Navigation changes to support multiple pages in WebContents

This Document is Public

Authors: carlscab@google.com
Jul 2020

One-page overview

This document explores what changes are necessary to the navigation code to support Portals and pre-rendering in a way that BackForwardCache (bfcache) also benefits from them. A key goal here is to make sure that these features can share as much as possible while keeping a consistent architecture so that we minimize the complexity of supporting them in the long term. This document does not go into feature specific details and leaves several related questions unanswered that will need to be addressed in feature specific detailed designs.

What all these features have in common is that a single WebContents instance will now host multiple pages. In the case of BackForwardCache and pre-rendering there will be one page being shown and multiple hidden ones that might be frozen in the case they are bfcached (prerendering might want to freeze them as well once the page is loaded). For Portals there will be a hierarchical structure of pages with one page at the root. For all these features we want to be able to replace or “navigate” the page at the root to one of the other pages.

So we are looking at two major changes:

1. Allow non WebContents objects to own FrameTree and NavigationController instances. Portals and pre-renderer would own these.
2. Provide a way to transfer RenderFrameHost instances from one FrameTree to another. Portal activation would transfer the page from one of the leaves to the main page, pre-render would activate the page in the background and in the case of bfcache the page would be stored in the cache and we would need to move it into the tree.

Code affected

- content/browser/renderer_host
- content/browser/web_contents

Design

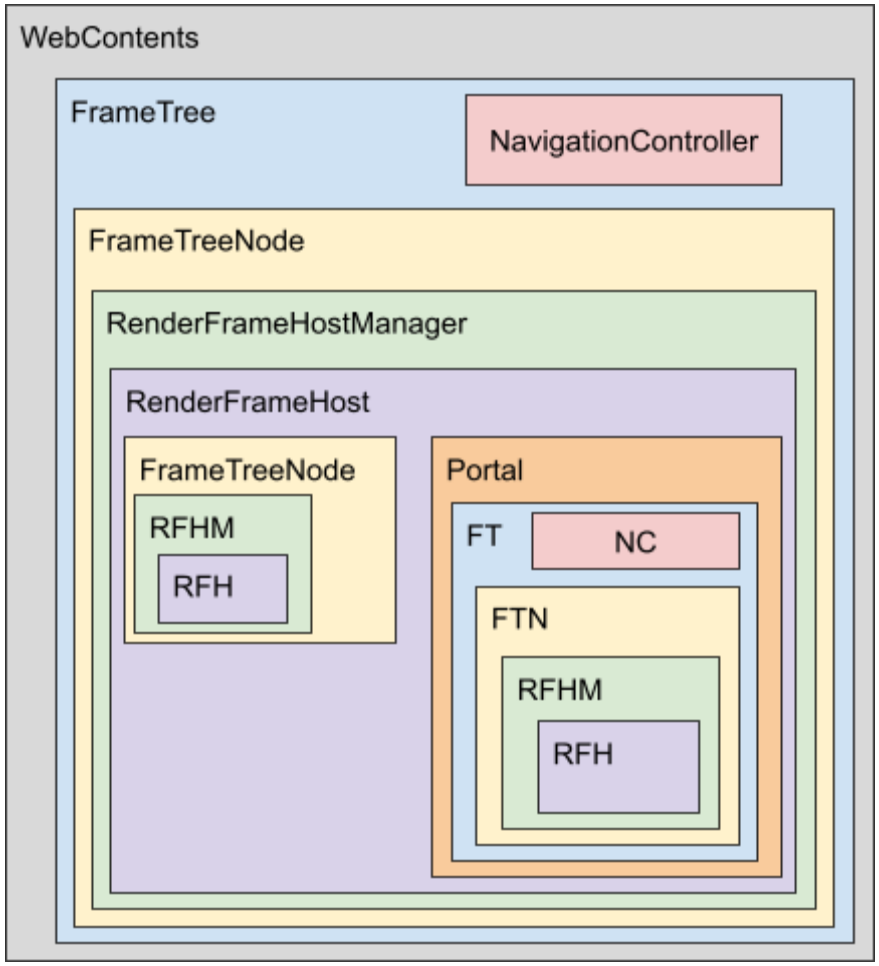
Multiple FrameTree instances per WebContents

WebContentsImpl currently owns a FrameTree and a NavigationController instance and implements a few related delegates: RenderFrameHostDelegate, RenderFrameHostManager::Delegate, RenderViewHostDelegate, RenderWidgetHostDelegate, and NavigationControllerDelegate.

