document.rootScroller and OOPIF

bokan@chromium.org - Sept 28, 2017

PUBLIC

This document outlines some of the issues and possible options to making document.rootScroller work correctly in an OOPIF-enabled world.

What is document.rootScroller?

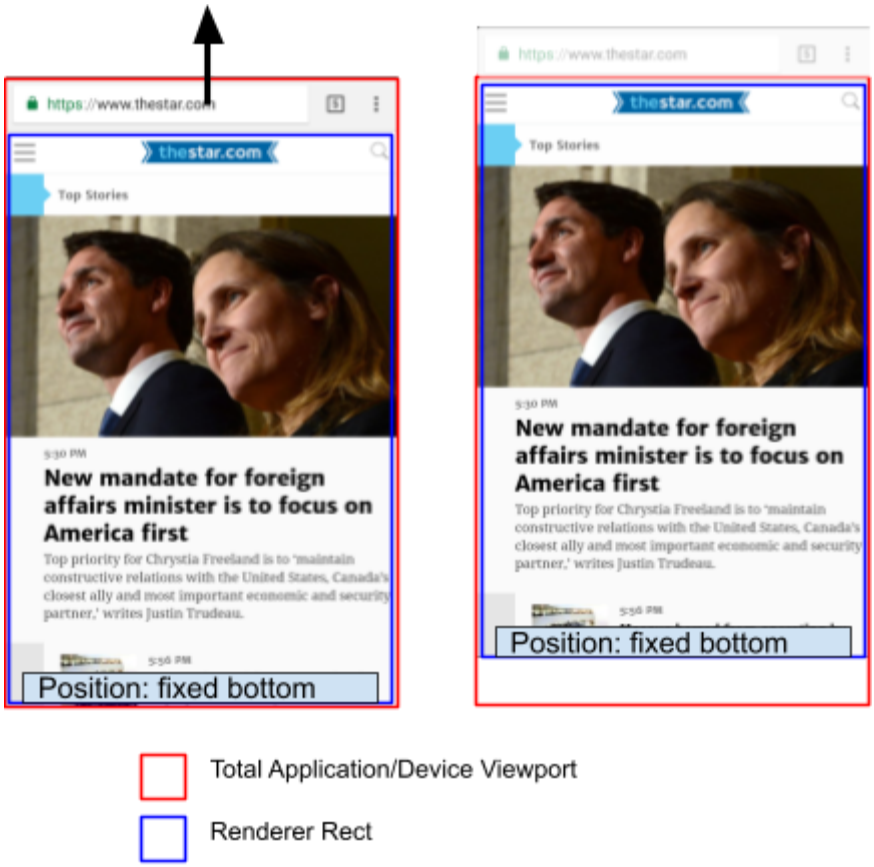
See the appendix

The URL bar transform

Today (putting rootScroller aside for a moment), URL bar movement is produced by the renderer's compositor (CC). CC keeps track of the URL bar's status (full-height, currently shown ratio). As the viewport is scrolled in CC, it changes the currently shown ratio.

The URL bar itself is drawn by the browser process, so when CC produces a CompositorFrame it sends the current height and shown ratio to the browser in the CompositorFrameMetadata. The browser's display compositor users this information to move the URL bar up/down and positions the renderer so that it's directly below the URL bar.

However, until the user lifts their finger -- and the URL bar is fully shown/hidden -- the renderer itself isn't resized (this is a performance optimization). This means that hiding the URL bar will expose an area outside the renderer at the bottom of the viewport:



As the URL bar is hidden, the renderer is shifted upwards but it isn't resized

Thus, in order to make this appear correctly, the renderer's compositor does some sleight of hand and expands the root clipping and scrolling layers' bounds. It does this by calculating the difference between how much the URL bar has been hidden since the last renderer's resize. Once the renderer is resized this adjustment is cleared.

Additionally, position: fixed Elements that stick to the bottom of the viewport get shifted by this same amount in order that they stay fixed to the viewport bottom.

