JavaScript module formats

1.

Importance of modularity

Rationale

Loose coupling

Loose coupling facilitates easier maintainability of apps by removing dependencies where possible.

When this is implemented efficiently, it’s quite easy to see how changes to one part of a system may affect another.

Performance

In the code below, loading of script **ac.js** is blocked by scripts **b.js** and **ab.js**.

Are these 2 scripts really needed for **ac.js** to work?

```
<!DOCTYPE html>
<html>
    <head>
        <script src="a.js"></script>
        <script src="b.js"></script>
        <script src="ab.js"></script>
        <script src="c.js"></script>
        <script src="ac.js"></script>
        <script src="abc.js"></script>
    </head>
    <body></body>
</html>
```

Global variables

The only way to share variables between **`a.js`** and **`c.js`** is making use of global variables.

When number of scripts grows, keeping track of these globals quickly becomes a pain.

```
<!DOCTYPE html>
<html>
    <head>
        <script src="a.js"></script>
        <script src="b.js"></script>
        <script src="ab.js"></script>
        <script src="c.js"></script>
        <script src="abc.js"></script>
    </head>
    <body></body>
</html>
```

2.

CommonJS

http://wiki.commonjs.org/wiki/Modules/1.1

An API for defining modules either server or client-side

math.js

```
exports.add = function() {
    var sum = 0, i = 0, args = arguments, l = args.length;
    while (i < l) {
        sum += args[i++];
    }
    return sum;
};
```

increment.js

```
var add = require('math').add;
exports.increment = function(val) {
    return add(val, 1);
};
```

program.js

```
var inc = require('increment').increment;
var a = 1;
inc(a); // 2
```

3.

AMD

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

AMD - 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.

“

AMD is particularly well suited for the browser environment where synchronous loading of modules incurs performance, usability, debugging, and cross-domain access problems.

API Specification

https://github.com/amdjs/amdjs-api/blob/master/AMD.md#api-specification

define() function

The specification defines a single function "**define**" that is available as a free variable or a global variable.

The signature of the function:

```
define(id?, dependencies?, factory);
```

{String} [id]

Optional. When present, works as module name.

Best practice is to always omit that arguments.

Module id format

- A module identifier is a String of "terms" delimited by forward slashes.
- A term must be a camelCase identifier, ".", or "..".
- Module identifiers may not have file-name extensions like ".js".
- Module identifiers may be "relative" or "top-level". A module identifier is "relative" if the first term is "." or "..".
- Top-level identifiers are resolved off the conceptual module name space root.
- Relative identifiers are resolved relative to the identifier of the module in which "require" is written and called.

Relative module ID resolution examples:

- if module "a/b/c" asks for "../d", that resolves to "a/d"
- if module "a/b/c" asks for "./e", that resolves to "a/b/e"

{String[]} [dependencies]

Optional. An array of the module ids that are dependencies required by the module that is being defined.

The dependencies ids may be relative ids, and should be resolved relative to the module being defined.

Specification defines three special dependency names that have a distinct resolution.

If the value of "**`require`**", "**`exports`**", or "**`module`**" appear in the dependency list, the argument should be resolved to the corresponding free variable as defined by the CommonJS modules specification.

{Function|Object} factory

Is a function that should be executed to instantiate the module or an object.

If the factory is a function it should only be executed once.

If the factory function returns a value (an object, function, or any value that coerces to true), then that value should be assigned as the exported value for the module.

If the factory argument is an object, that object should be assigned as the exported value of the module.

require(String)

Synchronously returns the module export for the module ID represented by the String argument.

It MUST throw an error if the module has not already been loaded and evaluated. In particular, the synchronous call to require MUST NOT try to dynamically fetch the module if it is not already loaded, but throw an error instead.

require(Array, Function)

The Array is an array of String module IDs. The modules that are represented by the module IDs should be retrieved and once all the modules for those module IDs are available, the Function callback is called, passing the modules in the same order as the their IDs in the Array argument.

Example:

```
define(function (require) {
        require(['a', 'b'], function (a, b) {
            //modules a and b are now available for use.
        });
    });
```

4.

RequireJS

http://requirejs.org

A JavaScript file and module loader

Usage

Let’s consider the following project structure:

```
project-directory/
    project.html
    scripts/
        main.js
        require.js
        helper/
            util.js
```

Usage

```
<!DOCTYPE html>
<html>
    <head>
        <title>My Sample Project</title>
        <!-- data-main attribute tells require.js to load
             scripts/main.js after require.js loads. -->
        <script data-main="scripts/main" src="scripts/require.js"></script>
    </head>
    <body>
        <h1>My Sample Project</h1>
    </body>
</html>
```

scripts/main.js

```
require(["helper/util"], function(util) {
    //This function is called when scripts/helper/util.js is loaded.
    //If util.js calls define(), then this function is not fired until
    //util's dependencies have loaded, and the util argument will hold
    //the module value for "helper/util".
});
```

Configuration

```
<script src="scripts/require.js"></script>
<script>
requirejs.config({
    //By default load any module IDs from js/lib
    baseUrl: 'js/lib',
    //except, if the module ID starts with "app", load it from the js/app directory.
    //Paths config is relative to the baseUrl.
    paths: {
        app: '../app'
    }
});
</script>

// Start the main app logic.
requirejs(
    ['jquery', 'app/sub'],
    function($, sub) {
        //jQuery and the app/sub module are all loaded and can be used here now.
    }
);
```

Configuration. Shim.

```
<script src="scripts/require.js"></script>
<script>
requirejs.config({
    //Remember: only use shim config for non-AMD scripts,
    //scripts that do not already call define().
    shim: {
        'backbone': {
            //These script dependencies should be loaded before loading backbone.js
            deps: ['underscore', 'jquery'],
            //Once loaded, use the global 'Backbone' as the module value.
            exports: 'Backbone'
        },
        'underscore': {
            exports: '_'
        }
    }
});
</script>
```

Further reading

1. http://addyosmani.com/writing-modular-js/
2. http://wiki.commonjs.org/wiki/Modules/1.1
3. https://github.com/amdjs/amdjs-api
    a. https://github.com/amdjs/amdjs-api/blob/master/AMD.md
    b. https://github.com/amdjs/amdjs-api/blob/master/require.md
4. http://requirejs.org/docs/api.html
5. Single-page RequireJS app sample: https://github.com/volojs/create-template

CREDITS

Special thanks to all the people who made and released these awesome resources for free:

- Simple line icons by Mirko Monti
- E-commerce icons by Virgil Pana
- Streamline iconset by Webalys
- Presentation template by SlidesCarnival