Vanilla JavaScript equivalent of jQuery's $.ready() - how to call a function when the page/DOM is ready for it [duplicate]

Asked 12 years, 9 months ago Modified 2 years, 5 months ago Viewed 2.3m times

1908

This question already has answers here:

$(document).ready equivalent without jQuery (39 answers)

Closed 8 years ago.

With jQuery, we all know the wonderful `.ready()` function:

```
$('document').ready(function(){});
```

However, let's say I want to run a function that is written in standard JavaScript with no library backing it, and that I want to launch a function as soon as the page is ready to handle it. What's the proper way to approach this?

I know I can do:

```
window.onload="myFunction()";
```

Or I can use the `body` tag:

```
<body onload="myFunction()">
```

Or I can even try at the bottom of the page after everything, but the end `body` or `html` tag like:

```
<script type="text/javascript">
    myFunction();
</script>
```

What is a cross-browser(old/new)-compliant method of issuing one or more functions in a manner like jQuery's `$.ready()`?

javascript jquery html

edited Jun 29, 2022 at 20:23
mruanova
7,075 • 6 • 40 • 57

asked Mar 27, 2012 at 23:57
chris
36.9k • 53 • 147 • 256

10 Answers

Sorted by: Highest score (default)

2649

The simplest thing to do in the absence of a framework that does all the cross-browser compatibility for you is to just put a call to your code at the end of the body. This is faster to execute than an `onload` handler because this waits only for the DOM to be ready, not for all images to load. And, this works in every browser.

```
<!doctype html>
<html>
<head>
</head>
<body>
Your HTML here

<script>
// self executing function here
(function() {
   // your page initialization code here
   // the DOM will be available here

})();
</script>
</body>
</html>
```

For modern browsers (anything from IE9 and newer and any version of Chrome, Firefox or Safari), if you want to be able to implement a jQuery like `$(document).ready()` method that you can call from anywhere (without worrying about where the calling script is positioned), you can just use something like this:

```
function docReady(fn) {
    // see if DOM is already available
    if (document.readyState === "complete" || document.readyState ===
"interactive") {
        // call on next available tick
        setTimeout(fn, 1);
    } else {
        document.addEventListener("DOMContentLoaded", fn);
    }
}
```

Usage:

```
docReady(function() {
    // DOM is loaded and ready for manipulation here
});
```

If you need full cross browser compatibility (including old versions of IE) and you don't want to wait for `window.onload`, then you probably should go look at how a framework like jQuery implements its `$(document).ready()` method. It's fairly involved depending upon the capabilities of the browser.

To give you a little idea what jQuery does (which will work wherever the script tag is placed).

If supported, it tries the standard:

```
document.addEventListener('DOMContentLoaded', fn, false);
```

with a fallback to:

```
window.addEventListener('load', fn, false )
```

or for older versions of IE, it uses:

```
document.attachEvent("onreadystatechange", fn);
```

with a fallback to:

```
window.attachEvent("onload", fn);
```

And, there are some work-arounds in the IE code path that I don't quite follow, but it looks like it has something to do with frames.

Here is a full substitute for jQuery's `.ready()` written in plain javascript:

```
(function(funcName, baseObj) {
    // The public function name defaults to window.docReady
    // but you can pass in your own object and own function name and those will
be used
    // if you want to put them in a different namespace
    funcName = funcName || "docReady";
    baseObj = baseObj || window;
    var readyList = [];
    var readyFired = false;
    var readyEventHandlersInstalled = false;
```

```
    // call this when the document is ready
    // this function protects itself against being called more than once
    function ready() {
        if (!readyFired) {
            // this must be set to true before we start calling callbacks
            readyFired = true;
            for (var i = 0; i < readyList.length; i++) {
                // if a callback here happens to add new ready handlers,
                // the docReady() function will see that it already fired
                // and will schedule the callback to run right after
                // this event loop finishes so all handlers will still execute
                // in order and no new ones will be added to the readyList
                // while we are processing the list
                readyList[i].fn.call(window, readyList[i].ctx);
            }
            // allow any closures held by these functions to free
            readyList = [];
        }
    }

    function readyStateChange() {
        if ( document.readyState === "complete" ) {
            ready();
        }
    }

    // This is the one public interface
    // docReady(fn, context);
    // the context argument is optional - if present, it will be passed
    // as an argument to the callback
    baseObj[funcName] = function(callback, context) {
        if (typeof callback !== "function") {
            throw new TypeError("callback for docReady(fn) must be a
function");
        }
        // if ready has already fired, then just schedule the callback
        // to fire asynchronously, but right away
        if (readyFired) {
            setTimeout(function() {callback(context);}, 1);
            return;
        } else {
            // add the function and context to the list
            readyList.push({fn: callback, ctx: context});
        }
        // if document already ready to go, schedule the ready function to run
        if (document.readyState === "complete") {
            setTimeout(ready, 1);
        } else if (!readyEventHandlersInstalled) {
            // otherwise if we don't have event handlers installed, install
them
            if (document.addEventListener) {
                // first choice is DOMContentLoaded event
                document.addEventListener("DOMContentLoaded", ready, false);
                // backup is window load event
                window.addEventListener("load", ready, false);
            } else {
                // must be IE
                document.attachEvent("onreadystatechange", readyStateChange);
                window.attachEvent("onload", ready);
            }
            readyEventHandlersInstalled = true;
        }
```

```
    }
})("docReady", window);
```

