Reusing ServiceWorker processes for navigation

clamy@chromium.org, horo@ - 2017-June-15

Status

Implemented: CL

Background

The general interaction between ServiceWorkers, navigations and renderer processes is described in this doc. This doc describes an issue between PlzNavigate and ServiceWorker.

Before r474395, the ServiceWorker for the navigation would create a new process instead of using the one we created speculatively for the navigation. This led us to implement two maps of RenderProcessHosts that track which pending or committed sites are expected/hosted by a RenderProcessHost. We introduced the ProcessReusePolicy::REUSE_PENDING_OR_COMMITTED_SITE that we use when we create ServiceWorker SiteInstances. When we create a process for them, we attempt to reuse a process that is currently hosting a frame of the same site, or that is expecting a navigation to the same site based on the two maps. This solves the issue of sharing a process between navigation and ServiceWorker when the navigation process is created **before** the ServiceWorker needs to start.

However, iff the ServiceWorker needs to start before a RenderFrameHost associated with the corresponding navigation has been created, it will create a new process in which to execute. With the way renderer process creation for navigation is currently set up, when the navigation is ready to commit, we will create a new renderer process to commit it. This is wasteful as the ServiceWorker and its navigation could execute in the same-process. This document proposes a way to reuse the process the ServiceWorker created for the commit of the navigation.

Navigation SiteInstances are created with a default ProcessReuse policy. By default, Chrome creates processes for each SiteInstance and doesn't proactively reuse processes until we exceed the process count limit. When over the limit, Chrome reuses a process randomly and not based on site. Other than this mechanism, navigations can reuse an existing process when they use the same SiteInstance as another navigation. This happens when the navigations are in the same BrowsingInstance and have the same site URL. However, when creating the SiteInstance for the navigation we do not have access to the SiteInstance of the ServiceWorker. Thus we can't use the BrowsingInstance to return the ServiceWorker SiteInstance to use for navigation, even if the ServiceWorker and the navigation have the same site URL. So we end

up creating two processes (one for ServiceWorker and one for navigation) even if we can execute them in the same process. The cost of process creation is high from a latency and memory usage point of view, especially on mobile devices. So we should attempt to reuse the ServiceWorker renderer process **to some extent** even if we don't exceed the process count limit.

The extent to which we can reuse the ServiceWorker renderer process is also limited. Putting all instances of a site into the renderer process with that site's ServiceWorker would be problematic, since it would lead to substantial contention between the various instances. In the end, we'd like to aim for one instance of the site to share a process with the ServiceWorker. Additional instances can be put in independent processes.

The existing problematic cases are the following ones.

Cross-site redirects in PlzNavigate

As explained above, we solved process reuse issue when PlzNavigate creates a process for navigation before the ServiceWorker is created.. But we still have an issue with cross-site redirects. Since PlzNavigate does not re-create a new process on redirects, when the cross-site redirect happens, we no longer have a suitable SiteInstance (and so no RenderProcessHost) to use for the redirected URL. In particular, if the redirected URL has a ServiceWorker, we will create a new render process for it to execute unless the process count limit has been reached (in which case we reuse a random one).

When the navigation commits, the SiteInstance we create for it will also create a new process unless the process limit has been reached. We would like it to reuse the process the ServiceWorker created instead.

Search provider pre-warm of ServiceWorker

Google Search is switching to using a ServiceWorker for their site, and they are complaining about the added latency that spawning a ServiceWorker adds to the page load. There is a proposal to spawn a renderer process and start the search provider ServiceWorker in it when we detect a user action in the omnibox that is likely to lead to a search. The goal is to have a ServiceWorker as close as ready to execute by the time the user actually start navigating to the search page.

However, the process we would create for the search ServiceWorker would again not be reused to commit the navigation to the search page, which is wasteful.

There is another problem with Android New Tab Page (NTP) and the pre-warm of ServiceWorker. The section "Android NTP and pre-warming ServiceWorker" describes it.

Tapping a Push Notification that triggers a navigation (bug)

When a PWA site such as Twitter Light sends a Push notification from the server and Chrome shows the notification, one renderer process is created for the service worker. And when the user taps the notification, another renderer process is created to show the related web contents. In this case we should also reuse the renderer process.

Proposal

We will update the RenderProcessHost creation code for SiteInstances so that by default it tries to reuse a RenderProcessHost that has been created to execute a ServiceWorker of the same origin and that has not been matched with a page from that site. A ServiceWorker has been matched with a page if:

1. It is put into a process because there's already a pending or committed page from that site.
2. A page is put into its process because the ServiceWorker was there and previously unmatched.

Note: once matched, the state doesn't change.

