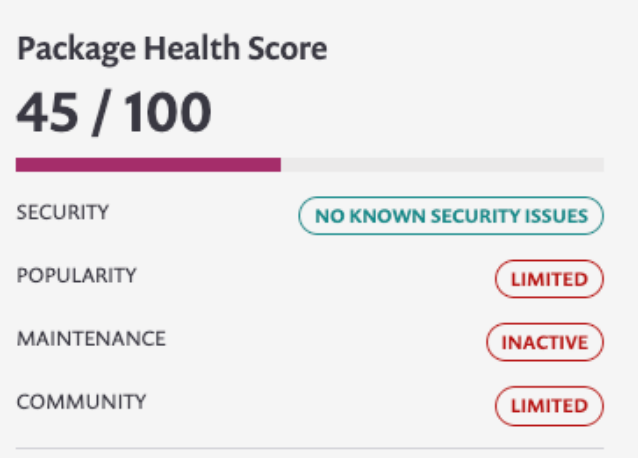
Threads.js

- Offload CPU-intensive tasks to worker threads in node.js, web browsers and electron using one uniform API.
- 2.8k Stars, last commit 2021



Workly

- A really simple way to move a stand-alone function/class to a worker thread.
- 1.9k Stars, last commit 2019



Honorable mentions: Partytown

- Built for different purpose
- Helps to offloading third-party scripts from main thread into a web worker
- 11.k Starts, actively developed

Summary

Web Workers are a viable paradigm to explore to tackle issues related to performance bottlenecks, either connected to huge uninterruptable workloads or with a need to run on a low-end devices.

There is decent tooling and ecosystem in place have DX at acceptable levels and reduce codebase complexity bloat.

As any threading, it requires comprehensive knowledge of the topic to avoid any potential “foot shooting”.

posti

→ Thank you!

References

- https://web.dev/workers-overview/
- https://www.smashingmagazine.com/2021/06/web-workers-2021/
- https://surma.dev/
- https://javascript.info/event-loop
- https://johnnyreilly.com/web-workers-comlink-typescript-and-react
- https://blog.logrocket.com/comlink-web-workers-match-made-in-heaven/
- https://blog.logrocket.com/integrating-web-workers-in-a-react-app-with-comlink/
- https://davidea.st/articles/comlink-simple-web-worker/#comlink-calling-methods-across-barriers