Reusing processes when creating a SiteInstance

clamy@chromium.org - 2017-Apr-19

Status

Implemented: process reuse for navigations without cross-site redirects: CL1, CL2, CL3,
Not implemented: process reuse for cross-site navigations.

Background

crbug.com/705318
SiteInstance is the core concept in Chrome which allows tracking which pages belong together and need to share a process. By default, creating a new SiteInstance for a particular URL will create a new renderer process. We'll use this process to render all pages belonging to the SiteInstance.

We also keep track of BrowsingInstances: groups of SiteInstances that are related together. This allows us to lookup a SiteInstance for a URL through SiteInstance::GetRelatedSiteInstance. When this function is used, we will attempt to find a SiteInstance whose URL matches the desired one among all the SiteInstances of the BrowsingInstance. If none is found, we will create a new SiteInstance (along with a new process).

Enter ServiceWorker. ServiceWorkers execute javascript in renderer processes. This means that the ServiceWorker should execute in an appropriate process. If an instance of the ServiceWorker is already running in a renderer process, then we'll use this ServiceWorker for all navigations to pages that it controls. Otherwise, the ServiceWorker needs to find a renderer process in which to start.

As ServiceWorkers are not pages, the concept of a SiteInstance applies badly to them. When navigating, the code of the ServiceWorkerProcessManager does not have enough information to access the SiteInstance that we expect the navigation to use: it does not know the ServiceWorker is starting because of navigation, nor does it know in which frame the navigation is happening. This is because ServiceWorkers can also be started in contexts very different from navigations. Using only the methods provided by SiteInstance to find a SiteInstance with an appropriate process, the start of a ServiceWorker would always end up creating a new SiteInstance and a new process. Since a ServiceWorker used by navigation could run in the

same process that will render the page (they have the same site), using a new process would be wasteful.

To avoid creating excess processes, the ServiceWorker code keeps track of the id of the RenderProcessHost of the page that requested the navigation. When starting the ServiceWorker, the ServiceWorkerProcessManager will simply try to use this RenderProcessHost. If it's not available, it will create a new one through SiteInstance::CreateForURL.

This breaks with PlzNavigate (aka the new navigation architecture). In PlzNavigate, we do not associate navigation requests with a RenderProcessHost until they are ready to commit. Since the ServiceWorker start can happen before that, the ServiceWorkerProcessManager will always end up creating a new process for a ServiceWorker used in navigations. This degrades performance when PlzNavigate is enabled.

Proposal

SiteInstance::SetUseExistingProcessIfPossible

We plan to introduce a new method in SiteInstance that will make it try to reuse an existing process instead of creating a new one. We only create a process for the SiteInstance when we call SiteInstance::GetProcess. This new method will be called before. For example, the ServiceWorker code to get a process to start a new ServiceWorker could look like that:

```
site_instance = SiteInstance::CreateForURL(service_worker_url);
site_instance->SetUseExistingProcessIfPossible(true);
site_instance->GetProcess();
```

Keep track of page's site in RenderProcessHost

To reuse RenderProcessHosts, we need to keep track of which sites a RenderProcessHost can render. These are the sites it has rendered already. In an ideal world where processes aren't costly, all RenderProcessHosts would only ever render one site, so finding a RenderProcessHost to use for our SiteInstance would be trivial. However, this is not the case: we do not switch processes in the vast majority of renderer-initiated navigations, and until SiteIsolation launches, we will still render subframes of a different site in their parent's process. Even then, many processes will contain pages from any different sites.

To keep track of all these sites, we will have to record the site of each navigation at commit time, and store it in the RenderProcessHost of the frame that just navigated. From there we have two possibilities:

1. We just keep a set of all the sites we have seen in this RenderProcessHost.

2. We refcount the sites based on the number of frames in the RenderProcessHost that are **currently** rendering a page from that site.

Option 1 is clearly simpler to implement. Since the set is constantly growing though, we run the risk of finding ourselves with a sort of “uber” process that we can use to display any kind of site, long after we’ve navigated away from them, instead of using a process that just rendered one site (which we would prefer in that case).

Option 2 doesn’t have this kind of issue, but refcounting is hard and prone to edge cases. In this case, we will have to be careful to decrement the refcount each time a frame is destroyed or navigates away.

Apart from tracking past or only current sites, we have another issue. How do we chose between two RenderProcessHosts that can both be used to render an site? The easiest answer is the first RenderProcessHost found, but it may decrease security. For example, let’s imagine that we keep track of all the urls that were rendered by the RenderProcessHost, and that we have a process that used to render foo.com, but is now rendering evil.com and has been compromised. All new tabs that we open and navigate to foo.com will have their ServiceWorker execute in the compromised renderer process. This is a downgrade compared to the current situation, where the ServiceWorker would execute in the RendererProcessHost of the main frame of the new tab.

If we keep track of all sites rendered by a RenderProcessHost, when looking for a RenderProcessHost to use for site foo.com, we should prioritize a RenderProcessHost that has **only** been used to render foo.com above RenderProcessHosts that have also rendered other sites. This will would make the kind of attacks described above less likely, but it is still only a heuristic.

If we only keep track of the current sites rendered by a RenderProcessHost, we can simply chose at random among eligible RenderProcessHosts.

Another point to keep in mind is performance. ServiceWorker’s current performance management prioritizes the process that is in the foreground and has the most frames. Creating a ServiceWorker in a process means that the process is foregrounded, because its requests needs to be processed without the throttling we apply to background tabs. We also need to keep alive the process while the ServiceWorker is executing, and had a rough assumption that a process with more frames may live longer.