Now with rootScroller…

This becomes more complicated if we want an arbitrary Element to control the URL bar. Consider the case where the rootScroller Element is inside of an iframe. When the URL bar hides, the newly exposed region at the bottom is now obscured by clipping layers in both the top frame and the iframe. Thus, each frame that has the rootScroller as a descendant is marked as non-clipping.

We must also mark each of these frames as being "affected by the URL bar" so that -- in the compositor -- we create transform nodes for position: fixed bottom Elements. These transforms

apply the URL bar “adjustment” talked about above to make sure position: fixed Elements stay fixed to the true viewport bottom, rather than moving up with the renderer.

Enter: OOPIF

The above approaches -- disabling clipping and adding transform nodes -- is possible because the entire frame/layer tree is in the same renderer, making it easy to make global changes. With OOPIF, this becomes more difficult since we now have to coordinate across processes.

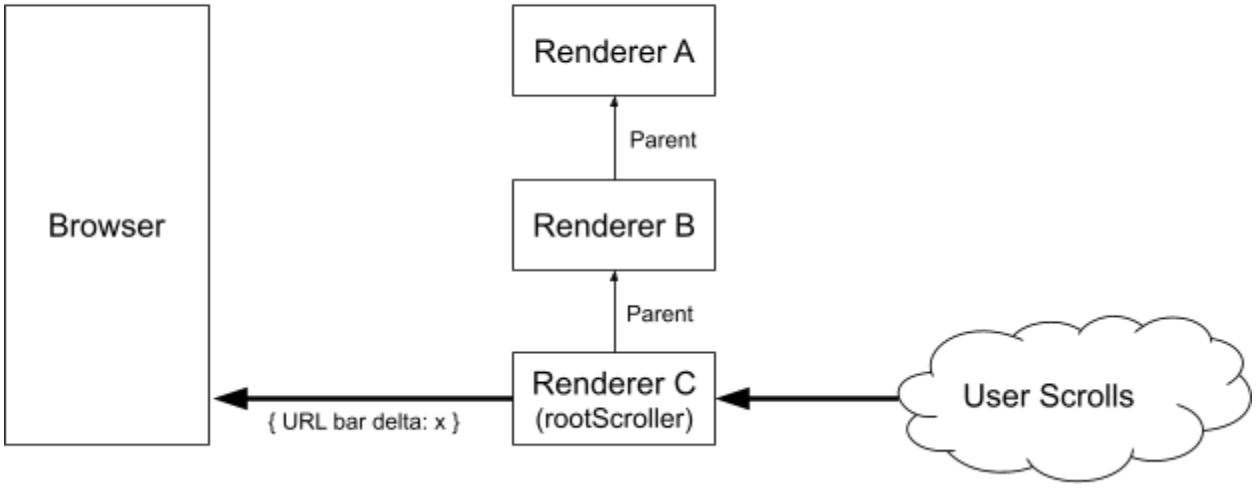
We somehow need to communicate the URL bar changes from the process that has the current rootScroller Element to each process that is a parent. Unfortunately, we can’t centralize this (yet*). Since each process sends rastered CompositorFrames to the browser, any bottom position: fixed Elements need to be rastered in the correct position so each *renderer* needs to know the URL bar adjustment to apply. Additionally, we can’t use the “make all iframes non-clipping” trick across processes either. Each renderer process will need to know how large of a surface it needs to raster into and this needs to change as the URL bar hides.

(*) Eventually: renderers will send unrasterized DisplayLists to the GPU process where they’ll be composited together and rasterized. The DisplayLists could have information about fixed Elements so this would be the ideal place to do all the URL bar based adjustments. Alas, we’re not there yet.

