V8 full code cache in CacheStorage for PWA

This Document is Public

Authors: horo@chromium.org
Link to this document: goo.gl/Pffqo7

One-page overview

Summary

Generates the full V8 code cache of the scripts which are stored to CacheStorage while the Service Worker is being installed. The scripts in HTTPCache and the scripts in CacheStorage which are stored by active (already installed) Service Workers are out of the scope.

Platforms

Mac, Windows, Linux, Chrome OS, Android

Team

worker-dev@chromium.org

Bug

ImplementationBug: crbug.com/768705 LaunchBug: crbug.com/788621

Code affected

Service workers and CacheStorages in blink and content.

Design

There are three storage types where V8 code cache can be stored:

- HTTPCache (Doc / Bug)
    - From Chrome 42, the V8 code cache mechanism was enabled for HTTPCache. Chrome generates the code cache for HTTPCache when a script file is executed twice in 72 hours.
- ServiceWorkerStorage (Doc / Bug)
    - From Chrome 42, the code cache of Service Worker scripts (including imported scripts) are generated while installing the Service Worker and stored to the ServiceWorkerStorage.
- CacheStorage (Doc / Bug)
    - From Chrome 53, Chrome generates the code cache for the script which are served from a CacheStorage to the page via a ServiceWorker when the script is executed in the page for the first time.

As of now, V8 generates the code cache using the "lazy" mode parser which doesn't parse the whole script codes, but parses the codes which V8 assumes is important using V8 heuristics such as toplevel code and IIFE (immediately invoked function expression). This has led to complaints that code caching is less effective than expected.

So V8 team introduced a new flag to generate the full code cache and tried to do the field experiment of it (Doc / Bug). But the new flag caused a regression in the performance test. This means that the full code generation in the page has a negative impact on the page load time.

To mitigate the risk of causing the regression, the scope of this project are limited to the scripts which are stored to CacheStorage while the Service Worker is being installed. And the code cache is generated on the Service Worker thread, so there is no direct negative impact on the main thread.

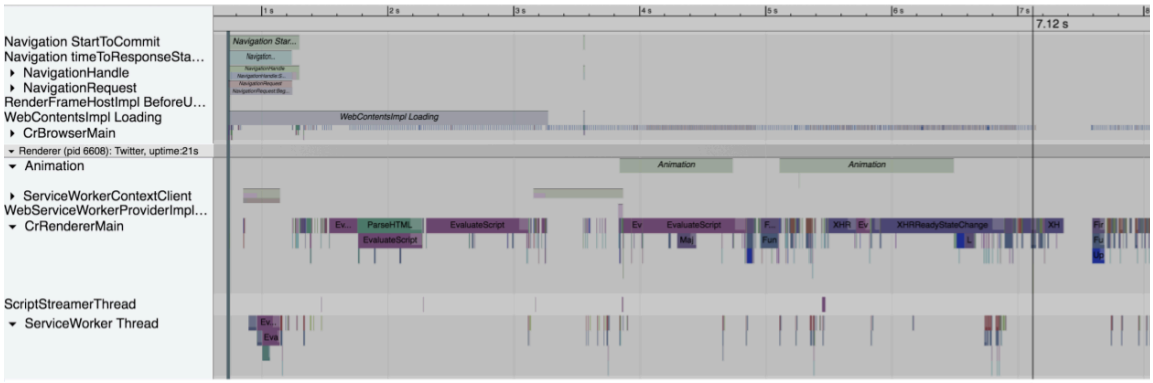
(We will do another experiment of the full code caching for service worker scripts to measure the impact of full code caching of service worker scripts)

Local experiment

horo@ implemented an experimental patch (CL), and briefly tested on Nexus 4 using Twitter Lite (https://mobile.twitter.com) in MTV office.

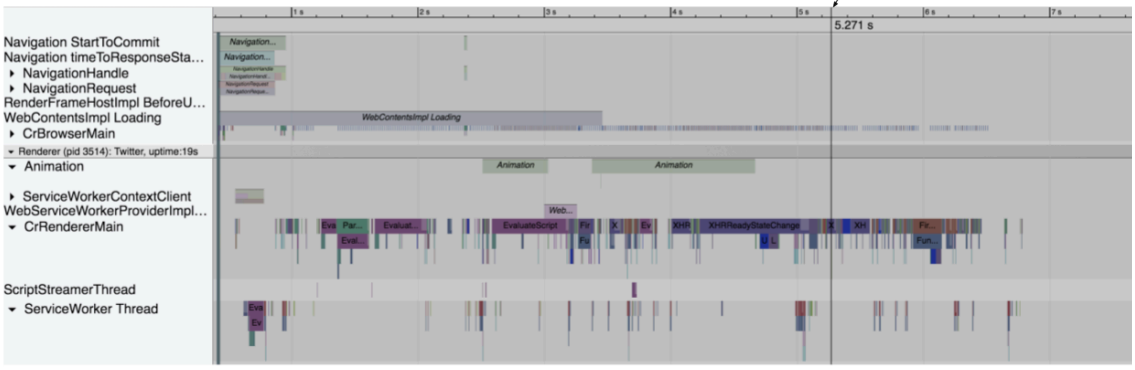
1. FMP (First meaningful paint) time including SW startup was improved from 6.38 sec to 4.84 sec.

