paulirish / what-forces-layout.md

Last active yesterday

What forces layout/reflow. The comprehensive list.

what-forces-layout.md

What forces layout / reflow

All of the below properties or methods, when requested/called in JavaScript, will trigger the browser to synchronously calculate the style and layout*. This is also called reflow or layout thrashing, and is common performance bottleneck.

Generally, all APIs that synchronously provide layout metrics will trigger forced reflow / layout. Read on for additional cases and details.

Element APIs

Getting box metrics

- `elem.offsetLeft`, `elem.offsetTop`, `elem.offsetWidth`, `elem.offsetHeight`, `elem.offsetParent`
- `elem.clientLeft`, `elem.clientTop`, `elem.clientWidth`, `elem.clientHeight`
- `elem.getClientRects()`, `elem.getBoundingClientRect()`

Scroll stuff

- `elem.scrollBy()`, `elem.scrollTo()`
- `elem.scrollIntoView()`, `elem.scrollIntoViewIfNeeded()`
- `elem.scrollWidth`, `elem.scrollHeight`
- `elem.scrollLeft`, `elem.scrollTop` also, setting them

Setting focus

- `elem.focus()` (source)

Also...

- `elem.computedRole`, `elem.computedName`
- `elem.innerText` (source)

Getting window dimensions

- `window.scrollX`, `window.scrollY`
- `window.innerHeight`, `window.innerWidth`
- window.visualViewport.height / width / offsetTop / offsetLeft (source)

document

- `document.scrollingElement` only forces style
- `document.elementFromPoint`

Forms: Setting selection + focus

- `inputElem.focus()`
- `inputElem.select()`, `textareaElem.select()`

Mouse events: Reading offset data

- `mouseEvt.layerX`, `mouseEvt.layerY`, `mouseEvt.offsetX`, `mouseEvt.offsetY` (source)

Calling getComputedStyle()

`window.getComputedStyle()` will typically force style recalc.

`window.getComputedStyle()` will often force layout, as well.

▶ Details of the conditions where gCS() forces layout

Getting `Range` dimensions

- `range.getClientRects()`, `range.getBoundingClientRect()`

SVG

Quite a lot of properties/methods force, but I haven't made an exhaustive list. This list in incomplete:

- SVGLocatable: `computeCTM()`, `getBBox()`
- SVGTextContent: `getCharNumAtPosition()`, `getComputedTextLength()`, `getEndPositionOfChar()`, `getExtentOfChar()`, `getNumberOfChars()`,

`getRotationOfChar()`, `getStartPositionOfChar()`, `getSubStringLength()`, `selectSubString()`

- SVGUse: `instanceRoot`

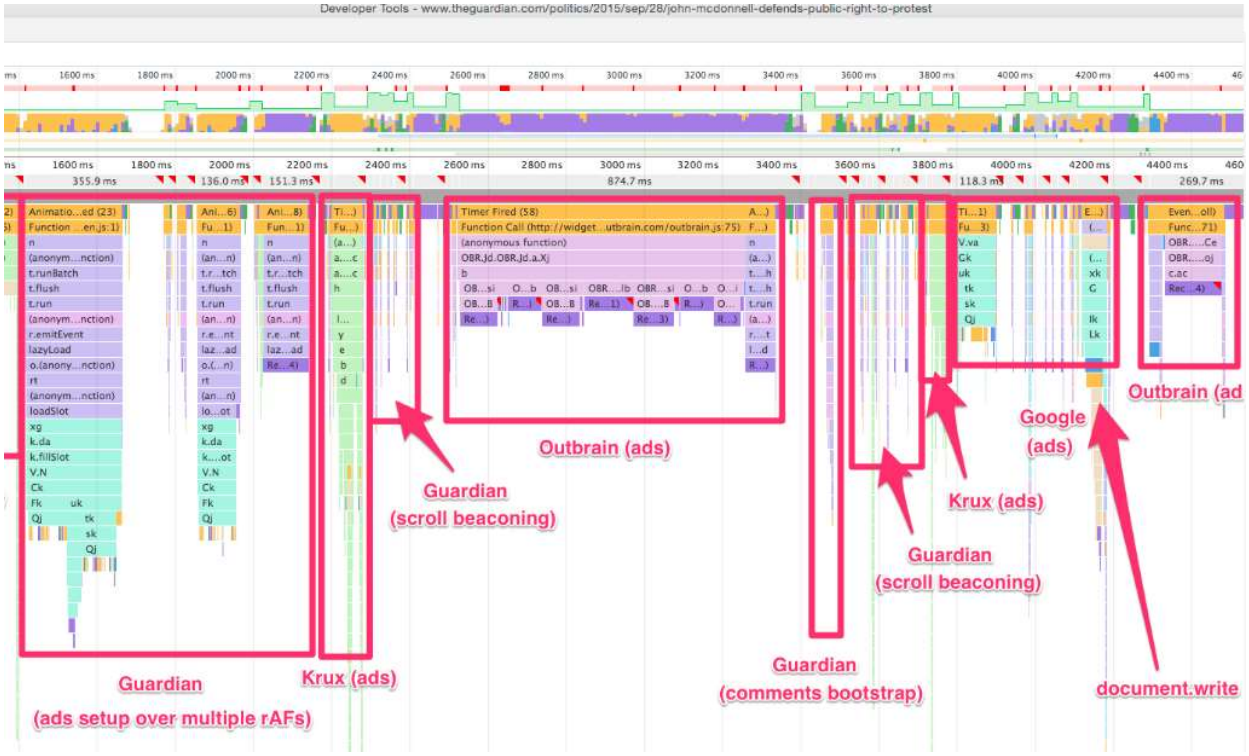
Use the "chromium source tree link" below to explore on your own!

