Note: this doc is in the process of being updated, an up-to-date shorter summary is here.

Multiple Page Architecture: //content/public API

altimin@chromium.org, carlscab@chromium.org, lfg@chromium.org
Created: Jul 15, 2020
Last updated: Nov, 2020

This document discusses how the //content/public API will look like if/when we will have support for multiple active pages within a single WebContents. This document focuses on how the final state should look from the embedder's perspective. See this document for the required changes to internals of //content/ and this document for the transition plan and related estimates.

Concepts

Document

The fundamental building block of the web, HTML document [spec]. Each document has a constant origin associated with it.

This document mostly assumes[1] that same-process RenderFrame swap (aka RenderDocument project) is implemented. After that, document will be represented by content::RenderFrameHost in the browser process; content::RenderFrame in the //content/renderer; blink::LocalFrame[2] and blink::Document in Blink.

Frame

A container holding documents. Each frame always has exactly one current document and each active document should be current for some frame. *Navigation*[3] is the process of changing the current document in a given frame.

A frame can either be owned by a document (thus defining non-changing parent relationships) or not (main frames).

---

[1] RenderDocument project doesn't block Multiple Page Architecture, but makes it easier
[2] Keep in mind that the term "frame" in this document never refers to RenderFrameHost and term "document" is exclusively reserved for that. Yes, this has a great potential for confusion — the proposal to rename RenderFrameHost to RenderDocumentHost solves this.
[3] Note that there also are same-document navigations, however for the purposes of this document "navigation" refers to cross-document navigations.

- From the API perspective, RenderFrameHost::GetParent() => RenderFrameHost* returns the parent document for the current document. This value might be null, but will never change during the lifetime of the given document.
- RenderFrameHost::GetMainFrame() returns the top document in the parent chain. Always non-null (but might be the same as the document in question) and stays constant during the entire lifetime of the given document.

Frames themselves do not have a corresponding object in //content/public API, but closely related concepts like navigations are indirectly referring to it (content::NavigationHandle).

Page

A collection of documents with the same main document (naturally, a page is 1:1 with main document).

Pages define their own embedding structure, separate from the regular iframe embedding — each RenderFrameHost might choose to embed a separate page. From the API perspective, RenderFrameHost::GetInnerPage() => Page* and content::Page::GetOuterDelegate[4]() => RenderFrameHost* return the embedded page and embedding document respectively).

Unlike document-level embedding, where the parent document can't change, the page-level embedding structure can change over time.

From the API perspective, on RenderFrameHost, methods GetParentOrOuterDelegate() => RenderFrameHost* (exists inside content/ now, will be exported to //content/public) and GetTopLevelDocument() => RenderFrameHost* (to be added). Unlike RenderFrameHost::GetParent and RenderFrameHost::GetMainFrame, returned values can change during the document's lifetime.

In the browser process, Page is represented by a content::Page (newly-introduced).
Due to multiprocess architecture each page will be backed by multiple renderers and no direct counterpart will be available in the renderer process. (see the appendix for the notes on the future of RenderViewHost).

Note: blink::Page is not a page in our definition as a) it's tied to a given process and a single blink::Page is not the page representation in the post-SiteIsolation world and b)

PageHolder

PageHolder is an abstraction for owning pages and navigating between them. It has a main frame (which owns the page's main document and thus indirectly the entire page) plus concept of a session history.

---

[4] Existing name — suggestions for a better name are welcome.

PageHolder can either be:

- Attached to a document, defining a page embedding tree: RenderFrameHostImpl::CreateInnerPagerHolder
- Or standalone.

A special type of navigation can transfer pages between page holders.

In the long term, PageHolder will replace the existing nested WebContents logic.
The API to create one might need to be exposed via //content/public if guest views or prerendering can't be moved inside content.

Tab

Tab is a concept combining rectangular content area with omnibox where URL can be displayed plus session history. Represented by WebContents class in //content/public API.

