JavaScript Modules

Steve Hartzog

What is a “module” in Java?

According to wikipedia, it is:

> “a distribution format for collections of Java code and associated resources. It also specifies a repository for storing these collections, or modules, and identifies how they can be discovered, loaded and checked for integrity.”

JavaScript !== Modules

The JavaScript language, formally known as EcmaScript, has no modules.

A discussion began on what a module should look like in JavaScript in January 2009[1].

EcmaScript 5 was released in December 2009 without them.

1. http://www.blueskyonmars.com/2009/01/29/what-server-side-javascript-needs/

JS Module Pattern History

January 2009: ServerJS group started, later became known as CommonJS in August 2009. Used in the very first release of Node.js in November 2009.

2010: AMD was one part of the CommonJS spec, known as Modules Transport/C. It was never released.

2011: RequireJS jumped the gun and released AMD, separate from the already evolved CommonJS module syntax being used in Node.js.

2013: CommonJS syntax finally comes to the browser.

JS Module Pattern

A pattern emerged. The “module” pattern, was part of several[1] and provided:

- ✓ Cleaner groupings of functions
- ✓ Cleaner global “namespace”[2]
- ✓ Support for dependency injection

1. See JavaScript Design Patterns, https://carldanley.com/javascript-design-patterns/.
2. There are no namespaces in JS. Move along.

JS Module Pattern vs Java Module

1. Not a distribution format
1. Not a repository
1. No definition of how it is discovered
1. No check for integrity
1. Requires a Script Loader

JS Module Patterns

CommonJS

Asynchronous Module Definition (AMD)

** ES6 Modules

AMD: What is it?

Designed for asynchronous loading & dependency management

Main use: intended for browsers

Part of an early CommonJS spec that was aborted, and Dojo's real world experience with XHR+eval

- Depends on **RequireJS**[1] (for browser support)

1. A library to help with the async request (XHR) and eval (JIT & execute) scripts not loaded in the DOM.

AMD: Module Syntax

```
// allows multiple defines per file
define('myModuleName', ['jquery', 'moment'], function(jquery, moment) {
          return { /* code here */ };
});



// or


// Simplified method; pass in require function, name is the filename it's in
define(function(require) {
          var $      = require('jquery')
              moment = require('moment');
          return { /* code here */ };
});
```

AMD: Built with RequireJS

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

// main.js
(function () {
          require.config({
                    name: ‘main’,
                    /* … */
                    paths: {
                              ‘jquery’: ‘tccom/assets/js/lib/jquery’,
                              ‘moment’: ‘sni-foundation/assets/js/vendor/moment.min’
                    }
          });
}());
```

AMD: But there are problems:

It’s fragile
- RequireJS is difficult to configure, tripping even experienced dev’s
- Circular dependencies are easy to run into
- Extremely difficult to troubleshoot

main.js needed to map paths

All files are async (mitigated by rjs optimizer)

Remember, it was only a first draft...

CommonJS: What is it?

Started as the ServerJS project
Designed for sync loading, but also supports async
Simple syntax
Main use: server

- Browserify formally brought CommonJS to the browser in 2011

CommonJS: Module Syntax

```
// Simple syntax
var $      = require('jquery')
    moment = require('moment');

/* code here */

exports.nameToExport = myFunctionName;
```

CommonJS: Built with Browserify

No config / data-main mapping file necessary.

All modules are minified into a single script at build.

However, bundles of 3rd party modules can be required as needed… with browserify-shim[1].

1. Of course, you must have a script tag for it, preferably pointing to a public CDN.

AMD

```
// Simple method; pass in require function,
name is the filename it's in
define(function(require) {
          var $      = require('jquery')
              moment = require('moment');
          return { /* code here */ };
});
```

CommonJS

```
// Even simpler
var $      = require('jquery')
    moment = require('moment');

/* code here */

exports.nameToExport = myFunctionName;
```

CommonJS: Bundle Creation

Every time your modules change, you must build a bundle (sourcemaps supported).

For us, this would be watched and automated in our build.

What about ES6?

So ES6, does have a module standard[1]...

and surprise, it looks almost exactly like CommonJS!

ES6 to be released June 2015.

1. http://www.2ality.com/2014/09/es6-modules-final.html

Recap

What do they look like again?

CommonJS

```
// Even simpler
var $      = require('jquery')
    moment = require('moment');

/* code here */

exports.nameToExport = myFunctionName;
```

ES6

```
// Best syntax
import $ from 'jquery';
import moment from 'moment';

/* code here */

exports.nameToExport = myFunctionName;
```

CommonJS

```
// Even simpler
var $      = require('jquery')
    moment = require('moment');

/* code here */

exports.nameToExport = myFunctionName;
```

- ✓ Simpler
- ✓ Cleaner
- ✓ Easier to update & maintain

Our AMD

```
// Simple method; pass in require function,
name is the filename it's in
define(function(require) {
          var $      = require('jquery')
              moment = require('moment');
          return { /* code here */ };
});
```

- ✗ Wrappers introduce complexity
- ✗ Separate mapping file needed
- ✗ Difficult to troubleshoot when it breaks

Conclusion

1. AMD is going away.

1. Baby steps. Prepare for ES6 with CommonJS.

1. Conversion is simple.

Questions?