Instead we want to support having multiple FrameTree instances in a WebContents. There is one FrameTree per page and there can be multiple FrameTree instances in a tab. There will always be one FrameTree instance owned by the WebContents (as is the case today) and there could be nested ones in case of Portals (the FrameTree will be owned by the RenderFrameHost that holds the Portal), and for prerendering there will be one visible (in WebContents) and one hidden for the page being pre-rendered. We will need to support navigations in these new FrameTrees which means we will need to commit navigation in them so it is required that we give them each a NavigationController instance to be able to classify commits and subsequent subframe commits. Thus we want to make FrameTree now own its NavigationController. These additional NavigationController will not be exposed to embedders and the primary session history for the tab will still be kept in the top most NavigationController (the one owned by the FrameTree owned by WebContents).

So for an example HTML snippet:

```
<body>
<portal></portal>
<iframe></iframe>
</body>
```



Portal now owns a FrameTree instead of a WebContents (as it currently does)

We need to rethink the various delegates and some of their methods will change behaviour depending on the FrameTree instance they belong to. In particular some of the events dispatched via these delegates will need to be routed to the WebContents, ignored, or require special handling depending on whether they come from the top level FrameTree (the one owned by WebContents) or not.

NavigationControllerDelegate

This one seems easy, FrameTree would implement this interface as it owns the NavigationController instance and most of the functionality required is best served at the FrameTree level. Still some of the events reported via this interface will need to be routed to the WebContents if this is the FrameTree owned by WebContents.

Required state controlled by an owner of the FrameTree

- RenderViewHost* GetRenderViewHost()
    - Remove, can be implemented entirely in renderer_host/.

- const std::string& GetContentsMimeType()
    - Is now a property of RVH and should probably be moved to RFH as this is a per document property
- FrameTree* GetFrameTree()
- IsHidden()

Ignored if not root, routed to WebContents otherwise
- void NotifyNavigationStateChanged(InvalidateTypes changed_flags)
- void NotifyBeforeFormRepostWarningShow()
- void NotifyNavigationEntryCommitted(const LoadCommittedDetails& load_details)
- void NotifyNavigationEntryChanged(const EntryChangedDetails& change_details)
- void NotifyNavigationListPruned(const PrunedDetails& pruned_details)
- void NotifyNavigationEntriesDeleted()
- bool ShouldPreserveAbortedURLs()
    - Return false if not at root

Can be removed, no longer used by NavigationController
- bool CanOverscrollContent() const

No changes needed (ie.e route to WebContents)
- void NotifyNavigationStateChanged(InvalidateTypes changed_flags)
- void UpdateOverridingUserAgent()
- bool IsBeingDestroyed()

Refactor needed
- WebContents* GetWebContents()
    - Maybe time to remove this layering violation (https://crbug.com/1076947)
- void SetHistoryOffsetAndLength(int history_offset, int history_length)
    - Route to correct renderers. Can be done at the FrameTree level. Special handling needed for inner/outer WebContents
- bool HasAccessedInitialDocument()
    - Only called from NavigationControllerImpl::IsUnmodifiedBlankTab which we will need to refactor as we now have multiple instances of NavigationControllerImpl in one tab
    - WebContents::has_accessed_initial_document_ will need to be moved to FrameTree so that the property can be moved along when pages change FrameTree..
- void Stop()
    - Stop all navigation in current FrameTree and all nested ones (e.g. Stop in top level should also stop navigation in portals). Stop in Prerenderer should not stop the pop level page, nor the other way around. Route to WebContents to send events to WebContentsObserver instances.
- void ActivateAndShowRepostFormWarningDialog()

- Needs to be routed to WebContents but augmented with NavigationController reference so that the result of the dialog can be routed back to the right controller.
    - We might be able to just ignore this for nested pages (Portals, BFCache, Prerenderer don't seem to need this)
    - We need to protect against controller instances disappearing so we probably need to pass a WeakPtr.
    - If the controller disappears while the dialog is showing we should just ignore the result (frame is also going away so there is no repost needed). Closing the dialog might not be a good option from the UX perspective.Should we also close the dialog if the controller disappears?

