Contents

[WEBPACK 2](#_Toc189508768)

[MODULES IN JAVASCRIPT 2](#_Toc189508769)

[MODULE FORMATS/SYSTEM 3](#_Toc189508770)

[IMPLEMENTING COMMON JS MODULE FORMATS 3](#_Toc189508771)

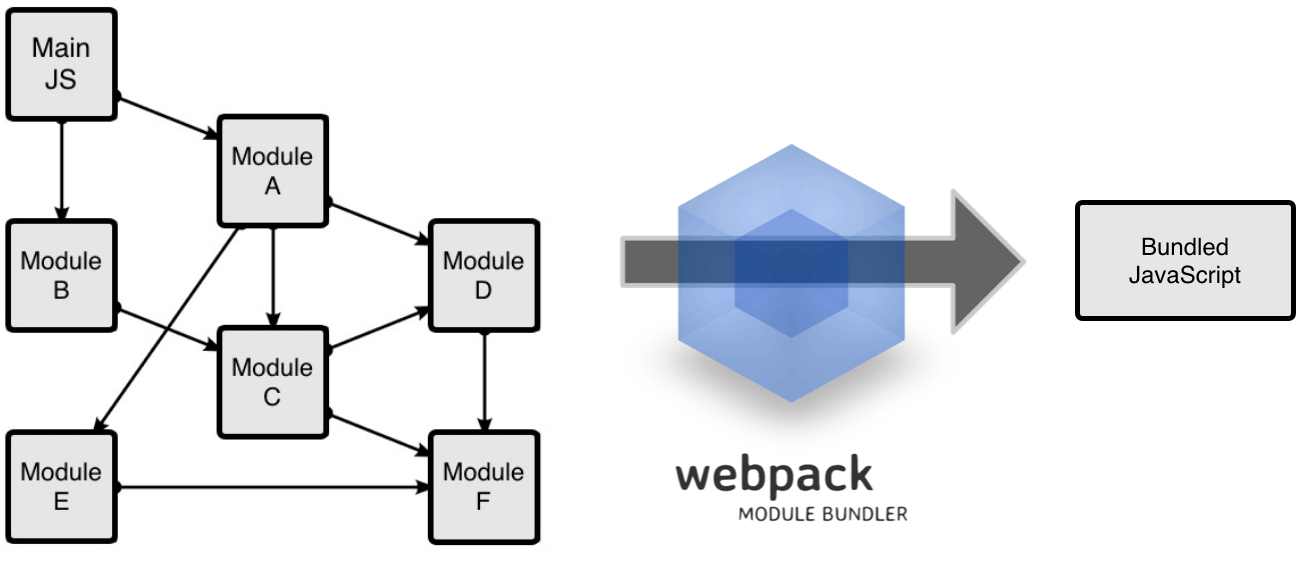
[INSTALLING AND CONFIGURING WEBPACK 4](#_Toc189508772)

[INSTALLING AND CONFIGURING WEBPACK 4](#_Toc189508773)

[MODULE LOADERS IN WEBPACK 5](#_Toc189508774)

# WEBPACK

* WEBPACK is a module web bundler.
* The webpack becomes more important for SPA’s because- This need a huge amount of Javascript code , which is responsible for dynamically showing up the HTML/Content on client browser unlike server side rendering(HTML is rendered by the server itself)



## MODULES IN JAVASCRIPT

* A module is just a file. One script can be one module.
* For large application – we logically divide the application based on functionality. Those logical divisions are called Modules.
* ***A module is a reusable piece of code that encapsulates implementation details and exposes a public API so it can be easily loaded and used by other code.***

**WHY MODULES?**

* **ABSTRACTION** : To delegate functionality to specialized libraries so that we don't have to understand the complexity of their actual implementation
* **ENCAPSULATION: T**o hide code inside the module if we don't want the code to be changed
* **REUSABILITY**: To avoid writing the same code over and over again
* **DEPENDENCY MANAGEMENT**: To easily change dependencies without rewriting our code

**PROBLEMS WITH MODULES**

|  |  |  |
| --- | --- | --- |
|  |  | * When we break the Javascript in modules we need to load the dependent modules should be load prior .So order of loading the modules on the page is important * **WEBPACK helps us such kind of dependency management among the modules** |
| ***Along with dependency management - Webpack also bundle all the modules together into a single file hence, minimize the number of network requests.*** |

**WEBPACK IN ACTION**

To understand webpack let build a small project to leverage webpack. We will follow below steps to do so