Normal code cache (--service-worker-code-cache-install=normal) FMP=6.38sec



FMP (First Meaningful Paint)

Full code cache (--service-worker-code-cache-install=full) FMP=4.84 sec



(When horo@ experimented 10 times in Tokyo office, the improvement was 6.82 sec => 5.60 sec in average.)

2. The generated code cache size became larger. So this may cause delay especially on slow IO and disk encryption enabled devices.
   Example: https://abs-0.twimg.com/responsive-web/web/ltr/main.97189a37ea3dd857.js
   Original JS file : 464,130 byte
   Normal code cache : 139,492 byte
   Full code cache : 1,297,348 byte
3. The install event (when the code cache is generated) took time.
   No code cache : 1.38 s
   Full code cache : 7.42 s

Implementation Plan

We will introduce a new Feature flag “PWAFullCodeCache” and implement behind the flag. After finishing the implementation we will do the field trial experiment and measure the impact of this feature.

Metrics

To measure the impact on the page load, we will monitor the existing metrics.

- PageLoad.Clients.ServiceWorker.ParseTiming.NavigationToParseStart
- PageLoad.Clients.ServiceWorker.PaintTiming.NavigationToFirstContentfulPaint
- PageLoad.Clients.ServiceWorker.PaintTiming.ParseStartToFirstContentfulPaint
- PageLoad.Clients.ServiceWorker.Experimental.PaintTiming.NavigationToFirstMeaningfulPaint
- PageLoad.Clients.ServiceWorker.Experimental.PaintTiming.ParseStartToFirstMeaningfulPaint
- PageLoad.Clients.ServiceWorker.DocumentTiming.NavigationToDOMContentLoadedEventFired
- PageLoad.Clients.ServiceWorker.DocumentTiming.NavigationToLoadEventFired

To collect the information of the installed scripts, we will add new metrics.

- ServiceWorker.CacheStorageInstalledScript.Count
- ServiceWorker.CacheStorageInstalledScript.ScriptSize
- ServiceWorker.CacheStorageInstalledScript.ScriptTotalSize
- ServiceWorker.CacheStorageInstalledScript.CachedMetadataSize
- ServiceWorker.CacheStorageInstalledScript.CachedMetadataTotalSize

To collect the memory impact, we added new metrics.

- Memory.Experimental.Renderer.*.ServiceWorkerControlledMainFrame.*

Rollout plan

Field trial experiment in M64/65 Canary,Dev. If there is no regression, we will do the field trial Beta and Stable. And then if there is no regression, we'll enable it by default in M65.

Core principle considerations

Speed

The script evaluation will be faster. But the metadata size on the desk will be larger so the script loading may become lower. And the memory usage on the renderer process may be larger.

Security

If a malicious software can rewrite the code cache, the page could behave unexpectedly. But this risk already exists even without this feature.

Privacy considerations

N/A

Testing plan

We have some LayoutTests running with PWAFullCodeCache feature using VirtualTestSuites. And also we will update fieldtrial_testing_config.json.

Followup work

We have to clean up the PWAFullCodeCache flags.

Concerns and Limitations

- When a new Chrome version is installed, the existing full code cache in CacheStorage become outdated. And the new normal (not full) code cache is generated when the script is executed in the page. So this may cause performance regression after Chrome update until a new Service Worker installs the new script.
- According to marja@'s report, the code cache sizes seem reasonable (3-5 x the source code size). But should we avoid storing the code cache when low disk space quota?
- ServiceWorker can't know in what charset encoding the script will be handled in the page (eg: <script charset="UTF-8">). If the page uses the different charset for the script, the code cache created in the ServiceWorker will be wasted.
- Currently we don't have plans to generate the full code cache for ES6 modules in ServiceWorker. But ServiceWorker can't know whether the file should be loaded as a module or as a normal script. So ServiceWorker tries to generate full code cache as a script even if the file is a module.
- We will not generate the full code cache in active Service Worker, because FetchEvents from the pages are blocked while generating the code cache. But to provide the Web developers with a method to intentionally generate code cache without installing a new Service Worker, it may be OK to generate the code cache when cache.put() is called in the page.