npm@
July, 2020
Public document

This document explains the VisibilityStateEntry proposal, which was already presented to the Web Perf WG (slides, minutes) after describing a couple of problems with the current Page Visibility API. These problems are documented here but this document is meant to be self-contained.

The problem

One important problem users face in the web is how slowly websites load. In order to improve the user experience, we need to empower developers to be able to measure performance in an accurate and reliable way. This way, developers will be able to improve the numbers over time. In other words, you can't improve what you can't measure.

One of the most important measures of the webpage loading experience is how quickly content shows up on the screen, so there are APIs to measure this: Paint Timing, Element Timing, Largest Contentful Paint, etc. However, in order to make sense of the numbers, the visibility state of the page needs to be taken into consideration. For example, if the user opens a page in the background and only comes back to it after an hour, the first contentful paint may be more than an hour, but this does not imply that this page load should be considered slow.

The PageVisibility API does not provide a full history of the visibility of the page. Currently, it provides developers with the ability to:

- Measure the current visibility state of the document via either `document.hidden` or `document.visibilityState`. The latter is the recommended way to query.
- Add an event listener to be notified of any future changes of the document via `onvisibilitychange`.

This means that a developer cannot know anything about past visibility states of a document, as shown by this issue. Developers want to know as much about the visibility history of a document as possible. Right now, this is impossible to do fully accurately and to achieve the most accurate results a developer would need to register the event listener as early as possible to lose the least amount of data form this forward-looking API. Therefore, we'd like some way to enable

developers to get this information without requiring running JavaScript very early during the page load.

Proposed solution

The proposed solution to the problem of exposing the full visibility history to web developers is to create a VisibilityStateEntry interface. A VisibilityStateEntry would be created upon navigation (to expose the very first visibility state of the page) as well as every time the visibility state of the page changes. Objects of this type would be exposed to PerformanceObserver. This has the advantage of enabling us to use the existing buffering and callback logic that's used for other performance entries.

Specifically, the following would be the IDL of the interface (all attributes are inherited from PerformanceEntry):

```
interface VisibilityStateEntry : PerformanceEntry {
   name - 'hidden' or 'visible' (similar to document.visibilityState)
   entryType - 'visibility-state'
   startTime - DOMHighResTimeStamp of when the change occurred (0 for the initial)
   duration - 0
}
```

One slight disadvantage of this proposal is that it ties the visibility object to the Performance interface, and it could be claimed that visibility states are not inherently related to Performance. However, we already have some precedent of tying non-performance to PerformanceObserver with the LayoutShift interface. In addition, the visibility state history is generally very relevant when looking at performance entries, as demonstrated for instance by the Paint Timing API (a user won't reach first-contentful-paint when the page is hidden). So tying it to PerformanceObserver is not too far-fetched in that regard.

Sample code

In the following code snippet we show how to tag a PaintTiming entry as 'tainted' due to the page being hidden at some point before it occurred.

```
let firstHiddenTime = null;
const observer = new PerformanceObserver(list => {
  list.getEntries().forEach(entry => {
    // getEntries() returns entries in order of startTime, so no need to
    // sort them.
    if (entry.entryType === 'visibility-state') {
      if (entry.name === 'hidden' && firstHiddenTime == null) {
```

```
        firstHiddenTime = entry.startTime;
      }
    } else {
      const isTainted = (firstHiddenTime != null);
      // Report the entry and whether it is tainted or not to analytics.
    }
  });
});
observer.observe({type: 'paint', buffered: true});
observer.observe({type: 'visibility-state', buffered: true});
```

Alternatives considered

- Create a new **`PageVisibilityObserver`** interface, which enables querying into past visibility states of the page. This is a good option in that it does not tie the solution to Performance or PerformanceObserver, but it requires rethinking and specifying every bit about the observer logic and callback. Since we want to use the same exact logic and there is precedent for not-fully-performance entries surfaced via PerformanceObserver, it does not make sense to reimplement it.
- Try to tackle the problem by just looking at the first visibility state change: this is mentioned in the GitHub issue as visibilitySwitch or even a boolean about whether the page has been hidden at some point in the past. These were fine as temporary solutions to the problem but they were never really worked on, so now we should aim to simply fix the full problem, which requires full visibility history. In addition, it does not solve use cases around knowing when the visibility state of the page changed very early.
- Add an option to the *options* parameter of addEventListener which enables surfacing visibility change events that have occurred in the past. This option has the advantage of not adding yet another way to query visibility. It has the disadvantage of requiring the buffering logic to be reimplemented in the DOM specification so that it can support the use-case. It also has the disadvantage of adding API surface to a method that is already extremely overloaded in the sense that it can be used for a myriad of purposes. Unless there exist use-cases for buffering logic for various other events, it does not seem worth the cost.

Security / privacy considerations

The same considerations as those in Page Visibility would apply here. The developer would already be able to query this information or approximate most of it by appending some JavaScript in the header which queries document.visibilityState and adds an onvisibilitychange event listener.