Notes - JavaScript - ES Modules - Lib Structure

• Dr Nick Hayward

A brief outline of using JavaScript's ES Modules to implement a custom module library.

Contents

- Intro
- Spire JS library
 - main.js
 - basic usage
 - module usage
 - spire.js import modules

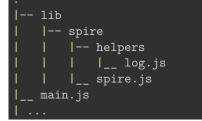
Intro Modules in JavaScript are not a new concept. For example, CommonJS is a popular option for modular development with Node.js.

However, with the introducion of ES2015 (ES6), we now have a built-in option for plain JavaScript, ES Modules.

We may also use this module structure to develop and structure custom module libraries.

For example, we might abstract utility modules, such as loggers, to a custom library for easy re-use in future projects.

Spire JS library For an example JS library, we may define the following directory structure



The lib directory contains the custom JS libraries, which may then be imported for use within an app.

For app usage, we might structure it as follows

```
|-- lib
| |-- spire
| | |-- helpers
| | | |__ log.js
| | |__ spire.js
|__ index.html
|__ main.js
|
```

main.js The **main.js** file is loaded from the **index.html** file, and acts as the loader file for JS in an example app.

We may also import the example Spire JS library into this app using this main loader file.

e.g.

import Spire from './lib/spire/spire.js';

The Spire object is the access point to the exported methods and variables for the custom JS library.

basic usage A custom JS library may then be accessed using this Spire object.

For example, we might call a method from the library,

const greeting = 'greetings from the planet Earth';
// basic log to console
Spire.log(`\${greeting}...we wish you well`);

The custom method log() provides a reusable method for various logging options in the application.

We might also call the following method using the same pattern,

```
Spire.dir({'name': 'test dir logger...'});
```

module usage A sample usage might include the above *helpers*, which we may package in the directory spire/helpers/.

For example, we currently have a log.js module for various custom loggers.

```
// basic logger to console
function log(value, ...values) {
   const logValue = console.log(value, ...values);
   return logValue;
}
// directory logger to console
function dir(value, ...values) {
   const dirValue = console.dir(value, ...values);
   return dirValue;
}
```

We may then simply export these methods from the log.js module, e.g.

<pre>export {</pre>			
log, dir			
dir			
}			

So, the interface for this module has now been defined relative to the above exported modules.

spire.js - import modules To allow a module to use these exported methods, and interact with the exposed interface, we may then import this module.

As part of the JS library structure, we may define a root module for organising a unified interface for the overall library.

In this example, we use the module **spire.js** to import the required modules and their interfaces

e.g.

```
import * as loggers from './helpers/log.js';
```

We may then define a Spire object for the overall library, e.g.

```
const Spire = {
    log: loggers.log,
    dir: loggers.dir,
```

This is then exported as the general interface for the Spire JS library,

export default Spire;