Where is the best place to put <script> tags in HTML markup?

When embedding JavaScript in an HTML document, where is the best place to put the `<script>` tags and included JavaScript? I seem to recall that you are not supposed to place these in the `<head>` section, but placing at the beginning of the `<body>` section is bad, too, since the JavaScript will have to be parsed before the page is rendered completely (or something like that). This seems to leave the *end* of the `<body>` section as a logical place for `<script>` tags.

So, where *is* the best place to put the `<script>` tags?

(This question references this question, in which it was suggested that JavaScript function calls should be moved from `<a>` tags to `<script>` tags. I'm specifically using JQuery, but more general answers are also appropriate.)

javascript jquery html

asked Jan 12 '09 at 18:15

mipadi
198k ● 44 ● 366 ● 390

Possible duplicate: stackoverflow.com/questions/143486/… – Mirko Cianfarani Nov 11 '14 at 10:23

13 Answers

Here's what happens when a browser loads a website:

1. Fetch the HTML page (e.g. index.html)
2. Begin parsing the HTML
3. The parser encounters a `<script>` tag referencing an external script file.
4. The browser requests the script file. Meanwhile, the parser blocks and stops parsing the other HTML on your page.
5. After some time the script is downloaded and subsequently executed.
6. The parser continues parsing the rest of the HTML document.

Step 4 causes a bad user experience. Your website basically stops loading until you've downloaded all scripts. If there's one thing that users hate it's waiting for a website to load.

Why does this even happen?

Any script can insert its own HTML via `document.write()` or other DOM manipulations. This implies that the parser has to wait until the script has been downloaded & executed before it can safely parse the rest of the document. After all, the script *could* have inserted its own HTML in the document.

However, most javascript developers no longer manipulate the DOM *while* the document is loading. Instead, they wait until the document has been loaded before modifying it. For example:

```
<!-- index.html -->
<html>
    <head>
        <title>My Page</title>
        <script type="text/javascript" src="my-script.js"></script>
    </head>
    <body>
        <div id="user-greeting">Welcome back, user</div>
    </body>
</html>
```

Javascript:

```
// my-script.js
document.addEventListener("DOMContentLoaded", function() {
    // this function runs when the DOM is ready, i.e. when the document has been parsed
    document.getElementById("user-greeting").textContent = "Welcome back, Bart";
});
```

Because your browser does not know my-script.js isn't going to modify the document until it has been downloaded & executed, the parser stops parsing.

Antiquated recommendation

The old approach to solve this problem was to put `<script>` tags at the bottom of your `<body>`, because this ensures the parser isn't blocked until the very end.

This approach has its own problem: the browser cannot start downloading the scripts until the entire document is parsed. For larger websites with large scripts & stylesheets, being able to

download the script as soon as possible is very important for performance. If your website doesn't load within 2 seconds, people will go to another website.

In an optimal solution, the browser would start downloading your scripts as soon as possible, while at the same time parsing the rest of your document.

The modern approach

Today, browsers support the `async` and `defer` attributes on scripts. These attributes tell the browser it's safe to continue parsing while the scripts are being downloaded.

async

```
<script type="text/javascript" src="path/to/script1.js" async></script>
<script type="text/javascript" src="path/to/script2.js" async></script>
```

Scripts with the async attribute are executed asynchronously. This means the script is executed as soon as it's downloaded, without blocking the browser in the meantime.
This implies that it's possible script 2 is downloaded & executed before script 1.

According to http://caniuse.com/#feat=script-async, 90% of all browsers support this.

defer

```
<script type="text/javascript" src="path/to/script1.js" defer></script>
<script type="text/javascript" src="path/to/script2.js" defer></script>
```

Scripts with the defer attribute are executed in order (i.e. first script 1, then script 2). This also does not block the browser.

Unlike async scripts, defer scripts are only executed after the entire document has been loaded.

According to http://caniuse.com/#feat=script-defer, 90% of all browsers support this. 92% support it at least partially.

An important note on browser compatibility: in some circumstances IE <= 9 may execute deferred scripts out of order. If you need to support those browsers, please read this first!

Conclusion

The current state-of-the-art is to put scripts in the `<head>` tag and use the `async` or `defer` attributes. This allows your scripts to be downloaded asap without blocking your browser.

