How do I include a JavaScript file in another JavaScript file?

Asked 14 years, 10 months ago Modified 2 months ago Viewed 4.4m times

6314

How do I include a JavaScript file inside another JavaScript file, similar to `@import` in CSS?

javascript file import include

Share Edit Follow

edited Jul 24, 2022 at 23:47

Mateen Ulhaq
25.8k ● 20 ● 111 ● 143

asked Jun 4, 2009 at 11:59

Alec Smart
95.1k ● 39 ● 122 ● 185

3 MDN docs about JavaScript modules, with corresponding examples on github – djvg Nov 20, 2021 at 19:08

72 Answers

Sorted by: Highest score (default)

1 2 3 Next

5437

The old versions of JavaScript had no import, include, or require, so many different approaches to this problem have been developed.

But since 2015 (ES6), JavaScript has had the ES6 modules standard to import modules in Node.js, which is also supported by most modern browsers.

For compatibility with older browsers, build tools like Webpack and Rollup and/or transpilation tools like Babel can be used.

ES6 Modules

ECMAScript (ES6) modules have been supported in Node.js since v8.5, with the `--experimental-modules` flag, and since at least Node.js v13.8.0 without the flag. To enable "ESM" (vs. Node.js's previous CommonJS-style module system ["CJS"]) you either use `"type": "module"` in `package.json` or give the files the extension `.mjs`. (Similarly, modules written with Node.js's previous CJS module can be named `.cjs` if your default is ESM.)

Using `package.json`:

```
{
    "type": "module"
}
```

Then `module.js`:

```
export function hello() {
  return "Hello";
}
```

Then `main.js`:

```
import { hello } from './module.js';
let val = hello();  // val is "Hello";
```

Using `.mjs`, you'd have `module.mjs`:

```
export function hello() {
  return "Hello";
}
```

Then `main.mjs`:

```
import { hello } from './module.mjs';
let val = hello();  // val is "Hello";
```

ECMAScript modules in browsers

Browsers have had support for loading ECMAScript modules directly (no tools like Webpack required) since Safari 10.1, Chrome 61, Firefox 60, and Edge 16. Check the current support at caniuse. There is no need to use Node.js' `.mjs` extension; browsers completely ignore file extensions on modules/scripts.

```
<script type="module">
  import { hello } from './hello.mjs'; // Or the extension could be just `.js`
  hello('world');
</script>
```

```
// hello.mjs -- or the extension could be just `.js`
export function hello(text) {
  const div = document.createElement('div');
  div.textContent = `Hello ${text}`;
  document.body.appendChild(div);
}
```

Read more at https://jakearchibald.com/2017/es-modules-in-browsers/

Dynamic imports in browsers

Dynamic imports let the script load other scripts as needed:

```
<script type="module">
  import('hello.mjs').then(module => {
```

```
      module.hello('world');
    });
</script>
```

Read more at https://developers.google.com/web/updates/2017/11/dynamic-import

Node.js require

The older CJS module style, still widely used in Node.js, is the `module.exports` / `require` system.

```
// mymodule.js
module.exports = {
   hello: function() {
      return "Hello";
   }
}
```

```
// server.js
const myModule = require('./mymodule');
let val = myModule.hello(); // val is "Hello"
```

There are other ways for JavaScript to include external JavaScript contents in browsers that do not require preprocessing.

AJAX Loading

You could load an additional script with an AJAX call and then use `eval` to run it. This is the most straightforward way, but it is limited to your domain because of the JavaScript sandbox security model. Using `eval` also opens the door to bugs, hacks and security issues.

Fetch Loading

Like Dynamic Imports you can load one or many scripts with a `fetch` call using promises to control order of execution for script dependencies using the Fetch Inject library:

```
fetchInject([
  'https://cdn.jsdelivr.net/momentjs/2.17.1/moment.min.js'
]).then(() => {
  console.log(`Finish in less than ${moment().endOf('year').fromNow(true)}`)
})
```

jQuery Loading

The jQuery library provides loading functionality in one line:

```
$.getScript("my_lovely_script.js", function() {
   alert("Script loaded but not necessarily executed.");
```

```
});
```

Dynamic Script Loading

You could add a script tag with the script URL into the HTML. To avoid the overhead of jQuery, this is an ideal solution.

The script can even reside on a different server. Furthermore, the browser evaluates the code. The `<script>` tag can be injected into either the web page `<head>`, or inserted just before the closing `</body>` tag.

Here is an example of how this could work:

```
function dynamicallyLoadScript(url) {
    var script = document.createElement("script");  // create a script DOM node
    script.src = url;  // set its src to the provided URL

    document.head.appendChild(script);  // add it to the end of the head
section of the page (could change 'head' to 'body' to add it to the end of the
body section instead)
}
```

This function will add a new `<script>` tag to the end of the head section of the page, where the `src` attribute is set to the URL which is given to the function as the first parameter.

