LEARNING requirejs

#requirejs

Table of Contents

About

Chapter 1: Getting started with requirejs

Remarks

RequireJS is an implementation of the Asynchronous Module (AMD) API used to load javascript files asynchronously (i.e. without blocking) and manage dependencies between multiple javascript files. It also includes an optimization tool for combining and minimizing script files for deployment while allowing the developer to maintain logically separate code.

Examples

Hello World

The following example will demonstrate a basic installation and setup of RequireJS.

Create a new HTML file called `index.html` and paste the following content:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Hello RequireJS</title>
    <script type="text/javascript"
src="http://requirejs.org/docs/release/2.3.2/minified/require.js"></script>
</head>
<body>
    <!-- place content here --->
    <script>
        requirejs(["scripts/say"], function(say) {
            alert(say.hello());
        });
    </script>
</body>
```

Create a new JS file at `scripts/say.js` and paste the following content:

```
define([], function(){
    return {
        hello: function(){
           return "Hello World";
        }
    };
});
```

Your project structure should look like this:

```
- project
    - index.html
    - scripts
        - say.js
```

Open the `index.html` file in a browser and it will alert you with 'Hello World'.

Explanation

1. Load the require.js script file.

   ```
   <script type="text/javascript"
   src="http://requirejs.org/docs/release/2.3.2/minified/require.js"></script>
   ```

2. Load the `say` module asynchronously from the `scripts` folder. *(Note that you do not need the `.js` extension when referencing module.)* The returned module is then passed to the provided function where `hello()` is invoked.

   ```
   <script>
      requirejs(["scripts/say"], function(say) {
          alert(say.hello());
      });
   </script>
   ```

3. The `say` module returns a single object with one function `hello` defined.

   ```
   define([], function(){
       return {
           hello: function(){
              return "Hello World";
           }
       };
   });
   ```

Hello World with Nested Dependencies

The following example expands upon `Hello World` by demonstrating multiple dependencies using the `define()` function.

Create a new HTML file called `index.html` and paste the following content:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Hello RequireJS</title>
    <script type="text/javascript"
src="http://requirejs.org/docs/release/2.3.2/minified/require.js"></script>
</head>
<body>
    <!-- place content here --->
    <script>
        requirejs(["scripts/say"], function(say) {
            console.log(say.hello("english"));
            console.log(say.hello("spanish"));
            console.log(say.hello("french"));
        });
    </script>
```

```
</body>
```

Create a new JS file at `scripts/say.js` and paste the following content:

```
define(["scripts/translations"], function(translations){
    return {
        hello: function(language){
           return translations[language].hello + " " + translations[language].world;
        }
    };
});
```

Create a new JS file at `scripts/translations.js` and paste the following content:

```
define([], function(){
    return {
        "english": {
            hello: "Hello",
            world: "World"
        },
        "spanish": {
            hello: "Hola",
            world: "Mundo"
        },
        "french": {
            hello: "Bonjour",
            world: "Le Monde"
        }
    };
});
```

Your project structure should look like this:

```
- project
    - index.html
    - scripts
        - say.js
        - translations.js
```

Open the `index.html` file in a browser and the console will output:

```
Hello World
Hola Mundo
Bonjour Le Monde
```

Explanation

1. Load the require.js script file.

   ```
   <script type="text/javascript"
   src="http://requirejs.org/docs/release/2.3.2/minified/require.js"></script>
   ```

2. Load the `say` module asynchronously from the `scripts` folder. *(Note that you do not need the `.js` extension when referencing module.)* The returned module is then passed to the provided function where `hello()` is invoked.

```
<script>
     requirejs(["scripts/say"], function(say) {
         console.log(say.hello("english"));
         console.log(say.hello("spanish"));
         console.log(say.hello("french"));
     });
</script>
```

3. The `say` module returns a single object with one function `hello` defined, but in this case we have defined a dependency (`scripts/translations`) and we will pull the translations from there.

```
define(["scripts/translations"], function(translations){
    return {
        hello: function(language){
           return translations[language].hello + " " + translations[language].world;
        }
    };
});
```

4. The `translations` module simply returns an object with various word translations.

```
define([], function(){
    return {
        "english": {
            hello: "Hello",
            world: "World"
        },
        "spanish": {
            hello: "Hola",
            world: "Mundo"
        },
        "french": {
            hello: "Bonjour",
            world: "Le Monde"
        }
    };
});
```

Using data-main Entry Point

A single entry point to your application is possible with RequireJS by using the `data-main` attributed within the `<script>` tag.

```
<script type="text/javascript" data-main="scripts/main"
src="http://requirejs.org/docs/release/2.3.2/minified/require.js"></script>
```

On load, RequireJS will look for the `data-main` attribute and inject the main script tag into the DOM with the `async` attribute set. It is this file that you will want to do any configuration before kicking off your application.

For example:

```
// contents of scripts/main.js
require.config({
    waitSeconds: 10,
    paths: {
        jquery: 'libs/jquery-1.4.2.min'
    }
});

requirejs(["jquery", "libs/say"], function($, say) {
    var $body = $('body');
    $body.append( $('<p/>').text(say.hello("english")) );
    $body.append( $('<p/>').text(say.hello("spanish")) );
    $body.append( $('<p/>').text(say.hello("french")) );
});
```

Incorporating Non-AMD Libraries

Not all libraries are defined in a way that is compatible with AMD and RequireJS's `define()` function. The author's have addressed this by including a `shim` directive for configuring those dependencies.

One example is using the jQuery UI Layout Plugin. That plugin depends on jQuery. You can configure it like this:

```
requirejs.config({
    paths: {
        'jquery': '../path/to/jquery.min',
        'jquery.layout': '../path/to/jquery.layout.min'
    },
    shim: {
        'jquery.layout': {
            deps: ['jquery']
        }
    }
});
```

And then use it in a layout module like this:

```
define(['jquery', 'jquery.layout'], function ($, layout) {
    $('body').layout();
});
```

Read Getting started with requirejs online: https://riptutorial.com/requirejs/topic/8791/getting-started-with-requirejs

Credits

S. No	Chapters	Contributors
1	Getting started with requirejs	Community, Michael Sacket