**The State of JavaScript Modules** (30 minute presentation)

Web & Mobile Developers Meetup

08/22/19

**Introduction**

This presentation will outline the history, evolution, and usage of JavaScript modules. The module world is confusing due to rapid growth and a lack of standard. There could be some mistakes in this presentation as a result.

**History**

As JavaScript became more complex, developers needed a good way of dependency and namespace management. This was happening at the same time JavaScript was moving towards the server-side.

Developers accomplished this originally by using *IIFE*s (Immediately Invoked Function Expressions) that attached their root object to the window and separated them into single files. For example, jQuery:



However, this required coding *script* tags in precise order which formed your dependency tree. JavaScript didn’t have a way of loading its own dependencies.

There are two parts of modules.

|  |  |
| --- | --- |
| Script Loaders | Module Definitions |

Scripts loaders can load your modules asynchronously (like RequireJS), and allow them to depend on each other and load in different orders.

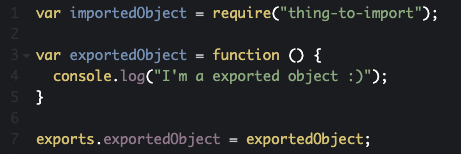
A module definition is essentially the API you use to define your module, which could be CommonJS, AMD, or UMD.

More info on script loaders:  
<https://msdn.microsoft.com/en-us/magazine/hh227261>

**Around 2009 (CommonJS and AMD introduced)**

**CommonJS**

CommonJS was the first module standard, originally called “ServerJS.” This is what NodeJS was based on. Most of the *npm* ecosystem is built upon this standard.

****

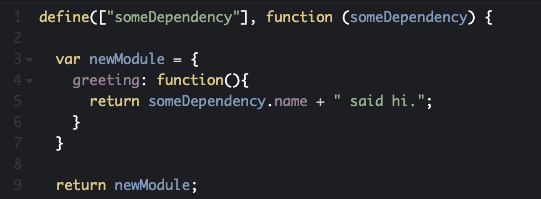
CommonJS introduces the *require()* method and *exports* object. CommonJS modules load dependencies on demand while executing the code.

**AMD**

The problem with CommonJS is that it lacks concurrency. For this reason, AMD was introduced (AMD was actually part of the CommonJS specification as another proposal). The acronym stands for Asynchronous Module Definition.

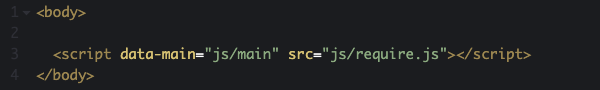
*The Asynchronous Module Definition (AMD) API specifies a mechanism for defining modules such that the module and its dependencies can be asynchronously loaded. This is particularly well suited for the browser environment where synchronous loading of modules incurs performance, usability, debugging, and cross-domain access problems.*

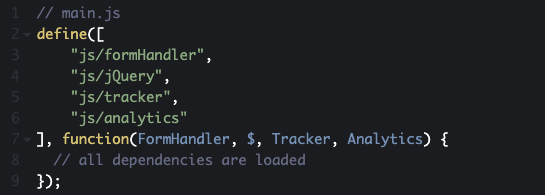
src: <https://github.com/amdjs/amdjs-api/blob/master/AMD.md>

****

AMD introduces the *define()* method, which allows your to define the dependencies the code relies on.

RequireJS is a JavaScript framework (not a standard) for managing your dependency order for you. It creates a single entry point to your main JavaScript file which calls the dependencies. RequireJS is based on AMD.





**Around 2011 (UMD introduced)**

**UMD**

UMD is a pattern of universal module definition for JavaScript modules. These modules are capable of working everywhere, be it in the client, on the server or elsewhere.

****

**Around 2015 (ES6 introduced)**

**ES6**

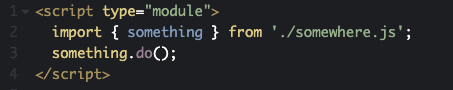
JavaScript has had modules for a long time at this point. However, they were implemented via libraries, and standards, and not built into the language.

After learning from CommonJS, AMD, and UMD, the TC39 committee introduced the biggest change to the language in 15 years which became ES6 or ECMAScript 5 or ECMAScript 2015. ES6 is the first time that JavaScript has built-in modules.

ES6 modules are pre-parsed in order to resolve further imports before code is executed.

**Using ES6 Modules in Browsers**

As of mid 2018, all major browsers support ES6 modules. You can use ES6 modules by using the type attribute and setting it to module.



**We we need Browserify, Webpack and Babel**

ES6

**Browserify (Released in 2011)**

Browserify was released as a tool to convert NodeJS scripts (using the CommonJS standard) into browser compatible window object globals.

**Babel (?)**

Babel transpiles old JavaScript code into browser compatible ES6 code.

**Webpack (Released in 2012)**

Webpack is, among other things, a dependency analyzer and module bundler. It uses loaders like Babel and Browserify when bundling and transpiling your code.

**Coding Examples**

**Overview**

Started January 2009 - CommonJS (Originally named ServerJS)

**Sources**

<https://medium.com/sungthecoder/javascript-module-module-loader-module-bundler-es6-module-confused-yet-6343510e7bde>

<https://ponyfoo.com/articles/brief-history-of-modularity>

<https://www.penta-code.com/how-javascript-modules-work-from-past-to-present/>

<https://javascript.info/modules-intro>

<https://objectpartners.com/2019/05/24/javascript-modules-a-brief-history/>

<https://addyosmani.com/writing-modular-js/>