Reusing the RenderProcessHost of an unmatched ServiceWorker

In order to identify that a RenderProcessHost has an unmatched ServiceWorker, I suggest extending SiteInstance with a is_for_service_worker flag.

When a RenderProcessHost is returned from SiteInstance::GetProcess() for an instance that has the is_for_service_worker flag set, there are four possibilities:

- The RenderProcessHost has just been created. Then it is the RenderProcessHost of an unmatched ServiceWorker.
- The RenderProcessHost is an existing one that was returned as part of the implementation of ProcessReuse::REUSE_PENDING_OR_COMMITTED_SITE. Then this RenderProcessHost contains a matched ServiceWorker, since it also contains a pending or committed page of the same site.
- The RenderProcessHost is an existing one that was returned because the process limit has been reached. In that case, it is a RenderProcessHost of an unmatched ServiceWorker. While it already contains pages, it does **not** contain a pending or committed navigation of the **same** site as the ServiceWorker. Since we want the new ServiceWorker and its corresponding navigation to share the same process as much as possible, we should also use that process for the corresponding navigation.
- The RenderProcessHost is that of an unmatched ServiceWorker. Then there is still an unmatched ServiceWorker inside, because we have put a new ServiceWorker inside the process and not a page. This can happen during the soft update ServiceWorker

algorithm that can follow a “push” notification. The two versions of the ServiceWorker should share processes, and if the push notification leads to a navigation, the navigation should be put in the renderer process.

Once we have created a RenderProcessHost with an unmatched ServiceWorker inside, the ServerWorker becomes matched when its RenderProcessHost is used for a SiteInstance whose policy does not have the is_for_service_worker flag set. The RenderProcessHost no longer contains an unmatched ServiceWorker because a page has been put into it. Therefore, it should be taken out of consideration of the process reuse mechanism this document describes.

To track processes which contains an unmatched ServiceWorker, the BrowserContext will have a new map (ReusableServiceWorkerProcessTracker) of type “std::map<GURL, std::set<ProcessID>>”. When a new process is created for a service worker or a random process is returned due to the process limit being exceeded, the site url and the process ID is inserted to the map.

When it comes to the creation of the renderer process, we will modify the default process creation code for SiteInstances. Right now, it does the following:

1. Attempt to reuse a process following the SiteIsolation::ProcessReusePolicy.
2. If none are found, check if the process limit has been reached. If yes, reuse a random existing one.
3. Otherwise, create a new RenderProcessHost.

Following the modification, we will have:

1. Attempt to reuse a process following the SiteIsolation::ProcessReusePolicy.
2. If none are found, check ReusableServiceWorkerProcessTracker if we can reuse a RenderProcessHost that has been used for an unmatched ServiceWorker with the same site URL.
3. If none are found, check if the process limit has been reached. If yes, reuse a random existing one.
4. Otherwise, create a new RenderProcessHost.

This ensures that RenderProcessHosts with unmatched ServiceWorkers are considered regardless of the SiteInstance ProcessReusePolicy. In particular, this covers both the case of navigation and of two ServiceWorkers with different patterns that resolve to the same SiteURL.

Android NTP and pre-warming ServiceWorker

Currently navigating away from the Android NTP doesn't create a new process. Instead, the Android NTP has an unused renderer process hidden behind its native UI, and this process is

used for the subsequent navigation. This logic is implemented by ShouldAssignSiteForURL() which returns false for Android NTP. The SiteInstance of the page doesn't have a site URL, so RenderFrameHostManager::DetermineSiteInstanceForURL() reuses the SiteInstance of NTP for the next navigation.

If Chrome creates a new process for pre-warming ServiceWorker from omnibox before the navigation while showing the NTP, this can cause a performance regression. So we have to use the NTP's renderer process for it. Or we should not pre-warm the ServiceWorker form omnibox while showing the NTP.

Use NTP's SiteInstance for pre-warming ServiceWorker from omnibox on Android

There is an idea to reuse the NTP's renderer process for ServiceWorker:

1. When the omnibox tries to start the ServiceWorker by calling ServiceWorkerContext::StartServiceWorkerForNavigationHint(), pass the SiteInstance of NTP to ServiceWorkerContext as an candidate SiteInstance.
2. When the ServiceWorkerProcessManager needs to allocate a worker process, if the candidate SiteInstance doesn't have a site URL yet, use the SiteInstance for the ServiceWorker.
3. To avoid reusing the same SiteInstance for a navigation to other site after started to launching a ServiceWorker in race condition, RenderFrameHostManager::DetermineSiteInstanceForURL() will check whether the current SiteInstance has been already passed to the ServiceWorkerContext before reusing the SiteInstance for the navigation.

horo@ created a proof-of-concept patch for it.
https://chromium-review.googlesource.com/c/536535/