Speed up Service Worker with Navigation Preloads

By Jake Archibald
Human boy working on web standards at Google

TL;DR

- In some situations, service worker boot-up time can delay a network response.
- A new experimental feature, navigation preload, fixes this by allowing you to make the request in parallel with service worker boot-up.
- You can distinguish preload requests from regular navigations using a header, and serve different content.
- Navigation preload is in Chrome 59 Canary behind a flag, and the API may change in response to developer feedback.
- It'll remain behind a flag in Chrome 59 stable (likely to be released in March), but you can apply for an origin trial to test it with real users.

Note: An earlier version of this article stated that navigation preload was in Chrome 57; however, the feature was delayed to Chrome 59.

The problem

When you navigate to a site that uses a service worker to handle fetch events, the browser asks the service worker for a response. This involves booting up the service worker (if it isn't already running), and dispatching the fetch event.

The bootup time depends on the device and conditions. It's usually around 50ms. On mobile it's more like 250ms. In extreme cases (slow devices, CPU in distress) it can be over 500ms. However, since the service worker stays awake for a browser-determined time between events, you only get this delay occasionally, such as when the user navigates to your site from a fresh tab, or another site.

The boot-up time isn't a problem if you're responding from the cache, as the benefit of skipping the network is greater than the boot-up delay. But if you're responding using the network…

…the network request is delayed by the service worker booting-up.

We're continuing to reduce the boot-up time by using code-caching in V8, by skipping service workers that don't have a fetch event, by launching service workers speculatively, and other optimizations. However, bootup time will always be greater than zero.

Facebook brought the impact of this problem to our attention, and asked for a way to perform navigation requests in parallel:

…and we said "yeah, seems fair".

"Navigation preload" to the rescue

Navigation preload is a new experimental feature that lets you say, "Hey, when the user makes a GET navigation request, start the network request while the service worker is booting up".

The startup delay is still there, but it doesn't block the network request, so the user gets content sooner.

Here's a video of it in action, where the service worker is given a deliberate 500ms startup delay using a while-loop:

Here's the demo itself. To get the benefits of navigation preload, you'll need Chrome 59 or later with **`chrome://flags/#enable-service-worker-navigation-preload`** enabled.

Activating navigation preload

```
addEventListener('activate', event => {
  event.waitUntil(async function() {
    // Feature-detect
    if (self.registration.navigationPreload) {
      // Enable navigation preloads!
      await self.registration.navigationPreload.enable();
    }
  }());
});
```

You can call `navigationPreload.enable()` whenever you want, or disable it with `navigationPreload.disable()`. However, since your `fetch` event needs to make use of it, it's best to enable/disable it in your service worker's `activate` event.

Using the preloaded response

Now the browser will be performing preloads for navigations, but you still need to *use* the response:

```
addEventListener('fetch', event => {
  event.respondWith(async function() {
    // Respond from the cache if we can
    const cachedResponse = await caches.match(event.request);
    if (cachedResponse) return cachedResponse;

    // Else, use the preloaded response, if it's there
    const response = await event.preloadResponse;
    if (response) return response;

    // Else try the network.
    return fetch(event.request);
  }());
});
```

`event.preloadResponse` is a promise that resolves with a response, if:

- Navigation preload is enabled.
- The request is a `GET` request.
- The request is a navigation request (which browsers generate when they're loading pages, including iframes).

Otherwise `event.preloadResponse` is still there, but it resolves with `undefined`.

Warning: Don't enable navigation preload then forget to use it. If you use `fetch(event.request)` instead of `event.preloadResponse`, you'll end up with double requests for navigations.

Custom responses for preloads

If your page needs data from the network, the quickest way is to request it in the service worker and create a single streamed response containing parts from the cache and parts from the network.

Say we wanted to display an article:

```
addEventListener('fetch', event => {
  const url = new URL(event.request.url);
  const includeURL = new URL(url);
  includeURL.pathname += 'include';

  if (isArticleURL(url)) {
    event.respondWith(async function() {
      // We're going to build a single request from multiple parts.
      const parts = [
        // The top of the page.
        caches.match('/article-top.include'),
        // The primary content
        fetch(includeURL)
          // A fallback if the network fails.
          .catch(() => caches.match('/article-offline.include')),
        // The bottom of the page
        caches.match('/article-bottom.include')
      ];

      // Merge them all together.
      const {done, response} = await mergeResponses(parts);
      // Wait until the stream is complete.
      event.waitUntil(done);
      // Return the merged response.
      return response;
    }());
  }
});
```

In the above, `mergeResponses` is a little function that merges the streams of each request. This means we can display the cached header while the network content streams in.

This is quicker than the "app shell" model as the network request is made along with the page request, and the content can stream without major hacks.

However, the request for `includeURL` will be delayed by the service worker's startup time. We can use navigation preload to fix this too, but in this case we don't want to preload the full page, we want to preload an include.

To support this, a header is sent with every preload request:

```
Service-Worker-Navigation-Preload: true
```

The server can use this to send different content for navigation preload requests than it would for a regular navigation request. Just remember to add a `Vary: Service-Worker-Navigation-Preload` header, so caches know that your responses differ.

Now we can use the preload request:

```
// Try to use the preload
const networkContent = Promise.resolve(event.preloadResponse)
  // Else do a normal fetch
  .then(r => r || fetch(includeURL))
  // A fallback if the network fails.
  .catch(() => caches.match('/article-offline.include'));

const parts = [
  caches.match('/article-top.include'),
  networkContent,
  caches.match('/article-bottom')
];
```

Note: `Promise.resolve(event.preloadResponse)` means we get a promise for undefined if `event.preloadResponse` is undefined. It's a good way to normalize behaviour with browsers that don't support `event.preloadResponse`.

Changing the header

By default, the value of the `Service-Worker-Navigation-Preload` header is `true`, but you can set it to whatever you want:

```
navigator.serviceWorker.ready.then(registration => {
  return registration.navigationPreload.setHeaderValue(newValue);
}).then(() => {
  console.log('Done!');
});
```

You could, for example, set it to the ID of the last post you have cached locally, so the server only returns newer data.

Getting the state

You can look up the state of navigation preload using `getState`:

```
navigator.serviceWorker.ready.then(registration => {
  return registration.navigationPreload.getState();
}).then(state => {
  console.log(state.enabled); // boolean
  console.log(state.headerValue); // string
});
```

Use it on live sites today!

We're still experimenting with this feature, but we're looking for real-world feedback. To get feedback, we're making it available as an origin trial starting in Chrome 59. Origin trials allow you to temporarily enable the feature for users of your website, so you can test its real-world impact. To do this, you'll need to request a token for your origin, and include the token as a header on your pages and service worker:

```
Origin-Trial: token_obtained_from_signup
```

To avoid the problems we saw with vendor prefixes, origin trials are globally shut off if usage exceeds 0.03% of all Chrome page loads, so large sites should only enable the feature for a fraction of its users.

If you do experiment with this feature, you can send us feedback on our mailing list, or join the spec discussion. If you find any bugs, please file an issue, including the words "service worker navigation preload" in the issue Summary.

Many thanks to Matt Falkenhagen and Tsuyoshi Horo for their work on this feature, and help with this article. And a huge thanks to everyone involved in the standardization effort