The good thing is that your website should still load correctly on the 20% of browsers that do not support these attributes, while speeding up the other 80%.

edited Feb 4 at 22:02

Onato
2,768 ● 17 ● 25

answered Jun 5 '14 at 21:20

Bart
9,473 ● 1 ● 10 ● 16

7 I'm surprised nobody cited Google's explanation... developers.google.com/speed/docs/insights/BlockingJS – Casey Falk Aug 15 '14 at 21:27

5 I'm not clear on what touches the DOM and what doesn't. Can you clarify? Is it safe to do an async load on something like jquery.js? – Doug Sep 6 '14 at 23:11

5 @Doug For example `document.write` operates on the dom. The question isn't *if* a script manipulates the dom, but *when* it does. As long as all dom manipulation happens after `domready` event has triggered, you're ok. jQuery is a library, and as such doesn't - or shouldn't - manipulate the dom by itself. – Bart Sep 9 '14 at 15:34

8 This answer is misleading. Modern browsers don't stop parsing when they reach a synchronous script tag that may affect the HTML, they just stop rendering/executing, and continue parsing optimistically to start downloading other resources that will probably be requested subsequently if no HTML is affected. – Fabio Beltramini Oct 16 '14 at 21:23

1 may I've reference for "If your website doesn't load within 2 seconds, people will go to another website." – KNU Oct 30 '14 at 7:24

Just before the closing body tag, as stated on

http://developer.yahoo.com/performance/rules.html#js_bottom

> Put Scripts at the Bottom
>
> The problem caused by scripts is that they block parallel downloads. The HTTP/1.1 specification suggests that browsers download no more than two components in parallel per hostname. If you serve your images from multiple hostnames, you can get more than two downloads to occur in parallel. While a script is downloading, however, the browser won't start any other downloads, even on different hostnames.

edited Apr 2 '14 at 13:58

answered Jan 12 '09 at 18:18

 vaxquis
5,270 ● 4 ● 23 ● 39

 Cammel
1,967 ● 2 ● 10 ● 5

5 Agreed with the concept and its explanation. But what happens if the user starts playing with the page. Suppose I've an AJAX dropdown which will start loading after the page has appeared to the user but while it is loading, the user clicks it! And what if a 'really impatient' user submits the form? – Hemant Tank Jan 3 '12 at 14:29

6 @Hermant Old comment but you may do the trick disabling the fields by default then enabling them using JS when the DOM is fully loaded. That's what Facebook seems to be doing nowadays. – novato Nov 11 '12 at 6:20

2 Just tested this with chrome, to check if this is still the same. It Is. You can check the differences in page load time of your browsers here. stevesouders.com/cuzillion – cypher Jan 8 '13 at 4:19

22 If this is best practice, why does stack overflow include all their script tags in <head>? :-P – Philip Apr 20 '13 at 5:04

9 In some cases, especially in ajax heavy sites, loading in head can actually result in faster load times. See: encosia.com/dont-let-jquerys-document-ready-slow-you-down (note that the "live()" function is deprecated in jquery, but the article still applies with the "on()" or "delegate" function). Loading in <head> may also be needed to guarantee correct behavior as pointed out by @Hermant. Finally, modernizr.com/docs recommends placing its scripts in the <head> for reasons explained on its site. – Nathan Jun 7 '13 at 15:54

The standard advice, promoted by the Yahoo! Exceptional Performance team, is to put the `<script>` tags at the end of the document body so they don't block rendering of the page.

But there are some newer approaches that offer better performance, I'm duplicating another answer here:

There are some great slides by Steve Souders (client-side performance expert) about:

- Different techniques to load external JavaScript files in parallel
- their effect on loading time and page rendering
- what kind of "in progress" indicators the browser displays (e.g. 'loading' in the status bar, hourglass mouse cursor).

answered Jan 12 '09 at 23:37

 orip
36.3k ● 16 ● 84 ● 121

Non-blocking script tags can be placed just about anywhere:

```
<script src="script.js" async></script>
<script src="script.js" defer></script>
<script src="script.js" async defer></script>
```

- `async` script will be executed asynchronously as soon as it is available
- `defer` script is executed when the document has finished parsing
- `async defer` script falls back to the defer behavior if async is not supported