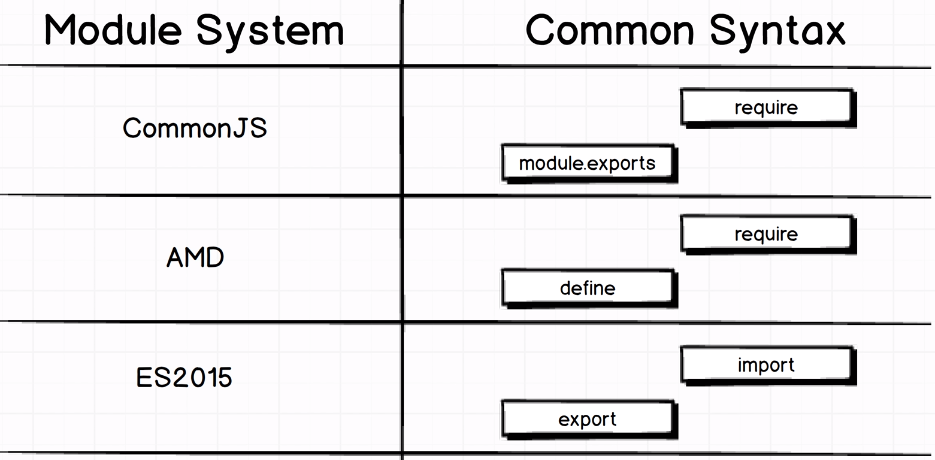
1. **Create new NPM project**
2. **Create 2 JS modules**
3. **Install and configure webpack**
4. **Run webpack**

|  |  |  |
| --- | --- | --- |
| **STEP 1: CREATING NPM PROJECT** | **npm init**  This will create a npm project with package.json file. This file is used for dependency management and configure sone build script | |
| **STEP 2: CREATING JS MODULES** |  | Here index.js is a dependent file which will call the function fron sum.js |
|  |

**After we create modules (as above) we need to link them up. So the way we can link all the modules we need to follow some pattern, which is know as MODULE FORMAT or MODULE SYSTEM**

### MODULE FORMATS/SYSTEM

* IN ES5 we didn’t have concept of modules, so developer came up with different approaches to modularize the code. The pattern/syntaxes to define a module are called Module Formats. The popular module formats used in ES5 are
  + **AMD**
    - Asynchronous Module Definition.
    - Loads the modules asynchronously
  + **COMMON JS**
    - This module system is used by Node JS
  + **ES 2015** – Universal Module definition – Used in Browsers and Node JS



### IMPLEMENTING COMMON JS MODULE FORMATS

|  |  |  |
| --- | --- | --- |
| sum.js (Module 1) | Index.js(Module 2) | |
| const sum = (a, b) => a + b;  **module.exports = sum;** | **const sum = require("./sum");**  const total = sum(10, 5);  console.log(total); | * All the export statement go at the bottom of the module * The “require” statement go at the top of the module * **The “require” function always takes the relative path of the module which it is importing** |

### INSTALLING AND CONFIGURING WEBPACK

|  |  |
| --- | --- |
| **INSTALLING WEBPACK (2.0) AS DEV DEPENDENCIES** | **npm install --save-dev webpack** |
| This will add web pack as project dependencies in package.json file | {  "name": "webpack-exercise",  "version": "1.0.0",  "description": "",  "main": "index.js",  "scripts": {  "test": "echo \"Error: no test specified\" && exit 1"  },  "author": "",  "license": "ISC",  **"devDependencies": {**  **"webpack": "^2.7.0"**  **}**  } |
| **CREATING WEBPACK CONFIG FILE**   * To know the webpack know regarding the bundling, entry point etc..configuration – we need to create a file names ”**webpack.config.js**” * This is kind of instruction file for webpack. |  |
| **webpack.config.js**  const path = require("path");  const config = {  **entry: "./src/index.js",**  output: {  **path: path.resolve(\_\_dirname, "build"),**  **filename: "bundle.js"**  }  };  module.exports = config; | 1. The entry property give the entry point JS file. We need to provide the relative path (from project directory) of the script file 2. Output property – This property will help us to configure the output file 3. **Path property** – To configure path of the o/p file . 4. This is the absolute path .We are using node to get the absolute path of the project(path.resolve()) 5. “build” is the name of the parent folder where “**bundle.js**” will be created 6. **\_\_dirname is a node variable which is the path of current working directory.** |

### INSTALLING AND CONFIGURING WEBPACK

|  |  |  |
| --- | --- | --- |
| UPDATE PACKAGE.JSON | RUNNING WEBPACK | INCLUDING THE FILE IN HTML |
| {  "name": "webpack-exercise",  "version": "1.0.0",  "description": "",  "main": "index.js",  **"scripts": {**  **"build": "webpack"**  **},**  "author": "",  "license": "ISC",  "devDependencies": {  "webpack": "^2.2.0-rc.0"  }  } | **npm run build**  This will create a bundle.js file in a bundle folder | <html class="no-js">  <head></head>  <body>  **<script src="./build/bundle.js"></script>**  </body>  </html> |
|  |

