I am looking for a proper way to implement lazy loading of images without harming printability and accessibility, and without introducing layout shift (content jump), preferrably using native `loading=lazy` and a fallback for older browsers. Answers to the question How lazy loading images using JavaScript works? included various solutions none of which completely satisfy all of these requirements.

An elegant solution should be based on valid and complete html markup, i.e. using `<img src`, `srcset`, `sizes`, `width`, `height`, and `loading` attributes instead of putting the data into `data-` attributes, like the popular javascript libraries lazysizes and vanilla-lazyload do. There should be no need to use `<noscript>` elements either.

Due to a bug in chrome, the first browser to support native lazyloading, images that have not yet been loaded will be missing in the printed page.

Edit Manage Stats

Ingo Steinke
Posted on 16 Dec 2021

Printable Lazy Loading

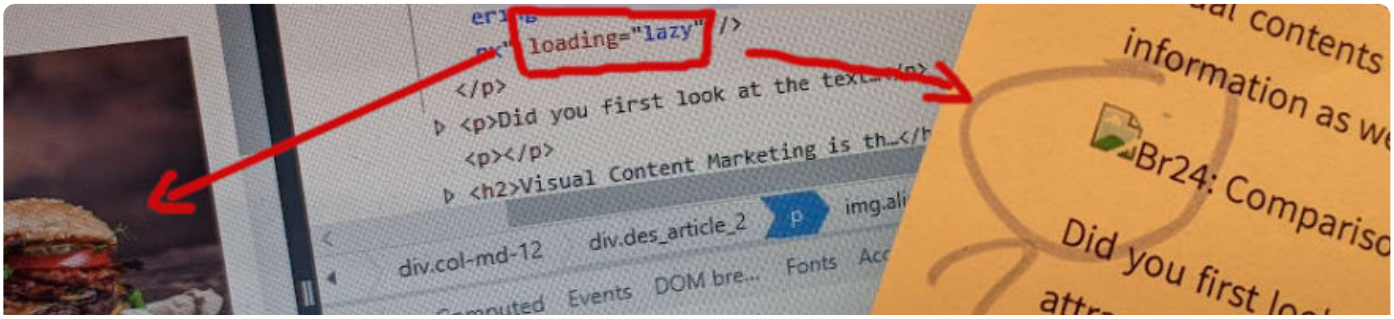
#webdev #html #performance #a11y

The past year has been a good one. Well, at least for page speed optimization. With Firefox support for natively loading images, it has never been so easy to defer image loading!

But wait ...

What about Printing?

Accessibility and progressive enhancement usually go together well with "web performance" (page speed optimization), but **lazy loading** has always been tricky. For a long time, it was not possible without using JavaScript, and if it's done the wrong way, you risk that your website's visitors will not see the images they are supposed to see.



Lazy loading images with accessibility and printer support was still an open issue when I relaunched my portfolio website last year.

Understanding the Problem

Due to a bug in Chrome, the first browser to support native lazyloading, images that have not yet been loaded will be missing in the printed page.

What to use Lazy Loading for?

Benefits of lazy loading: web performance, page speed, load time optimization can be good for user experience and search engine optimization when done in the right way.

