Jake Archibald wrote... who?

Using ServiceWorker in Chrome today

Posted 24 September 2014

The implementation for ServiceWorker has been landing in Chrome Canary over the past few months, and there's now enough of it to do some cool shit!

Unnecessary representation of "cool shit"

What is ServiceWorker?

ServiceWorker is a background worker, it gives us a JavaScript context to add features such as push messaging, background sync, geofencing and network control.

In terms of network control, it acts like a proxy server sitting on the client, you get to decide what to do on a request-by-request basis. You can use this to make stuff work faster, offline, or build new features.

I'm biased, but I think ServiceWorker changes the scope of the web more than any feature since XHR.

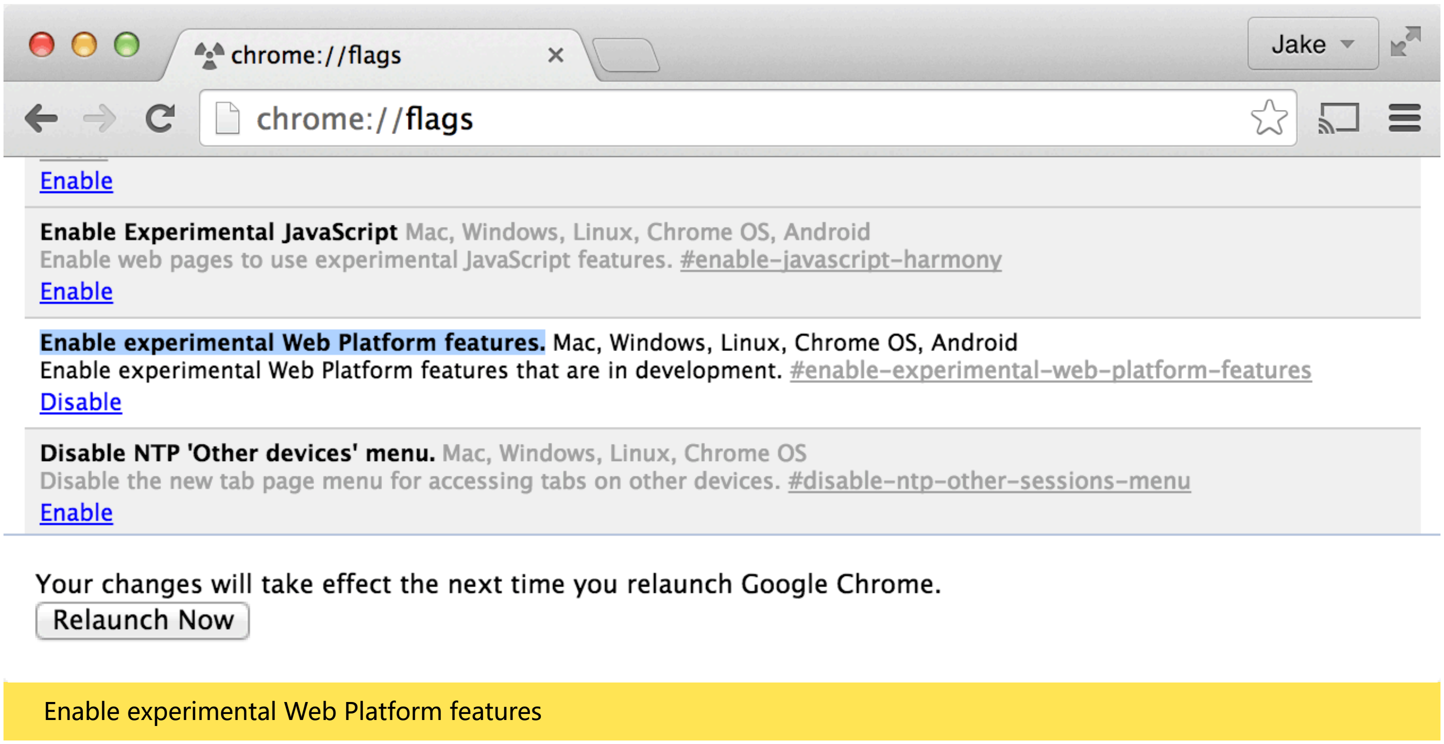
If you want more of an overview on ServiceWorker, check out:

ServiceWorker is coming, look busy - 30min talk
Is ServiceWorker ready? - feature support status across browsers
ServiceWorker first draft - blog post
ServiceWorker API - on MDN

In Canary today

Google & Mozilla are actively developing ServiceWorker. The developer tooling in Canary makes it easier to use at the moment, but expect Firefox to (catch up soon).

