SVELTEKIT • ADVANCED

Service workers

ON THIS PAGE

Service workers act as proxy servers that handle network requests inside your app. This makes it possible to make your app work offline, but even if you don't need offline support (or can't realistically implement it because of the type of app you're building), it's often worth using service workers to speed up navigation by precaching your built JS and CSS.

In SvelteKit, if you have a `src/service-worker.js` file (or `src/service-worker/index.js`) it will be bundled and automatically registered. You can change the location of your service worker if you need to.

You can disable automatic registration if you need to register the service worker with your own logic or use another solution. The default registration looks something like this:

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

Inside the service worker

Inside the service worker you have access to the `$service-worker` module, which provides you with the paths to all static assets, build files and prerendered pages. You're also provided with an app version string, which you can use for creating a unique cache name, and the deployment's `base` path. If your Vite config specifies `define` (used for global variable replacements), this will be applied to service workers as well as your server/client builds.

other requests as they happen. This would make each page work offline once visited.

```
/// <reference types="@sveltejs/kit" />
import { build, files, version } from '$service-worker';

// Create a unique cache name for this deployment
const CACHE = `cache-${version}`;

const ASSETS = [
  ...build, // the app itself
  ...files  // everything in `static`
];

self.addEventListener('install', (event) => {
  // Create a new cache and add all files to it
  async function addFilesToCache() {
    const cache = await caches.open(CACHE);
    await cache.addAll(ASSETS);
  }

  event.waitUntil(addFilesToCache());
});

self.addEventListener('activate', (event) => {
  // Remove previous cached data from disk
  async function deleteOldCaches() {
    for (const key of await caches.keys()) {
      if (key !== CACHE) await caches.delete(key);
    }
  }

  event.waitUntil(deleteOldCaches());
});

self.addEventListener('fetch', (event) => {
  // ignore POST requests etc
  if (event.request.method !== 'GET') return;

  async function respond() {
    const url = new URL(event.request.url);
    const cache = await caches.open(CACHE);
```

```
      if (response) {
        return response;
      }
    }

    // for everything else, try the network first, but
    // fall back to the cache if we're offline
    try {
      const response = await fetch(event.request);

      // if we're offline, fetch can return a value that is not a Response
      // instead of throwing - and we can't pass this non-Response to respondWith
      if (!(response instanceof Response)) {
        throw new Error('invalid response from fetch');
      }

      if (response.status === 200) {
        cache.put(event.request, response.clone());
      }

      return response;
    } catch (err) {
      const response = await cache.match(event.request);

      if (response) {
        return response;
      }

      // if there's no cache, then just error out
      // as there is nothing we can do to respond to this request
      throw err;
    }
  }

  event.respondWith(respond());
});
```

Be careful when caching! In some cases, stale data might be worse than data that's unavailable while offline. Since browsers will empty caches if they get too full, you should also be careful about caching large assets like video files.

Docs

The service worker is bundled for production, but not during development. For that reason, only browsers that support modules in service workers will be able to use them at dev time. If you are manually registering your service worker, you will need to pass the `{ type: 'module' }` option in development:

```js
import { dev } from '$app/environment';

navigator.serviceWorker.register('/service-worker.js', {
  type: dev ? 'module' : 'classic'
});
```

`build` and `prerendered` are empty arrays during development

Type safety

Setting up proper types for service workers requires some manual setup. Inside your `service-worker.js`, add the following to the top of your file:

```js
/// <reference types="@sveltejs/kit" />
/// <reference no-default-lib="true"/>
/// <reference lib="esnext" />
/// <reference lib="webworker" />

const sw = /** @type {ServiceWorkerGlobalScope} */ (/** @type {unknown} */ (self));
```

```ts
/// <reference types="@sveltejs/kit" />
/// <reference no-default-lib="true"/>
/// <reference lib="esnext" />
/// <reference lib="webworker" />

const sw = self as unknown as ServiceWorkerGlobalScope;
```

This disables access to DOM typings like `HTMLElement` which are not available inside a reference to the SvelteKit types ensures that the `$service-worker` import has proper type definitions. If you import `$env/static/public` you either have to `// @ts-ignore` the import or add `/// <reference types="../.svelte-kit/ambient.d.ts" />` to the reference types.

Other solutions

SvelteKit's service worker implementation is deliberately low-level. If you need a more full-fledged but also more opinionated solution, we recommend looking at solutions like Vite PWA plugin, which uses Workbox. For more general information on service workers, we recommend the MDN web docs.