RenderFrameHostDelegate

Most of the methods pass a reference to the RenderFrameHost. There are some that are only called for the main frame (e.g. OnDidBlockNavigation), or that might not make sense for nested FrameTrees (e.g. DidCancelLoading, DidAccessInitialDocument).

RenderViewHostDelegate

Most of the methods pass a reference to the RenderViewHost. There are some that might not make sense for nested FrameTrees (e.g. Activate).

Some need to be closely looked at like GetFrameTree() specially given that the contract states that the returned FrameTree is guaranteed to be constant for the lifetime of the RenderViewHost

Given that frame_host and render_host have been merged we can reevaluate if some of the methods can be moved to other delegate interfaces.

RenderFrameHostManager::Delegate

Most of the methods pass a reference to the RenderViewHost or RenderFrameHost. There are some that might not make sense for nested FrameTrees (e.g. SetFocusToLocationBar). Other will need special treatment depending on whether they are in a nested FrameTree or not (GetControllerForRenderManager)

RenderWidgetHostDelegate

Many methods pass a reference to the RenderWidgetHostImpl. Some will need special treatment depending the FrameTree they belong to (GetCurrentPageUkmSourceId), some should not really change behaviour (IsFullscreen()). Some need to be closely looked at like GetFrameTree().

FrameTree instances can be attached to a WebContents, attached to one of the RenderFrameHost instances in the frame tree (e.g. Portals) or held separately or detached from the frame tree (e.g. pre-rendering, we need to load pages but not render them). WebContentsImpl will now need to check what FrameTree a call is coming from and behave accordingly. Instead of adding all this logic to WebContentsImpl we could have a new SuperDelegate class that implements all these interfaces. This class would change the FrameTree it is associated with as pages move between holders and change its behaviour accordingly. Note that the association to /content/browser/*_host classes do not change as this class is moved along with the RFH and RFHP instances. This SuperDelegate would be a nice to have from a code health perspective but not strictly required.

FrameTree Hierarchy

FrameTrees can now form a hierarchy with one at the root. We need to expose an API to walk frames across FrameTrees. The API will be exposed on the RenderFrameHostImpl as a new method:

```
RenderFrameHostImpl* RenderFrameHostImpl::GetOuterFrame()
RenderFrameHostImpl* RenderFrameHostImpl::GetParentOrOuterFrame()
std::vector<RenderFrameHostImpl*>
RenderFrameHostImpl::GetInnerFramesInSubtree()
```

The current ones:

```
RenderFrameHostImpl*
RenderFrameHostImpl::ParentOrOuterDelegateFrame()
std::vector<RenderFrameHost*>
RenderFrameHostImpl::GetFramesInSubtree()
```

```
We will try to deprecate the current ones and have them replaced by
the new ones hiding the fact that sometimes the nested frames are in
a nested WebContents, that would keep the interface the same once we
get rid of nested WebContents.
```

