RequireJS: JavaScript Dependency Injection and Module Loading

**By Jeff Valore**

1. **Getting Started**

RequireJS is a JavaScript file and module loader. It is optimized for in-browser use, but it can be used in other JavaScript environments, like Rhino and [Node](http://requirejs.org/docs/node.html). Using a modular script loader like RequireJS will improve the speed and quality of your code.

Prerequisite: need to down load the require.js file and include I your project.

Mention the **main.js** (main entry point for our application) for **data-main** property in the **script tag**.

**then require.js will initiate for load main.js file**

**2. Defining and Requiring modules**

* **Using External module as dependency:**

**Config section**

require.config({

paths:{

jquery:"jquery-2.1.1.min"

}

});

**Require function**

Require function is used to load the initial set of modules.

Each module is identified as string

require({dependency array}, call back function);

require(["jquery"], function($){);

* **Asynchronous Module dependency(AMD)**

**Asynchronous module definition** (**AMD**) is a [JavaScript](https://en.wikipedia.org/wiki/JavaScript) specification that defines an [API](https://en.wikipedia.org/wiki/Application_programming_interface) for defining code modules and their dependencies, and loading them asynchronously if desired. Implementations of AMD provide the following benefits:

* Website performance improvements. AMD implementations load smaller JavaScript files, and only load them when they are needed.
* Fewer page errors. AMD implementations allow developers to define dependencies that must load before a module is executed, so the module does not try to use outside code that is not yet available.

AMD modules can be loaded by any AMD module loader

* Require.js
* Curl.js
* **Defining AMD module**

**Syntax**: We can use either **self executing function** or **require’s define function**

* + 1. **Self executing function**

**var myModule = (function(){..})();**

* + 1. **require’s define function**

**2.1 When define a module inline with other code**

**define({modulenameString}, {dependencyArray }, callback function );**

**define('taskData', [] , function() {} );**

**2.2 When define module in a separate file**

**define( {dependencyArray }, callback function );**

**define( [] , function() {} );**

**2.3 Simplified commonJS Wrapper**

**define({moduleNameString}, function(require, export, module) {**

**var {module} = require({moduleName});**

**export.{publicMember} = {publicValue};**

**})**

**Design patterns for AMD**: We can use either **Module Pattern** or **Revealing Module Pattern**

1. **Module Pattern**: define only private members at top and define only public embers in return statement.

**var myModule = (function(){**

**var iamPrivate =”private”;**

**return{**

**iAmPublic : “public”**

**}**

**})();**

1. **Revealing Module Pattern**: define all members at top and return only public methods.

**var myModule = (function(){**

**var iamPrivate =”private”;**

**var iamPublic =”public”;**

**return{**

**iAmPublic : iamPublic**

**}**

**})();**

Generally we follow the revealing Design pattern.

1. **Loading Remote Modules**

**From where it will load**

When reuireJS needs to load module

First it will check cache for module

if It exist then load from module

If not not exist then it will load from server and cache it.

While loading from the server it will use module name as url notation (**basepath+moduleName+”.js”)**

**Note**: here base path is the path where the main module is existing.

**Note**: when we define this module in separate file then we should not mention the module name, we should keep the file name as module name.

1. **Optimization**

* **Introduction**
* When we organize files into subdirectories then we face the performance problem, requireJS makes separate calls to each file.
* To eliminate this we can use the **r.js optimizer**
* Just it will take all the individual files and make it one minimized file.
* **r.js runs** in Node.js
* **Build Environment and Running r.js**

**Step1:** We need to run r.js using node and mention few options

**node build\r.js**

**-o**

**name=main (**this the main module where r.js will start here and optimize all the modules which are pointinghere**)**

**baseUrl=src/js**

**mainConfigFile=src/js/main.js**

**out=src/js/main-optimized.min.js (**output file name**)**

**optimize=none (**if we don’t want minified file **)**

**Step2:** In index.html we need to mention this minified file

* **Optimization Result**

It produces the minified file, if we want we can turn off the optimization.

* **Debugging and source Maps**

The problem with minified file is we cannot debug this.

We can solve this in 2 ways

1. Use un optimized files in development (main.js) and optimized file in prod (main-optimized.min).
2. Use Source Maps

This files are secondary files, deployed along with your optimized files these tells to your browser where this lines of java script originally located before optimization. **generateSourceMaps=true**

**Note:** for generating the sourceMap we need to set the **preserveLicenseComments=false**

**uglify1 is not supported for sourceMap, if you are using old r.js the we need to mention the optimize= uglify2**

* **Build Profiles**

For using the r.js we need to provide lot of options,

We can create one file and mention all the options and we can give this file name to r.js in command line.

**node build\r.js -o build\build.config.js**

1. **Configuration Options**

We can mention require.config in index.html or in the beginning of the main module.

* **baseUrl**

1. root path to use for all module lookups
2. if **data-main** attribute is used, base url set to the path of the file specified by data-main

* **paths:**

This are all key value pair object.

While module lookup (search with given name) this value will override

* **shim**

If we depend on any third party library that does not register self with reuire.js on its own. We should mention only non AMD modules.

**export**: it will bring the global variable

Example: if we mention underscore script tag in browser the it will set “\_” symbol in global level for that browser. So if we mention the “\_” symbol in the export the require js brings that symbol from windows

**require([“underscore”], function(\_){ …});**

**deps**: the dependency modules

**shim: {**

**'underscore': {**

**exports: '\_'**

**}**

**}**

* **waitSeconds**

Number of seconds to wait for a module to finish loading before throwing an error.

* **deps**

before reuire.js load these modules will load

* **callback**

after dependencies load this function will call.

* **urlArgs**

Value which is mentioned here will append as query string to the module request.

1. **Plugins**
2. **Unit Testing Require JS Module**

Modules and Require JS is use full

Difficulties of testing with Require JS

Module loading is asynchronous, so we need to use asynchronous test frame work for testing

Modules are cached.