The latest version of the code is shared publicly on GitHub at https://github.com/jfriend00/docReady

Usage:

```
// pass a function reference
docReady(fn);

// use an anonymous function
docReady(function() {
    // code here
});

// pass a function reference and a context
// the context will be passed to the function as the first argument
docReady(fn, context);

// use an anonymous function with a context
docReady(function(context) {
    // code here that can use the context argument that was passed to docReady
}, ctx);
```

This has been tested in:

```
IE6 and up
Firefox 3.6 and up
Chrome 14 and up
Safari 5.1 and up
Opera 11.6 and up
Multiple iOS devices
Multiple Android devices
```

Working implementation and test bed: http://jsfiddle.net/jfriend00/YfD3C/

Here's a summary of how it works:

1. Create an IIFE (immediately invoked function expression) so we can have non-public state variables.
2. Declare a public function `docReady(fn, context)`
3. When `docReady(fn, context)` is called, check if the ready handler has already fired. If so, just schedule the newly added callback to fire right after this thread of JS finishes with `setTimeout(fn, 1)`.
4. If the ready handler has not already fired, then add this new callback to the list of callbacks to be called later.

5. Check if the document is already ready. If so, execute all ready handlers.
6. If we haven't installed event listeners yet to know when the document becomes ready, then install them now.
7. If `document.addEventListener` exists, then install event handlers using `.addEventListener()` for both `"DOMContentLoaded"` and `"load"` events. The "load" is a backup event for safety and should not be needed.
8. If `document.addEventListener` doesn't exist, then install event handlers using `.attachEvent()` for `"onreadystatechange"` and `"onload"` events.
9. In the `onreadystatechange` event, check to see if the `document.readyState === "complete"` and if so, call a function to fire all the ready handlers.
10. In all the other event handlers, call a function to fire all the ready handlers.
11. In the function to call all the ready handlers, check a state variable to see if we've already fired. If we have, do nothing. If we haven't yet been called, then loop through the array of ready functions and call each one in the order they were added. Set a flag to indicate these have all been called so they are never executed more than once.
12. Clear the function array so any closures they might be using can be freed.

Handlers registered with `docReady()` are guaranteed to be fired in the order they were registered.

If you call `docReady(fn)` after the document is already ready, the callback will be scheduled to execute as soon as the current thread of execution completes using `setTimeout(fn, 1)`. This allows the calling code to always assume they are async callbacks that will be called later, even if later is as soon as the current thread of JS finishes and it preserves calling order.

Share

Improve this answer

Follow

edited Sep 11, 2019 at 21:24

Jan Kyu Peblik

1,527 ● 14 ● 22

answered Mar 28, 2012 at 0:46

jfriend00

706k ● 103 ● 1k ● 1k

1 Why setTimeout(fn, 1) is used other than setTimeout(fn, 0)? – David Long Feb 9, 2022 at 2:11

6 @David - It does not really matter as the browser has a min timeout time of ~4ms anyway. The general idea is that we want to communicate to the reader of the code that this `setTimeout()` will fire on a future tick of the event loop, not immediately. While even `setTimeout(fn, 0)` will fire on a future tick of the event loop, I thought it was clearer to a less educated reader of the code if I used a non-zero value for the time to illustrate it will happen in the future, not immediately. Not a big deal either way. – jfriend00 Feb 9, 2022 at 2:41

1 Thanks for this! Currently working on a layout library for the web using constraint layouts(android style) and I needed to optimize load time and remove initial unordered flicker

due to default DOM state before the js kicks in and your answer really helped.
– gbenroscience Aug 4, 2022 at 0:49

