Session isolation in Headless Chrome

skyostil@
June 8th, 2016

Problem: For efficiency reasons we want to run several parallel sessions in a single headless browser instance. These sessions (“tabs”) should have isolated storage (cookies, local storage, cache, etc.) to avoid accidental cross-talk (e.g., two sessions might want to load the same URL but use different cookies). It may also be advantageous to allow efficiently saving and restoring a given session, for example, between test harness iterations.

Issues

Main design requirements:

- Minimize memory usage: the static memory cost of introducing another session should be minimal.
- Isolation: one session should not be able to affect the functionality of another. Performance isolation is not a requirement however.

- Which resources should we be isolating?
    - Cookies, cache, local storage -- essentially everything from the StoragePartition.
    - Ideally all kinds of side effects should be minimized. For example delivering mutation observer events in one session should not cause these events to be delivered in all sessions.
    - More specifically:
        - Cache
        - Referrer
        - User agent
        - Per-resource response (url, final url, headers, content)
        - Javascript enabled
        - Time of day
        - Navigator platform
        - Timezone
        - Initial set of cookies (domain, path, name, value, expiration, secure, http-only)
        - Initial DOM local storage items (security origin, key, value)
        - Initial DOM session storage items (ditto)
        - Plugins enabled
        - Touch event support
        - Frame background color
        - Font size
        - HTTP request headers

        - Browser language
        - CSS media type
- How do we associate per-session data like cookies with a network request?
- At which level should this per-context data live?
- Do we need further isolation (static globals)?

Potential solutions

Option 1: Implement multiple sessions inside a single Blink instance

- **Pro:** Least amount of resource overhead, since many resources can be shared between sessions.
- Con: Very invasive -- many static objects need to become per-session objects (e.g., the memory cache).

Option 2: Implement multiple Blink instances inside a single renderer process

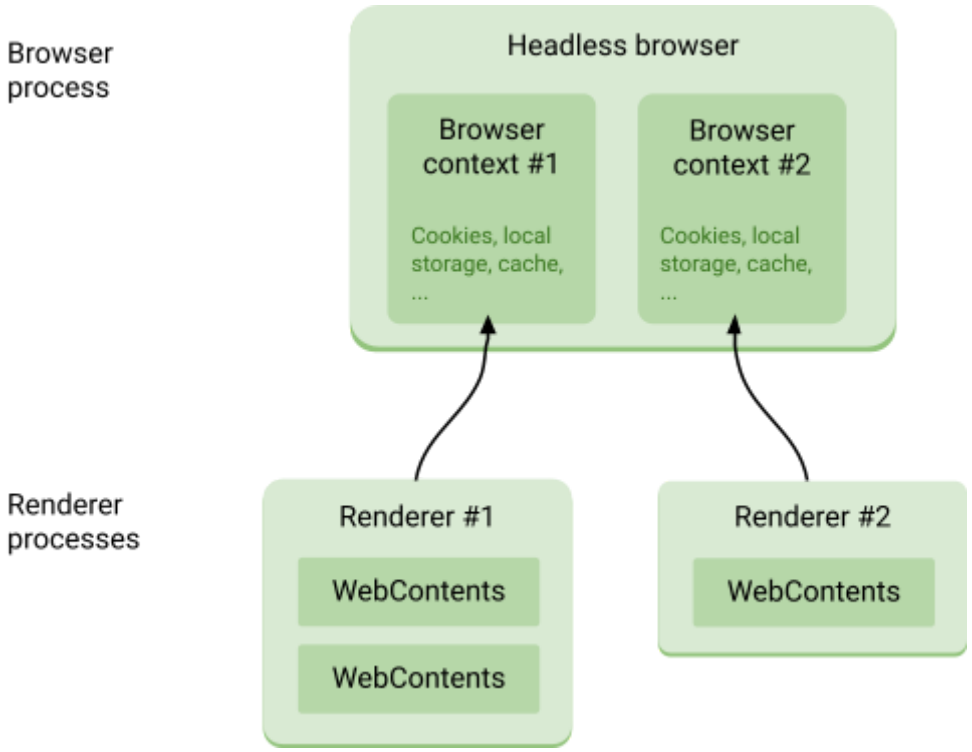
- **Pro:** Still within a single process, so less memory overhead.
- Con: Chrome doesn't currently support having multiple storage partitions in a single renderer process.
- Con: Might still end up needing many changes to deal with global statics.

