AMD Modules

Some old notes on AMD modules, kept here for posterity.

AMD Modules

AMD stands for *Asynchronous Module Definition*. It is a specification[1] born out of a need to have modular development in a project that is meant to be deployed in the browser. As scripts in the browser are simply concatenated together in one common environment, the CommonJS style described above would not work.

The AMD specification consists of a number of parts:

AMD Modules You write your module files in a specific format, with the use of the define function that we will discuss shortly. Part of that specification is what other modules your module depends on.

AMD Modules cannot be inserted into a webpage directly. They need the use of a loader or builder.

The key features of these modules is that they have explicitly declared dependencies, and that they can be loaded asynchronously: You specify the file's dependencies, along with how your module will finish its loading once those dependencies have been loaded.

AMD Loader The loader is responsible for loading the modules in the correct order. It has to provide a "define" function, and it processes the information provided in those "define" calls to determine the correct order in which modules should be loaded to resolve the dependencies.

There are a number of existing loaders, including require.js[2] and RaveJS[3]. We will spend more time looking closer at require.js in future segments.

AMD Builder There are various programs whose goal is to build/consolidate the various AMD modules into one file to be served in a <script> tag. Loaders are used in the development of the application, while builders are used in the deployment phase. require.js[4] includes such a builder/optimizer, but there are many others, for instance browserify[5].

These programs often also do compression, removing spaces, comments etc, to reduce the file size and make it faster to download.

The key component of and AMD module is the define function. It takes up to three arguments:

[1] https://github.com/amdjs/amdjs-api/blob/master/AMD.html
[2] http://requirejs.org/
[3] https://github.com/RaveJS
[4] http://requirejs.org/
[5] http://browserify.org/

- id is the first argument, and it is optional. It is a string characterizing the module's "identifier". This will default to the filename if omitted, which is typically the case.

- dependencies is the second argument, also optional but usually included. It is an *array* of the "id"s of modules that your module depends on. Those modules will be processed first before your module is processed. There are three special id names that are treated separately: "require", "exports" and "module". If those ids appear, they are resolved to their CommonJS module meaning.

  If this second argument is omitted, then it defaults to the triple ["require", "exports", "module"].

  One key difference with the ids in the AMD specification is that they are looked for either relative to the current file or from "top-level". There is nothing analogous to the "node_modules" folders.

- factory is the third, and only required, argument. It can be an object, in which case it is what is exported by this module. Or more typically it is a function, which will be executed exactly once, and its return value is the exported object from the module. This function will receive as arguments the modules that were listed in the dependencies array.

- For a proper AMD implementation, the define function has a property, define.amd, which must be an object but has no other required fields.

Here is an example of how such a file might look like. In its simplest form it has no dependencies:

```
define(function() {
    var OurModule;
    // ... define OurModule here
    return OurModule;
});
```

Or if we wanted to give it a specific name rather than have it obtain its name from the filename:

```
define('ourModuleName', function() {
    var OurModule;
    // ... define OurModule here
    return OurModule;
});
```

Here's an example of a module that depends on jQuery and one other module:

```
define(["jquery", "otherModule"], function($, otherM) {
    // "$" here will equal the jquery object
    // and otherM equal the "otherModule"
    // ...
    return MyModule;
});
```

We can easily transform a CommonJS/Node module into an AMD module by using the simplified CommonJS wrapping. For instance here is how our TaskListController file might look:

```
define(function(require, exports, module) {
    var Template, Task, TaskListController;

    Template = require('Template');
    Task = require('Task');

    // Write TaskListController here

    // Export at the end
    module.exports = TaskListController;
    // AMD return
    return TaskListController;
});
```

A sample of using AMD modules can be found here[6]. Start by looking inside the js/app folder and the bottom of the index-test.html file for bootstrapping such an application. You will find there the following line:

```
<script type="text/javascript" data-main="js/app/main" src="js/lib/require.js"></script>
```

This line loads the require.js file, which then takes care of loading the application by starting at the "js/app/main" module and following its dependencies.

[6] https://github.com/skiadas/HealCalc3/tree/master/