All in all, I think the approach to track the sites currently rendered in a RenderProcessHost makes the most sense in the ServiceWorker context. To chose which one to use, we’ll look at foregrounded processes that can display the site we’re looking for and pick one at random. If there are no foreground eligible process, we’ll pick a background process one that can display the site at random. If we still can’t find any, we’ll have to create a new process.

The initial plan is not to have the number of frames heuristics. It's less clear that it is right/useful. We can land metrics if we are worried that dropping it will lead to too many processes being kept alive because of ServiceWorkers.

When to add an site to the RenderProcessHost

Initially, we suggested to add a site to the set of sites rendered in the RenderProcessHost at each commit of a navigation. This made it quite clear what the set of sites represented, but it does not solve the problem of creating too many processes for ServiceWorkers when PlzNavigate is enabled.

Two cases are problematic: navigations that require a new process and cross-site renderer-initiated navigations that do not swap processes. When the navigation requires a new process (e.g. new tab, cross-process navigation), the RenderProcessHost should keep track of the site of the navigation we expect to commit. Otherwise, when doing a navigation that requires a new process, we will end up creating two processes: the process created for the navigation will not be found when looking for a process for the ServiceWorker.

The issue is similar with the renderer-initiated cross-site navigations. We need to keep track of the fact that we expect the RenderProcessHost to be used for the commit of the navigation.

To sum it up, we also need to keep track of the sites of the navigations we expect to commit in the RenderProcessHost on top of the sites of the Documents currently rendered.

Note that the new process case can work in PlzNavigate because we create the speculative RenderFrameHost with a bound SiteInstance. When we move to a speculative SiteInstance model, we will have to assess carefully whether we can create a speculative SiteInstance that has no site URL initially (so that we can reuse it if the navigation is redirected). We probably don't want to reuse the process when the navigation is redirected cross-site if a ServiceWorker for the initial site executed inside it.

Cross-site redirects during navigation

Even with the previous solution, we may still end up creating a process for a ServiceWorker when PlzNavigate is enabled when we encounter a cross-site redirect. If a ServiceWorker needs to execute on the cross-site redirect, we may have to create a new process for the ServiceWorker, since it's likely that we won't have one for this new site.

We have two cases here. In the first one, the cross-site redirect will end up committing in a RenderFrameHost that already exists, including the one we selected for the initial URL. This happens on renderer-initiated navigations for example. I'm arguing that the ServiceWorker for

the cross-site redirect should be allowed to reuse this RenderProcessHost, since the navigation would eventually commit there. Note that this means we may end up using a RenderProcessHost for the ServiceWorker that is not associated with the same site as the SW (this is what is happening currently). Currently, we do not update the set of pending sites on each redirect, since we're waiting for the redirects to finish to pick which process to use.

To implement this, on each cross-site redirect, we will compute which SiteInstance should be used for the redirected URL. If it is an existing one, we will inform its RenderProcessHost to expect a navigation to the redirected URL.

If the SiteInstance is a brand new one, this corresponds to the case where we don't have an existing process we can use to commit the navigation. We could re-create a speculative RenderFrameHost on each cross-site redirect (when the navigation can swap processes). But the re-creation of a speculative on each cross-site redirect may lead to wasted resources when no ServiceWorker executes, so I don't plan on implementing it.

In practice, this means that the ServiceWorker will have to create a new process in that case. When the navigation eventually commits, with the way the process code is written today, it will not be able to use the process we created for the ServiceWorker. We plan to mitigate this by allowing new SiteInstances created for navigation to reuse processes that were only used by a ServiceWorker of matching site. So this should allow the commit of the redirected URL to reuse the process that was created for the SW to execute. See this doc[1] for the proposed ServiceWorker process reuse for navigation.

With all of this in place, the only case where ServiceWorkers would lead to an extra process creation is when we have several cross-site redirects. For example if we get: siteA.com -> siteB.com -> siteC.com, we will end up creating an extra process if siteB has a ServiceWorker. I think the performance impact of this case should be acceptable.

Site URL vs Origin

While Origins are what we use in other places of the code for security checks, this document uses site URLs (used by SiteInstances). creis@ mentions that we have to use Site URLs to handle the case of the New Tab Page. The New Tab Page has an initial navigation URL of a Google URL that uses a ServiceWorker. However this resolves to a site URL of chrome://newtab (that is used when creating its SiteInstance). The New Tab Page ServiceWorker URL has also been modified so that it site URL is also chrome://newtab. This ensures that this ServiceWorker ends up in the RendererProcessHost of the SiteInstance we created for the new tab.

[1] Note: this is still WIP

Additional considerations

When granting special privileges to a RenderProcessHost, such as bindings or file access, it should be taken out of the list of RenderProcessHosts we may use for ServiceWorkers.

Process-per-site mode

This API could help simplify the process-per-site mode. In the process-per-site mode, we can register specific URLs in order to ensure that we only create a single process for them. Current use cases include the NTP, some WebUI pages and some hosted apps.

The current implementation stores these urls in a struct in the Chrome implementation, along with the RenderProcessHosts we should use to display them. When asked for a RenderProcessHost to use for those process-per-site urls, we build a list of suitable RenderProcessHosts by querying over the list of all RenderProcessHosts and asking the ChromeContentClient if they are suitable. Then we select one at random among the list of eligible RenderProcessHosts.

Since this mirrors the proposed implementation of SiteInstanceImpl::SetUseExistingProcessIfPossible, we could simply create SiteInstanceImpls with SetUseExistingProcessIfPossible to true when a url needs to use the process-per-site mode. We would then remove all the process registration code for process-per-site.