Improving WebUI code caching

Attention - this doc is public and shared with the world!

Contact: seth.brenith@microsoft.com

Status: Inception | **Draft** | Accepted | Done

Bug: chromium:1210399

LGTMs needed

Name	Write (not) LGTM in this row
leszeks@	LGTM starting with option 4
<add yourself>	

Previous work

In Code caching for WebUI scripts - Google Docs, I proposed a strategy for enabling code caching on scripts loaded via the chrome:// and chrome-untrusted:// protocols. The strategy follows the usual pattern for open web scripts:

1. First load: save the current timestamp
2. Second load: if the saved timestamp was within the last 72 hours, then generate code cache data
3. Third load: use the code cache data to avoid parsing time

The difference for WebUI scripts is that they don't use the network cache, so it's harder to tell whether the script content has changed since the previous load. To make code caching work correctly, we added a SHA-256 hash of the script text to each record saved (both timestamps and full code cache data).

However, there's a problem: computing a SHA-256 hash takes time. It's faster than parsing JavaScript, but it's a lot slower than doing nothing at all. The design of code caching for the open web is intended to have almost no impact on the first load time, because the user is probably never coming back to that page. Once the script is loaded a second time, then the browser judges that the effort of serializing and saving code cache data is probably worthwhile. But this implementation of code caching for WebUI affects the first load too, by requiring a hash computation! In Edge's experimentation, we've seen that WebUI code caching improved the median load time for frequently used WebUI panes, such as Downloads, but worsened the median load time for less frequently used WebUI, such as History and Settings.

Option 1: put the burden on the data sources

I’m not advocating this option, but I think it’s important to include it here for completeness. If hashing takes too long, then perhaps we shouldn’t do it at all. Instead, each WebUI content source could have the option to provide an “ETag” for the data: a short string that is guaranteed to differ if the script data has changed. If the ETag on a subsequent load matches the ETag from a previous load, then the scripts are assumed to be identical. For scripts loaded from the application installation directory, the version number of the browser would be a reasonable (and very cheap) ETag. Precomputed hashes are also a fine option.

I see two potential problems with this approach:

- There are many WebUI data sources, each of which would have to specify ETag behavior if they want to be eligible for code caching. I’m hoping that we can create something that is more of a “pit of success” rather than requiring deliberate opt-in from each data source.
- It may be difficult to get sufficient test coverage to ensure that all WebUI data sources specify ETags correctly. Any bugs where the same ETag is applied to differing script data would cause confusing errors which are hard to reproduce and understand.

Option 2: add a non-hashing first load

We could add a step to the beginning of the process described above.

1. *newly added:* First load: save a timestamp **without** a source text hash
2. Second load: save a timestamp with a source text hash, if the previous timestamp was recent enough
3. Third load: generate code cache data, if the timestamp with hash was recent enough
4. Fourth load: use the code cache data

This process takes more steps to get to the point where it benefits from code caching, but it protects against doing any hashing work in the first load, keeping the first-load speed as fast as possible. We might consider a shorter recency requirement for the second load, such as 24 or 48 hours rather than 72, since at least four loads are required to see any benefit. Keep in mind also that the Chrome stable channel updates approximately once every two weeks, and code cache data is not usable across upgrades, so there is a strict upper limit on how long we can take to get those four loads done.

Option 3: use a cheaper hash on first load

When saving the timestamp after the first load, we could include a cheap hash such as CRC32 rather than a powerful cryptographic hash. On the second load, if the CRC32 matches, then we don't know for sure that the content didn't change, but we have a clear enough signal that we can decide to generate code cache data. The generated code cache data must be accompanied by a full SHA-256 hash, because hash collisions on the third or subsequent loads would be catastrophic.

Option 4: don't hash at all on first load

I'm not advocating this option, but including it for completeness. Including a hash with the timestamp lets us judge whether the script has changed before saving full code cache data (which takes time to generate and space on disk). Currently, we avoid generating full code cache data for scripts that have changed between the first and second loads, assuming they'll probably change again on subsequent loads. However, most WebUI scripts don't change per load, so we could optimistically assume that saving full code cache data on the second load is worthwhile based only on the timestamp and not using any hash comparison. This means we don't have to compute a hash on the first load which saves the timestamp.