If you want to hack around with ServiceWorker, open Chrome Canary, go to `chrome://flags` and enable "experimental Web Platform features". Give the browser a restart, and you're ready to go!



Enable

Enable Experimental JavaScript Mac, Windows, Linux, Chrome OS, Android
Enable web pages to use experimental JavaScript features. #enable-javascript-harmony
Enable

Enable experimental Web Platform features. Mac, Windows, Linux, Chrome OS, Android
Enable experimental Web Platform features that are in development. #enable-experimental-web-platform-features
Disable

Disable NTP 'Other devices' menu. Mac, Windows, Linux, Chrome OS
Disable the new tab page menu for accessing tabs on other devices. #disable-ntp-other-sessions-menu
Enable

Your changes will take effect the next time you relaunch Google Chrome.
Relaunch Now

Enable experimental Web Platform features

By playing with this today, you'll become one of the first developers in the world to do. You may create some cool shit, but there's also a chance you'll step in some uncool shit. If that happens, please file bugs. If you're unsure if you've found a bug or some unexpected-but-correct behaviour, comment on this article (demos of the issue will help massively).

Demos

Check out ServiceWorkersDemos, which includes things like offline-first news, a clientside wiki, and some trains.

Trained-to-thrill

The trains one in particular highlights the performance benefits of ServiceWorker. Not only does it work offline, it feels lightning-fast on any connection. Here's a comparrison on a connection throttled to 250kbits:

Without vs with ServiceWorker

With the ServiceWorker, we get to first-render over a second faster, and our first render includes images, which takes 16 seconds without ServiceWorker. They're cached images of course, but ServiceWorker allows us to present a fully-rendered view while we go to the network for latest content.

Getting started

To begin, you need to register for a ServiceWorker from your page:

```
navigator.serviceWorker.register('/worker.js').then(
  function (reg) {
    console.log('◕‿◕', reg);
  },
  function (err) {
    console.log('ಠ_ಠ', err);
  },
);
```

As you can see, `.register` take a URL for your worker script and returns a promise. The ServiceWorker script must be on the same origin as your page.

If you're new to promises, check out the HTML5Rocks article - I use visits to this to justify my employment, so thanks!

You can also limit the scope of the ServiceWorker to a subset of your origin:

```
navigator.serviceWorker
```

```
  .register('/worker.js', {
    scope: '/trained-to-thrill/',
  })
  .then(
    function (reg) {
      console.log('◕‿◕', reg);
    },
    function (err) {
      console.log('ಠ_ಠ', err);
    },
  );
```

This is particularly useful for origins that contain many separate sites, such as Github pages.

