Closed Tab Cache for Chrome

This Document is Public

Authors: aebacanu@, carlscab@, altimin@, sreejakshetty@
June 2020

One-page overview

We want to make the “Reopen Closed Tab” button instant for very recently closed tabs. We are currently (June 2020) actively working on delivering the BackForwardCache for Chrome. This cache will make back and forward navigation instant. We want to reuse the work being done there to also be able to quickly restore recently closed tabs. The main use case being accidental clicks where the restore happens shortly after the close. We expect to get a big UX win by being able to restore such tabs instantly with their entire state.

Our end goal is to have a working behind-the-flag implementation of the feature. The timeline for a full launch TBD, as it will depend on when bfcache is fully functional.

BackForwardCache (bfcache) is currently being trialed in Canary so the code is in a functional state that allows us to reuse it for the closed tab cache. There is still ongoing work to address a lot of corner cases and other issues but nothing that would prevent us from moving forward with the closed tab cache. We want to move forward with this project in parallel so that we have a functional prototype by the time bfcache is launched.

Android already has a similar functionality (tabPendingClosure) but the underlying implementation is much more basic (e.g. pages are not frozen).

Summary

We want to build a tab-level cache (in chrome/browser/ui) and use it in TabStripModel hooking into TabStripModel::InternalCloseTabs and CreateRestoredTab. After a timeout the page will be evicted from the cache. We will start with something like 15 seconds for now, but will use metric data to make a more educated guess.

Note: Android already has support for this, see tabPendingClosure. If we decide to have this cache in content/ layer, we should share as much of the implementation as possible (some of the Android code is in Java so that might not be that easy)

Platforms

Mac, Windows, Linux, Chrome OS

Team

chrome-bfcache@google.com

Bug

Pending.

Code affected

- chrome/browser/ui
- content/

Design

This feature will heavily rely on the work carried out for bfcache. We will rely on all the work being carried out to support lifecycle management in RenderFrameHost instances.

We will approach the project in an incremental manner. At first we will try to come up with a prototype that reuses existing functionality (bfcache) as much as possible and then look into refining functionality. For example bfcache imposes very strict restrictions on what a page can do while in the cache and maybe not all of those apply here.
We will stage the work as follows

Feature flag

Add a new feature flag in chrome/browser/browser_features.h so that we gate all changes behind it.

Metrics

Make sure we have the required metrics to determine the usefulness and usage of this feature. In particular we want to measure the frequency of “Undo Close Tab” button use and the time between closing a tab and reopening it, as this will inform us on what cache timeout makes the most sense.

ClosedTabCache

Implement a cache similar to BackForwardCacheImpl that stores WebContents instances instead of RenderFrameHost instances. When adding WebContents to this cache the currently shown pages LifecycleState will be set to kInBackForwardCache and thus all renders frozen. Conversely when restoring the WebContents the renderers need to be unfrozen and the page reactivated.

We also need to keep track of how long a WebContents has been in the cache and implement some time out for eviction. Initially we will evict after 15 sec but this should be flag configurable.

At this stage we want to reuse as much functionality as possible from bfcache. This might impose stricter restrictions than actually needed, but for a first prototype this will be fine. In particular we do not plan to run unload handler for now (same as bfcache). And hopefully we will never need to as this is the approach we are pursuing for bfcache.

Potential structure

Entry {

std::unique_ptr<WebContents> web_contents;

base::Time timestamp; // time when tab was closed

// to be amended based on more state we later discover we should keep (potentially the index within the tab strip, so we can restore it in the same place)

}

ClosedTabCache = map(key -> Entry)

Potential keys to use for mapping of the entries

- SessionID
    - an int32_t value
    - a new unique SessionID is created at every TabRestoreService::Entry creation
    - TabRestoreService uses SessionID to identify entries, either tabs or windows
    - SessionID is however not accessible from TabStripModel or from CreateRestoredTab. It would have to be passed down as an argument.
- NavigationEntry id
    - it is already accessible in CreateRestoredTab and can be retrieved from WebContents' navigation controller in TabStripModel::DetatchWebContentsImpl: web_contents->GetController().GetLastCommittedEntry().GetUniqueId()

We decided on using SessionID by making TabRestoreServiceHelper::CreateHistoricalTab return the tab's SessionID rather than nothing. This ensures that when TabStripModel::DetachWebContentsImpl calls delegate_->CreateHistoricalTab(), it'll get access to the SessionID, which we can use to add an entry into the cache. Similarly, we will need to expose the SessionID to CreateRestoredTab, which we can do by passing it as an argument from all the way up the call hierarchy when TabRestoreServiceHelper::RestoreEntryById calls BrowserLiveTabContext::AddRestoredTab.