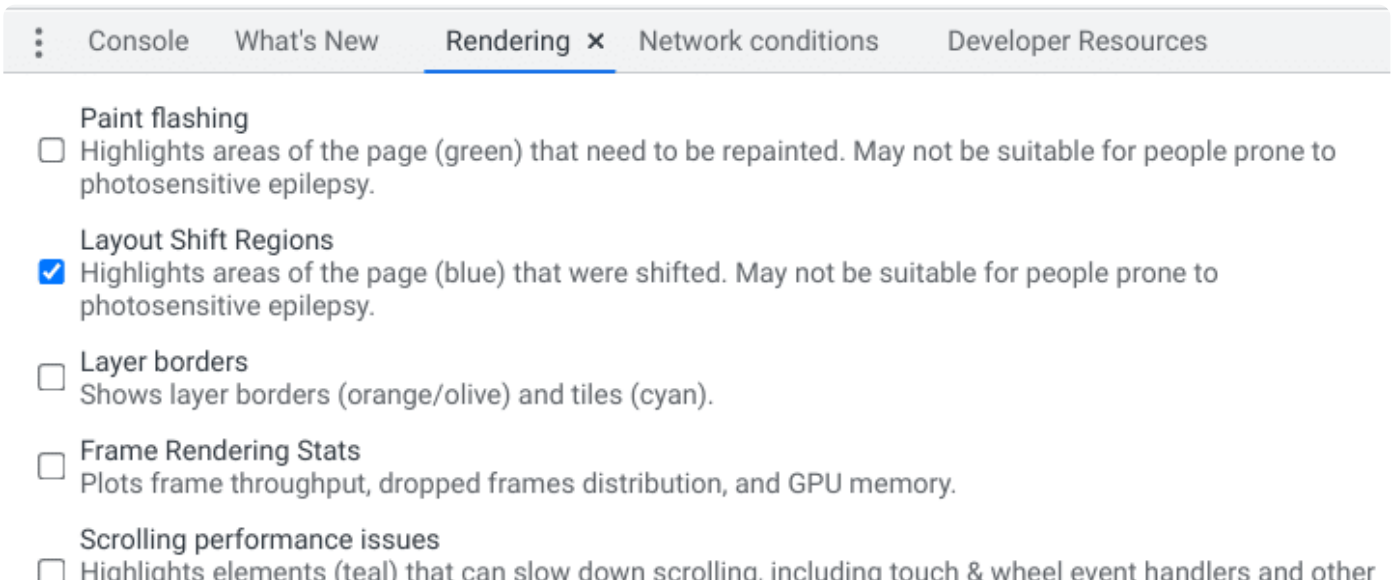
Layout Shift

Layout shift can make users click on the wrong links by accident and make them lose their focus while reading, so it has become a core web vital in 2021.

This problem is caused by the disturbing effect that elements take up more space after being loading completely, thus moving the content below and causing the whole page to shift.

Don't forget to specify the `width` and `height` attributes for every image element, and properly style containing elements to ensure a placeholder that is exactly the same size as the image to be lazily loaded later on. Doing so will prevent layout shift caused by this image or its container element.

How to examine layout shift? developer tools offer to highlight layout shift. In the current Chromium browsers, this option can be found in the "Rendering" tab.



main-thread scrolling situations.

"Above the Fold"

Also you don't want to defer loading elements **"above the fold"** which means elements that are visible as soon as you open a website. The fold, a word dating from printed newspapers, now means the bottom of the initial viewport before scrolling the page.

But you do want to optimize "below the fold" and load images and videos on demand only, otherwise you might make the user load data (which will take time and bandwidth, probably causing costs for the data transfer) that might never be needed at all.

Unpredictabilities of Responsive Web Design

What is above or below the fold depends on the user's devices. See how my blog looks differently on different screens:

Other aspects to consider: Maybe some images are very important and occur several times on the same website so chances are that they will be viewed sooner or later anyway. Maybe they have such a small file size that lazy loading will not make much of a difference.