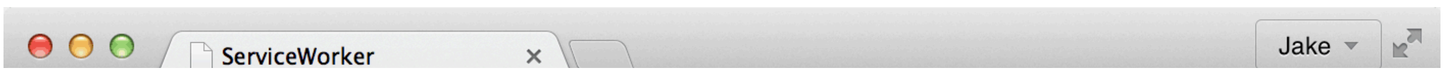
HTTPS only

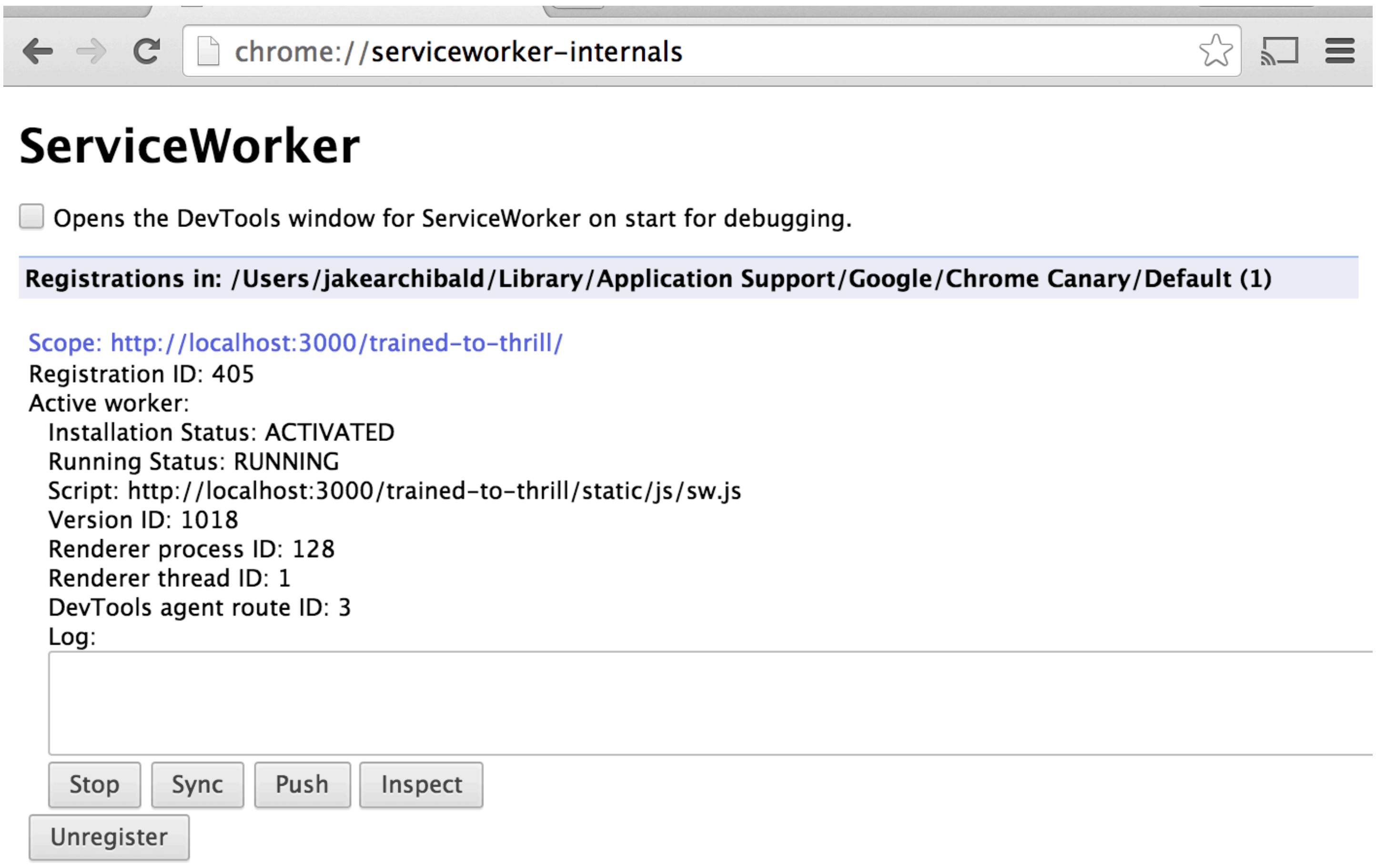
Using ServiceWorker you can hijack connections, respond differently, & filter responses. Powerful stuff. While you would use these powers for good, a man-in-the-middle might not. To avoid this, you can only register for ServiceWorkers on pages served over HTTPS, so we know the ServiceWorker the browser receives hasn't been tampered with during its journey through the network.

Github Pages are served over HTTPS, so they're a great place to host demos.

Devtools

It's early days for Chrome's ServiceWorker devtools, but what we have today is better than we ever had with AppCache. Go to `chrome://serviceworker-internals`, you'll get a list of all the ServiceWorker registrations the browser is aware of along with the state of each worker within it. Click "Inspect" to open a devtools window for the worker, this allows you to set breakpoints & test code in the console.




chrome://serviceworker-internals

ServiceWorker

Opens the DevTools window for ServiceWorker on start for debugging.

Registrations in: /Users/jakearchibald/Library/Application Support/Google/Chrome Canary/Default (1)

```
Scope: http://localhost:3000/trained-to-thrill/
Registration ID: 405
Active worker:
  Installation Status: ACTIVATED
  Running Status: RUNNING
  Script: http://localhost:3000/trained-to-thrill/static/js/sw.js
  Version ID: 1018
  Renderer process ID: 128
  Renderer thread ID: 1
  DevTools agent route ID: 3
  Log:
```

chrome://serviceworker-intervals

Installation Status - This is useful during updates.

