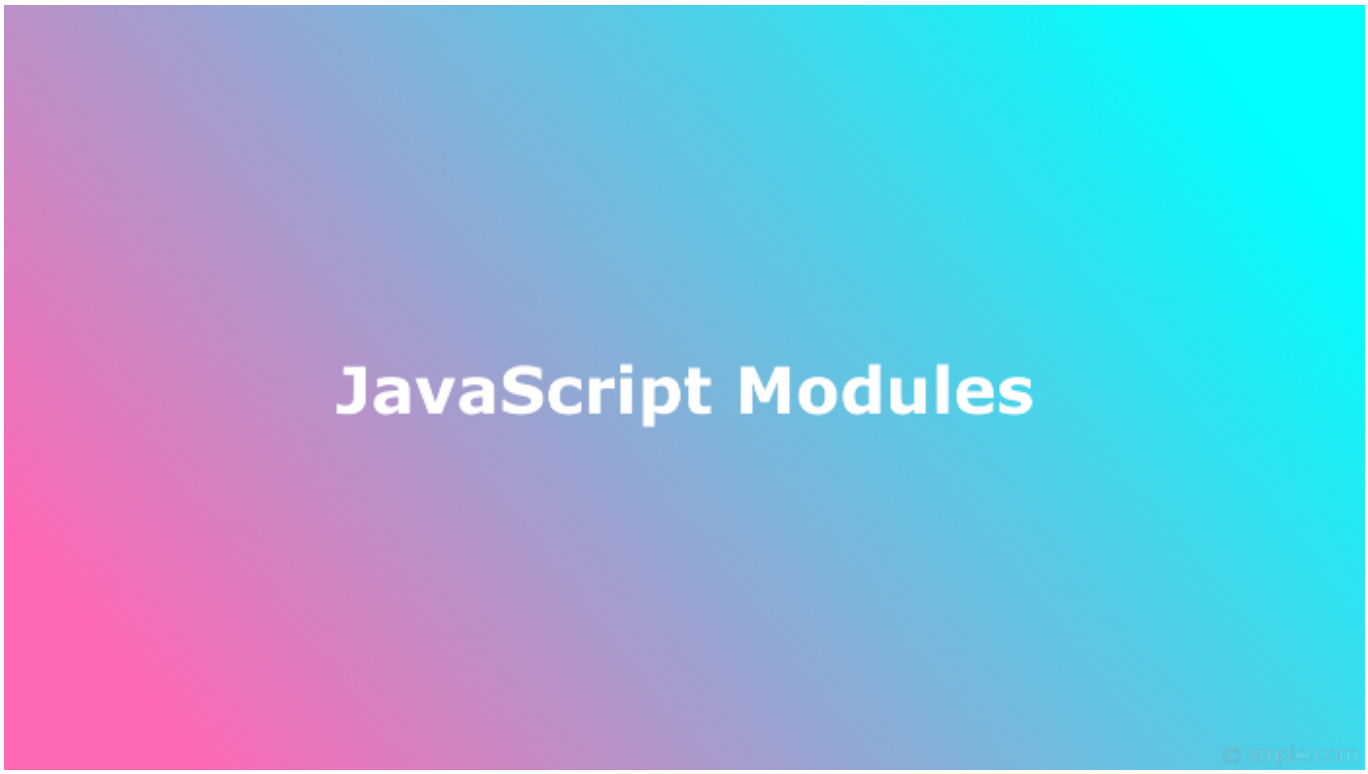
- — — — — CommonJS vs AMD vs RequireJS vs ES6 Modules — — — — -

Mohanesh Sridharan Follow
Sep 5, 2017 · 4 min read



Before i step into the modular section, kindly check out my unique comparison, Garbage Collection vs Automatic Reference Counting.

JavaScript Modules refer to a small units of independent, reusable code. They have distinct functionality, allowing them to be added, removed without disrupting the system. It seems to mimic how classes are used in Java or Python.