Tab (WebContents) directly owns one (primary) PageHolder, and always has one top-level page (== page which main document's URL is displayed in the omnibox):

- WebContents::GetTopLevelPage() => Page*

Session history of this top-level PageHolder is the tab-level session history. Session history (incl. NavigationEntries) for non-top-level PageHolders is not exposed via //content/public.

Pages within the tab can be:

- active top-level (held in top-level PageHolder)
- active embedded (held in the PageHolder which is embedded into active page or another active embedded page).
- prerendered (held in a non-top-level non-embedded (== detached) PageHolder).
- bfcached (held in back-forward cache rather than a page holder)
- pending deletion (is not held in back-forward cache nor in page holder)

These lifecycle states are exposed explicitly via //content/public as an enum (current basic proposal, might change as we think more) on RenderFrameHost object.as an enum on the PageHolder object.

It's possible to get a list of all pages in the given WebContents:

- WebContents::GetAllPages
- And the list of documents in each page
    - Page::GetAllDocuments

Tab vs page/document

Signals

Depends on the feature, often can be both — depends whether the aggregation is done inside //content or outside. Example: “is audible” state: exposed both as per-document (for features altering behaviour for audible frames) and per-tab (for showing UI indicator).

Guideline: if the content embedder needs to know about individual documents/pages (typically: has a mojo connection to a document and/or custom code running in the renderer), it needs to be per-document/page. Otherwise, per-tab is preferred with “document/page -> tab” aggregation being handled within //content.

- Rationale: It’s easier to work with per-tab signals for //content/ embedders as it removes the need to listen to navigation events with their associated subtleties and complexities.

Per-tab signals:

- WebContents::GetX().
- WebContentsObserver::OnXChanged
    - Rule: every change in the return value of GetX() should have a corresponding WebContentsObserver::OnXChanged callback.
    - Either navigation should not affect the return value of GetX or OnXChanged should be dispatched as a part of the navigation.
    - Good: WebContents::GetThemeColour and OnThemeColorChanged as a part of navigation.
    - Bad: WebContents::WasEverAudible implicitly being reset on navigation.
- Session history:
    - Considered to be a per-tab concept from embedder’s perspective.
    - Only notifications about top-level page holder are propagated to the embedder.

Per-page/frame signals:

- Page::GetX or RenderDocumentHost::GetX
- Signals should be explicitly attributed to a RFH /RVH / RWH / Page by providing a parameter on observer methods: WebContentsObserver::OnXChanged(Page/RenderDocumentHost*, …)
- Example: WebContentsObserver::OnCookiesAccessed (for the requests made by document)

Special type of signals: per-navigation:

- not tied to a page/document as navigation is responsible for changing documents.
- Should take NavigationHandle* as a parameter
- Example: WebContentsObserver::OnCookiesAccessed (for initial network request made by navigation)

States

Like signals, state can be per-tab, per-document/page or per-navigation. The guideline is very similar for a guideline for signals:

- Typically, per-tab state is calculated based only on the per-tab signals.
- Generally, per-tab state should not depend on navigation events.
- Discouraged but okay:
    - Idempotent updates (e.g. call CalculateState which is a no-op when called multiple times) for a navigation-related event.
    - Permanently updating tab-level based on the page-level state without ever resetting it later.
- Good: updating browser theme color
- Bad: resetting per-tab state in DidFinishNavigation.

Per-tab state can be stored using WebContentsUserData.

Per-page/document state: all accessed to it are keyed by content::Page (for pages) or RenderFrameHost (for documents).

Can be explicitly associated with Page or RenderFrameHost using PageUserData (to-be-introduced[5]) or RenderDocumentHostUserData[6].
Also possible: WebContentsUserData + map<RenderDocumentHost*, data>.

- Good: attaching state directly to the document
- Good: keying data by document in WebContentsUserData-bound map
- Bad: using WebContents::GetMainFrame to get per-document data.

Note: in the majority of circumstances with the introduction of PageUserData and RenderDocumentHostUserData embedders would not need to listen to navigation events anymore.