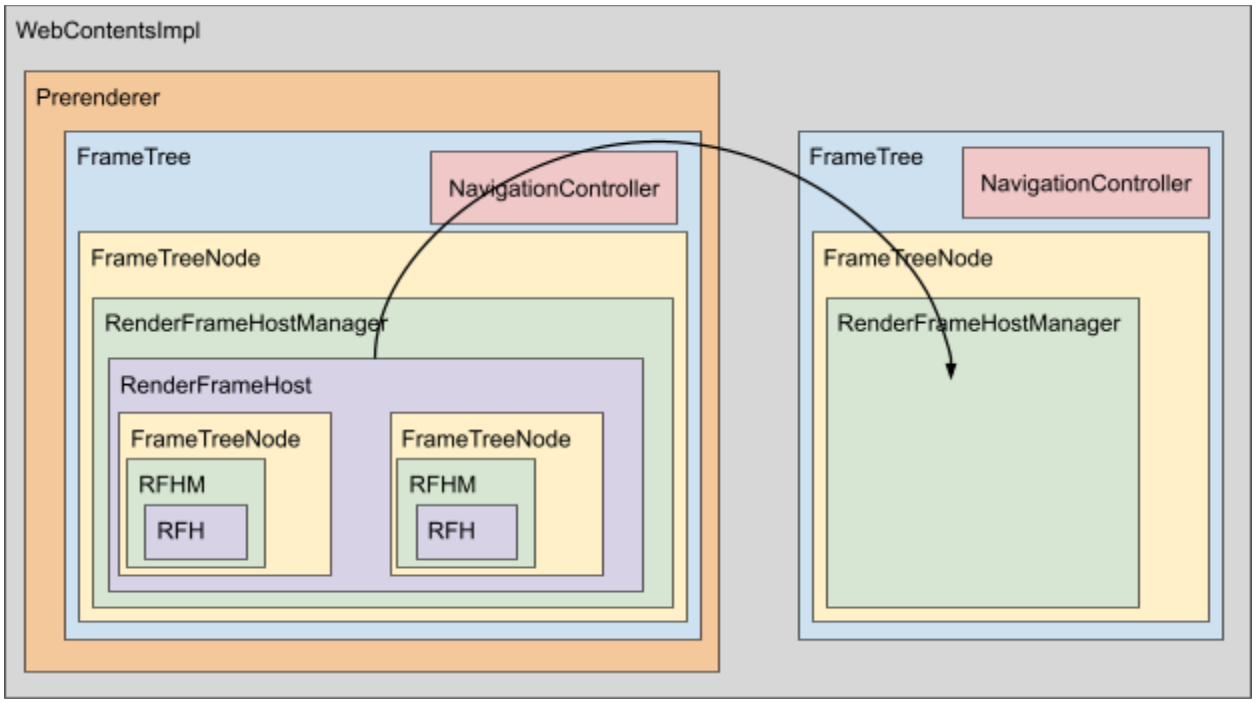
We will try to deprecate the current ones and have them replaced by the new methods hiding the fact that sometimes the nested frames are in a nested WebContents. This way we do not need to change the interface when we get rid of nested WebContents.

Moving Pages

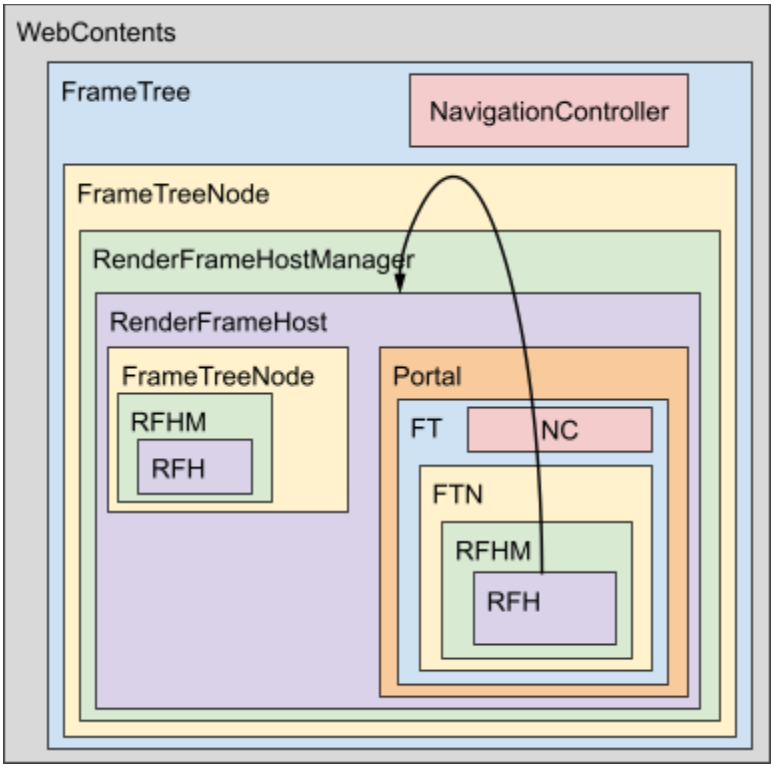
In order to support Portal activation (page in the portal becomes the top level page), navigation to a pre-rendered page (page rendered on the side becomes top level page),

back navigation from cache (cached page becomes top level page), we need to be able to move Pages into and out of FrameTree instances.