Both of these solutions are discussed and illustrated in JavaScript Madness: Dynamic Script Loading.

Detecting when the script has been executed

Now, there is a big issue you must know about. Doing that implies that *you remotely load the code*. Modern web browsers will load the file and keep executing your current script because they load everything asynchronously to improve performance. (This applies to both the jQuery method and the manual dynamic script loading method.)

It means that if you use these tricks directly, *you won't be able to use your newly loaded code the next line after you asked it to be loaded*, because it will be still loading.

For example: `my_lovely_script.js` contains `MySuperObject`:

```
var js = document.createElement("script");

js.type = "text/javascript";
js.src = jsFilePath;

document.body.appendChild(js);

var s = new MySuperObject();

Error : MySuperObject is undefined
```

Then you reload the page hitting `F5`. And it works! Confusing...

So what to do about it ?

Well, you can use the hack the author suggests in the link I gave you. In summary, for people in a hurry, he uses an event to run a callback function when the script is loaded. So you can put all the code using the remote library in the callback function. For example:

```
function loadScript(url, callback)
{
    // Adding the script tag to the head as suggested before
    var head = document.head;
    var script = document.createElement('script');
    script.type = 'text/javascript';
    script.src = url;

    // Then bind the event to the callback function.
    // There are several events for cross browser compatibility.
    script.onreadystatechange = callback;
    script.onload = callback;

    // Fire the loading
    head.appendChild(script);
}
```

Then you write the code you want to use AFTER the script is loaded in a lambda function:

```
var myPrettyCode = function() {
   // Here, do whatever you want
};
```

Then you run all that:

```
loadScript("my_lovely_script.js", myPrettyCode);
```

Note that the script may execute after the DOM has loaded, or before, depending on the browser and whether you included the line `script.async = false;`. There's a great article on Javascript loading in general which discusses this.

Source Code Merge/Preprocessing

As mentioned at the top of this answer, many developers use build/transpilation tool(s) like Parcel, Webpack, or Babel in their projects, allowing them to use upcoming JavaScript syntax, provide backward compatibility for older browsers, combine files, minify, perform code splitting etc.

Share Edit Follow

edited Sep 19, 2022 at 9:18

Keatinge

4,331 ● 6 ● 27 ● 44

answered Jun 4, 2009 at 12:13

Bite code

587k ● 113 ● 306 ● 332

If anyone is looking for something more advanced, try out RequireJS. You'll get added benefits

631

such as dependency management, better concurrency, and avoid duplication (that is, retrieving a script more than once).

You can write your JavaScript files in "modules" and then reference them as dependencies in other scripts. Or you can use RequireJS as a simple "go get this script" solution.

Example:

Define dependencies as modules:

some-dependency.js

```
define(['lib/dependency1', 'lib/dependency2'], function (d1, d2) {

     //Your actual script goes here.
     //The dependent scripts will be fetched if necessary.

     return libraryObject;  //For example, jQuery object
});
```

implementation.js is your "main" JavaScript file that depends on some-dependency.js

```
require(['some-dependency'], function(dependency) {

    //Your script goes here
    //some-dependency.js is fetched.
    //Then your script is executed
});
```

Excerpt from the GitHub README:

> RequireJS loads plain JavaScript files as well as more defined modules. It is optimized for in-browser use, including in a Web Worker, but it can be used in other JavaScript environments, like Rhino and Node. It implements the Asynchronous Module API.
>
> RequireJS uses plain script tags to load modules/files, so it should allow for easy debugging. It can be used simply to load existing JavaScript files, so **you can add it to your existing project without having to re-write your JavaScript files.**
>
> ...

Share Edit Follow

edited Sep 28, 2013 at 13:11

Peter Mortensen
31k ● 22 ● 109 ● 132

answered Jun 7, 2012 at 20:55

John Strickler
25.3k ● 4 ● 53 ● 69

239

There actually *is* a way to load a JavaScript file *not* asynchronously, so you could use the functions included in your newly loaded file right after loading it, and I think it works in all browsers.

You need to use `jQuery.append()` on the `<head>` element of your page, that is:

```
$("head").append($("<script></script>").attr("src", url));

/* Note that following line of code is incorrect because it doesn't escape the
 * HTML attribute src correctly and will fail if `url` contains special
characters:
 * $("head").append('<script src="' + url + '"></script>');
 */
```

However, this method also has a problem: if an error happens in the imported JavaScript file, Firebug (and also Firefox Error Console and Chrome Developer Tools as well) will report its place incorrectly, which is a big problem if you use Firebug to track JavaScript errors down a lot (I do). Firebug simply doesn't know about the newly loaded file for some reason, so if an error occurs in that file, it reports that it occurred in your main HTML file, and you will have trouble finding out the real reason for the error.