Running Status / Start / Stop - The ServiceWorker closes when it isn't needed to save memory, this gives you control over that.

Sync/Push - These fire events in the ServiceWorker. They're not particularly useful right now.

Inspect - Launch a devtools window for the worker. This lets you set breakpoints & interact with the console.

Unregister - Throw the ServiceWorker away.

Opens the DevTools window ... on start - What it says on the tin. Useful for debugging startup issues.

Also, shift+refresh loads the current page without the ServiceWorker, this is useful for checking CSS and page script changes without having to wait for a background update.

worker.js

Let's start with:

```
// The SW will be shutdown when not in use to save memory,
// be aware that any global state is likely to disappear
console.log('SW startup');

self.addEventListener('install', function (event) {
  console.log('SW installed');
});

self.addEventListener('activate', function (event) {
  console.log('SW activated');
});

self.addEventListener('fetch', function (event) {
  console.log('Caught a fetch!');
  event.respondWith(new Response('Hello world!'));
});
```

Now if you navigate to your page, the SW above should install & activate. Refreshing the page will cause the SW to take over the page and respond "Hello world". Here's a demo.

You won't see the above console logs in your page, they happen within the worker itself. You can launch a devtools window for the worker from `chrome://serviceworker-internals`.

event.respondWith

This is what you call to hijack the fetch and respond differently. It must be called synchronously, passing in a `Response` or a promise that resolves to one. If you don't call it, you get normal browser behaviour.

You'll likely get your response from:

`new Response(body, opts)` - manually created as above. This API is in the fetch spec.
`fetch(urlOrRequest)` - the network. This API is also in the fetch spec.
`caches.match(urlOrRequest)` - the cache. We'll pick this up later in the article.

`event.request` gives you information about the request, so you can do what you want per request. Also, since `match` and `fetch` are promise-based, you can combine them. Fallback from cache, to network, to a manually created response, for instance.

Updating your ServiceWorker

The lifecycle of a ServiceWorker is based on Chrome's update model: Do as much as possible in the background, don't disrupt the user, complete the update when the current version closes.

Whenever you navigate to page within scope of your ServiceWorker, the browser checks for updates in the background. If the script is byte-different, it's considered to be a new version, and installed (note: only the script is checked, not external `importScripts`). However, the old version remains in control over pages until all tabs using it are gone. Then the old version is garbage collected and the new version takes over.

This avoids the problem of two versions of a site running at the same time, in different tabs. Our current strategy for this is "cross fingers, hope it doesn't happen".

You navigate through this using the install & activate events:

install

This is called when the browser sees this version of the ServiceWorker for the first time. You can pass a promise to `event.waitUntil` to extend this part of the lifecycle:

```
self.addEventListener('install', function (event) {
  console.log('Installing…');

  event.waitUntil(
    somethingThatReturnsAPromise().then(function () {
      console.log('Installed!');
    }),
  );
});
```

If the promise rejects, that indicates installation failure, and this ServiceWorker will become redundant and be garbage collected.

This all happens in the background while an existing version remains in control. You're not disrupting the user, so you can do some heavy-lifting. Fetch stuff from the network, populate caches, get everything ready. Just don't delete/move anything that would disrupt the current version.

The installing worker can take over immediately if it wants, kicking the old version out by calling `event.replace()`, although this isn't implemented in Chrome yet.

By default, the old version will remain in control until no pages are open within its scope.

activate

This happens when the old version is gone. Here you can make changes that would have broken the old version, such as deleting old caches and migrating data.

```
self.addEventListener('activate', function (event) {
  console.log('Activating…');

  event.waitUntil(
    somethingThatReturnsAPromise().then(function () {
      console.log('Activated!');
    }),
  );
});
```

`waitUntil` is here too. Fetch (and other) events will be delayed until this promise settles. This may delay the load of a page & resources, so use `install` for anything that can be done while an old version is still in control.

On navigate

When a new document is loaded, it decides which ServiceWorker it will use for its lifetime. So when you first visit an origin, and it registers, installs, & activates a ServiceWorker, the page won't be controlled by this worker until the next navigate (or refresh). The exception is if the worker calls `event.replace()` in the install event, then it takes control of all in-scope pages immediately.

