**JavaScript: defer and async –**

[Stephanie De Smedt](https://medium.com/@simply_stef?source=post_page-----9e30ad2647a9--------------------------------) May 11, 2022

When developing an interactive website or web application the users browser needs to download any scripts or stylesheets as part of the loading process. Whilst HTML and CSS elements are usually lightweight and quick to load, a complex JavaScript file can slow this process down.

By default all content will load sequentially… Placing a <script> tag at the beginning of an HTML file means that any content below it will not load until the script has been processed.

A workaround could be to place the <srcipt> at the end of the HTML. The downside of this is that the browser will not start processing any content until after the rest of the page is rendered which will slow down total load time.

Luckily there are a couple of built in attributes to help deal with this in a better way.

**Defer**

The defer attribute tells the browser not to wait for the script to finish loading. Instead it indicates that the rest of the content should be built whilst it is processed in the background. Once the DOM is fully built the deferred script runs.

In the case where two different scripts are included and both are set to defer they retain their relative order. For example, consider the case where we have a file with a large script and a subsequent smaller script like in the snippet below.

<script defer src="./very-large-file.js"></script>  
<script defer src="./smaller-file.js"></script>

The browser will start loading both of scripts in the background whilst rendering any remaining content. Once the rest of the content is loaded the very-large-file will be run first followed by the smaller-file. If the very-large-file is still loading both scripts will wait until it is ready to run, ensuring they are run in order.

Whilst that can sound negative, it comes in useful for cases where we might need to load a JavaScript library in combination with a script that depends on it.

**Async**

As with defer the async attribute instructs the browser to carry on parsing content whilst it loads in the background. However, one key difference is that as soon as the script has finished loading it will run.

If there are multiple async scripts on a page they will run in whatever order they finish loading and will not wait for any content… In other words, the DOM and other scripts won’t wait for them — and they won’t wait for anything.

This means running a script with the async attribute is useful when loading high priority content, since you will guarantee that it loads as soon as possible. For example, any content that interacts with the styling/rendering of a website.

If the script has dependencies on anything else async should be avoided — or caution should be used at the very least! Different processors or browsers can result in different execution times so there is no guarantee the order in which it will be run in relation to other content.

The nature of async means that if it finishes loading before the DOM load is finished that render process will be paused in order to run the script — another reason why async need to be reserved for only “high priority” content.

**Can you combine “async defer”**

Technically it is possible to combine async and defer within the same <script> tag — however I would however advise to avoid it!

Combining async and defer means that their properties can compete against each other leading to unpredictable behaviour. Both attributes are best used with a clear purpose in mind and combining them can confuse what the outcome will be. For example, if one browser type loads a script at a slower rate than another the site will run differently and can cause challenges in troubleshooting why.

**Summary**

In general, defer is ideal for JavaScript resources that aren't critical to the initial construction of the page. This should be considered particularly in connection with mobile devices which have slower processing times as frontend HTML/CSS content will appear first, whilst background JavaScript is executed later at a more appropriate point.

On the flip side async should be reserved for high priority JavaScripts that should run as early as possible in the loading process, for example those that impact the rendering of a site. Since the files will be run in no particular order, it is important to be sure that the file truly is independent so as not to cause unpredictable errors.