But if that is not a problem for you, then this method should work.

I have actually written a jQuery plugin called *$.import_js()* which uses this method:

```
(function($)
{
    /*
     * $.import_js() helper (for JavaScript importing within JavaScript code).
     */
    var import_js_imported = [];

    $.extend(true,
    {
        import_js : function(script)
        {
            var found = false;
            for (var i = 0; i < import_js_imported.length; i++)
                if (import_js_imported[i] == script) {
                    found = true;
                    break;
                }

            if (found == false) {
                $("head").append($('<script></script').attr('src', script));
                import_js_imported.push(script);
            }
        }
    });

})(jQuery);
```

So all you would need to do to import JavaScript is:

```
$.import_js('/path_to_project/scripts/somefunctions.js');
```

I also made a simple test for this at Example.

It includes a `main.js` file in the main HTML and then the script in `main.js` uses `$.import_js()` to import an additional file called `included.js`, which defines this function:

```
function hello()
{
    alert("Hello world!");
}
```

And right after including `included.js`, the `hello()` function is called, and you get the alert.

(This answer is in response to e-satis' comment).

Share Edit Follow

edited May 12, 2021 at 12:57

Flimm
143k ● 47 ● 259 ● 278

answered Apr 28, 2011 at 15:25

Kipras
2,684 ● 1 ● 16 ● 10

176

Another way, that in my opinion is much cleaner, is to make a synchronous Ajax request instead of using a `<script>` tag. Which is also how Node.js handles includes.

Here's an example using jQuery:

```
function require(script) {
    $.ajax({
        url: script,
        dataType: "script",
        async: false,           // <-- This is the key
        success: function () {
            // all good...
        },
        error: function () {
            throw new Error("Could not load script " + script);
        }
    });
}
```

You can then use it in your code as you'd usually use an include:

```
require("/scripts/subscript.js");
```

And be able to call a function from the required script in the next line:

```
subscript.doSomethingCool();
```

Share Edit Follow

edited Sep 28, 2013 at 12:55

Peter Mortensen
31k ● 22 ● 109 ● 132

answered Sep 8, 2011 at 18:22

Ariel
4,502 ● 2 ● 22 ● 23

132

It is possible to dynamically generate a JavaScript tag and append it to HTML document from inside other JavaScript code. This will load targeted JavaScript file.

```
function includeJs(jsFilePath) {
    var js = document.createElement("script");
```

```
    js.type = "text/javascript";
    js.src = jsFilePath;

    document.body.appendChild(js);
}

includeJs("/path/to/some/file.js");
```

Share Edit Follow

edited Sep 28, 2013 at 12:44

Peter Mortensen
31k ● 22 ● 109 ● 132

answered Jun 4, 2009 at 12:02

Svitlana Maksymchuk
4,380 ● 2 ● 16 ● 11

112

There is a good news for you. Very soon you will be able to load JavaScript code easily. It will become a standard way of importing modules of JavaScript code and will be part of core JavaScript itself.

You simply have to write `import cond from 'cond.js';` to load a macro named `cond` from a file `cond.js`.

So you don't have to rely upon any JavaScript framework nor do you have to explicitly make Ajax calls.

Refer to:

- *Static module resolution*
- *Module loaders*

Share Edit Follow

edited Oct 19, 2014 at 18:05

Peter Mortensen
31k ● 22 ● 109 ● 132

answered Jul 3, 2012 at 13:32

Imdad
5,962 ● 4 ● 35 ● 53

2 Seven years later, this answer doesn't work: "SyntaxError: import declarations may only appear at top level of a module". – David Spector May 27, 2021 at 15:52

91

Statement `import` is in ECMAScript 6.

Syntax

```
import name from "module-name";
import { member } from "module-name";
import { member as alias } from "module-name";
import { member1 , member2 } from "module-name";
import { member1 , member2 as alias2 , [...] } from "module-name";
import name , { member [ , [...] ] } from "module-name";
import "module-name" as name;
```

Share Edit Follow

edited Dec 11, 2016 at 9:47
Peter Mortensen

answered Apr 17, 2015 at 1:56
draupnie

 31k ● 22 ● 109 ● 132  1,120 ● 8 ● 11

69

Maybe you can use this function that I found on this page *How do I include a JavaScript file in a JavaScript file?*:

```
function include(filename)
{
    var head = document.getElementsByTagName('head')[0];

    var script = document.createElement('script');
    script.src = filename;
    script.type = 'text/javascript';

    head.appendChild(script)
}
```