`navigator.serviceWorker.controller` points to the ServiceWorker controlling the page, which is null if the page isn't controlled.

In practice:

Here's how that looks:

Updating a ServiceWorker

And here's the code. In that demo I'm forcing install/activate to take 5 seconds.

Unfortunately refreshing a single tab isn't enough to allow an old worker to be collected and a new one take over. Browsers make the next page request before unloading the current page, so there isn't a moment when current active worker can be released. We're looking to solve this issue with some special-casing, or a "Replace" button in serviceworker-internals.

The easiest way at the moment is to close & reopen the tab (cmd+w, then cmd+shift+t on Mac), or shift+reload then normal reload.

The Cache

ServiceWorker comes with a caching API, letting you create stores of responses keyed by request. The entry point, `caches`, is present in Canary, but it isn't useful yet. In the mean time, check out the polyfill.

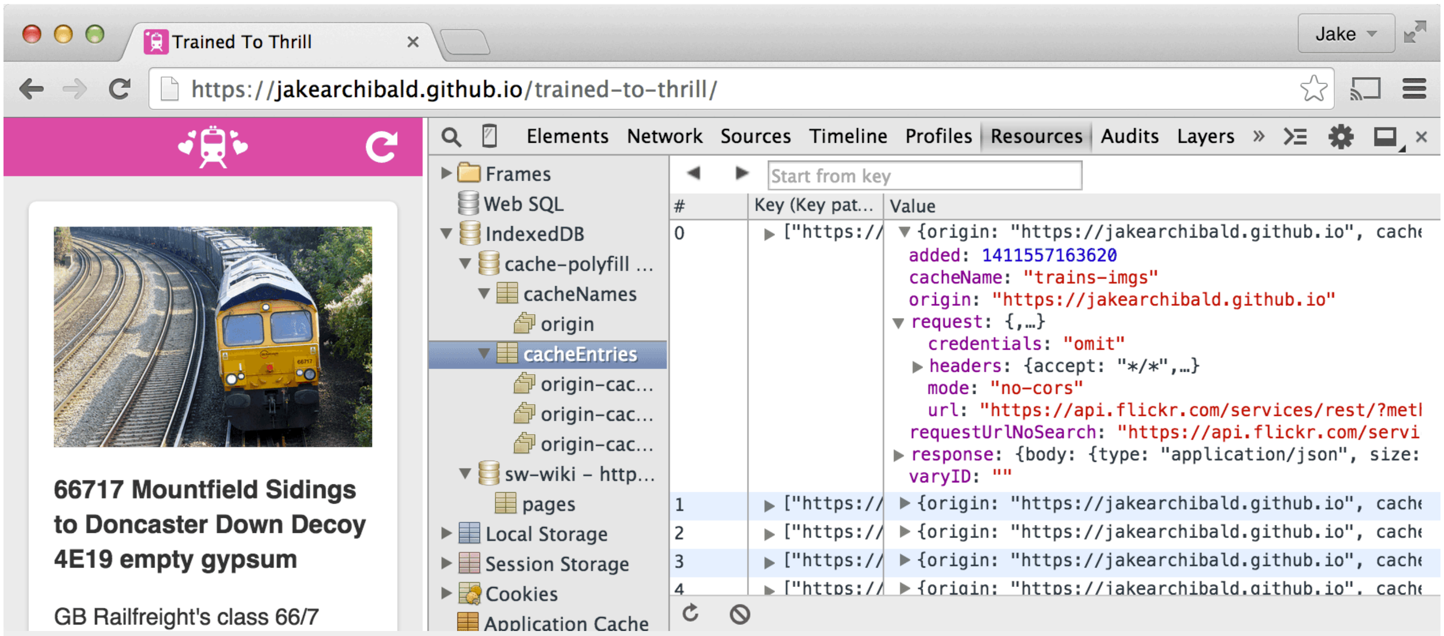
```
importScripts('serviceworker-cache-polyfill.js');

self.addEventListener('install', function (event) {
  // pre cache a load of stuff:
  event.waitUntil(
    cachesPolyfill.open('myapp-static-v1').then(function (cache) {
      return cache.addAll([
        '/',
        '/styles/all.css',
        '/styles/imgs/bg.png',
        '/scripts/all.js',
      ]);
    }),
  );
});

self.addEventListener('fetch', function (event) {
  event.respondWith(
    cachesPolyfill.match(event.request).then(function (response) {
      return response || fetch(event.request);
    }),
  );
});
```

Trained-to-thrill manages caches dynmanically and at install time to create its offline-first experience.

