A modern solution to lazy loading images using Intersection Observer

Vamsi Vempati Follow
May 8, 2018 · 6 min read

Photo by Deva Darshan on Unsplash

Fan of Angular-In-Depth? Support us on Twitter!

Performance of a web application has become a key factor in deciding conversion rates for e-commerce websites. The faster a page loads, the better the conversion rate. According to the recent mobile page speed benchmarks released by *Google*, the bounce probability increases as page load time increases. The bounce probability increase varies from 32% for a 1 to 3 second load time to 123% for a 1 to 10 second load time.

Out of the things that increase the page load time, images can be a major issue, especially if there are quite a few of them on the page. When possible, it is best to lazy load the images on a page and only display them when the user gets to them, so it does not increase the initial page load time. This also has the benefit of reducing data usage, especially for mobile, where a lot of users can have poor connectivity and limited data plans.

Current solution to lazy loading

In order to decide whether or not to load an image, we need to check if that image is visible (in viewport) and if it is, we load the image. To check if the image is in viewport, we can use events and event handlers to detect the scroll position, offset value, element height and viewport height and calculate whether an image is in the viewport or not, we can then display the image if it is in the viewport.

But, this has a few drawbacks:

- This can cause performance issues as the calculations will be run on the main thread
- The calculation is performed each time there is a scroll on the page which is bad if the image is well below the view port
- The calculation can be very expensive if there are a lot of images on the page

A modern solution

I recently came across a relatively new DOM API—Intersection Observer API which provides a way to detect when an element intersects with the viewport. This lets us register a callback function that is executed when the element of interest enters or exits the viewport. Therefore, we do not have to do anything on the main thread which makes the page faster.

In addition to detecting whether an element is in the viewport or not, the **Intersection Observer API** also can detect the percentage of visibility of the element in relation to the viewport. This can be specified as **threshold** in the options when creating the intersection observer. Threshold can take values from 0 to 1, with the default being 0 which means that the callback is executed when the first pixel of the element is visible. A value of 1 means that the callback is only executed only when all the pixels are visible.

Threshold can also take an array of values ranging from 0 to 1—[0, 0.5, 1], in which case the callback is executed at each threshold value. Refer to this codepen from the docs for a demo of *Intersection Observer* with various thresholds.

Lazy loading using *Intersection Observer* comprises the following steps:

- Create a new intersection observer
- Watch the element we wish to lazy load for visibility changes
- When the element is in viewport, load the element
- Once the element is loaded, stop watching it for visibility changes

And with Angular, we can wrap all this functionality into a custom directive.

Wrapping it all in an Angular directive

Since, we are changing the behavior of the DOM elements, we can make an Angular directive that can be used on the elements which we want to lazy load.

This directive will have an output event *deferLoad* which the component can use to perform what needs to be done when the element is in viewport—in this case, to display the element.

```
1  import {ElementRef, EventEmitter, Output} from '@angul
2
3  @Directive({
4      selector: '[deferLoad]'
5  })
6  export class DeferLoadDirective {
7      @Output() public deferLoad: EventEmitter<any> = ne
8
9      private _intersectionObserver? : IntersectionObser
10
```

Create a new intersection observer and watch the element

Once the component's view has been fully initialized, we create a new intersection observer.

This takes in two parameters:

i. A callback function which is executed when the percentage of the visible target element crosses a threshold.

ii. An optional object to customize the observer where we can specify the desired threshold value(s) at which the callback function should be executed. If we do not specify any options, the default threshold value is used—0.

```
public ngAfterViewInit () {
    this._intersectionObserver = new IntersectionObserv
        this.checkForIntersection(entries);
    }, {});
    this._intersectionObserver.observe(<Element>(this._
```

For more options you can customize the observer, please refer to the docs.

Check, load, and unobserve

The callback function *checkForIntersection* would have logic to check if the element is intersecting. If the element is intersecting, we emit the output event *deferLoad*, stop watching the element and disconnect the intersection observer.