Share Edit Follow

edited Nov 15, 2014 at 17:01

Ajeet Lakhani
3,798 ● 2 ● 23 ● 37

answered Jun 4, 2009 at 12:04

Arnaud Gouder de Beauregard
1,529 ● 1 ● 13 ● 22

61

Here is a **synchronous** version **without jQuery**:

```
function myRequire( url ) {
    var ajax = new XMLHttpRequest();
    ajax.open( 'GET', url, false ); // <-- the 'false' makes it synchronous
    ajax.onreadystatechange = function () {
        var script = ajax.response || ajax.responseText;
        if (ajax.readyState === 4) {
            switch( ajax.status) {
                case 200:
                    eval.apply( window, [script] );
                    console.log("script loaded: ", url);
                    break;
                default:
                    console.log("ERROR: script not loaded: ", url);
            }
        }
    };
    ajax.send(null);
}
```

Note that to get this working cross-domain, the server will need to set `allow-origin` header in its response.

Share Edit Follow

edited Jun 18, 2015 at 14:44

Flimm
143k ● 47 ● 259 ● 278

answered Dec 11, 2013 at 11:54

heinob
19.3k ● 5 ● 42 ● 61

I just wrote this JavaScript code (using Prototype for DOM manipulation):

53

```
var require = (function() {
    var _required = {};
    return (function(url, callback) {
        if (typeof url == 'object') {
            // We've (hopefully) got an array: time to chain!
            if (url.length > 1) {
                // Load the nth file as soon as everything up to the
                // n-1th one is done.
                require(url.slice(0, url.length - 1), function() {
                    require(url[url.length - 1], callback);
                });
            } else if (url.length == 1) {
                require(url[0], callback);
            }
            return;
        }
        if (typeof _required[url] == 'undefined') {
            // Haven't loaded this URL yet; gogogo!
            _required[url] = [];

            var script = new Element('script', {
                src: url,
                type: 'text/javascript'
            });
            script.observe('load', function() {
                console.log("script " + url + " loaded.");
                _required[url].each(function(cb) {
                    cb.call(); // TODO: does this execute in the right context?
                });
                _required[url] = true;
            });

            $$('head')[0].insert(script);
        } else if (typeof _required[url] == 'boolean') {
            // We already loaded the thing, so go ahead.
            if (callback) {
                callback.call();
            }
            return;
        }

        if (callback) {
            _required[url].push(callback);
        }
    });
})();
```

Usage:

```
<script src="prototype.js"></script>
<script src="require.js"></script>
<script>
    require(['foo.js','bar.js'], function () {
        /* Use foo.js and bar.js here */
    });
</script>
```

Gist: http://gist.github.com/284442.

Share Edit Follow

edited Jun 25, 2017 at 21:04
Talha Awan

answered Jan 23, 2010 at 5:20
nornagon

53

If you want it in pure JavaScript, you can use `document.write`.

```
document.write('<script src="myscript.js" type="text/javascript"></script>');
```

If you use the jQuery library, you can use the `$.getScript` method.

```
$.getScript("another_script.js");
```

Share Edit Follow

edited Oct 3, 2020 at 20:47

Peter Mortensen
31k ● 22 ● 109 ● 132

answered Nov 13, 2013 at 9:18

Venu immadi
1,595 ● 11 ● 20

52

Here's the generalized version of how Facebook does it for their ubiquitous Like button:

```
<script>
  var firstScript = document.getElementsByTagName('script')[0],
      js = document.createElement('script');
  js.src =
'https://cdnjs.cloudflare.com/ajax/libs/Snowstorm/20131208/snowstorm-min.js';
  js.onload = function () {
    // do stuff with your dynamically loaded script
    snowStorm.snowColor = '#99ccff';
  };
  firstScript.parentNode.insertBefore(js, firstScript);
</script>
```

Run code snippet Expand snippet

If it works for Facebook, it will work for you.

The reason why we look for the first `script` element instead of `head` or `body` is because some browsers don't create one if missing, but we're guaranteed to have a `script` element - this one. Read more at http://www.jspatterns.com/the-ridiculous-case-of-adding-a-script-element/.

Share Edit Follow

edited Jul 8, 2015 at 2:48

answered Jul 8, 2015 at 2:41

Dan Dascalescu
148k ● 59 ● 325 ● 414

36

You can also assemble your scripts using PHP:

File `main.js.php`:

```
<?php
    header('Content-type:text/javascript; charset=utf-8');
    include_once("foo.js.php");
    include_once("bar.js.php");
?>

// Main JavaScript code goes here
```

Share Edit Follow

edited Sep 28, 2013 at 12:50

Peter Mortensen

31k ● 22 ● 109 ● 132

answered Dec 27, 2010 at 21:03

Calmarius

19k ● 19 ● 114 ● 160

33

Most of solutions shown here imply dynamical loading. I was searching instead for a compiler which assemble all the depended files into a single output file. The same as Less/Sass preprocessors deal with the CSS `@import` at-rule. Since I didn't find anything decent of this sort, I wrote a simple tool solving the issue.