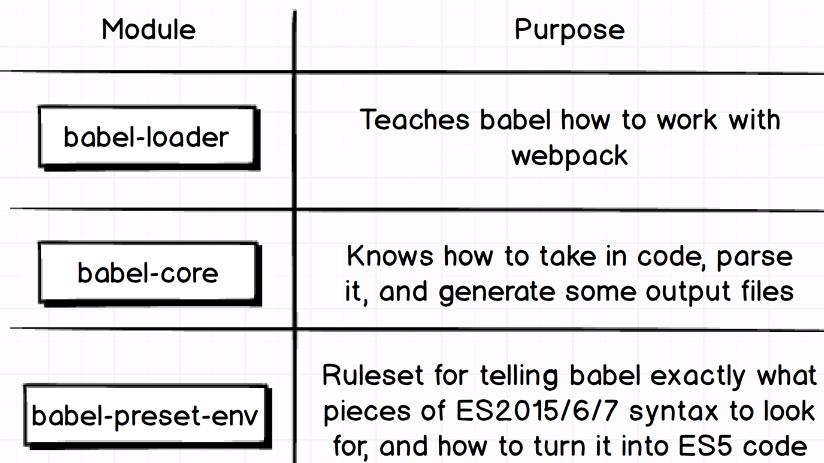
### MODULE LOADERS IN WEBPACK

* **Loader are used to do some pre-processing before bundling the code**
* Preprocessing like transpiling ES6 code(Babel Loader), preprocessing CSS and images

#### BABEL LOADER

**Babel loader transpiles the ES6 code to ES5 code**

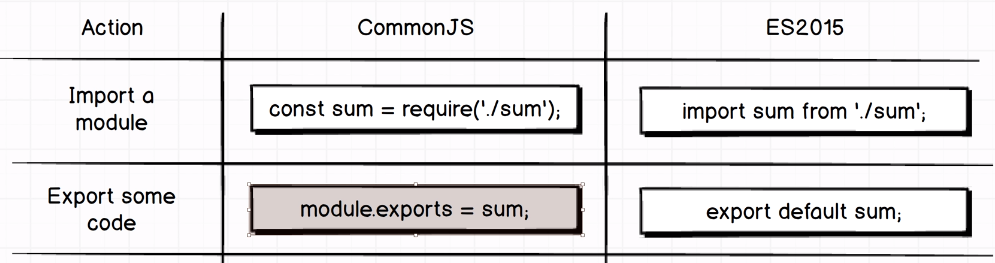
1. Babel Loader – Babel support multiple build sytem.This helps in compartibility with the build system as webpack
2. Babel Core –
3. Babel preset env – ES6 🡺 ES5



|  |  |  |  |
| --- | --- | --- | --- |
| **INSTALLING BABEL LOADER** | npm install --save-dev **babel-loader @babel/core @babel/preset-env** | | |
|  | | | |
| * To configure any loader we have to tell webpack on which type of files loader has to be applied. It can be applied to any type of files like JS, CSS, images etc. * We provide this configuration in webpack.config.js using some regex | | | |
| **CONFIGURATIONS** | | | |
| **STEP 1 :**  **CONFIGURATION IN WEBCONFIG JS**  const path = require("path");  const config = {  entry: "./src/index.js",  output: {  path: path.resolve(\_\_dirname, "build"),  filename: "bundle.js"  },  module: {  rules: [  {  **use: "babel-loader",**  test: **/\.js$/**  }  ]  }  };  module.exports = config; | | **STEP 2 :**  We need to create a file with name **.babelrc** in the root folder of the project  **PRESET CONFIGURATION IN .baberc**  {  "presets": ["@babel/preset-env"]  }   * **test :** This takes a regex value .This property tell webpack that babel loader will be applied to JS file only * **use:** This property makes babel compartible with webpack. * The preset-env module transpiles the ES6 code to ES5 code so that the transpiled code is compartible to lower version of browsers too. |  |

**CONVERTING TO ES6 MODULES SYSTEM**

Let’s convert the above commonJs module system into ES6 module system. Below are few highlight of the difference between them



|  |  |
| --- | --- |
| **Sum.js** | **Index.js** |
| const sum = (a, b) => a + b;  **export default sum;** | **import sum from "./sum";**  const total = sum(10, 10);  console.log(total); |