Where to hook into the codebase

1. Adding an entry to the cache

We will hook into TabStripModel::DetatchWebContentsImpl to add an entry to the ClosedTabCache in addition to calling CreateHistoricalTab. We also no longer want to run unload listeners for WebContentses that are cached nor do we want to destroy them.

2. Restoring an entry from the cache

We will hook into CreateRestoredTab to add a check and try to retrieve the desired WebContents from the ClosedTabCache and only recreate it if there is a cache miss (or the flag for the experiment is not set). In case of a cache hit, we would also skip all the navigation controller restore bits since the restored WebContents already holds that state.

Storing tabs vs. windows

There is the question of whether we would want to focus on only restoring a single tab (i.e. the most recently closed one) or we would also want to restore a whole window having been closed (with all its tabs). The same keyboard shortcut that restores the most recently closed tab (Ctrl+Shift+T/Command+Shift+T) also restores the last window if that is the last close action that happened. Restoring a whole window opens up 3 potential approaches:

- store all tabs of that window in the cache and restore them
    - unfeasible if we limit the cache size to 1 (as per the bfcache)
- store one tab of all the window's tabs in the cache and reload the others normally
    - need to choose which tab to store in the cache (potentially the currently active one?)
- have 2 types of entries in the ClosedTabCache, similar to the RestoreTabService and have the Window entry hold all its tabs (WebContentses)
    - can still limit the cache size to 1 entry in this case
    - more logic to implement

For now, the simplest implementation should focus only on restoring tabs.

Storing WebContents vs. WebContentsData

TabStripModel::WebContentsData is a class created to own a WebContents that is in a tabstrip. Other than the actual WebContents object, it has:

- ReplaceWebContents function, which changes the WebContents that the WebContentsData tracks through a pointer swap. Can be used to transfer ownership of WebContents when put into the ClosedTabCache.
- WebContents* opener_ - the WebContents from which the tracked WebContents was opened (if any)
- bool pinned_ - whether the tab was pinned. Useful since "Reopen closed tab" should restore a tab as pinned if it was previously so
- bool blocked_ - whether the tab was blocked by a modal dialog. We may not need to worry about those since the WebContents may be restored along all the open modal dialogs (as per a simple tab switch). We should check if WebContents knows by itself if it is blocked. A simple solution would be to not store any page with a modal dialog open.

The infrastructure takes care of restoring all these (e.g. AddRestoredTab takes boolean pin as an argument and takes care of restoring the tab as a pinned tab if it was pinned when

closed), so it makes more sense for us to only store WebContents in ClosedTabCache::Entry, alongside any other state we find useful (time when closed, index within the tabstrip etc).

Ownership of WebContents

Current flow

TabStripModel::CloseWebContentses receives a list of WebContentses to close and first attempts a fast shutdown of all the tabs in the list that can close. It then runs the regular shutdown procedure on all the remaining tabs. Here, in the ClosedTabCache case, we want to avoid running the unload listeners potentially for the last WebContents to be closed, as that will be put onto the ClosedTabCache (?).

The ownership is handled in TabStripModel::DetachWebContentsImpl, where the raw_web_contents are passed to BrowserTabStripModelDelegate::CreateHistoricalTab and eventually all the way down to TabRestoreServiceHelper::CreateHistoricalTab, which creates a new TabRestoreService::Tab and adds it to the service's entries list. We then determine the next index in the tabstrip that should be selected and ungroup the tab to be deleted from its tab group. We erase the WebContentsData entry from the list owned by TabStripModel, we update the selection model and call ReplaceWebContents, which swaps the WebContents tracked by the old WebContentsData with the nullptr and returns the old WebContents. The old WebContents is passed to a TabStripModel::DetachedWebContents object and TabStripModel::SendDetachWebContentsNotifications is where dwc->contents.reset() gets, called, which destroys the WebContents.

Changes to the current flow

As mentioned earlier, we no longer want to run the unload listeners for the WebContents being put into the ClosedTabCache.