So here is the compiler, https://github.com/dsheiko/jsic, which replaces `$import("file-path")` with the requested file content securely. Here is the corresponding Grunt plugin: https://github.com/dsheiko/grunt-jsic.

On the jQuery master branch, they simply concatenate atomic source files into a single one starting with `intro.js` and ending with `outtro.js`. That doesn't suits me as it provides no flexibility on the source code design. Check out how it works with jsic:

src/main.js

```
var foo = $import("./Form/Input/Tel");
```

src/Form/Input/Tel.js

```
function() {
    return {
          prop: "",
          method: function(){}
    }
}
```

Now we can run the compiler:

```
node jsic.js src/main.js build/mail.js
```

And get the combined file

build/main.js

```
var foo = function() {
    return {
          prop: "",
```

```
        method: function(){}
    }
};
```

Share Edit Follow

edited Oct 19, 2014 at 18:13

Peter Mortensen

31k ● 22 ● 109 ● 132

answered Jul 13, 2013 at 21:44

Dmitry Sheiko

2,170 ● 1 ● 25 ● 28

30

If your intention to load the JavaScript file is **using the functions from the imported/included file**, you can also define a global object and set the functions as object items. For instance:

global.js

```
A = {};
```

file1.js

```
A.func1 = function() {
  console.log("func1");
}
```

file2.js

```
A.func2 = function() {
  console.log("func2");
}
```

main.js

```
A.func1();
A.func2();
```

You just need to be careful when you are including scripts in an HTML file. The order should be as in below:

```
<head>
  <script type="text/javascript" src="global.js"></script>
  <script type="text/javascript" src="file1.js"></script>
  <script type="text/javascript" src="file2.js"></script>
  <script type="text/javascript" src="main.js"></script>
</head>
```

Share Edit Follow

edited Dec 11, 2016 at 9:42

Peter Mortensen

31k ● 22 ● 109 ● 132

answered Jul 24, 2015 at 6:53

Adem İlhan

1,480 ● 3 ● 16 ● 26

ES6 Modules

26

Yes, use type="module" in a script tag (support):

```
<script type="module" src="script.js"></script>
```

And in a `script.js` file include another file like this:

```
import { hello } from './module.js';
...
// alert(hello());
```

In 'module.js' you must export the function/class that you will import:

```
export function hello() {
    return "Hello World";
}
```

A working example is here.

Share Edit Follow

edited Dec 31, 2022 at 18:16

answered Jul 10, 2019 at 16:12

Kamil Kiełczewski

88.8k ● 31 ● 380 ● 353

25

This should do:

```
xhr = new XMLHttpRequest();
xhr.open("GET", "/soap/ajax/11.0/connection.js", false);
xhr.send();
eval(xhr.responseText);
```

Share Edit Follow

edited Nov 2, 2014 at 13:50

Peter Mortensen

31k ● 22 ● 109 ● 132

answered Mar 24, 2013 at 19:32

tggagne

2,874 ● 1 ● 22 ● 15

25

Or rather than including at run time, use a script to concatenate prior to upload.

I use Sprockets (I don't know if there are others). You build your JavaScript code in separate files and include comments that are processed by the Sprockets engine as includes. For development you can include files sequentially, then for production to merge them...

See also:

- *Introducing Sprockets: JavaScript dependency management and concatenation*

Share Edit Follow

edited Feb 15, 2023 at 6:59

Imdad

5,962 ● 4 ● 35 ● 53

answered Jun 7, 2012 at 20:48

JMawer

259 ● 3 ● 2

23

I had a simple issue, but I was baffled by responses to this question.

I had to use a variable (myVar1) defined in one JavaScript file (myvariables.js) in another JavaScript file (main.js).

For this I did as below:

Loaded the JavaScript code in the HTML file, in the correct order, myvariables.js first, then main.js:

```
<html>
    <body onload="bodyReady();" >

        <script src="myvariables.js" > </script>
        <script src="main.js" > </script>

        <!-- Some other code -->
    </body>
</html>
```

File: myvariables.js

```
var myVar1 = "I am variable from myvariables.js";
```

File: main.js

```
// ...
function bodyReady() {
    // ...
    alert (myVar1);    // This shows "I am variable from myvariables.js", which
I needed
    // ...
}
// ...
```

As you saw, I had use a variable in one JavaScript file in another JavaScript file, but I didn't need to include one in another. I just needed to ensure that the first JavaScript file loaded before the second JavaScript file, and, the first JavaScript file's variables are accessible in the second JavaScript file, automatically.

This saved my day. I hope this helps.