Such scripts will be executed asynchronously/after document ready, which means you cannot do this:

```
<script src="jquery.js" async></script>
<script>jQuery(something);</script>
<!--
  * might throw "jQuery is not defined" error
  * defer will not work either
-->
```

Or this:

```
<script src="document.write(something).js" async></script>
<!--
  * might issue "cannot write into document from an asynchronous script" warning
  * defer will not work either
-->
```

Or this:

```
<script src="jquery.js" async></script>
<script src="jQuery(something).js" async></script>
<!--
  * might throw "jQuery is not defined" error (no guarantee which script runs first)
  * defer will work in sane browsers
-->
```

Or this:

```
<script src="document.getElementById(header).js" async></script>
<div id="header"></div>
<!--
  * might not locate #header (script could fire before parser looks at the next line)
  * defer will work in sane browsers
-->
```

Having said that, asynchronous scripts offer these advantages:

- **Parallel download of resources**:
  Browser can download stylesheets, images and other scripts in parallel without waiting for a script to download and execute.
- **Source order independence**:
  You can place the scripts inside head or body without worrying about blocking (useful if you are using a CMS). Execution order still matters though.

It is possible to circumvent the execution order issues by using external scripts that support callbacks. Many third party JavaScript APIs now support non-blocking execution. Here is an example of loading the Google Maps API asynchronously.

edited Feb 6 '15 at 13:37

answered Feb 6 '15 at 11:19

Salman A

113k ● 39 ● 244 ● 329

This is the correct answer for today - using this approach means it's easier to keep your widgets self contained, no need to do fancy `<head>` include logic. – Daniel Sokolowski Jul 7 '15 at 18:17

1 I'm confused why you can't use `async` or `defer` when including jQuery as you specify in your second block: `<script src="jquery.js" async></script>`. Are you able to explain why? I thought I need to have the async tag in for performance—per the accepted answer—so my page can load even while jQuery is still loading]. Thanks! – elbowlobstercowstand Aug 6 '15 at 8:10

1 @elbow 99% of times `<script src=jquery.js>` is followed by `$(function(){ ... })` blocks somewhere in the page. Asynchronous loading does not guarantee that jQuery will be loaded at the time browser tries to parse those blocks hence it will raise *$ is not defined* error (you may not get the error if jQuery was loaded from cache). I answered a question about loading jQuery asynchronously and preserve `$(function(){ ... })`. I'll see if I could find it, or you can look at this question: stackoverflow.com/q/14811471/87015 – Salman A Aug 6 '15 at 9:12

@SalmanA Thank you! Yes, I fall in that 99%. I first need `jquery` lib to load, *then* my remaining `.js` scripts. When I declare `async` or `defer` on the `jquery` lib script tag, my `.js` scripts don't work. I thought `$(function(){ ... })` protected that—guess not. **Current solution:** I don't add `defer` or `async` on `jquery` lib script, but I do add `async` on my follow up `.js` scripts. Note: the reason I'm doing *any* of this is to make Google Page Speed happy. Thx again for the help! Any other advice is welcome. (Or a link to your previous answer). :) – elbowlobstercowstand Aug 6 '15 at 21:22

@elbow See stackoverflow.com/a/21013975/87015, it will only give you an idea but not the complete solution. You could instead search for "jquery async loader libraries". – Salman A Aug 7 '15 at 5:40

If you are using JQuery then put the javascript wherever you find it best and use `$(document).ready()` to ensure that things are loaded properly before executing any functions.

On a side note: I like all my script tags in the `<head>` section as that seems to be the cleanest place.

edited Apr 2 '14 at 14:05

vaxquis

5,270 ● 4 ● 23 ● 39

answered Jan 12 '09 at 18:18

Andrew Hare

204k ● 38 ● 484 ● 544

10 in the head...er? `<header>`? – Northborn Design Jan 20 '13 at 17:43

3 Note that using `$(document).ready()` doesn't mean you can put your JavaScript *anywhere* you like – you still have to put it after the `<script src=".../jquery.min.js">` where you include jQuery, so that `$` exists. – Rory O'Kane Jul 10 '13 at 15:42

2 It is not optimal to put script tags in the <head> section - this will delay display of the visible part of the page until the scripts are loaded. – CyberMonk Oct 14 '13 at 0:33

~~XHTML Won't Validate if the script is anywhere other than within the head element.~~ turns out it can be everywhere.