We also no longer need to pass it to DetatchedWebContents, but to ClosedTabCache::Entry. This ensures that we don't call reset() on it either, since reset() is only called on DetatchedWebContentses.

We could pass the whole WebContentsData to ClosedTabCache::Entry from this point onwards and no longer replace its tracked WebContents with the nullptr. Alternatively, if only storing WebContents, we would leave the replacement with nullptr and potentially check the will_delete argument of TabStripModel::DetachWebContentsImpl. If true, we check if the WebContents could be added into the ClosedTabCache rather than handed over to DetatchedWebContents.

Freezing/unfreezing of WebContents

Freezing

When a WebContents is put into the ClosedTabCache, it needs to be frozen similarly to the approach used for bfcache in RenderViewHostImpl::EnterBackForwardCache. We should implement WebContents::EnterClosedTabCache (or WebContentsData::EnterClosedTabCache). We don't need to unregister the RenderViewHost since we are freezing the whole FrameTree along with the WebContents. For now, we want to dispatch a page hide, freeze the page, but not hook eviction (see here). We will eventually also hook eviction and stop all JavaScript execution when the eviction API is fully implemented.

Unfreezing

When restoring a WebContents from the ClosedTabCache, we unfreeze it as is done in RenderFrameHostManager::RestoreFromBackForwardCache and swap it in.

Hook into TabStripModel

Integrate the cache into TabStripModel so that closed tabs go into the cache. We probably want to limit the number of WebContents that can be in the cache at any given time. Reopen tab will need to look in the cache and in case of a hit can execute a fastpath to restore the page from the cache.

So instead of deleting the WebContents in TabStripModel::SendDetachWebContentsNotifications we would store it in the cache and we would do a cache lookup in CreateRestoredTab.

Generalize Eviction API

We need to make sure that eviction events due to forbidden feature use which are currently routed to the bfcache are also routed to the new cache. Probably have a generic Cache (RenderFrameHostCache) and let BackForwardCacheImpl and ClosedTabCache make use of it.

Investigate WebContentsDelegate

We need to determine whether work is required to prevent / block calls to the WebContentsDelegate from the cached WebContents and what the impact of not blocking would be.

Finch Experiment

Launch an experiment

Metrics

Success metrics

We will be tracking UserActions.Counts and are expecting a higher frequency of RestoreTab events as a first measure of success of this feature.

As a second measure, we have added two histograms:

- TabManager.TimeSinceTabClosedUntilRestored
- TabManager.TimeSinceWindowClosedUntilRestored

They measure the duration since a tab/window was closed until it was restored from the "Reopen closed tab/window" button. We will be using these metrics to make an educated guess of the timeout we should use for an entry in ClosedTabCache.

It is currently estimated that roughly 60% of tab restores happen within 15 seconds. This means that about 60% of tab restores will be made instant by this feature.

Regression metrics

The same UserActions.Counts can be used to flag a potential drop in the frequency of RestoreTab events. This could be a sign that ClosedTabCache is not working as expected and users avoid restoring most recent tabs.

Rollout plan

Experiment-controlled rollout. Feature flag: kClosedTabCache

Core principle considerations

Speed

We expect this feature to improve Chrome's performance by getting rid of the overhead of completely reloading recently closed tabs. This would make navigation faster in a way very similar to how bfcache impacts back and forward navigations.

Privacy considerations

Privacy concerns are related to the freezing/unfreezing of pages as they get added/restored from the ClosedTabCache. We are storing recently closed pages in the background and

require those pages to be completely frozen: no direct or indirect interaction with the user is permitted. For example, we cannot allow pages to still collect geolocation data while cached. Eviction happens very fast anyway.

This is the same as BackForwardCache.

Testing plan

N/A for now.

Followup work

We should consider making the kBackForwardCache LifeCycleState more generic like kCached if we were to reuse it for this project.

We also want to have a tighter integration with the current Android tabPendingClosure. Probably mostly replacing all that functionality. This might slightly reduce the coverage of that feature (the ClosedTabCache has stricter guarantees on what pages are allowed to do while cached without being evicted). But this probably makes sense from a correctness point of view.

It was suggested that we have a tighter integration with TabRestoreService. Instead of having a new object storing the recently closed WebContents, an idea is to make TabRestoreService store them.