SNS COLLEGE OF ENGINEERING

An Autonomous Institution

Coimbatore-107

19TS601-FULL STACK DEVELOPMENT

UNIT-1

JAVASCRIPT AND BASICS OF MERN STACK

Document and resource loading

Document and resource loading

Page:DOMContentLoaded, load, beforeunload, unload

- Browsers sometimes suspend pages or abort them entirely, in case the system resources are limited.
- There are modern lifecycle hooks that help to handle such interventions without affecting the user experience.

- There are three significant events in the framework of an HTML page lifecycle:
  - DOMContentLoaded: the event when the browser completely loaded HTML, the DOM tree is set up, but external resources, such as pictures <img> and stylesheets might not be loaded yet.
  - load: the event when not only HTML is loaded, but other external resources (for instance, images, stylesheets and so on), too.
  - beforeunload: the event when the user is leaving the page.

- Each of the events above may be useful on a specific purpose:
  - DOMContentLoaded: when the DOM is ready, the handler is capable of looking up DOM nodes and initializing the interface.
  - load: the external resources are loaded, hence the styles are applied, the sizes of the images are known and so on.
  - beforeunload: the user is getting out, and it's not possible to check if the latter has saved the changes, and ask to make sure the user really wants to leave or not.
  - unload: the user has almost left, yet several operations can not be initiated (for example, sending out statistics).

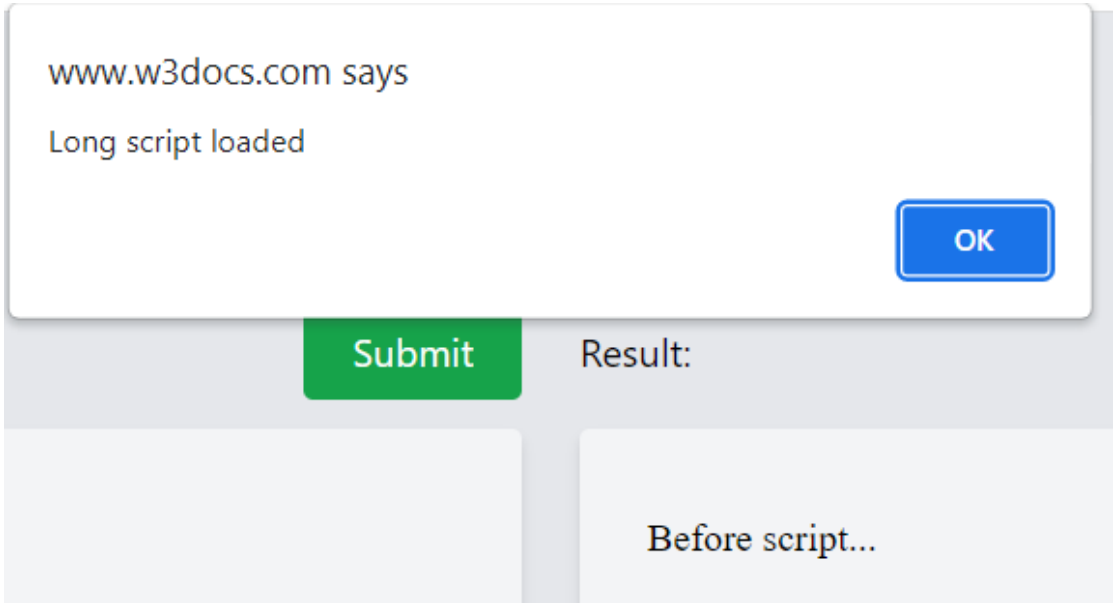
- Refer examples in the below link
- https://www.w3docs.com/learn-javascript/page-domcontentloaded-load-beforeunload-unload.html

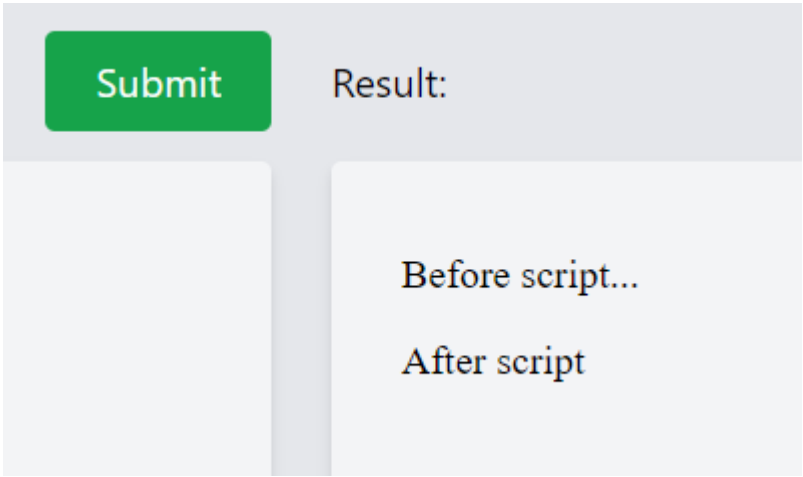
Scripts:async, defer

- The scripts are heavier in modern browsers than in HTML: the download size is larger, and it takes more time to process.
- Once the browser loads HTML meets a <script>...</script> tag, it won't be able to continue setting up the DOM.
- The script should be executed by it right now. The same scenario works for the external scripts. In other words, the browser must hold on until the script loads, execute it and process the rest of the page only after.