Modules are self-contained. Updating a module is much easier if it is decoupled from other pieces of code. This encourages the programmer to go through the program a lot

less intimidating. It solves the namespace ambiguity as well allowing the objects to be created in publicly accessible namespaces while the functions in it remain **private**. Modules can be **reused,** eliminating duplicate pieces of code thereby saving huge amount of time.
Before the modules arrived, The **Revealing Module Pattern** was getting used.

```
var revealingModule = (function () {
    var privateVar = "Ben Thomas";
    function setNameFn( strName ) {
        privateVar = strName;
    }
return {
        setName: setNameFn,
    };
})();

revealingModule.setName( "Paul Adams" );
```

In this program, the public functions are exposed while the private properties and methods are encapsulated.

Multiple modules can be defined in a single file but the downslides are that asynchronous loading of modules is not possible, cannot import modules programmatically.

CommonJS

They came up with a separate approach to interact with the module system using the keywords *require* and *exports*. require is a function used to import functions from another module. exports is an object where any function put into it will get exported.

```
//------ payments.js ------
var customerStore = require('store/customer'); // import module

//------ store/customer.js ------
exports = function(){
    return customers.get('store);
}
```

In the above example, the customerStore is imported to the payments.js. The function which is set to the exports object in customer module is loaded in payments file.

These modules are designer for server development and these are synchronous.ie., the files are loaded one by one in order inside the file.

NodeJS implementation

They are heavily influenced by CommonJS specification. The major difference arises in the exports object. NodeJS modules use *module.exports* as the object to get exported while CommonJS uses just the *exports* variable.

```
//payments.js
var customerStore = require('store/customer'); // import module

//store/customer.js
function customerStore(){
    return customers.get('store);
}
modules.exports = customerStore;
```

They are also synchronous in nature. The parameter passed to the *require* checks for the module name inside the *node_modules* directory. Circular dependencies are supported and a developer can easily understand the concepts. The cons are just one file per module, only objects are made as modules and the browsers cannot use these modules directly without transpiling.

But recently Browserify, used to bundle code from the modules, uses this method in the browser. Webpack also handles complex pipelines of source transformations which includes CommonJS modules.

Asynchronous Module Definition (AMD)

AMD was born as CommonJS wasn't suited for the browsers early on. As the name implies, it supports asynchronous module loading.

```
define(['module1', ',module2'], function(module1, module2) {
  console.log(module1.setName());
});
```

The function is called only when the requested modules are finished loading. The *define* function takes the first argument as an array of dependency modules. These modules are loaded in a non-blocking manner in the background and once the loading is completed, the callback function is executed.

It is designed to be used in browsers for better startup times and these modules can be objects, functions, constructors, strings, JSON, etc. Modules can be split in multiple files, which are compatible for *require* and *exports* and circular dependencies are supported as well.

RequireJS implements the AMD API. It loads the plain JavaScript files as well as modules by using plain script tags. It includes an optimizing tool which can be run while deploying our code for better performance.

```
<script data-main="scripts/main" src="scripts/require.js"></script>
```

This is the only code required to include files in RequireJS. The *data-main* attribute defines the initialization and it looks for scripts and dependencies.

As you probably noticed, none of the modules above were native to JavaScript. We tried to emulate a module system using the module pattern, CommonJS and AMD. Fortunately, ECMAScript 6 have introduced built-in modules which takes it through to the next and final section.

ECMAScript 6 modules (Native JavaScript)

ECMAScript 6 a.k.a., ES6 a.ka., ES2015 offers possibilities for importing and exporting modules compatible with both synchronous and asynchronous modes of operation.

```
//------ lib.js ------
export const sqrt = Math.sqrt;
```

```
export function square(x) {
    return x * x;
}
export function diag(x, y) {
    return sqrt(square(x) + square(y));
}

//------ main.js ------

import { square, diag } from 'lib';
console.log(square(11)); // 121
console.log(diag(4, 3)); // 5
```

The *import* statement is used to bring modules into the namespace. It is not dynamic, cannot be used anywhere in the file. This is in contrast with the require and define. Th *export* statement makes the elements public. This static behavior makes the static analyzers build the tree of dependencies while bundling the file without running code. This is used by modern JavaScript frameworks like ReactJS, EmberJS, etc. The drawback is that it isn't fully implemented in the browsers and it requires a transpiler like Babel to render in the unsupported browsers.

If you are looking at starting a new module or project, ES2015 is the right way to go and CommonJS/Node remains the choice for the server.

Thanks for reading.

If you enjoyed this article, feel free to hit that clap button 👏 to help others find it.

Follow me at https://medium.com/@mohanesh