The cache polyfill is backed by IndexedDB, so you can kinda sorta use devtools to see what's in the cache.



https://jakearchibald.github.io/trained-to-thrill/

66717 Mountfield Sidings to Doncaster Down Decoy 4E19 empty gypsum

GB Railfreight's class 66/7

```
#   Key (Key pat...   Value
0   ▶ ["https://     ▼ {origin: "https://jakearchibald.github.io", cache
                        added: 1411557163620
                        cacheName: "trains-imgs"
                        origin: "https://jakearchibald.github.io"
                       ▼ request: {,…}
                          credentials: "omit"
                         ▶ headers: {accept: "*/*",…}
                          mode: "no-cors"
                          url: "https://api.flickr.com/services/rest/?metl
                        requestUrlNoSearch: "https://api.flickr.com/servi
                       ▶ response: {body: {type: "application/json", size:
                        varyID: ""
1   ▶ ["https://     ▶ {origin: "https://jakearchibald.github.io", cache
2   ▶ ["https://     ▶ {origin: "https://jakearchibald.github.io", cache
3   ▶ ["https://     ▶ {origin: "https://jakearchibald.github.io", cache
4   ▶ ["https://     ▶ {origin: "https://jakearchibald.github.io", cache
```

Cache polyfill in IndexedDB

Rough edges & gotchas

As I said earlier, this stuff is really new. Here's a collection of issues that get in the way. Hopefully I'll be able to delete this section in the coming weeks, in fact I've deleted loads since I originally posted this article.

fetch() doesn't send credentials by default

When you use `fetch`, those request won't contain credentials such as cookies. If you want credentials, instead call:

```
fetch(url, {
  credentials: 'include',
});
```

This behaviour is on purpose, and is arguably better than XHR's more complex default of sending credentials if the URL is same-origin, but omiting them otherwise. Fetch's behaviour is more like other CORS requests, such as `<img crossorigin>`, which never sends cookies unless you opt-in with `<img crossorigin="use-credentials">` However, I'm concerned this is going to catch developers out. Interested to hear your feedback on this!

fetch() is only available in ServiceWorkers

`fetch` should be available in pages, but Chrome's implementation isn't. The cache API should also be available from pages too, but the polyfill depends on `fetch`, so that too is ServiceWorker-only for the moment. Here's the ticket.

This got in the way a little when building Trained-To-Thrill. I wanted to fetch content from the cache while fetching the same content from the network (the search feed from Flickr). I did this by making two XHR requests, but adding a made-up header to one so I could tell them apart when they hit the ServiceWorker.

The cache polyfill cannot store opaque responses

```
self.addEventListener('fetch', function (event) {
  if (/\.jpg$/.test(event.request.url)) {
    event.respondWith(
      fetch('https://www.google.co.uk/….gif', {
        mode: 'no-cors',
      }),
    );
  }
});
```

The above works fine, even though the image doesn't have CORS headers. It's an opaque response, you can use it in response to requests that don't require CORS (such as `<img>`), but you cannot get access to the response's content with JavaScript.

You can also store these responses in the cache for later use, however this isn't possible with the polyfill, which depends on JavaScript access to the response.

No workarounds for this, we need to wait for the native implementation.

🟧 If installation fails, we're not so good at telling you about it

If a worker registers, but then doesn't appear in serviceworker-internals, it's likely it failed to install due an error being thrown, or a rejected promise being passed to `event.waitUntil`.

To work around this, check "Opens the DevTools window for ServiceWorker on start for debugging", and put a `debugger;` statement at the start of your install event. This, along with "Pause on uncaught exceptions", should reveal the issue.

Over to you

Developer feedback at this point is crucial. It lets us catch not only issues with our implementation, but also design issues with ServiceWorker itself.

File bugs against Chrome, bugs against the spec, or leave a comment below. Let us know what works & what doesn't!

View this page on GitHub

Comments powered by Disqus