380

If you are doing **VANILLA** plain **JavaScript** without jQuery, then you must use (Internet Explorer 9 or later):

```
document.addEventListener("DOMContentLoaded", function(event) {
    // Your code to run since DOM is loaded and ready
});
```

Above is the equivalent of jQuery `.ready`:

```
$(document).ready(function() {
    console.log("Ready!");
});
```

Which ALSO could be written SHORTHAND like this, which jQuery will run after the ready even occurs.

```
$(function() {
    console.log("ready!");
});
```

NOT TO BE CONFUSED with BELOW (which is not meant to be DOM ready):

DO NOT use an IIFE like this that is self executing:

```
 Example:

(function() {
   // Your page initialization code here  - WRONG
   // The DOM will be available here   - WRONG
})();
```

This IIFE will NOT wait for your DOM to load. (I'm even talking about latest version of Chrome browser!)

Share
Improve this answer
Follow

edited Feb 8, 2019 at 18:31

answered Jul 1, 2015 at 20:31

Tom Stickel
20.3k ● 6 ● 114 ● 115

play() failed because the user didn't interact with the document first – CS QGB Nov 24, 2021 at 10:16

1 yes, first one for me :D if u want to add pure script to sharepoint script editor, use this.. document.addEventListener("DOMContentLoaded", function(event) – user1409736 Feb 16,

2022 at 22:14

In order to make play work, you have to add the `muted` parameter to the `video` tag.
– Ruben Alves Jan 24, 2023 at 19:19

192

I would like to mention some of the possible ways here together with a **pure javascript trick which works across all browsers**:

```
// with jQuery
$(document).ready(function(){ /* ... */ });

// shorter jQuery version
$(function(){ /* ... */ });

// without jQuery (doesn't work in older IEs)
document.addEventListener('DOMContentLoaded', function(){
    // your code goes here
}, false);

// and here's the trick (works everywhere)
function r(f){/in/.test(document.readyState)?setTimeout('r('+f+')',9):f()}
// use like
r(function(){
    alert('DOM Ready!');
});
```

The trick here, as explained by the original author, is that we are checking the **document.readyState** property. If it contains the string `in` (as in `uninitialized` and `loading`, the first two DOM ready states out of 5) we set a timeout and check again. Otherwise, we execute the passed function.

And here's the jsFiddle for the trick which **works across all browsers.**

Thanks to Tutorialzine for including this in their book.

Share

Improve this answer

Follow

edited Nov 4, 2016 at 5:18

answered May 19, 2015 at 7:58

Ram Patra

16.6k ● 13 ● 70 ● 87

58 Very bad approach, using a timeout loop with an arbitrary 9ms interval, and using eval. Also checking for just /in/ doesn't make much sense. – Vitim.us Nov 22, 2015 at 14:51

81

Tested in IE9, and latest Firefox and Chrome and also supported in IE8.

```
document.onreadystatechange = function () {
  var state = document.readyState;
  if (state == 'interactive') {
      init();
```

```
  } else if (state == 'complete') {
      initOnCompleteLoad();
  }
};
```

Example: http://jsfiddle.net/electricvisions/Jacck/

UPDATE - reusable version

I have just developed the following. It's a rather simplistic equivalent to jQuery or Dom ready without backwards compatibility. It probably needs further refinement. Tested in latest versions of Chrome, Firefox and IE (10/11) and should work in older browsers as commented on. I'll update if I find any issues.

```
window.readyHandlers = [];
window.ready = function ready(handler) {
  window.readyHandlers.push(handler);
  handleState();
};

window.handleState = function handleState () {
  if (['interactive', 'complete'].indexOf(document.readyState) > -1) {
    while(window.readyHandlers.length > 0) {
      (window.readyHandlers.shift())();
    }
  }
};

document.onreadystatechange = window.handleState;
```

Usage:

```
ready(function () {
  // your code here
});
```

It's written to handle async loading of JS but you might want to sync load this script first unless you're minifying. I've found it useful in development.