- It leads to the following two significant problems:
  - The scripts are not capable of seeing the DOM elements below them. Hence, they can’t add handlers.
  - In case there exists a massive script at the the page top, “the page will be blocked”. The page content won't be seen by the users until it downloads and runs, like this:

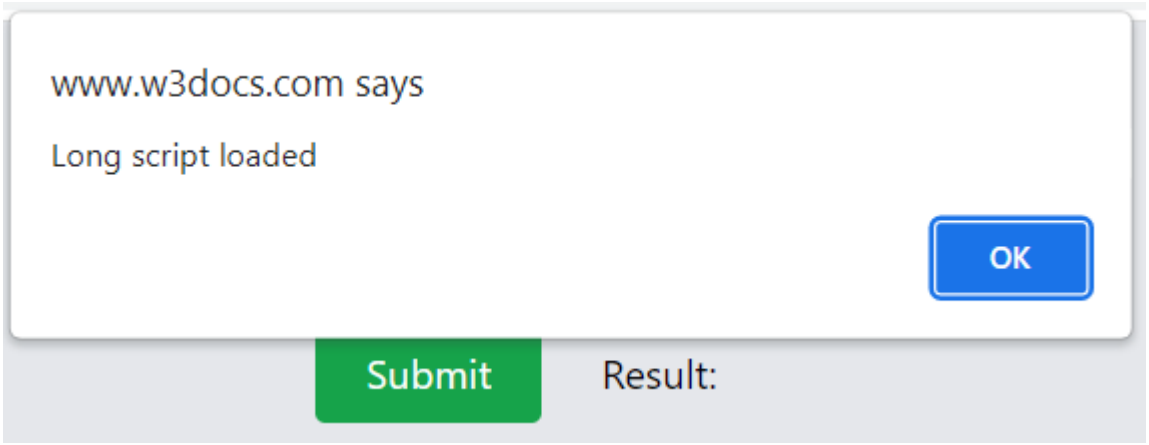
```
<html>
 <head>
  <title>Title of the Document</title>
 </head>
 <body>
  <p> Before script...</p>
  <script src="https://www.w3docs.com/js/long.js?speed=1"></script>
  <!-- This is not visible until the script loads. -->
  <p> After script</p>
 </body>
</html>
```

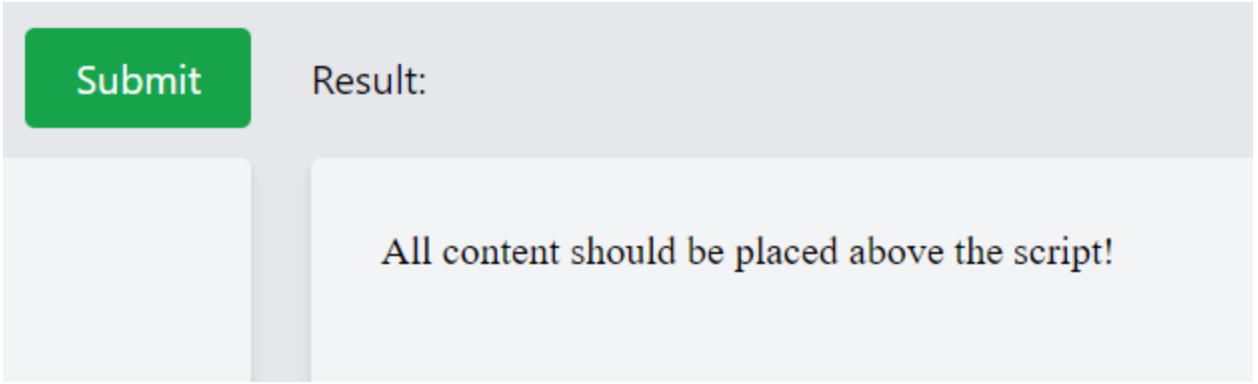

www.w3docs.com says
Long script loaded
OK
Submit
Result:
Before script...


Submit
Result:
Before script...
After script

- It is also possible to place a script at the bottom of the page. In that case, it will see the elements above and the page content won't be blocked by it, like this:

```
<html>
 <head>
  <title>Title of the Document</title>
 </head>
 <body>
  All content should be placed above the script!
  <script src="https://www.w3docs.com/js/long.js?speed=1"></script>
 </body>
</html>
```


www.w3docs.com says
Long script loaded
OK
Submit
Result:


Submit
Result:
All content should be placed above the script!

- However, this is not a perfect solution. For instance, if the script is seen by the browser, only after the entire HTML document is downloaded. Note that for large HTML documents, it can be a noticeable delay. Things like this can be invisible to people who use faster connections. But, consider that many people still have a slow internet connection. Fortunately, there exist two <script> attributes that can solve the possible issues. These attributes are called defer and async.

Defer

- Defer informs the browser that it should continue working with the page, load the script “in the background” and then run the script when it loads.

Async

- Async signifies that the script is completely independent:
- The async scripts are not waited for by the page: the content is processed and presented.
- DOMContentLoaded and the async scripts don't have to wait for one another. (the DOMContentLoaded event might occur before and after an async script, in case the latter finishes loading after the page is complete. Either it can happen after an async script, in case the latter is short or was located in the HTML-cache).
- Other scripts never wait for async scripts and vice versa.

- So, in case of having multiple scripts, they might execute in any sequence.

Dynamic scripts

- The script begins to load as soon as it's appended to the document (*).
- Dynamic scripts act like "async" by default.
- In other words, they don't wait for anything, and vice versa.
- The script, which loads first-runs first, as well.

Resource loading: onload and onerror

- Resources such as scripts, styles, images provide load and error events for tracking their loading:
  - The load event happens on a successful load.
  - The error event occurs on a failed load.
- There is an exception: <iframe>, which only triggers the load, for each load completion, even if the page is not found. For resources, you can also use the readystatechange event. But it is used rarely as the load and error events are simpler and handier.

Thank You