Share Edit Follow

edited Dec 11, 2016 at 9:45

Peter Mortensen

31k ● 22 ● 109 ● 132

answered Jul 22, 2015 at 2:15

Manohar Reddy Poreddy

26.4k ● 9 ● 160 ● 144

19

In a modern language with the check if script has already been loaded, it would be:

```
function loadJs( url ){
  return new Promise(( resolve, reject ) => {
    if (document.querySelector( `head > script[ src = "${url}" ]`) !== null ){
```

```
        console.warn( `script already loaded: ${url}` );
        resolve();
    }
    const script = document.createElement( "script" );
    script.src = url;
    script.onload = resolve;
    script.onerror = function( reason ){
        // This can be useful for your error-handling code
        reason.message = `error trying to load script ${url}`;
        reject( reason );
    };
    document.head.appendChild( script );
  });
}
```

Usage (async/await):

```
try { await loadJs("https://.../script.js"); }
catch(error) { console.log(error); }
```

or

```
await loadJs( "https://.../script.js" ).catch( err => {} );
```

Usage (Promise):

```
loadJs( "https://.../script.js" ).then( res => {} ).catch( err => {} );
```

Share Edit Follow

edited Oct 3, 2020 at 19:47

Peter Mortensen

31k ● 22 ● 109 ● 132

answered Jul 20, 2017 at 15:15

Dmitry Sheiko

2,170 ● 1 ● 25 ● 28

19

The `@import` syntax for achieving CSS-like JavaScript importing is possible using a tool such as Mixture via their special `.mix` file type (see here). I assume the application does this via one of above-mentioned methods.

From the Mixture documentation on `.mix` files:

> Mix files are simply .js or .css files with .mix. in the file name. A mix file simply extends the functionality of a normal style or script file and allows you to import and combine.

Here's an example `.mix` file that combines multiple `.js` files into one:

```
// scripts-global.mix.js
// Plugins - Global

@import "global-plugins/headroom.js";
@import "global-plugins/retina-1.1.0.js";
@import "global-plugins/isotope.js";
@import "global-plugins/jquery.fitvids.js";
```

Mixture outputs this as `scripts-global.js` and also as a minified version (`scripts-global.min.js`).

Note: I'm not in any way affiliated with Mixture, other than using it as a front-end development tool. I came across this question upon seeing a `.mix` JavaScript file in action (in one of the Mixture boilerplates) and being a bit confused by it ("you can do this?" I thought to myself). Then I realized that it was an application-specific file type (somewhat disappointing, agreed). Nevertheless, figured the knowledge might be helpful for others.

Note: Mixture was discontinued on 2016/07/26 (after being open sourced on 2015/04/12).

Share Edit Follow

edited Oct 5, 2020 at 17:34

answered Mar 14, 2014 at 4:40

Isaac Gregson

2,039 ● 1 ● 22 ● 31

18

In case you are using Web Workers and want to include additional scripts in the scope of the worker, the other answers provided about adding scripts to the `head` tag, etc. will not work for you.

Fortunately, Web Workers have their own `importScripts` function which is a global function in the scope of the Web Worker, native to the browser itself as it is part of the specification.

Alternatively, as the second highest voted answer to your question highlights, RequireJS can also handle including scripts inside a Web Worker (likely calling `importScripts` itself, but with a few other useful features).

Share Edit Follow

edited May 23, 2017 at 12:34

Community Bot

1 ● 1

answered Jan 1, 2015 at 8:58

Turnerj

4,268 ● 5 ● 36 ● 53

18

Although these answers are great, there is a simple "solution" that has been around since script loading existed, and it will cover 99.999% of most people's use cases. Just include the script you need before the script that requires it. For most projects it does not take long to determine which scripts are needed and in what order.

```
<!DOCTYPE HTML>
<html>
    <head>
        <script src="script1.js"></script>
        <script src="script2.js"></script>
    </head>
    <body></body>
</html>
```

If script2 requires script1, this really is the absolute easiest way to do something like this. I'm very surprised no-one has brought this up, as it's the most obvious and simplest answer that will apply in nearly every single case.

Share Edit Follow

answered Apr 12, 2018 at 19:38

KthProg

2,080 ● 1 ● 25 ● 32

17

```
var js = document.createElement("script");

js.type = "text/javascript";
js.src = jsFilePath;

document.body.appendChild(js);
```

Share Edit Follow

edited Aug 22, 2012 at 6:44

Spudley

167k ● 39 ● 234 ● 308

answered Aug 22, 2012 at 4:33

Sam4Code

541 ● 5 ● 9

14

My usual method is:

```
var require = function (src, cb) {
    cb = cb || function () {};

    var newScriptTag = document.createElement('script'),
        firstScriptTag = document.getElementsByTagName('script')[0];
    newScriptTag.src = src;
    newScriptTag.async = true;
    newScriptTag.onload = newScriptTag.onreadystatechange = function () {
        (!this.readyState || this.readyState === 'loaded' || this.readyState
=== 'complete') && (cb());
    };
    firstScriptTag.parentNode.insertBefore(newScriptTag, firstScriptTag);
}
```

It works great and uses no page-reloads for me. I've tried the AJAX method (one of the other answers) but it doesn't seem to work as nicely for me.

