Service worker registration

- Common registration boilerplate
- A user's first visit
- Improving the boilerplate
- Subsequent visits
- Reasons to register early
- Testing things out
- Conclusion

Service workers can meaningfully speed up repeat visits to your web app, but you should take steps to ensure that a service worker's initial installation doesn't degrade a user's first-visit experience.

Generally, deferring service worker registration until after the initial page has loaded will provide the best experience for users, especially those on mobile devices with slower network connections.

Common registration boilerplate

If you've ever read about service workers, you've probably come across boilerplate substantially similar to the following:

```
if ('serviceWorker' in navigator) {
    navigator.serviceWorker.register('/service-worker.js');
}
```

This might sometimes be accompanied by a few `console.log()` statements, or code that detects an update to a previous service worker registration, as a way of letting users know to refresh the page. But those are just minor variations on the standard few lines of code.

So, is there any nuance to `navigator.serviceWorker.register`? Are there any best practices to follow? Not surprisingly (given that this article doesn't end right here), the answer to both is "yes!"

A user's first visit

Let's consider a user's first visit to a web app. There's no service worker yet, and the browser has no way of knowing in advance whether there will be a service worker that is eventually installed.

As a developer, your priority should be to make sure that the browser quickly gets the minimal set of critical resources needed to display an interactive page. Anything that slows down retrieving those responses is the enemy of a speedy time-to-interactive experience.

Now imagine that in the process of downloading the JavaScript or images that your page needs to render, your browser decides to start a background thread or process (for the sake of brevity, we'll assume it's a thread). Assume that you're not on a beefy desktop machine, but rather the type of underpowered mobile phone that much of the world considers their primary device. Spinning up this extra thread adds contention for CPU time and memory that your browser might otherwise spend on rendering an interactive web page.

An idle background thread is unlikely to make a significant difference. But what if that thread isn't idle, but instead decides that it's also going to start downloading resources from the network? Any concern about CPU or memory contention should take a backseat to worries about the limited bandwidth available to many mobile devices. Bandwidth is precious, so don't undermine critical resources by downloading secondary resources at the same time.

All of this is to say that spinning up a new service worker thread to download and cache resources in the background can work against your goal of providing the shortest time-to-interactive experience the first time a user visits your site.

Improving the boilerplate

The solution is to control start of the service worker by choosing when to call `navigator.serviceWorker.register()`. A simple rule of thumb would be to delay registration until after the `load event` fires on `window`, like so:

```
if ('serviceWorker' in navigator) {
    window.addEventListener('load', function() {
    navigator.serviceWorker.register('/service-worker.js');
    });
}
```

But the right time to kick off the service worker registration can also depend on what your web app is doing right after it loads. For example, the Google I/O 2016 web app features a short animation before transitioning to the main screen. Our team found that kicking off the service worker

registration during the animation could lead to jankiness on low-end mobile devices. Rather than giving users a poor experience, we delayed service worker registration until after the animation, when the browser was most likely to have a few idle seconds.

Similarly, if your web app uses a framework that performs additional setup after the page has loaded, look for a framework-specific event that signals when that work is done.

Subsequent visits

We've been focusing on the first visit experience up until now, but what impact does delayed service worker registration have on repeat visits to your site? While it might surprise some folks, there shouldn't be any impact at all.

When a service worker is registered, it goes through the `install` and `activate` lifecycle events. Once a service worker is activated, it can handle `fetch` events for any subsequent visits to your web app. The service worker starts *before* the request for any pages under its scope is made, which makes sense when you think about it. If the existing service worker weren't already running prior to visiting a page, it wouldn't have a chance to fulfill `fetch` events for navigation requests.

So once there's an active service worker, it doesn't matter when you call `navigator.serviceWorker.register()`, or in fact, *whether you call it at all*. Unless you change the URL of the service worker script, `navigator.serviceWorker.register()` is effectively a no-op during subsequent visits. When it's called is irrelevant.

Reasons to register early

Are there any scenarios in which registering your service worker as early as possible makes sense? One that comes to mind is when your service worker uses `clients.claim()` to take control of the page during the first visit, and the service worker aggressively performs runtime caching inside of its `fetch` handler. In that situation, there's an advantage to getting the service worker active as quickly as possible, to try to populate its runtime caches with resources that might come in handy later. If your web app falls into this category, it's worth taking a step back to make sure that your service worker's `install` handler doesn't request resources that fight for bandwidth with the main page's requests.

Testing things out

A great way to simulate a first visit is to open your web app in a Chrome Incognito window, and look at the network traffic in Chrome's DevTools. As a web developer, you probably reload a local instance of your web app dozens and dozens of times a day. But by revisiting your site when there's already a service worker and fully populated caches, you don't get the same experience that a new user would get, and it's easy to ignore a potential problem.

Here's an example illustrating the difference that registration timing could make. Both screenshots are taken while visiting a sample app in Incognito mode using network throttling to simulate a slow connection.

Network traffic with early registration.

The screenshot above reflects the network traffic when the sample was modified to perform service worker registration as soon as possible. You can see precaching requests (the entries with the gear icon next to them, originating from the service worker's `install` handler) interspersed with requests for the other resources needed to display the page.

Network traffic with late registration.

In the screenshot above, service worker registration was delayed until after the page had loaded. You can see that the precaching requests don't start until all the resources have been fetched from the network, eliminating any contention for bandwidth. Moreover, because some of the items we're precaching are already in the browser's HTTP cache—the items with `(from disk cache)` in the Size column—we can populate the service worker's cache without having to go to the network again.

Bonus points if you run this sort of test from an actual, low-end device on a real mobile network. You can take advantage of Chrome's remote debugging capabilities to attach an Android phone to your desktop machine via USB, and ensure that the tests you're running actually reflect the real-world experience of many of your users.

Conclusion

To summarize, making sure that your users have the best first-visit experience should be a top priority. Delaying service worker registration until after the page has loaded during the initial visit can help ensure that. You'll still get all the benefits of having a service worker for your repeat visits.

A straightforward way to ensure to delay your service worker's initial registration until after the first page has loaded is to use the following:

```
if ('serviceWorker' in navigator) {
    window.addEventListener('load', function() {
    navigator.serviceWorker.register('/service-worker.js');
```

```
    });
}
```

□□□ #3
□ □□□ □□□ 2026-10-31 22:15:47 UTC
□ cocobook-sync □□□ 2026-11-04 17:38:01 UTC