contenteditable

- Lots & lots of stuff, ...including copying an image to clipboard (source)

* Appendix

- Reflow only has a cost if the document has changed and invalidated the style or layout. Typically, this is because the DOM was changed (classes modified, nodes added/removed, even adding a psuedo-class like :focus).
- If layout is forced, style must be recalculated first. So forced layout triggers both operations. Their costs are very dependent on the content/situation, but typically both operations are similar in cost.
- What should you do about all this? Well, the `More on forced layout` section below covers everything in more detail, but the short version is:
  i. `for` loops that force layout & change the DOM are the worst, avoid them.
  ii. Use DevTools Performance Panel to see where this happens. You may be surprised to see how often your app code and library code hits this.
  iii. Batch your writes & reads to the DOM (via FastDOM or a virtual DOM implementation). Read your metrics at the begininng of the frame (very very start of `rAF`, scroll handler, etc), when the numbers are still identical to the last time layout was done.



Timeline trace of The Guardian. Outbrain is forcing layout repeatedly, probably in a loop.

Cross-browser

- The above data was built by reading the Blink source, so it's true for Chrome, Opera, Brave, Edge and most android browsers. You can browse them yourself in the Chromium source tree.
- Tony Gentilcore's Layout Triggering List was for 2011 WebKit and generally aligns with the above.
- Modern WebKit's instances of forced layout are mostly consistent: `updateLayoutIgnorePendingStylesheets` - GitHub search - WebKit/WebKit
- Gecko's reflow appears to be requested via FrameNeedsReflow. Results: `FrameNeedsReflow` - mozilla-central searchfox
- No concrete data on IE or EdgeHTML, but they likely were roughly the same, as the return values for these properties are spec'd.

More on forced layout

- **Avoiding layout thrashing — Web Fundamentals** The **best** resource on identifying and fixing this topic.
- CSS Triggers - covers what operations are required as a result of setting/changing a given CSS value. The above list, however, are all about what forces the purple/green/darkgreen circles synchronously from JavaScript.
- Fixing Layout thrashing in the real world | Matt Andrews
- Timeline demo: Diagnosing forced synchronous layouts - Google Chrome
- Preventing 'layout thrashing' | Wilson Page

- wilsonpage/fastdom
- Rendering: repaint, reflow/relayout, restyle / Stoyan
- We spent a week making Trello boards load extremely fast. Here's how we did it. - Fog Creek Blog
- Minimizing browser reflow | PageSpeed Insights | Google Developers
- Optimizing Web Content in UIWebViews and Websites on iOS
- Accelerated Rendering in Chrome
- web performance for the curious
- Jank Free

Updated slightly April 2020. Codesearch links and a few changes to relevant element properties.

Load earlier comments...

Waltari10 commented on May 24, 2019

I am confused about mixing reads and writes. So first doing all the reads and then the writes will be more efficient than mixing them up?

