Measuring Performance in a Service Worker

By Paul Kinlan

Paul is a Developer Advocate

Other than Jake Archibald worrying about his Developer skills rotting and falling off, he made a strong case that by intelligently using service worker you can drastically improve the performance of your site or app. Watch the video for an overview.

If you are going to Supercharge your Page load time as Jake suggests, you really need to be able to understand how service workers affect your page's requests.

The Resource Timing and the User Timing API are critcial components in many sites RUM (Real User Monitoring) infrastructure because it lets you understand holistically how all of your users see the performance of your site (Another use-case is detecting content injection). In short, it lets you understand nearly every aspect of every web request made from your site, well, unless you have a service worker or a Web worker.

Here is a quick example of how it can be used in to get a list of all requests that were made to a domain that is not the current domain.

```
var getThirdPartyRequests = function() {
  var thirdPartyRequests = [];
  var requests = window.performance.getEntriesByType("resource");

  var currentHost = window.location.host

  for(var requestIdx = 0; requestIdx < requests.length; requestIdx++) {
    var request = requests[requestIdx];
    var url = new URL(request.name);
    var host = url.host;

    if(host != currentHost) {
      thirdPartyRequests.push(request);
    }
  }

  return thirdPartyRequests;
};
```

The Resource Timing and User Timing APIs were created and implemented before service worker was a twinkle in an engineers eye and the above code would not be able to understand how the Service Woker impacted it.

A recent set of changes in Chrome 45 (Beta in July 2015) will help you by introducing the ability for all forms of workers (Web and service worker) to have access to the Resource Timing and User Timing APIs and thus be able to let you keep on top of the network performance for all your users.

Accessing Performance Metrics from a Service Worker

The biggest change is the addition of the `performance` object into a Workers context (Web and ServiceWorkers) that will now let you understand the performance timings of all the requests that are made in the context of the worker and will also let you set your own marks for measurment of JS execution. If you can only see what happens from the context of the current window you would miss critical timing information from:

- `fetch()` requests made inside the `oninstall` event of the service worker
- `fetch()` requests made when caching data in an `onpush` event can now be traced to understand the performance your users see.
- finally, `fetch()` request that are made and intercepted by the `onfetch` handler.

The last point is important. One way to think of a service worker is as a proxy that sits between the web UI and the network. The `performance` object on the `window` only sees the timings and information for the part of the request that it invokes, it has no knowledge of the service worker sitting in between the client and the network so it can't understand the impact of the service worker.

How can I use this?

A typical service worker that support offline first would have an install step where it will download and save all the assets for later use

An example of where this could be used is to record and log the timing data of the install step so you can make some measured decisions about how you can improve the performance of your install based on real in the wild user usage.

```
self.addEventListener("install", function() {
  var urls = [
    '/',
    '/images/chrome-touch-icon-192x192.png',
    '/images/ic_add_24px.svg',
    '/images/side-nav-bg@2x.jpg',
    '/images/superfail.svg',
    '/scripts/voicememo-core.js',
    '/styles/voicememo-core.css',
    '/third_party/Recorderjs/recorder.js',
    '/third_party/Recorderjs/recorderWorker.js',
    '/third_party/Recorderjs/wavepcm.js',
    '/third_party/moment.min.js',
    '/favicon.ico',
    '/manifest.json'
  ];

  urls = urls.map(function(url) {
    return new Request(url, {credentials: 'include'});
  });

  event.waitUntil(
    caches
      .open(CACHE_NAME + '-v' + CACHE_VERSION)
      .then(function(cache) {
        // Fetch all the URL's and store in the cache
        return cache.addAll(urls);
      })
      .then(function () {
        // Analyse all the requests
        var requests = self.performance.getEntriesByType("resource");

        // Loop across all the requests and save the timing data.
        return;
      })
  );
});
```

Today many sites use RUM for understanding how the majority of their users experience their site. Tools like Google Analytics already report site speed data using the Navigation Timing API but will need to be updated to include performance analysis from a Worker context.

Will the Navigation Timing API arrive to Service Workers

Right now there are no plans to add the Navigation Timing API to the service worker context, because there are no traditional navigations in a service worker. The interesting thing is that to the service worker, every navigation in the service worker's controlled set of pages looks like a resource fetch. This alone makes service workers an incredibly compelling way to centralize the majority of your performance logic in your web app.

Can I see what changed?

- crbug.com/465640
- crbug.com/465641
- crbug.com/465643

I am interested in the discussion and specs:

- https://groups.google.com/a/chromium.org/forum/#!topic/blink-dev/htsW078UcFA
- https://w3c.github.io/resource-timing/