Modern browsers also support async loading of scripts which further enhances the experience. Support for async means multiple scripts can be downloaded simultaneously all while still rendering the page. Just watch out when depending on other scripts loaded asynchronously or use a minifier or something like browserify to handle dependencies.

edited Dec 3, 2016 at 7:46

community wiki
9 revs, 4 users 90%
PhilT

26

The good folks at HubSpot have a resource where you can find pure Javascript methodologies for achieving a lot of jQuery goodness - including `ready`

http://youmightnotneedjquery.com/#ready

```
function ready(fn) {
  if (document.readyState != 'loading'){
    fn();
  } else if (document.addEventListener) {
    document.addEventListener('DOMContentLoaded', fn);
  } else {
    document.attachEvent('onreadystatechange', function() {
      if (document.readyState != 'loading')
        fn();
    });
  }
}
```

example inline usage:

```
ready(function() { alert('hello'); });
```

Share

Improve this answer

Follow

edited Oct 28, 2016 at 17:39

rogerdpack

66.5k ● 39 ● 282 ● 401

answered Jun 10, 2015 at 13:27

Lorcan O'Neill

3,393 ● 1 ● 28 ● 25

25

I'm not quite sure what you're asking, but maybe this can help:

```
window.onload = function(){
    // Code. . .

}
```

Or:

```
window.onload = main;

function main(){
    // Code. . .

}
```

Share

Improve this answer

Follow

edited Oct 13, 2017 at 14:01

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Jun 19, 2016 at 6:03

Zak The Hat

327 ● 1 ● 7 ● 15

To me, this seems to be the correct answer and a lot simpler than the alternatives.
– Julian Knight Apr 10, 2022 at 14:05

3 While it's simpler and may seem correct, the `onload` can be fired before the DOM is ready, unlike using `DOMContentLoaded`; though the OP can be forgiven due to the age of the post, the above comment is much newer and should consider this – forbiddenera Jul 31, 2023 at 10:07

13

Your method (placing script before the closing body tag)

```
<script>
   myFunction()
</script>
</body>
</html>
```

is a reliable way to support old and new browsers.

Share

Improve this answer

Follow

edited Oct 28, 2016 at 16:38

rogerdpack

66.5k • 39 • 282 • 401

answered Mar 28, 2012 at 0:46

Kernel James

4,044 • 29 • 40

7

Ready

```
function ready(fn){var d=document;(d.readyState=='loading')?
d.addEventListener('DOMContentLoaded',fn):fn();}
```

Use like

```
ready(function(){
    //some code
});
```

For self invoking code

```
(function(fn){var d=document;(d.readyState=='loading')?
d.addEventListener('DOMContentLoaded',fn):fn();})(function(){

    //Some Code here
    //DOM is avaliable
    //var h1s = document.querySelector("h1");

});
```

Support: IE9+

Share Improve this answer Follow

answered Nov 24, 2015 at 5:15

Vitim.us

22.1k ● 15 ● 99 ● 114

1 This answer is redundant, it's already been mentioned here: stackoverflow.com/a/30757781/1385441 – Ram Patra Feb 26, 2016 at 16:47

2 @RamPatra Not really. This answer is much more compressed and way nicer to write. – I try so hard but I cry harder Sep 6, 2022 at 9:14

4

Here's a cleaned-up, non-eval-using version of Ram-swaroop's "works in all browsers" variety--works in all browsers!

```
function onReady(yourMethod) {
  var readyStateCheckInterval = setInterval(function() {
    if (document && document.readyState === 'complete') { // Or 'interactive'
      clearInterval(readyStateCheckInterval);
      yourMethod();
    }
  }, 10);
}
// use like
onReady(function() { alert('hello'); } );
```

It does wait an extra 10 ms to run, however, so here's a more complicated way that shouldn't:

```
function onReady(yourMethod) {
  if (document.readyState === 'complete') { // Or also compare to 'interactive'
    setTimeout(yourMethod, 1); // Schedule to run immediately
  }
  else {
    readyStateCheckInterval = setInterval(function() {
      if (document.readyState === 'complete') { // Or also compare to
'interactive'
        clearInterval(readyStateCheckInterval);
        yourMethod();
      }
    }, 10);
  }
}

// Use like
onReady(function() { alert('hello'); } );

// Or
onReady(functionName);
```

See also *How to check if DOM is ready without a framework?*.

Share
Improve this answer
Follow

edited Apr 13, 2018 at 16:00

answered Oct 28, 2016 at 17:52

rogerdpack
66.5k ● 39 ● 282 ● 401

3

`document.ondomcontentready=function(){}` should do the trick, but it doesn't have full browser compatibility.

Seems like you should just use jQuery min

Share
Improve this answer
Follow

edited May 16, 2013 at 22:47

Nakilon
35k ● 16 ● 111 ● 146

answered Mar 28, 2012 at 0:04

Max Hudson
10.2k ● 15 ● 60 ● 110