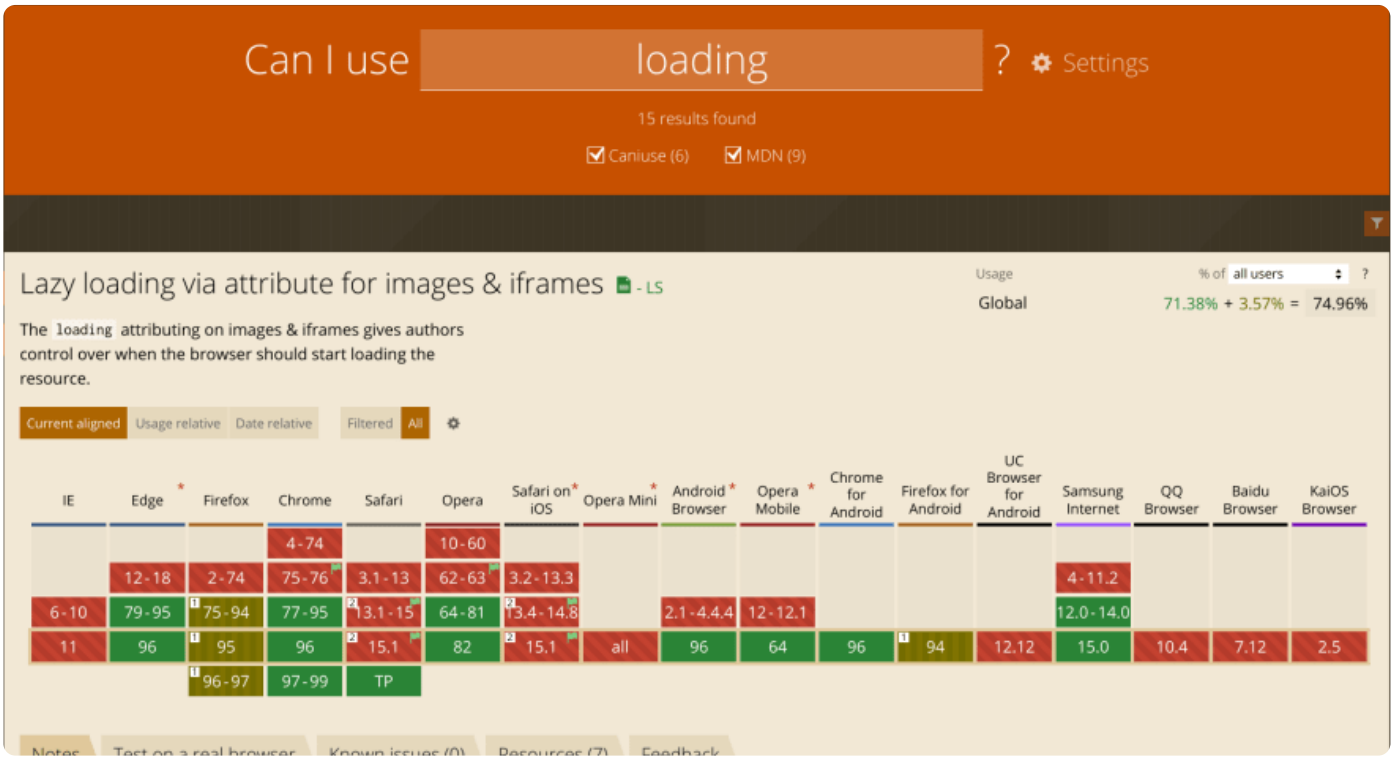
Native Lazy Loading

Either way, you can control which images to defer and which to load as soon as possible by setting the `loading` attribute.

`loading=lazy` activates deferred loading, while
`loading=eager` is the default without any explicit deferring.

Browser Support

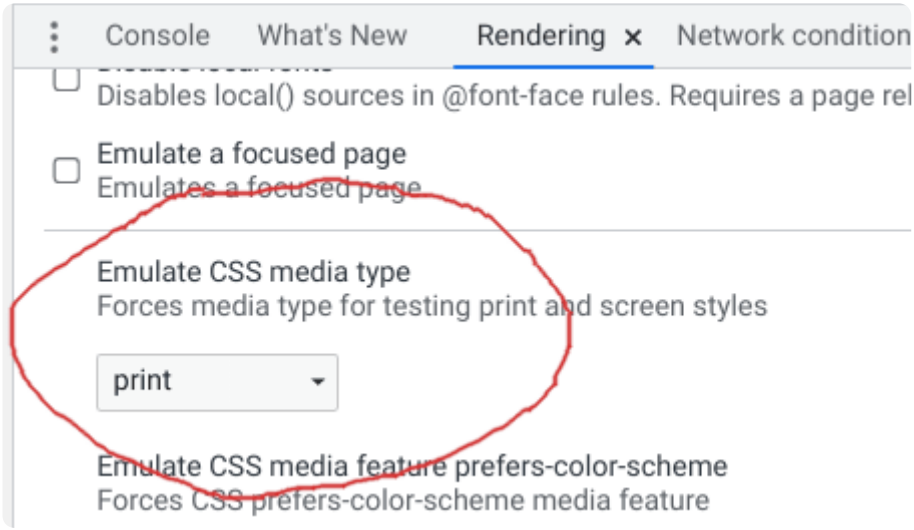
Native lazy loading has first been supported by Google Chrome since Chrome 77 in 2019, and it's currently supported by popular browsers except for the usual suspects (Internet Explorer and Safari).



Testing

Good news: you don't need to buy an actual printer device, to test your website.