So is it better to:
read()
read()
write()
write()

Than:
read()
write()
read()
write()

Why is it better to id this way? Cant you batch reads and writes together anyway using FastDom?

kirill-chirkov-at-cl... commented on Jul 4, 2019 • edited

What about `elementFromPoint`? https://developer.mozilla.org/en-US/docs/Web/API/DocumentOrShadowRoot/elementFromPoint
Does it force reflow?

EECOLOR commented on Jul 13, 2019

@paulirish Thank you for the gist.

It states:

> Read your metrics at the begininng of the frame (*very very start of rAF*, scroll handler, etc),

The *very very start of rAF* seems unrealistic in real life applications, some other parts of the application (or a library) might have done a write already. Wouldn't it be better to state that in general reads should be done anywhere except for rAF (or simpler: reads should not be done in rAF)? And, to make that work: writes should only be done in rAF.

cmcculloh commented on Oct 31, 2019

@paulirish the SVG link is dead. I made some updates to your gist to link to the blog post on Internet Archive, but also just went ahead and copied in the relevant content.

https://gist.github.com/cmcculloh/ed188bc6286a9afb9e0f1c971322c9be

Thanks for this gist!

Losses commented on Jan 27, 2020

@paulirish Hmmmm, what about `entry.rootBounds` or `entry.boundingClientRect` of `IntersectionObserver`'s callback parameter?

paulirish commented on Jan 28, 2020

@cmcculloh you da real mVP. thx

@Losses i am 95% sure they dont. the intersectionobserver was designed to give you the most up to date values, but reading them won't force a brand new layout.

softwareCobbler commented on Apr 29, 2020

Should setting an `input[type=text]`'s `value` property be on this list? I have 650ms worth of `this.querySelector("input").value = someValue;` to donate.

Losses commented on Apr 29, 2020

Another question, will these code trigger double reflow while user resizing their window?

```
window.addEventListener('resize', () => console.log(window.innerWidth))
```

One reflow may from window resize itself, another reflow comes from getting `window.innerWidth`

I wonder if the browser engine will merge these two reflow into one, this deeply confused me...

EECOLOR commented on Apr 30, 2020

@Losses The basic idea is that all of those methods will ensure they give the correct values. This means that if something has changed since the last calculation, they will calculate again to be sure.

A cycle in the browser consists of a few different sections (they go in circles, so me starting with one does not have any meaning) _ `paint` _ `regular` _ `animation frame` _ `paint` _ ....

With javascript we can execute code in 2 places: `regular` and `animation frame`. All code is not in `requestAnimationFrame()` will execute in `regular`. If you change something in the DOM and then ask the DOM about some measurement, it will calculate it for you. If you change something in the DOM before `paint` and ask the DOM about some measurement after `paint` it does not need to calculate, because it already knows.

So to be safe, make sure you only read from the DOM in `regular` and only write to the DOM in `animation frame`. Important: do not write to the DOM in `regular` and do not read from the DOM in `animation frame`.

This presentation helped for me: https://www.youtube.com/watch?v=cCOL7MC4Pl0

Losses commented on Apr 30, 2020

@EECOLOR Wow, thanks for your answer, it really help!

skortchmark9 commented on May 5, 2020 • edited

Is there any way to avoid relayout due to changing text in a text node? I've tried many variations of fixed width/height, overflow, `contain`, monospaced fonts, all to no avail. I would even settle for being to limit the layout to the parent node, but right now it always seems to leak to the whole document...I'm literally considering using canvas for rendering text which just seems so wrong

andfinally commented on Jun 23, 2020

I'm interested to see `matchMedia` isn't in this list, since that must involve measuring window dimensions.

melkishengue commented on Jul 28, 2020

Great list, good to know !!