Here's an explanation of how the code works for those that are curious: essentially, it creates a new script tag (after the first one) of the URL. It sets it to asynchronous mode so it doesn't block the rest of the code, but calls a callback when the readyState (the state of the content to be loaded) changes to 'loaded'.

Share Edit Follow

edited Aug 26, 2017 at 17:02

answered May 31, 2013 at 19:31

Alexis Purslane

1,360 ● 11 ● 32

13

I wrote a simple module that automates the job of importing/including module scripts in JavaScript. For detailed explanation of the code, refer to the blog post *JavaScript require / import / include modules*.

```
// ----- USAGE -----
```

```
require('ivar.util.string');
require('ivar.net.*');
require('ivar/util/array.js');
require('http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js');

ready(function(){
    //Do something when required scripts are loaded
});

    //--------------------

var _rmod = _rmod || {}; //Require module namespace
_rmod.LOADED = false;
_rmod.on_ready_fn_stack = [];
_rmod.libpath = '';
_rmod.imported = {};
_rmod.loading = {
    scripts: {},
    length: 0
};

_rmod.findScriptPath = function(script_name) {
    var script_elems = document.getElementsByTagName('script');
    for (var i = 0; i < script_elems.length; i++) {
        if (script_elems[i].src.endsWith(script_name)) {
            var href = window.location.href;
            href = href.substring(0, href.lastIndexOf('/'));
            var url = script_elems[i].src.substring(0, script_elems[i].length -
script_name.length);
            return url.substring(href.length+1, url.length);
        }
    }
    return '';
};

_rmod.libpath = _rmod.findScriptPath('script.js'); //Path of your main script
used to mark
                                                   //the root directory of your
library, any library.


_rmod.injectScript = function(script_name, uri, callback, prepare) {

    if(!prepare)
        prepare(script_name, uri);

    var script_elem = document.createElement('script');
    script_elem.type = 'text/javascript';
    script_elem.title = script_name;
    script_elem.src = uri;
    script_elem.async = true;
    script_elem.defer = false;

    if(!callback)
        script_elem.onload = function() {
            callback(script_name, uri);
        };
    document.getElementsByTagName('head')[0].appendChild(script_elem);
};

_rmod.requirePrepare = function(script_name, uri) {
    _rmod.loading.scripts[script_name] = uri;
    _rmod.loading.length++;
};

_rmod.requireCallback = function(script_name, uri) {
```

```
    _rmod.loading.length--;
    delete _rmod.loading.scripts[script_name];
    _rmod.imported[script_name] = uri;

    if(_rmod.loading.length == 0)
        _rmod.onReady();
};

_rmod.onReady = function() {
    if (!_rmod.LOADED) {
        for (var i = 0; i < _rmod.on_ready_fn_stack.length; i++){
            _rmod.on_ready_fn_stack[i]();
        });
        _rmod.LOADED = true;
    }
};

_.rmod = namespaceToUri = function(script_name, url) {
    var np = script_name.split('.');
    if (np.getLast() === '*') {
        np.pop();
        np.push('_all');
    }

    if(!url)
        url = '';

    script_name = np.join('.');
    return  url + np.join('/')+'.js';
};

//You can rename based on your liking. I chose require, but it
//can be called include or anything else that is easy for you
//to remember or write, except "import", because it is reserved
//for future use.
var require = function(script_name) {
    var uri = '';
    if (script_name.indexOf('/') > -1) {
        uri = script_name;
        var lastSlash = uri.lastIndexOf('/');
        script_name = uri.substring(lastSlash+1, uri.length);
    }
    else {
        uri = _rmod.namespaceToUri(script_name, ivar._private.libpath);
    }

    if (!_rmod.loading.scripts.hasOwnProperty(script_name)
     && !_rmod.imported.hasOwnProperty(script_name)) {
        _rmod.injectScript(script_name, uri,
            _rmod.requireCallback,
                _rmod.requirePrepare);
    }
};

var ready = function(fn) {
    _rmod.on_ready_fn_stack.push(fn);
};
```


edited Oct 19, 2014 at 18:26

Peter Mortensen
31k ● 22 ● 109 ● 132

answered Apr 12, 2013 at 16:34

stamat
1,910 ● 22 ● 28

13

This script will add a JavaScript file to the top of any other `<script>` tag:

```
(function () {
    var li = document.createElement('script');
    li.type = 'text/javascript';
    li.src =
"http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js";
    li.async = true;
    var s = document.getElementsByTagName('script')[0];
    s.parentNode.insertBefore(li, s);
})();
```