Per-navigation state: state generated by an in-flight navigation before the navigation created a document. Example: cookie access
Recommendation: use NavigationHandleUserData (to-be-implementedTBI) and transfer it to document/page level during the commit.
Example:

- Transferring information about cookie requests made by a navigation to the page.

Observers and notifications

Basic proposal: everything is kept on WebContentsObserver, but:

- All per-tab notifications correspond to the per-tab signals according to the criteria above.

[5] To Be Introduced
[6] Existing name — called “RenderDocumentHostUserData” because its lifetime is different from the lifetime of the RenderFrameHost as it currently exists.

- All non per-tab observers on WebContents have to have one of the following references: Page*, RenderFrameHost*, RenderViewHost*, RenderWidgetHost*.

Alternative advanced proposal:

- Per-tab states / associated notifications are kept in WebContentsObserver
- Navigation-related events are moved to a separate NavigationObserver (with lifetime matching the lifetime of WebContents).
- Per-page document events are moved to PageObserver (with lifetime matching the lifetime of a content::Page).
    - Ways to create:
        - A method on WebContentsObserver telling us when the page was created: WebContentsObserver::OnPageCreated(Page*)
        - An explicit list somewhere (like for WebContentsObservers)
        - Lazy initialization when a feature is accessed.

Page lifecycle

- Possible lifecycle states for a page / document:
    - Speculative (created but not committed yet).
    - Prerendered (started loading, but not activated yet).
    - Active top-level.
    - Active embedded (portal / guestview).
    - In back-forward cache.
    - Pending deletion.
    - Exposed as an enum on RenderFrameHost (RenderFrameHost::GetLifecycleState()) and a corresponding notification (WebContentsObserver::OnRenderFrameHostLifecycleStateChanged(RFH*, LifecycleState old, LifecycleState new)).
- RenderDocumentHost::IsCurrent
    - True for “active” and “embedded” pages, false for everything else.
    - Current documents are allowed to show UI elements to the user, everyone else can’t.
        - IsCurrent should be checked each time when going from per-document to per tab state.
            - Most importantly, before showing a UI element (all displayed UI elements are inherently tab-level).
    - Note: baseline expectation is that the iframes and embedded pages have roughly the same restrictions. Individual features (like permission requests) might have additional restrictions for embedded pages.
- RenderDocumentHost::IsActiveAndDisallowReactivation
    - True for active and active embedded.
    - Should not be called for speculative frames.

    - Use case: want to ignore the request, but don't want to break the page.
        - Forces pages to be evicted from bfcache and prerenderer to be cancelled.
- WebContentsObserver::RenderFrameHostChanged
    - Notification dispatched when the RenderFrameHost changes its lifecycle state
    - For each document-related UI element, this element should be cleared when the document becomes inactive.
        - Roughly speaking, every IsCurrent call checking if UI element can be shown should listen to RenderFrameHostChanged to dismiss such an element.
- Each change in the page lifecycle state corresponds to a navigation.

Navigations

- Navigation is the process of changing the current document in a given frame, represented by NavigationHandle (with the exception of same-document navigations).
    - Each document change is a navigation.
    - Navigations can't be tied to a specific page and considered to be a separate scope in their own right (which means that it will provide the ability to attach data to it via NavigationHandleUserData).
- Common navigation properties:
    - Navigations can be observed via DidStartNavigation / DidFinishNavigation and other methods on WebContentsObserver.
        - We might want to move them to a separate NavigationObserver, but in this case this observer will still observe all events in the given WebContents and should be owned as a WebContentsUserData.
        - For all navigation DidStartNavigation / DidFinishNavigation notifications are dispatched.
    - Navigation is tied to a particular frame (and therefore, PageHolder). Some of the associated frame / PageHolder properties are exposed via NavigationHandle:
        - NavigationHandle::IsInTopLevelFrame
        - NavigationHandle::IsInMainFrame (exists already)
        - NavigationHandle::IsInPortal
        - NavigationHandle::IsInPrerender
        - NavigationHandle::IsInFencedFrame
    - For navigations except navigations to about:blank and about:srcdoc navigation throttles are created and dispatched.
- Same-document navigation do not change active document
    - NavigationHandle::IsSameDocument == true
    - ReadyToCommitNavigation, RenderFrameHostChanged notifications are not dispatched.
- Regular navigation loading a new page
    - ReadyToCommitNavigation, RenderFrameHostChanged are dispatched