htho commented on Sep 8, 2020

I found another really nasty one: inserting `<iframe>` elements.

```
setTimeout(() => document.body.insertAdjacentHTML("beforeend", "<div><iframe src=\"\" style=\"dis
```

htho commented on Sep 11, 2020

At least in jQuery 2.x the event handling component makes a copy of each property in an event (at least for click events). This includes the properties mentioned above.
This happens even though I have no jquery click events registered.
My workaround is to make a custom build of jquery, where I override the original copy routine with a routine that instead setups getters.

```
// someScriptBeforeJQueryIsIncluded.js

// activate workaround
window.FF_fix_useGetter = true;
```

```
// https://github.com/jquery/jquery/blob/2.2-stable/src/event.js#L445
fix: function( event ) {
	if ( event[ jQuery.expando ] ) {
		return event;
	}

	// Create a writable copy of the event object and normalize some properties
	var i, copy,
		type = event.type,
		originalEvent = event,
		fixHook = this.fixHooks[ type ];

	if ( !fixHook ) {
		this.fixHooks[ type ] = fixHook =
			rmouseEvent.test( type ) ? this.mouseHooks :
			rkeyEvent.test( type ) ? this.keyHooks :
			{};
	}
	copy = fixHook.props ? this.props.concat( fixHook.props ) : this.props;

	event = new jQuery.Event( originalEvent );

	i = copy.length;
	if(!window.FF_fix_useGetter) {
		// original approach
		var prop;
		while ( i-- ) {
			prop = copy[ i ];
			event[ prop ] = originalEvent[ prop ];
		}
	} else {
		// getter approach
		while ( i-- ) {
			const prop = copy[ i ];
			Object.defineProperty(event, prop, {
				get: function() { return originalEvent[ prop ]; }
			})
		}
	}

	// Support: Cordova 2.5 (WebKit) (#13255)
	// All events should have a target; Cordova deviceready doesn't
	if ( !event.target ) {
		event.target = document;
```

```
        }

        // Support: Safari 6.0+, Chrome<28
        // Target should not be a text node (#504, #13143)
        if ( event.target.nodeType === 3 ) {
                event.target = event.target.parentNode;
        }

        return fixHook.filter ? fixHook.filter( event, originalEvent ) : event;
},
```

a-boertien commented on Sep 23, 2020

Currently in Chromium: entering text input into a text area causes Layout:

https://bugs.chromium.org/p/chromium/issues/detail?id=1116001&q=textarea%20performance&can=2

adrianoellero commented on Jan 5, 2021

look at this: https://codepen.io/chriscoyier/pen/EyRroJ

aquaductape commented on Mar 2, 2021 • edited

I tested `scrollY` inside window scroll event on Chrome Version 88.0.4324.150 and it doesn't fire Layout calculation.

The page is very simple, has 2000 words, so that there's a scrollbar to test.
I ran two tests by using the Performance Tab. One that has window scroll event that logs `scrollY` and the other that doesn't have js files.

Here are the comparisons of Tasks.

The one that doesn't have js files.

```
Scroll > Event:scroll > Update Layer Tree > Composite Layers
```

The one that does log `scrollY` on the scroll event.

```
Scroll > Event:scroll > Update Layer Tree > Composite Layers
         Function call
         window.addEventListener.passive
```

The one that logs `scrollY` is almost identical, except for the js part where you see Function call. If there was Layout thrashing, inside the Task there would be 'Layout'.

Here's a demo https://8mxws.csb.app/
Here's the source https://codesandbox.io/s/testscrollyreflow-8mxws?file=/app.js

I didn't test within codesandbox, I tested locally, but i'm sharing it that way cuz it's the easiest way

mfbx9da4 commented on Mar 29, 2021 • edited

If every item in the scrollable list has a `&:hover {...` will this cause layout thrashing when scrolling?

lazysergey commented on May 15, 2021

> `Scroll > Event:scroll > Update Layer Tree > Composite Layers`
> **@aquaductape** how do I get same output for debugging? any way to see what stages getting executed by browser?