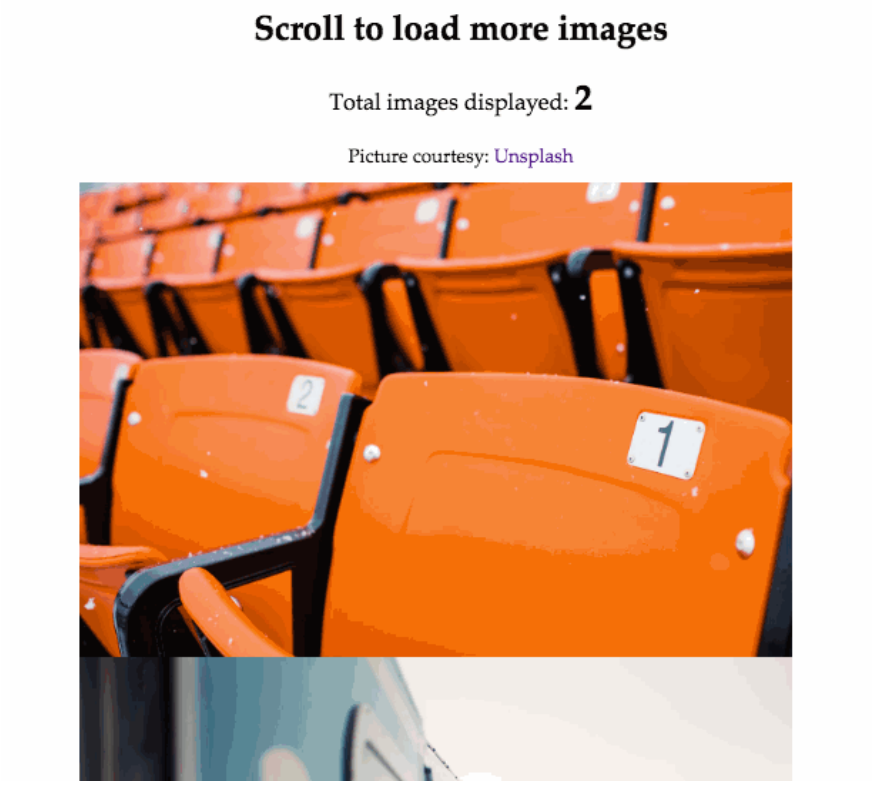
```
private checkForIntersection = (entries: Array<Interse
    entries.forEach((entry: IntersectionObserverEntry)
        if (this.checkIfIntersecting(entry)) {
            this.deferLoad.emit();
            this._intersectionObserver.unobserve(<Elem
            this._intersectionObserver.disconnect();
        }
    });
}
```

Usage

Now you can add *DeferLoadModule* to your list of imports in the module corresponding to your component and use it with the element that you wish to lazy load like below:

```
<div
    (deferLoad)="showMyElement=true">
    <my-element
       *ngIf=showMyElement>
      ...
```

Show time



Demo of lazy loading using Intersection Observer API in Angular

Other uses

Intersection Observer can be used for various things other than lazy loading such as:

- infinite scrolling where you see more content as you scroll
- ad visibility reporting—to report when a percentage or a whole of an ad is visible
- performing tasks such as animations only when the user sees the result.

Browser support

Intersection Observer is a fairly new API and the browser support is mostly for recent versions of the browsers. Also, it is not supported at all on Internet Explorer. For unsupported browsers, we can use a polyfill to lazy load the elements.

Desktop | Mobile

Feature	Chrome	Edge	Firefox (Gecko)	Internet Explorer	Opera	Safari (WebKit)
Basic support	51	15	55 (55)[1][2]	No support	38	WebKit bug 159475

Desktop compatibility from Mozilla docs

Desktop | Mobile

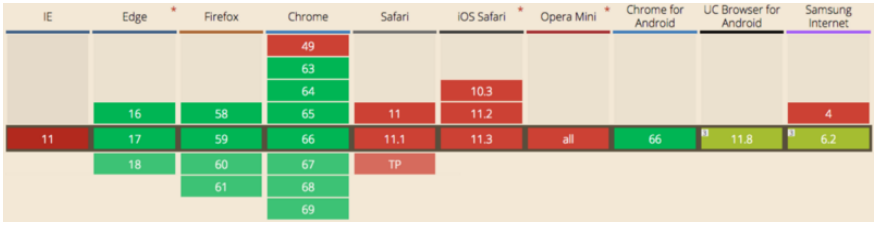
Feature	Chrome	Edge	Firefox (Gecko)	Internet Explorer	Opera	Safari (WebKit)
Basic support	51	15	55 (55)[1][2]	No support	38	WebKit bug 159475

Mobile compatibility from Mozilla docs

[1] This feature has been implemented since Gecko 53.0 (Firefox 53.0 / Thunderbird 53.0 / SeaMonkey 2.50) behind the preference `dom.IntersectionObserver.enabled`, which was `false` by default. Enabled by default beginning in Firefox 55. See bug 1243846.

[2] Firefox doesn't currently take the `clip-path` of ancestor elements into account when computing the visibility of an element within its root. See bug 1319140 for the status of this issue.

Foot notes for browser support



Can I use Intersection Observer ?

Polyfill

An official W3C polyfill is available to support *Intersection Observer* for unsupported browsers. But, if you do not want to increase your app bundle size, you can use your own event handlers to check if an element is in viewport with a few lines of code. However, this feature is implemented partially in various browsers and you might want to handle certain browsers individually. Therefore, using an official polyfill could be more beneficial, if you are willing to trade off the bundle size, which is not a lot—6.8 kb minified and 2.3 kb minified and gzipped.

Even better

I have created an npm package with the *deferLoad* directive that can be used in Angular applications with very simple setup. In the

interest of reducing package size, it falls back to a scroll detection mechanism for unsupported browsers instead of using the polyfill.

Usage instructions are very similar to how it is shown in the **Usage** section of this article and it is available from the Github repo.

You can see a demo of this package in play here.

Intersection Observer

The web's traditional position calculation mechanisms rely on explicit queries of DO...

www.w3.org

Intersection Observer API

The Intersection Observer API provides a way to asynchronously observe changes i...

developer.mozilla.org

. . .

Thank you for reading! If you enjoyed this article, please feel free to 👏 *and help others find it. Please do not hesitate to share your thoughts in the comments section below. Follow me on Medium or Twitter for more articles. Happy coding folks!!* 💻 ☕

. . .