Share Edit Follow

edited Oct 3, 2020 at 20:47

Peter Mortensen

31k ● 22 ● 109 ● 132

answered Oct 26, 2013 at 3:42

Vicky Gonsalves

11.7k ● 2 ● 37 ● 58

13

Keep it nice, short, simple, and maintainable! :]

```
// Third-party plugins / script (don't forget the full path is necessary)
var FULL_PATH = '', s =
[
    FULL_PATH + 'plugins/script.js'      // Script example
    FULL_PATH + 'plugins/jquery.1.2.js', // jQuery Library
    FULL_PATH + 'plugins/crypto-js/hmac-sha1.js',      // CryptoJS
    FULL_PATH + 'plugins/crypto-js/enc-base64-min.js'  // CryptoJS
];

function load(url)
{
    var ajax = new XMLHttpRequest();
    ajax.open('GET', url, false);
    ajax.onreadystatechange = function ()
    {
        var script = ajax.response || ajax.responseText;
        if (ajax.readyState === 4)
        {
            switch(ajax.status)
            {
                case 200:
                    eval.apply( window, [script] );
                    console.log("library loaded: ", url);
                    break;
                default:
                    console.log("ERROR: library not loaded: ", url);
            }
        }
    };
    ajax.send(null);
}

// Initialize a single load
load('plugins/script.js');

// Initialize a full load of scripts
if (s.length > 0)
{
    for (i = 0; i < s.length; i++)
    {
        load(s[i]);
```

```
    }
}
```

This code is simply a short functional example that *could* require additional feature functionality for full support on any (or given) platform.

Share Edit Follow

edited Oct 3, 2020 at 20:55

Peter Mortensen
31k ● 22 ● 109 ● 132

answered Feb 8, 2015 at 1:19

tfont
11k ● 7 ● 57 ● 52

12

I came to this question because I was looking for a simple way to maintain a collection of useful JavaScript plugins. After seeing some of the solutions here, I came up with this:

1. Set up a file called "plugins.js" (or extensions.js or whatever you want). Keep your plugin files together with that one master file.
2. plugins.js will have an array called `pluginNames[]` that we will iterate over `each()`, then append a `<script>` tag to the head for each plugin

```
//set array to be updated when we add or remove plugin files
var pluginNames = ["lettering", "fittext", "butterjam", etc.];

//one script tag for each plugin
$.each(pluginNames, function(){
    $('head').append('<script src="js/plugins/' + this + '.js"></script>');
});
```

3. Manually call just the one file in your head:
   `<script src="js/plugins/plugins.js"></script>`

BUT:

Even though all of the plugins get dropped into the head tag the way they ought to, they don't always get run by the browser when you click into the page or refresh.

I've found it's more reliable to just write the script tags in a PHP include. You only have to write it once and that's just as much work as calling the plugin using JavaScript.

Share Edit Follow

edited Jan 10, 2020 at 5:03

answered Dec 1, 2011 at 5:36

rgb_life
353 ● 3 ● 10

12

There are several ways to implement modules in JavaScript. Here are the two most popular ones:

ES6 Modules

Browsers do not support this moduling system yet, so in order for you to use this syntax you must use a bundler like Webpack. Using a bundler is better anyway because this can combine all of your

different files into a single (or a couple of related) files. This will serve the files from the server to the client faster because each HTTP request has some associated overhead accompanied with it. Thus by reducing the overall HTTP request we improve the performance. Here is an example of ES6 modules:

```
// main.js file

export function add (a, b) {
  return a + b;
}

export default function multiply (a, b) {
  return a * b;
}


// test.js file

import {add}, multiply from './main';   // For named exports between curly
braces {export1, export2}
                                        // For default exports without {}

console.log(multiply(2, 2));  // logs 4

console.log(add(1, 2));  // logs 3
```

CommonJS (used in Node.js)

This moduling system is used in Node.js. You basically add your exports to an object which is called `module.exports`. You then can access this object via a `require('modulePath')`. Important here is to realize that these modules are being cached, so if you `require()` a certain module twice it will return the already created module.

```
// main.js file

function add (a, b) {
  return a + b;
}

module.exports = add;  // Here we add our 'add' function to the exports object


// test.js file

const add = require('./main');

console.log(add(1,2));  // logs 3
```

Share Edit Follow

edited Oct 3, 2020 at 20:17

Peter Mortensen

31k ● 22 ● 109 ● 132

answered Aug 10, 2018 at 17:11

Willem van der Veen

35.1k ● 17 ● 196 ● 165

reputation requirement helps protect this question from spam and non-answer activity.