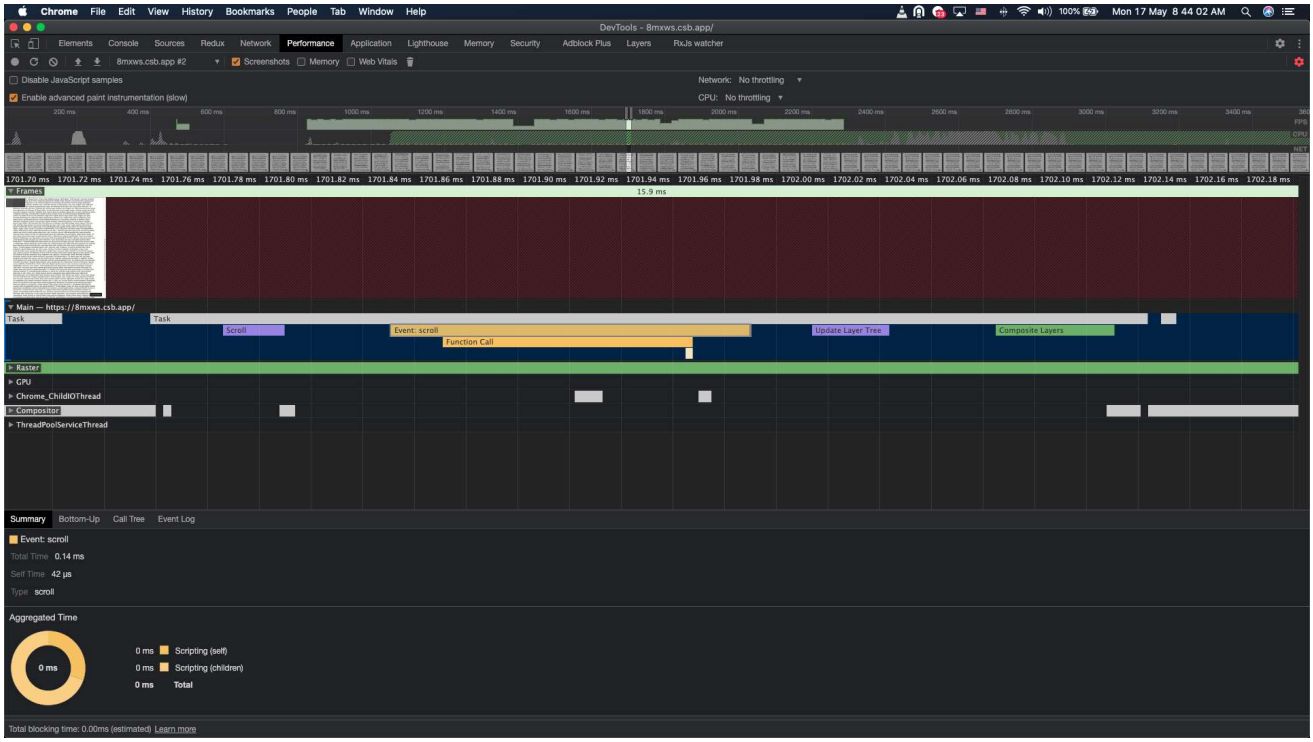
aquaductape commented on May 15, 2021 • edited

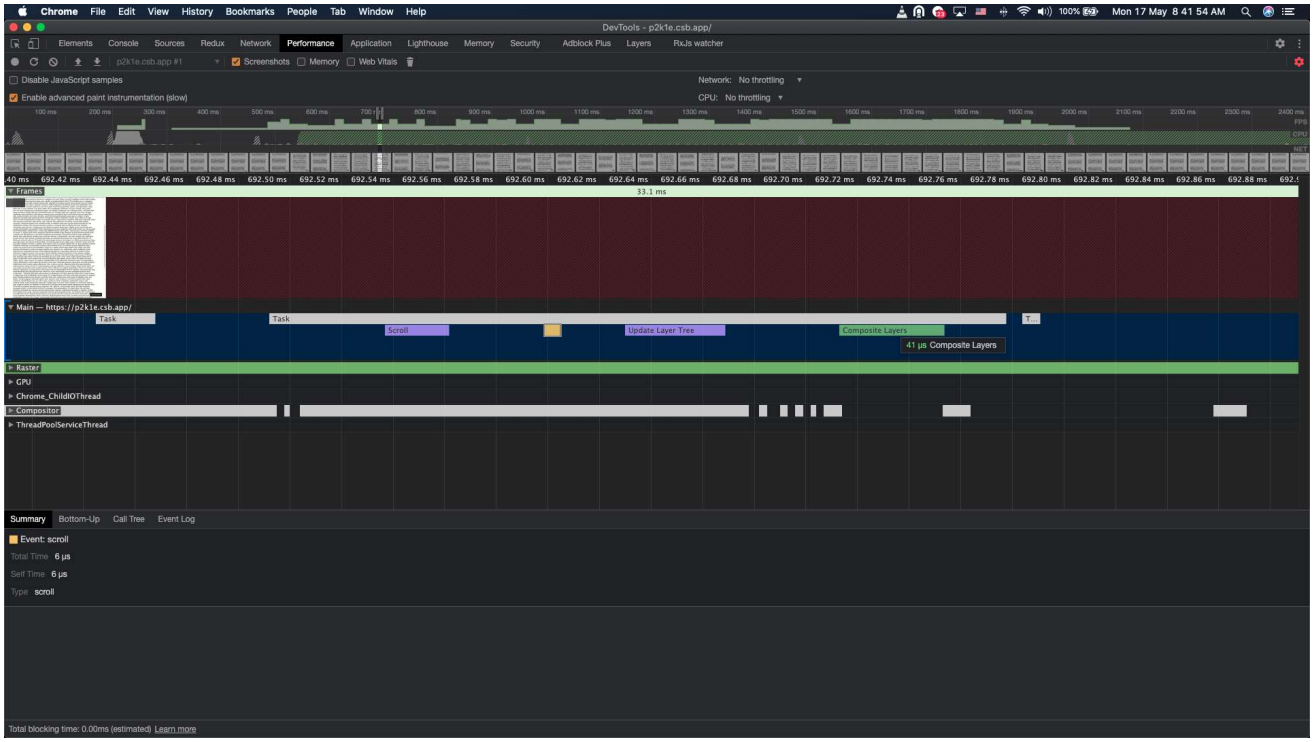
@lazysergey From what I know, the Performance result in chrome devtools doesn't format it in a text based way. The result is shown in GUI timeline and you hover over the cells and it tells what it is. So instead of showing a screenshot, I typed out the result as is.