You can defer the execution with something like jQuery so it doesn't matter where it's placed (except for a small performance hit during parsing).

edited Jan 12 '09 at 19:05

answered Jan 12 '09 at 18:22

Allain Lalonde

38.5k ● 53 ● 147 ● 203

1 XHTML will validate with script tags in the body, both strict and transitional. Style tags however may only be in the head. – I.devries Jan 12 '09 at 18:43

The conventional (and widely accepted) answer is "at the bottom", because then the entire DOM will have been loaded before anything can start executing.

There are dissenters, for various reasons, starting with the available practice to intentionally begin execution with a page onload event.

answered Jan 12 '09 at 18:18

dkretz
29.7k • 11 • 62 • 120

Depending on the script and its usage the best possible (in terms of page load and rendering time) may be to not use a conventional <script>-tag per se, but to dynamically trigger the loading of the script asynchronously.

There are some different techniques, but the most straight forward is to use document.createElement("script") when the window.onload event is triggered. Then the script is loaded first when the page itself has rendered, thus not impacting the time the user has to wait for the page to appear.

This naturally requires that the script itself is not needed for the rendering of the page.

For more information, see the post Coupling async scripts by Steve Souders (creator of YSlow but now at Google).

answered Jan 12 '09 at 21:06

stpe
1,824 • 2 • 17 • 27

Depends, if you are loading a script that's necessary to style your page / using actions in your page (like click of a button) then you better place it on the top. If your styling is 100% CSS and you have all fallback options for the button actions then you can place it in the bottom.

Or best thing (if that's not a concern) is you can make a modal loading box, place your javascript at the bottom of your page and make it disappear when the last line of your script gets loaded. This way you can avoid users using actions in your page before the scripts are loaded. And also avoid the improper styling.

answered Nov 18 '13 at 4:41

ahmedmzl
124 • 1 • 2 • 10

Script blocks DOM load untill it's loaded and executed.

If you place scripts at the end of `<body>` all of DOM has chance to load and render (page will "display" faster). `<script>` will have access to all of those DOM elements.

In other hand placing it after `<body>` start or above will execute script (where there's still no DOM elements).

You are including jQuery which means you can place it wherever you wish and use .ready()

answered Jun 29 '15 at 12:38

Szymon Toda
633 • 5 • 16 • 39

- If you still care a lot about support and performance in IE<10, it's best to ALWAYS make your script tags the last tags of your HTML body. That way, you're certain that the rest of the DOM has been loaded and you won't block and rendering.
- If you don't care too much anymore about IE<10, you might want to put your scripts in the head of your document and use `defer` to ensure they only run after your DOM has been loaded (`<script type="text/javascript" src="path/to/script1.js" defer></script>`). If you still want your code to work in IE<10, don't forget to wrap your code in a `window.onload` even, though!

edited Jan 20 at 20:40

answered Jan 20 at 19:50

user5803163

In the accepted answer, this is refered to as the “antiquated recommendation”. If you still mean it, you probably should produce some reference to back it. – dakab Jan 20 at 20:10

@dakab : Thanks for the comment. I modified my answer! – user5803163 Jan 20 at 20:40

It makes more sense to me to include the script after the HTML. Because most of the time I need the Dom to load before I execute my script. I could put it in the head tag but I don't like all the Document loading listener overhead. I want my code to be short and sweet and easy to read.

I've heard old versions of safari was quarky when adding your script outside of the head tag but I say who cares. I don't know anybody using that old crap do you.

Good question by the way.

answered Dec 29 '12 at 6:06

zachdyer
326 ● 4 ● 13

You can place where you want the scripts and one is not better than another practice.

the situation is as follows:

The page load linearly, "top-down", so if you put the script in the head ensures that it starts to load before everything, now, if you put it inside the body mixed with the code can cause page loads a unsightly manner.

identify good practice does not depend on where.

to support you, I will mention the following:

you can place:

and the page will load linearly

page is loaded asynchronously with other content

the content of the page will load before and after completion of loading the scripts are loaded

good practice here would be, when will implement each?

I hope I've been helpful, anything just answer me this issue.

answered Dec 27 '13 at 3:52

Juan Miguel
18 ● 2

protected by Josh Crozier Sep 4 '14 at 17:18

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site.

Would you like to answer one of these unanswered questions instead?