Option 3: Run each session in a separate renderer process

- **Pro:** Better matches normal Chrome architecture.
- Con: Introduces a separate renderer process per session, adding memory overhead. It's not yet clear if we can reduce this overhead enough.

The working assumption is that we will go with **Option 3** and strive to minimize the memory overhead of each additional session (renderer process).

Session isolation C++ API



To implement session isolation, we introduce a new HeadlessBrowserContext object to the headless API. The role of this class matches that of content::BrowserContext in that it encapsulates the storage partition, blob storage and other data associated with a browsing session. In fact, its implementation is essentially an opaque wrapper around content::BrowserContext.

```
std::unique_ptr<HeadlessBrowserContext> my_browser_context(
    browser->CreateBrowserContextBuilder().Build());
```

The headless API is further modified to allow associating a new tab at creation time with a specific HeadlessBrowserContext (instead of the default one):

```
HeadlessWebContents* web_contents = browser->CreateWebContentsBuilder()
    .SetBrowserContext(my_browser_context).Build();
```

An arbitrary number of tabs can be associated with a single browser context. The browser context must outlive all the tabs that are associated with it.

Note that some of the items listed in the issues section will need additional customization points since they are not part of the browser context.

Session isolation DevTools API

Ideally session isolation should also be possible over the DevTools wire protocol. As a part of providing Mojo services to WebContents, we intent to introduce a Browser DevTools protocol domain to make it possible to open and close tabs. Similarly the browser context can be exposed with the following commands:

- Browser.canCreateBrowserContext() => bool (whether or not the target supports parallel browser contexts)
- Browser.createBrowserContext() => id string
- Browser.destroyBrowserContext(id)
- Browser.newPage() will take an optional browser_context_id parameter, identifying the context to associate with.

Further study is needed to see if all Chromium builds can support parallel browser contexts or whether it is limited to headless mode.

Case study: cookies

- Cookie store is owned by URLRequestContextStorage and indirectly by the URLRequestContext.
- Renderer accesses cookie store via IPC: SetCookie, GetCookies.
- Cookies are mapped to a URLRequestContext by RenderFrameMessageFilter.
- URLRequestContextGetter constructs the URLRequestContext.
- RenderProcessHostImpl owns a storage partition, which owns the URLRequestContextGetter.
- The StoragePartition is retrieved based on the BrowserContext.
- StoragePartitions can be divided into a partition_domain/partition_name namespace.
- On the network request side, cookies don't seem to be communicated to custom protocol handlers. However, if we can associate protocol handlers with a URLRequestContext, they can look up the correct cookies from the cookie store.

⇒ Having a separate BrowserContext per session should allow cookie isolation.

To support this use case, the browser context can be associated with a specific set of net::URLRequestJobFactories.

Case study: renderer memory usage

Things to investigate:

- private_dirty usage in a fresh renderer
- Svelte mode
- Which allocations could potentially be shared across renderers (e.g., glyph cache?)

Privilege separation using browser contexts

In addition to isolating user state between pages, browser contexts can also serve as a privilege separation mechanism. Normally, headless browser contexts won't have any special privileges. However, it is possible to add custom Mojo services to a particular context. These services are implemented by the embedder (outside the renderer sandbox) and in general should not be exposed to regular web sites.

To make accidental privilege escalation less likely, browser contexts which include Mojo services will not support HTTP/HTTPS fetching by default. If the embedder needs both Mojo services and HTTP/HTTPS fetching in the same context, they need to enable it explicitly:

```
browser->CreateBrowserContextBuilder()
       .AddJsMojoBindings(...)
       .EnableUnsafeNetworkAccessWithMojoBindings(true);
```

Resources

- Patch to add HeadlessBrowserContext: https://codereview.chromium.org/2043603004/
- Suborigins (turns out these don't really intersect with this use case):
    - https://www.chromium.org/developers/design-documents/per-page-suborigins
    - https://w3c.github.io/webappsec-suborigins/