We want to accomplish this by moving RenderFrameHost instances in the root FrameTreeNode from one FrameTree to another. This is currently mostly supported by the fact that only the root FrameTreeNode is owned by the FrameTree and all children are owned by their parent RenderFrameHost. So by moving the RenderFrameHost at the root we move the entire tree. This property is currently used by bfcache to store entire frame trees.



Example of pre-rendered page being moved to the root FrameTree



Portal activation

There is some page related state that is currently held in the FrameTree (e.g. load_progress), FrameTreeNode (e.g. FrameReplicationState) and RenderFrameHostManager (e.g. RenderFrameProxyHostMap). As we move pages around we will need to move this state too. For FrameTreeNode and RenderFrameHostManager only the state of the instances at the root of the tree need to be moved as all other instances are moved along with the root RenderFrameHost. It would make sense to move all this state to a new object BrowsingInstanceFrameState (name tbd) that is owned by RenderFrameHost, that way the state will move along with the RenderFrameHost. Alternatively we would just need to collect all this information (window.name, owner_type, etc) and move it along with the RenderFrameHost instance (similar to what BackForwardCacheImpl::Entry is doing). Once this is in place, BackForwardCacheImpl::Entry can be replaced by this new entry. As this is related to the work being carried out for FrameGroup we will look into this in a separate design doc.

For some of the page moves between FrameTrees we will also need to transfer the last committed NavigationEntry from the source NavigationController to the destination one. E.g. In portal activation we will move the current Portal NavigationEntry to the NavigationController in the top most FrameTree (the one owned by WebContents i.e the primary one).

Navigation Support

There are two big pieces of work to support the new type of navigations where we transfer a page between FrameTrees. The actual navigation code, and making sure that existing code supports FrameTree / FrameTreeNode changes.

Navigation flow changes

Basically we want to generalize the current bfcache back navigation to a new type of navigation that moves pages from one FrameTree to another.

We will generalize the current BackForwardCache flow

1. NavigationRequest is created that knows whether this navigation will be using an existing RenderFrameHost.
2. Navigation starts and runs the regular throttles
    a. This is needed as some implement security checks or restrictions that need to happen
3. We run a no op NavigationURLLoader (like CachedNavigationURLLoader)
4. NavigationRequest::CommitNavigation now determines that this is a special navigation
    a. Calls a new method on RenderFrameHostManager that sets speculative_frame_host to point to the RFH that is being moved into the FrameTree. That way the RFH will be consumed during RenderFrameHostManager::CommitPending can be reused.
    b. Calls RenderFrameHost::Commit{Portal, BFcache, Pre-Render}
5. RenderFrameHost::Commit* performs any feature specific work needed
    a. bfcache: Send unfreeze message to Renderers
    b. Portals: Perform some activation related work
6. DidCommit can happen inline as it is just a swap
    a. This will call RenderFrameHostManager::CommitPending and finalize the swap.

We will need to address the following issues (most are feature related so will depend on what the specific feature requires):

- Adoption involves 2 renderers and the browser exchanging IPCs so we will need to take care of possible races. For example with bfcache we had to take care of various races (e.g. browser activates page while renderer sends evict message). We will need to carefully consider these wrt Portals, bfcache, prerender and maybe add some common infrastructure to the navigation code to deal with them.
    - IPC can arrive before or after the activation
    - The Portals API might just “tolerate” those races

        - Maybe we need to add an extra step to the basic navigation flow described above and let the renderers reach a state where it is safe to commit so NavigationRequest::CommitNavigation would trigger all that work and if successful a new task would run NavigationRequest::DidCommitNavigation.
            - We are currently exploring this possibility for bfcache navigations.
- CommitNavigation swaps RenderFrameHost instances similarly to bfcache.
    - No need to send a CommitNavigationIPC to the renderers, as the swap happens browser side.
    - We will need to send some IPC though to update the lifecycle state on the render side (eg. bfcache sends unfreeze)
- What happens with the source FrameTree
    - It will be left empty (not RFH) which should be fine as it will be deleted soon after
    - Alternatively it might be simpler to create an empty RenderFrameHost (without creating a renderer for it).
- The activated portal might adopt the old page
    - We need to keep the old page around until the portal activate event completes.
        - During this time the old page is no longer current (we have already committed the navigation)
        - What is the old page allowed to do while waiting to be adopted or not.
            - Do we need to attach it to a temporary page holder? Could we reuse the source one?
    - If it is not adopted we can either move it into the bfcache or destroy it.
    - Is this adoption a new navigation?
        - Do we need to expose it to embedders?
- What happens if there are inflight navigations when we activate?
    - Navigation in old page -> cancel navigation if possible, otherwise fail activation
    - Navigation in portal -> cancel navigation and restart