- Print preview and saving as PDF will give you a good idea
- You can emulate media type print in the browser's developer tools:



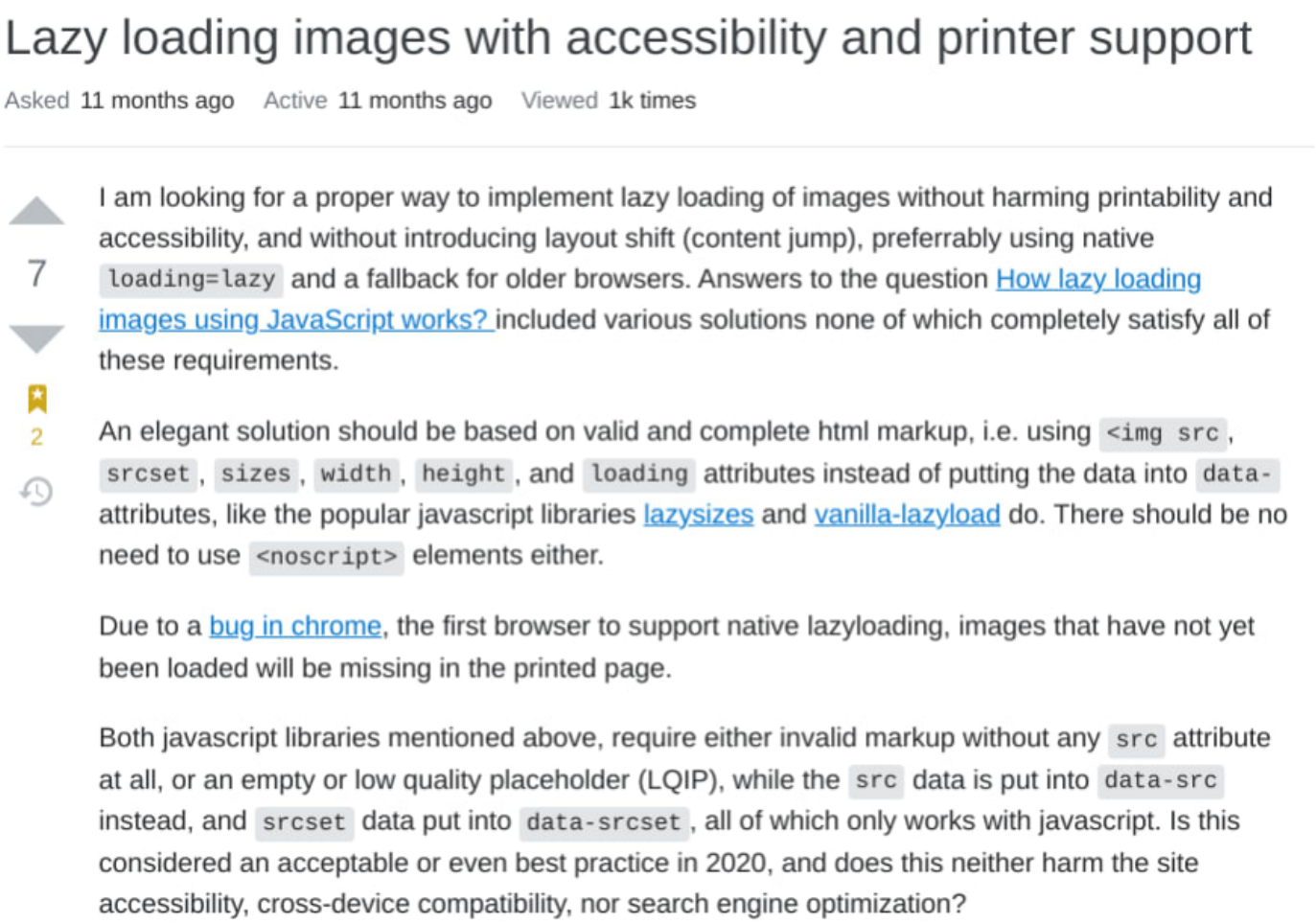
Missing Images

Printing a document without images might actually be beneficial and save our users and our environment wasting toner and paper for irrelevant decoration.

As a web developer or content creator, I want to actively decide what to print and what to omit. But using native lazy loading, every single one of the deferred images is missing in print preview.

Discussion

Lazy loading images with accessibility and printer support, my question on StackOverflow:



Lazy loading images with accessibility and printer support

Asked 11 months ago Active 11 months ago Viewed 1k times

7

2

I am looking for a proper way to implement lazy loading of images without harming printability and accessibility, and without introducing layout shift (content jump), preferrably using native `loading=lazy` and a fallback for older browsers. Answers to the question How lazy loading images using JavaScript works? included various solutions none of which completely satisfy all of these requirements.

An elegant solution should be based on valid and complete html markup, i.e. using `<img src`, `srcset`, `sizes`, `width`, `height`, and `loading` attributes instead of putting the data into `data-` attributes, like the popular javascript libraries lazysizes and vanilla-lazyload do. There should be no need to use `<noscript>` elements either.

