Dispatcher framework and BFCache WPTs landing plan

hiroshige@chromium.org

PUBLIC. This doc was used to share planning ideas within Chromium, and the contents of this doc is mostly included in common/dispatcher/README.md in https://github.com/web-platform-tests/wpt/pull/28950.

(Previous state: Private, I'll post a brief summary at RFC 89 (because most of this doc is about `RemoteContext.execute()`) + a link to this doc, and then include the details as comments/READMEs in wpt PRs.)

Overview

Proposed steps:

- Land pure JavaScript executor framework originating from COEP/COOP under `common/dispatcher/` (CL 3033199),
- Define a new API interface on top of that (CL to be linked), and
- Land BFCache WPTs on top of the new API (CL 2885636 and to-be-split CL 2798554).
- Discuss testharness/testdriver integration later separately.

The `common/dispatcher/` library will have two sets of APIs:

- Existing APIs for COEP/COOP tests, such as `send()`, `receive()`, etc. These APIs and COEP/COOP tests will be kept mostly as-is, because according to the discussion on RFC 90: [testdriver] Primitives for cross-context messaging #90, we need more work on both the COEP/COOP tests and framework sides to adopt the RFC to existing COEP/COOP tests.
- New API, `RemoteContext` (and some helpers needed on the executor side), that is basically the same as the following RFCs (see below for diffs/additions), is built on top of the existing APIs above, and is used for implementing BFCache tests.
    - RFC 88: [testdriver] Extend the mechanisms for giving browsing contexts ids. #88
    - RFC 89: [testdriver] Add an execute_script function to testdriver. #89
    - RFC 91: [testdriver] RemoteContext object #91

I expect we can decouple the test framework discussion from the BFCache landing plan, because

- The new API has a solid API semantics (see below) so that we can switch the underlying implementation to the jgraham's testdriver-based version in the future. I confirmed this

by CL 3082215, which switches the impl to the testdriver-based version without modifying test bodies.
- I expect the discussions are more about a good API interface, not necessarily testdriver-integrated, so having a consensus on the new API would be sufficient to unblock the BFCache WPTs. I expect the RFC discussions so far + this proposal will address the review comments like:
    - https://github.com/web-platform-tests/rfcs/pull/86#issuecomment-881681930
    - https://github.com/web-platform-tests/rfcs/pull/86#issuecomment-887311284
    - https://github.com/web-platform-tests/wpt/pull/28950#discussion_r669395488
    - https://chromium-review.googlesource.com/c/chromium/src/+/3055392/3#message-7ec7ea77911b70b715695364ef9a92ea8a729bc5

The new API semantics

Mainly diffs from/additions to the RFCs 88, 89, and 91.

The core part is the additional semantic clarifications and test writing guidelines in Evaluation timing of Injected scripts section, because in BFCache tests it's very important to understand and control the interaction and timing around injected scripts and navigation.

Usage

We'd inject scripts like:

```
// injector.html
const argOnLocalContext = ...;

window.open('executor.html?uuid=' + uuid);
const ctx = new RemoteContext(uuid);
await ctx.execute(
    (arg) => functionOnRemoteContext(arg),
    [argOnLocalContext]);
```

and on executor we provide a helper class `Executor`. `Executor` is not strictly needed for the testdriver-based version itself (and thus will be mostly no-op), but is needed for the COEP-based impl.

```
// executor.html
function functionOnRemoteContext(arg) { ... }
```

```
const executor = new Executor(uuid from location.search);
```

Limited Scope

- Only `RemoteContext` constructor with a string UUID argument and `RemoteContext.execute()` are implemented.
- `RemoteContext` isn't integrated with testdriver/testharness.
- RFC 90: [testdriver] Primitives for cross-context messaging #90 is not supported, as BFCache WPTs don't need directly using send/receive() primitives right now.

Always `await execute()`

`RemoteContext.execute()` processing can be serialized, and thus always wait for the resolution of the promise returned by `RemoteContext.execute()` before calling next `RemoteContext.execute()`.
So it's a good practice to always write `await ctx.execute(...)`.

This isn't strictly needed for COEP-based impl, but anyway I expect enforcing this would make the tests easier to read and debug. I'm not sure whether this can be enforced to existing COEP/COOP tests though.

Evaluation timing of Injected scripts

The script injected by `RemoteContext.execute()` can be evaluated any time, e.g. even before DOMContentLoaded events or even during navigation, and it's the responsibility of test-specific code/helpers to ensure evaluation timing constraints (which can be test-specific).

Alternatively, we might want the API interface itself to ensure more strict evaluation timing restrictions to avoid the additional synchronization code like the following subsections.
But probably it's better to keep the script injection layer (i.e. `RemoteContext.execute`) simple for now (and consider more strict timing restrictions later),

- In order to unblock BFCache WPTs earlier.
- Because I'm not so confident about what kind of timing restrictions should work for other cases, like prerendering.
- The additional synchronization code below can be mostly in helper JavaScript file and doesn't affect test bodies so much.

Evaluation timing around page load

To avoid race conditions between injected script evaluation and page load, we can use pure JavaScript code like below, to ensure `mainCode` is evaluated after the first pageshow event:

```
// executor.html
window.pageShowPromise = new Promise(resolve =>
    window.addEventListener('pageshow', resolve, {once: true}));

// injector.html
const waitForPageShow = async () => {
  while (!window.pageShowPromise) {
    await new Promise(resolve => setTimeout(resolve, 100));
  }
  await window.pageShowPromise;
};
await ctx.execute(waitForPageShow);
await ctx.execute(mainCode);
```

Evaluation timing around navigation out

To avoid race conditions between script injection framework and navigation/unloading:

- Do not call the next `RemoteContext.execute()` for the context after triggering the navigation, until we are sure that the context is not active (e.g. after we confirm that the new page is loaded). AND
- Call `` `Executor.suspend(callback)` `` synchronously within the injected script. This suspends executor-related code to avoid race between navigation and executor-related code.
    - TODO: the name suspend() might be confusing, because it doesn't suspend injected script evaluation in testdriver version.

Additional notes on `` `RemoteContext.execute` ``

Already posted at https://github.com/web-platform-tests/rfcs/pull/89#issuecomment-896327379:

- When the return value of an injected script is a Promise, it should be resolved before any navigation starts (i.e. shouldn't be resolved after navigating out and navigating back again). It's fine to create a Promise to be resolved after navigations, if it's not the return value of injected scripts.
- `` `RemoteContext.execute` `` should wait for the target context to be created or to become active (i.e. window.open, back-navigation from BFCache, etc.).
    - This is already the case for COEP-based impl, but there are no way to do this in testdriver-based version.
    - I made WPTRunner's switch_to_window() to retry until the target is found in my draft as a temporary workaround.