Remove constant FrameTree FrameTreeNode associations

We have the following potentially troublesome references (without / with tests):

`RenderFrameHostImpl::frame_tree_node_`	114
`RenderFrameHostImpl::frame_tree_node()`	195 / 304
`RenderFrameHostImpl::frame_tree_`	9
`RenderFrameHostImpl::frame_tree()`	25 / 26
`NavigationControllerDelegate::GetFrameTree`	28 / 532

`RenderViewHostDelegate::GetFrameTree`	27 / 529
`RenderWidgetHostDelegate::GetFrameTree`	22 / 530

Tests probably do not need to be looked at, as most of the existing ones will not exercise the code that changes FrameTree FrameTreeNode pointers.

Ideally we would get rid of all RenderFrameHostImpl:frame_tree accesses as this would also help with the bugs where rfh->frame_tree_node()->current_frame_host() != rfh because we have navigated away (example bug). This is an issue even without portals or bfcache.

These are some of the uses we have identified so far:

- Getting a frame tree node id
    - Tricky because it's exposed in a public API RenderFrameHost::GetFrameTreeNode (~78 / 132 usages). We could remove it as observers can track the same "node" with RenderFrameHostChanged.
    - Alternative is to expose it only for subframes (e.g. one of the observers uses FTN id for tracking navigations in ad subframes) and return -1 for main frames (only main frames will change FTN).
    - Root frame FTN is used (sometimes) as WebContents id, can expose it directly instead.
- Walking the children
    - Child nodes are now owned by RenderFrameHost so we could query them directly.
- Selecting the focused frame
    - Should move to WebContents-level (one focused frame per tab) and the appropriate delegate.
- Calculating depth
    - Is a constant for RFHI, can be passed into or computed in RFHI's constructor.
- GettingProxyToParent from render_manager — the parent is still constant, should be null for non-current frames (otherwise it can point to the wrong frame).
- Navigation management
    - Cancel / restart navigations in the given node which could be moved to a small dedicated interface
- Tracking the loading state
    - Loading state is currently tracked per RFH and the FrameTree just aggregates across all of them. This functionality could be moved to FrameTree or to a helper method that takes a RFH and just walks the tree.
- Querying ongoing navigation request in FTN
    - NavigationRequest should be passed explicitly (e.g. to functions like DispatchBeforeUnload).

- Querying GetFrameOwnerElementType / HasTransientUserActivation / FeaturePolicy / AdFrame type
    - Should be transferred in RenderFrameHostChanged
- Accessing VirtualAuthenticator, which is per-frame tree node
    - Figure out why this is the case. Probably has to be per-RFH (with additional transfers if needed).

content/public changes

We are not expecting any changes to //content/public related to these navigation core changes. Some of the planned features that build on top of this work will though. For example Prerendering might want to expose a property on NavigationRequest to flag a navigation as “prerendering activation” and expose an API to start such navigations.

Implementation Plan

We do not expect any of these changes to be gated behind a dedicated MPArch flag. Code changes needed to support multiple FrameTrees will mostly consist of refactors and should all be no ops for the current code. Similarly for page moves we will need to move the state currently stored in FrameTreeNode to a different location so it can move along the page. These changes should also be no ops of the current code. The new code paths to perform the actual page moves do not really need to be gated by a dedicated flag as this will be up to the actual feature using it (Portals, Prerender) to enter those new code paths, and thus the features themselves will have flags for this.

Alternatives considered

Swapping in FrameTreeNodes instead of RenderFrameHosts.

Discussed in depth here: RenderFrameHost vs FrameTreeNode swap

NavigationController-per-WebContents

From a high-level, both prerendering and portals do not have session history: they have a single entry and all navigation there commit with replacement.

However, given that both portals and prerendering need to commit navigation, giving them their own NavigationController (which will not be visible to embedders) is needed to be able to classify commits and subsequent subframe commits.

Multiple vs single WebContent for Portals discussion

Portals architecture (internal)