Migrating Fields out of DidCommitProvisionalLoad_Params

fergal@ rakina@
2020-09-07

Background

After the browser sends CommitNavigation to the renderer, it waits for the DidCommitProvisionalLoad_Params IPC before making the speculative render frame host current. Everything in that IPC is based on information known to the browser at the time it sends the CommitNavigation.

dcheng@ is working on https://crrev.com/c/1956814 which will ensure that browser commits turn into renderer commit.

Goal

Since all the information is available to the browser, we can calculate everything we need in the browser and we would not need to wait for the reply before switching render frame hosts. This resolves a bunch of potential race conditions related to actions performed on the current host while the RPC is sent but not yet replied. It also should allow for the removal of a lot of logic from the renderer (e.g. chunks of DocumentLoader) and maybe simplification of logic in the browser (e.g. history and NavigationType logic).

The end state should be that DidCommitProvisionalLoad_Params has no fields or has been removed entirely (if we have eliminated all cases of failure to commit (or turned them into CHECKs) I'm not sure the IPC serves any purpose anymore, receiving it could be a sanity check but it's unclear when you could react to it's non-arrival).

How

As always, the danger is the unknowns. We can't really be sure that we've got this all correct by just reading code and migrating logic. Even with all the tests passing it's very possible for us to have get things wrong. So, each field in the response we can go through the following stages:

1. DCHECK_EQ - the field's value is calculated in the browser and and compared the IPC

value, all tests pass with this DCHECK_EQ
2. DumpWithoutCrashing - behind a flag (1 flag per parameter), mismatches cause DumpWithoutCrashing in released binaries. We enable these in the usual way for a flag (canary, dev, …) and confirm that we do not see dumps. Debugging the dumps may be difficult since the stack will not really reveal much about what led us to that point but we can attach CrashKeys.
3. Delete from renderer - When we are confident that the browser version is correct, start passing the computed value in the CommitPending IPC so that the renderer can use this instead of calculating it itself. This enables deletion of code and removal from DidCommitProvisionalLoad_Params

We have a sheet which lists the fields in the RPC params with some notes on what is needed to calculate them on the browser side.

Possible blockers

Error page handling

Removing error page commits within the renderer

Document.open

As pointed out in a comment from dcheng@, document.open is specced to cancel ongoing navigations

- https://html.spec.whatwg.org/multipage/dynamic-markup-insertion.html#document-open-steps
- https://github.com/whatwg/html/issues/3447

Also some bugs that might depend on this

- https://bugs.chromium.org/p/chromium/issues/detail?id=763106#c53
- https://bugs.chromium.org/p/chromium/issues/detail?id=1099193

Currently this is done via blink::FrameLoader::StopAllLoaders. StopAllLoaders stops loading in all subframes too (this agrees with the spec, I think) and resets the NavigationClients.
What happens if a renderer-initiated navigation has already committed on the browser side at this point?

There are 2 distinct cases to consider:

- racy: the navigation is triggered in 1 task and then document.open is called in another. This is racy, the navigation could reach the commit stage before the cancellation makes it to the browser. By just accepting the commit if it comes, even after the document.open,

we just widen the window for races slightly

- non-racy: this seems solvable but the cost might not be acceptable
    - We could delay outgoing navigation requests until the end of the task, clearing them if document.open is called. This would slightly delay the start of renderer initiated navigations.
    - we could avoid this in most cases with extra complication of sending the navigation requests to the browser immediately but requiring a second "go ahead, document.open wasn't called" signal before they can be committed. It seems like this signal would arrive at the browser long before it was ready to commit, only when the JS task is long (10s or 100s of ms) would the commit actually be blocked waiting for it.

We also need to consider subframes (including remote ones). These could also have commits arriving after document.open but that's OK since the document.open is going to result in the destruction of these frames.