But here's the screenshot in case

lazysergey commented on May 17, 2021 • edited

thanks, for me they're also almost identical, I don't see any layout tasks





waruyama commented on Jul 1, 2021

Oh dear, it looks like changing the body's cursor with `document.body.style.cursor = 'move';` invalidates the layout for the whole page. Ouch!

nullhook commented on Jul 2, 2021

@waruyama do you have a test link?

waruyama commented on Jul 2, 2021

@nullhook
Here is a little test case that creates an SVG with a massive amount of circles.

https://jsfiddle.net/17svhrqy/3/

1. Run it and wait until "children created" is written to the console.
2. Click on the red square and move the pointer.

If the cursor is changed on pointerdown, there is a gap of 500ms before mouse move event is triggered. Without changing the cursor (you have to comment that line), this gap is only 170ms. I am pretty sure that recalculation of the style is causing the difference.

mootari commented on Aug 8, 2021

@paulirish The Fog Creek Blog link is dead. Here is the archive.org link: https://web.archive.org/web/20180401153527/https://blog.fogcreek.com/we-spent-a-week-making-trello-boards-load-extremely-fast-heres-how-we-did-it/

htho commented on Sep 7, 2021

I've started a little repository to reproduce these issues in order to learn from them. Maybe we can start to collect best practices there and see if some issues are resolved in the future.

https://github.com/htho/what-forces-layout-reflow

matonga commented on Nov 26, 2021

> Is there any way to avoid relayout due to changing text in a text node? I've tried many variations of fixed width/height, overflow, `contain`, monospaced fonts, all to no avail. I would even settle for being to limit the layout to the parent node, but right now it always seems to leak to the whole document...I'm literally considering using canvas for rendering text which just seems so wrong

@skortchmark9 thank you! I tried with an embedded SVG first, but it didn't do the trick. Using canvas was the only solution that worked. Anyway if someone knows some alternative I'd like to hear about it.

dSalieri commented on Feb 11, 2022 • edited

Assignment to `elem.style.transform` value, invokes paint stage, also assignment to `elem.style.left` value, invokes paint stage twice. But if you will create property `will-change`, you don't see paint stage nowhere.

Kaiido commented on Mar 11, 2022

A sneaky one I just found in the canvas2D API which currently only concerns Firefox:
setting `ctx.fillStyle`, `ctx.strokeStyle` or `ctx.shadowColor` will force a relayout there, when the associated canvas element is attached to the DOM.
This is because these values can accept a value of `"currentColor"`, so the browser has to perform a recalc to get the computed color. Chrome and Safari have a bug where they only get that value from the canvas element's `style` attribute, so they don't need to trigger the recalc there. However their behavior is against the specs, so it may change in the future (or maybe the specs will change).

Test page: https://jsfiddle.net/sxe36Lk1/