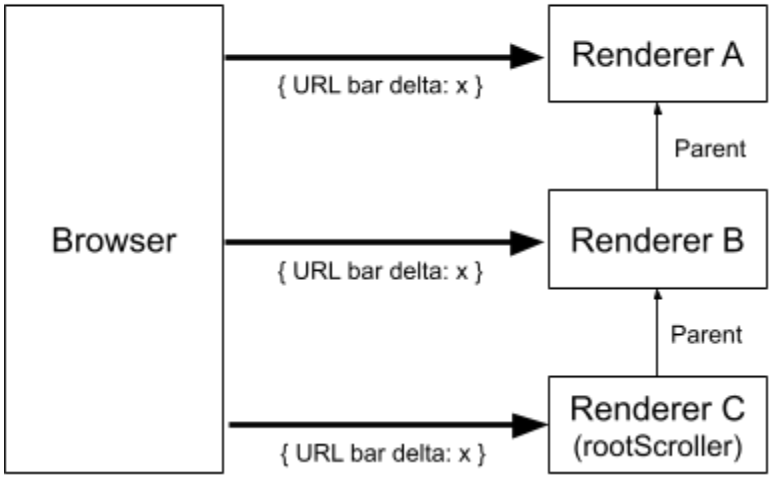
Option 1

The renderer with the rootScroller sends the URL bar delta to some “broker” service in the browser. The broker then sends the URL bar state to each renderer that’s an ancestor of the rootScroller.

Frame 0



Frame 1



Note that we send the URL bar state back to the rootScroller renderer. We don't want to immediately apply the URL bar adjustment in the rootScroller renderer since it'll be out of sync with its parent renderers (i.e. bottom position: fixed elements from the rootScroller renderer and parent renderer will move relative to each other).

The main issue I see with this approach is synchronization issues. What do we do when one renderer takes a long time to generate a frame? When do we decide to actually move the URL bar up in the browser? These will have unfortunate visual artefacts if we break synchronization. Presumably there's other features that would also suffer from this kind of synchronization issues (window resize, orientation change, etc), how do they solve them? Has this been thought about?

Option 2

This is much simpler: just force all renderers that are “affected by the URL bar” into a single process. This way the current implementation doesn’t change at all.

We already don’t make any guarantees on process isolation so this shouldn’t be a major security concern. Additionally, setting a frame as the rootScroller should signal some level of trust (at least from the parent to the child).

Do we currently support process swapping like this? How complicated is it? What are the perf characteristics like?

Option 3 - (Future)

At some point, renderers will stop sending rastered resources and send DisplayLists instead (or some other similar data structure). We could add a marker in these lists to specify which item is the rootScroller in each renderer. The display compositor could then determine which clipping rects to adjust and which position: fixed items to shift when the URL bar is moved.

This avoid most/all the synchronization issues in Option 1 while keeping all the benefits of OOPIFs. Unfortunately, it’s a long way out and its unclear exactly how this will look so we can’t do this today.

“Option” 4

OOPIF has been well thought about and I’m just coming into these issues so there’s likely better approaches than what I’ve thought of.

Appendix: What and How of document.rootScroller

See the explainer for details. In short, document.rootScroller provides a way to specify that an arbitrary scroller on a page controls the URL bar’s hiding/show motion. It also give the scroller other powers (like overscroll glow) but the URL bar is the most important in terms of usefulness and most challenging in terms of implementation.

A page can specify that a scrollable Element should control the URL bar by setting it as the document.rootScroller:

```
<div id=”scroller”>...</div>
<script>
     document.rootScroller = document.getElementById(‘scroller’);
</script>
```

Note that #scroller must exactly fill the initial containing block (i.e. it must be viewport filling). This is a restriction we’ve placed to make implementation easier. See the usage guide for full details on usage and restrictions.

An important point for the purposes of this document is that the page can set an <iframe> as its document.rootScroller. When it does this, we use the iframe’s rootScroller to control the URL bar. If the iframe itself doesn’t set a document.rootScroller, scrolling its document controls the URL bar. If it did set its document.rootScroller, scrolling that Element controls the URL bar. In other words, document.rootScroller is *composable*. The top level Document starts off with ownership of URL bar movement and it delegates it down through <iframe>s.

For implementation details, see the README.md.