Due to a bug in chrome, the first browser to support native lazyloading, images that have not yet been loaded will be missing in the printed page.

Both javascript libraries mentioned above, require either invalid markup without any `src` attribute at all, or an empty or low quality placeholder (LQIP), while the `src` data is put into `data-src` instead, and `srcset` data put into `data-srcset`, all of which only works with javascript. Is this considered an acceptable or even best practice in 2020, and does this neither harm the site accessibility, cross-device compatibility, nor search engine optimization?

I had been "looking for a proper way to implement lazy loading of images without harming printability and accessibility, and without introducing layout shift (content jump), preferably using native `loading=lazy` and a fallback for older browsers".

An elegant solution should be based on valid and complete HTML markup, thus using `<img src`, `srcset`, `sizes`, `width`, `height`, and `loading` attributes instead of using `data-` attributes, like the popular JavaScript libraries lazysizes and vanilla-lazyload do.

My question got two extensive answers (thanks Graham from INHU for taking the effort) and no official statement from Chromium developers so far.

But Chrome/Chromium developer Addy Osmani replied to my tweet by providing a link to Houssein Djirdeh's solution (or rather workaround):

> it implements a custom print button that iterates through all images and sets `loading=eager` prior to calling `print()`.
> Otherwise waiting on issues 2532: Asynchronous print events to load deferred images

Addy Osmani
@addyosmani

Replying to @fraktalisman

Here's one way to tackle printing with loading=lazy courtesy of @hdjirdeh: lazy-load-with-print-ctl1l4wu1.now.sh - it implements a custom print button that iterates through all images and sets loading=eager prior to calling print(). Otherwise waiting on github.com/whatwg/html/is... (cc @zcorpan)

Conclusion

As both Wikimedia's WHATWG issue and the Chrome bug are still open

Follow and join the discussion using the links provided in this article and try to make the proper decision on how to use native lazy loading in your next project.

Discussion (5)

InHuOfficial • Dec 16 '21 • Edited on Dec 16

I started reading this and thought "I have a solution for that", I will drop a comment with a link to it.

Then half way down I saw you had linked to the question I was going to link to 🤣 (and I only just realised it was your question I answered lol)!

thanks for the shout out ❤️

Sadly I never got a true 100% solve as you can't intercept or listen for printing via the browser menu 😢

Great article as always, have the usual ❤️🦄 from me!

P.S. stop telling people my real name, I like an air of mystery on DEV 🤣 (joking, it's fine!)

Ingo Steinke • Dec 16 '21

Thanks for your approval! Also "as always", I have updated my post several times after the initial release. While the content is mostly recycling of notes and bookmarks from the past months, I'm still waiting for an elegant solutions, while most most other devs I talked to didn't even see it as an issue.

InHuOfficial • Dec 16 '21

I don’t print anymore but I do sometime print to PDF so this bother me a little which is why I found your question so interesting!

I think the reason most Devs don’t see it as a problem is what I like to call “developer blinkers” where we assume everyone is tech savvy because “it’s easy” to us.

I now want to resurrect that idea as I had forgotten about it! I also want to write an article on “development blinkers” now so thanks for the double inspiration!

Ingo Steinke • Dec 16 '21

The German word "Internetausdrucker", literally a person who prints out the internet, has been used to mock stick-in-the-mud users, and especially politicians and civil servants for their sluggish adaption of the digital lifestyle. Many people (including myself) also used to print their "online" train tickets, just in case their phone battery might die during the ride (not that unlikely given the neglected state of public transport and missing power sockets in the trains). But HP will eventually stop me printing anything anyway by not releasing device drivers for current Linux and MacOS anymore.

Looking forward to your take on our blinkers!

InHuOfficial • Dec 16 '21

both of my examples no longer work, 😢 defo going to revisit this as I was close to something quite special looking back at the concept!

Thanks for the nudge, I guess I know what I am doing in my spare time between Christmas and New Year now! 🤣

Ingo Steinke

Web Development, Sustainability, Art and Music, Nature and Travel, Sustainability

LOCATION
Germany

WORK
Creative Web Developer at Ingo Steinke

JOINED
21 Sep 2019

More from Ingo Steinke

Animated Gradient Text Color
#webdev #showdev #css #tutorial

Aspect ratio: no need for container units!
#css #html #webdev #todayilearned

CSS :has(.parent-selectors)
#css #webdev #todayilearned #javascript