Service Workers: Introduction and Learning Resources

Marco Antonio Moreno Maya

28/02/2024

What are Service Workers?

Service Workers are JavaScript scripts that run in the background of the browser and act as proxies between the web application, the browser, and the network. They enable control over how network requests and cache resources are managed, allowing the creation of web applications capable of functioning offline, improving performance, and providing push notifications, among other features.

Resources to Learn About Service Workers:

Researching these sources will provide you with a solid understanding of Service Workers and how to effectively implement them in your web applications.

Users expect apps to start on slow or flaky network connections, or even when offline. They expect the content they've most recently interacted with, such as media tracks or tickets and itineraries, to be available and usable. When a request isn't possible, they expect the app to tell them instead of silently failing or crashing. And users wish to do it all quickly. As we can see in this study Milliseconds make millions, even a 0.1 second improvement in load times can improve conversion by up to 10%. In summary: users expect PWAs to be reliable and that's why we have service workers.

Hello Service Workers

A service worker acts as a middleware proxy, running device-side, between your PWA and servers, which includes both your own servers and cross-domain servers.

When an app requests a resource covered by the service worker's scope, including when a user is offline, the service worker intercepts the request, acting as a network proxy. It can then decide if it should serve the resource from the cache via the Cache Storage API, from the network as normally would happen without a service worker, or create it from a local algorithm. This lets you

provide a similar experience to that provided by a platform app. It can even work entirely offline.

Not all browsers support service workers. Even when present your service worker won't be available on first load or while it's waiting to activate. Therefore, treat it as optional and do not require it for core functionality.

Registering a Service Worker

Before a service worker takes control of your page, it must be registered for your PWA. That means the first time a user comes to your PWA, network requests will go directly to your server because the service worker is not yet in control of your pages.

After checking if the browser supports the Service Worker API, your PWA can register a service worker. When loaded, the service worker sets up shop between your PWA and the network, intercepting requests and serving the corresponding responses.

```
if ('serviceWorker' in navigator) {
   navigator.serviceWorker.register("/serviceworker.js");
}
```

Note: There is only one service worker per PWA, but that doesn't mean you need to place the code only in one file. A service worker can include other files using importScripts in every browser or using ECMAScript module imports in some modern browsers.

Verify if a Service Worker is Registered

To verify if a service worker is registered, use developer tools in your favorite browser.

In Firefox and Chromium-based browsers (Microsoft Edge, Google Chrome, or Samsung Internet):

- Open developer tools, then click the Application tab.
- In the left pane, select Service Workers.
- Check that the service worker's script URL appears with the status "Activated". (You'll learn what this status means in the lifecycle section in this chapter). On Firefox the status can be "Running" or "Stopped".

In Safari:

- Click the Develop menu then the Service Workers submenu.
- Check that an entry with the current origin appears in the submenu. It opens an inspector over the service worker's context.

Scope

The folder your service worker sits in determines its scope. A service worker that lives at example.com/my-pwa/sw.js can control any navigation at the my-pwa path or below, such as example.com/my-pwa/demos/. Service workers can only control items (pages, workers, collectively "clients") in their scope. Scope applies to browser tabs and PWA windows.

Only one service worker per scope is allowed. When active and running, only one instance is typically available no matter how many clients are in memory (such as PWA windows or browser tabs).

Warning: You should set the scope of your service worker as close to the root of your app as possible. This is the most common setup as it lets the service worker intercept all the requests related to your PWA. Don't put it inside, for instance, a JavaScript folder or have it loaded from a CDN.

Lifecycle

Service workers have a lifecycle that dictates how they are installed, this is separate from your PWA installation. The service worker lifecycle starts with registering the service worker. The browser then attempts to download and parse the service worker file. If parsing succeeds, its install event is fired. The install event only fires once.

Service worker installation happens silently, without requiring user permission, even if the user doesn't install the PWA. The Service Worker API is even available on platforms that do not support PWA installation, such as Safari and Firefox on desktop devices.

Service worker registration and installation, while related, are different events. Registration happens when a page requests a service worker by calling register() as described previously. Installation happens when a registered service worker exists, can be parsed as JavaScript, and doesn't throw any errors during its first execution.

After the installation, the service worker is not yet in control of its clients, including your PWA. It needs to be activated first. When the service worker is ready to control its clients, the activate event will fire. This doesn't mean, though, that the page that registered the service worker will be managed. By default, the service worker will not take control until the next time you navigate to that page, either due to reloading the page or re-opening the PWA.

You can listen for events in the service worker's global scope using the self object.

```
// This code executes in its own worker or thread
self.addEventListener("install", event => {
   console.log("Service worker installed");
});
self.addEventListener("activate", event => {
```

```
    console.log("Service worker activated");
});
```

Updating a Service Worker

Service workers get updated when the browser detects that the service worker currently controlling the client and the new (from your server) version of the same file are byte-different.

Warning: When updating your service worker, do so without renaming it. Do not even add file hashes to the filename. Otherwise, the browser will never get the new version of your service worker!

After a successful installation, the new service worker will wait to activate until the existing (old) service worker no longer controls any clients. This state is called "waiting", and it's how the browser ensures that only one version of your service worker is running at a time. Refreshing a page or reopening the PWA won't make the new service worker take control. The user needs to close or navigate away from all tabs and windows using the current service worker and then navigate back. Only then will the new service worker take control.

Service Worker Lifespan

Once installed and registered, a service worker can manage all network requests within its scope. It runs on its own thread, with activation and termination controlled by the browser. This lets it work even before or after your PWA is open. While service workers run on their own thread, there is no guarantee that in-memory state will persist

Conclusion

Service Workers are a powerful technology that enables developers to create rich, engaging web applications with offline capabilities and enhanced performance. By learning how to use Service Workers effectively, developers can unlock new possibilities for web development and deliver better user experiences.

[web.dev, (2021, MDN Web Docs, (2024, February 8]

References

MDN Web Docs. Service worker api - web apis, (2024, February 8). URL https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API.

web.dev. Service workers, (2021). URL https://web.dev/learn/pwa/service-workers.