WebAPIs and Viewports

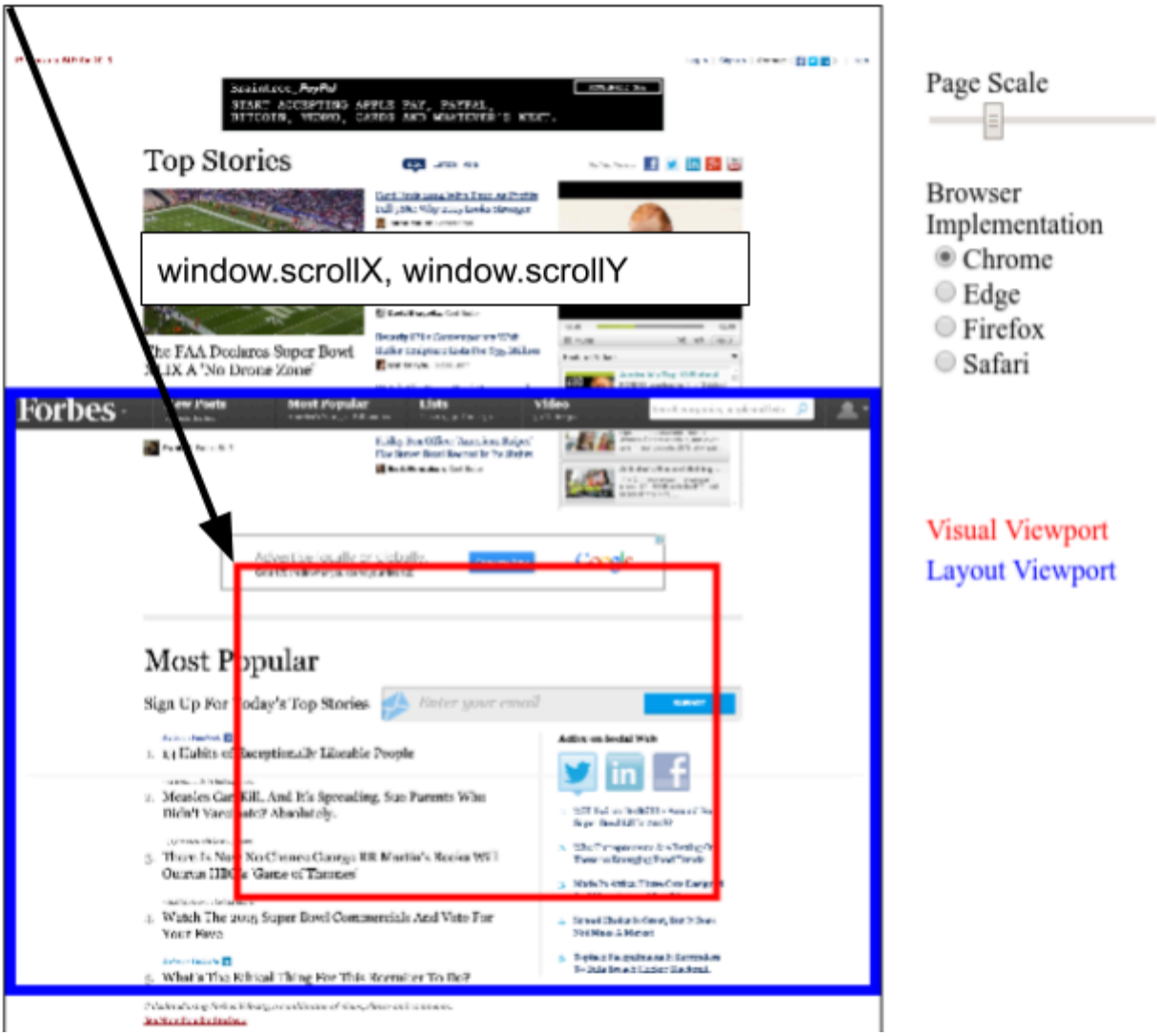
bokan@chromium.org

Introduction

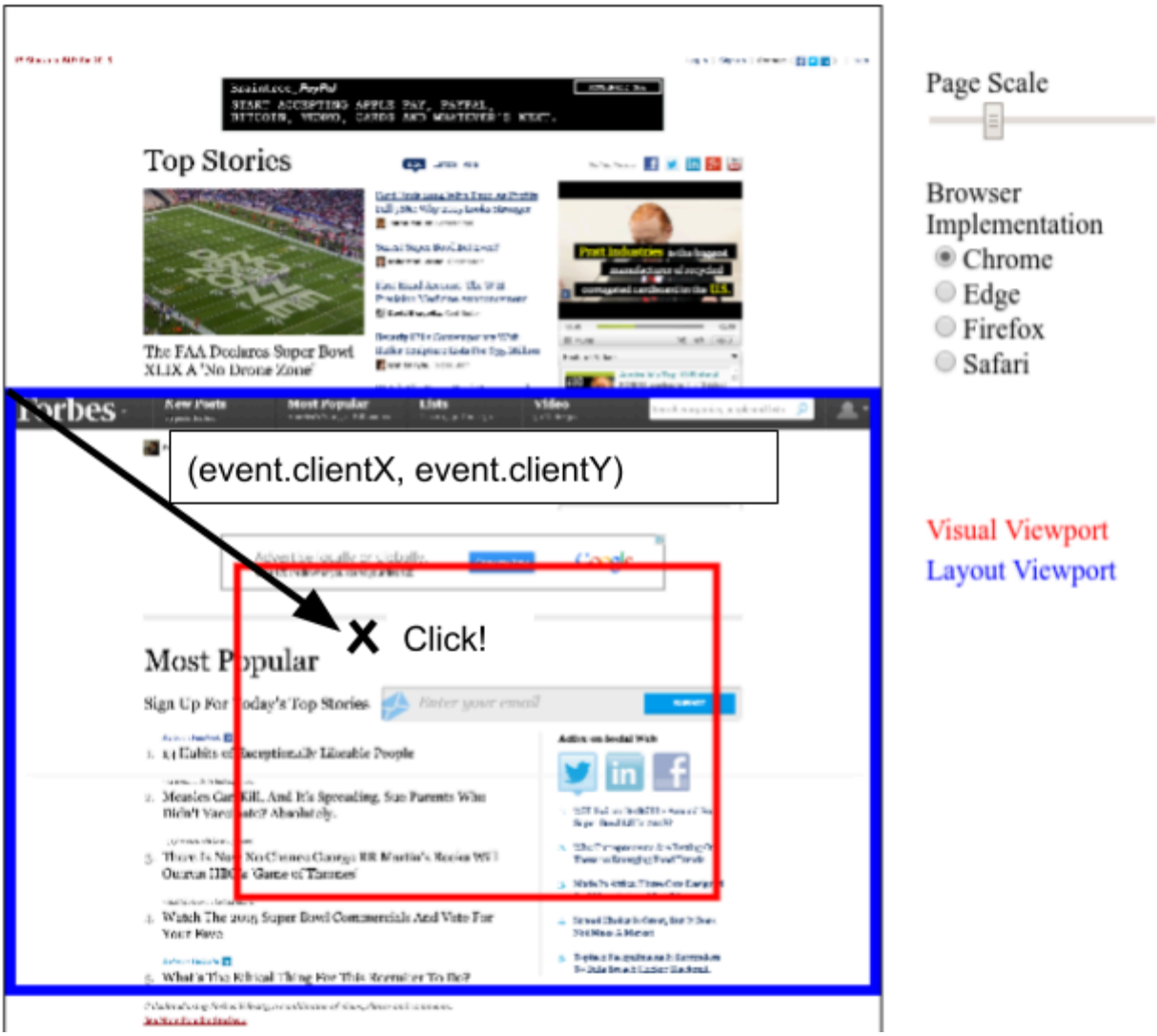
There are several APIs in the web platform that refer to or return coordinates relative to the “viewport”. Until recently, this was unambiguous and well-defined. With the “virtual viewport” introduced in some browsers (IE, Chrome) due primarily to mobile pinch-zoom, there is no longer one viewport - there’s now a “visual” as well as a “layout” viewport. See my demo page for a visual demonstration of how these work.

Unfortunately, this means that APIs referring to “viewport” are now somewhat ambiguous. Which viewport should they refer to? For the most part, Chrome attempted to match IE/Edge, where most of the scrolling and dimension APIs are relative to the visual viewport while “client” coordinates are considered relative to layout.

For example, window.scrollX will return the horizontal distance from the left edge of the *visual* viewport to the left edge of the document. This means that on a non-scrollable page, if the user zooms in and scrolls right, scrollX will be positive:



However, MouseEvent.clientX will return the horizontal distance between the left edge of the *layout* viewport and the mouse event location:



I'm not sure how intentional this choice of mixing viewports for APIs was. It's turned out to be a problem as many web developers will assume that the APIs refer to the same viewport (particularly on desktop pages which are rarely tested on mobile devices and don't zoom - the two viewports are essentially equivalent). For example, many pages make assumptions like: window.scrollY + Element.getBoundingClientRect.y == verticalOffsetInDocument which is no longer true. This has led to a long tail of bugs.

"Inert Visual Viewport"

Chrome tried to solve this discrepancy by making all APIs relative to the layout viewport. In effect, this made pinch-zoom invisible to web pages. As far as the page knew, the user was always looking at the page fully zoomed-out.