        - RenderFrameHostStateChanged notification is dispatched for changing the new RenderFrameHost lifecycle state from “speculative” to “active / prerendered / portalled”.
            - When we get rid of speculative RenderFrameHosts, the new RenderFrameHost will be created in the target state instead of changing to it from “speculative”.
        - If the navigation happens in a portal or a prerenderer, navigation entry-related notifications will not be dispatched.
- Back-forward cache restore
    - NavigationHandle::IsServedFromBackForwardCache == true
    - This navigation restores an existing page from back-forward cache.
    - Can happen only for the top-level pages.
    - ReadyToCommitNavigation, RenderFrameHostChanged are dispatched
    - RenderFrameHostStateChanged notification dispatched for “in back-forward cache” to “active” lifecycle state change.
    - NavigationThrottles are dispatched.
        - Network response is assumed to happen immediately for navigation throttle purposes.
- Portal / prerenderer activation
    - NavigationHandle::IsPortalActivation
    - NavigationHandle::IsPrendererActivation
    - These navigations take existing prerendered / portalled pages, transfer them across PageHolders and make them new top-level pages.
    - NavigationThrottles are dispatched.
        - Network response is assumed to happen immediately for navigation throttle purposes.
    - (tentative)
        - Old frame will be navigated to about:blank synchronously without dispatching any navigation throttles.

Summary of changes

- Add content::Page class
    - content::Page::GetMainDocument
    - content::Page::GetAllDocuments
    - RenderFrameHost::GetPage
    - RenderWidgetHost::GetPage
    - RenderViewHost::GetPage
- Page embedding
    - Replace WebContents::GetInnerWebContents with RenderFrameHost::GetInnerPage

        - Replace WebContents::GetOuterWebContents with RenderFrameHost::GetOuterDelegate
        - Replace WebContents::GetOutermostWebContents with RenderFrameHost::GetTopLevelDocument
        - Export RenderFrameHost::GetParentOrOuterDelegate (and find a better name for this)
- NavigationHandle:
    - NavigationHandle::IsInTopLevelFrame
    - NavigationHandle::IsInPortal
    - NavigationHandle::IsInPrerender
    - NavigationHandle::IsInFencedFrame
    - NavigationHandle::IsPortalActivation
    - NavigationHandle::IsPrendererActivation
- Lifecycle:
    - Add “Prerender”, “Portal”, “FencedFrame” to RenderFrameHost::LifecycleState
- Replace WebContents::GetMainFrame with WebContents::GetTopLevelPage
- WebContents
    - Move page-specific calls to content::Page / RenderFrameHost
    - This includes things from GetLastCommittedURL() to GetEncoding().
- WebContentsObserver:
    - Add missing RFH* / Page* / RVH* / RWH* references to page-level methods
    - Add OnXChanged for tab-level signals which can be affected by a navigation.

Appendix

Future of RenderViewHost

RenderView/Host represents a local collection of documents with the same main document in the same process.

The main purpose of RenderView/Host is to provide a way to update the state shared between these documents in a single IPC, for example page visibility.

At the moment RenderViewHosts are not replaced on every navigation. Plan: stop sharing RenderView between pages [doc with details].

- After that each RenderViewHost will be tied to a particular Page: will add a constant RenderViewHost::GetPage() getter.

Readability proposal: re-define it as a “page view” — view of a page from a given process / isolate perspective.
Rename:

- RenderViewHost => RenderPageViewHost

- RenderView => RenderPageView
- blink::Page => blink::PageView (to highlight cross-process nature of the page).

Keep a very small surface in //content/public for broadcasting IPCs: only interface broker. Expected number of use cases: << 10.

Aspirational goal: remove RenderViewHost from //content/public entirely.