This has the advantage that the page never changes due to pinch-zoom and it becomes a browser-side user feature rather than part of the web platform. The way we think of pinch-zoom in Chrome is that it's mostly an accessibility and legacy page feature used for non-mobile ready pages. If a site wants to enable richer zoom interaction (e.g. Google Maps), it can implement it itself using touch events and CSS transforms. The design of the browser's pinch-zoom being entirely off-thread driven means that interacting with it from JS leads to bad user experiences and should be avoided.

Chrome shipped the "inert-visual-viewport" change in M48. However, due to backlash from developers we reverted the change in M49. Fragmenting the behavior between browsers more than it already is was painful for developers. The flag still exists in Chrome: chrome://flags/#inert-visual-viewport

Alternatively…

We could switch to use "visual" everywhere instead. This would be more compatible with how the web works today but has the downside that pinch-zoom can (perhaps unexpectedly) cause side effects on the page. Particularly on desktop pages that are generally built without testing with a touchscreen.

Today's Behavior

Tested using viewporttest.html

	Chrome	*Edge*	*Firefox(*)*	*Safari (Mac)*	*Safari (iOS)*
window.scroll{X|Y}	Visual	Visual	Both	Visual	Visual
window.scroll[To|By]	Visual	Visual	Both	Visual	Visual
window.inner{Width|Height}	Visual	Visual	Both	Visual	Visual
MouseEvent.client{X|Y}	Layout	Layout	Both	Visual	Visual
Touch.client{X|Y}	Layout	N/A	Both	N/A	Visual
Document.element[s]FromPoint	Layout	Layout	Both	Visual	Visual
Element.getBoundingClientRect	Layout	Layout	Both	Visual (but size unscaled) ~~Buggy (bug)~~	Visual

Element.scrollIntoView(IfNeeded)	Visual	Layout (IfNeeded = N/A)	Both (IfNeeded= N/A)	Visual	Visual
Element.scroll{Top|Left} (For scrollingElement)	Visual	Layout	Both	Visual	Visual

*Firefox has only one viewport but it can be thought of as having both and just always having the same size and location.