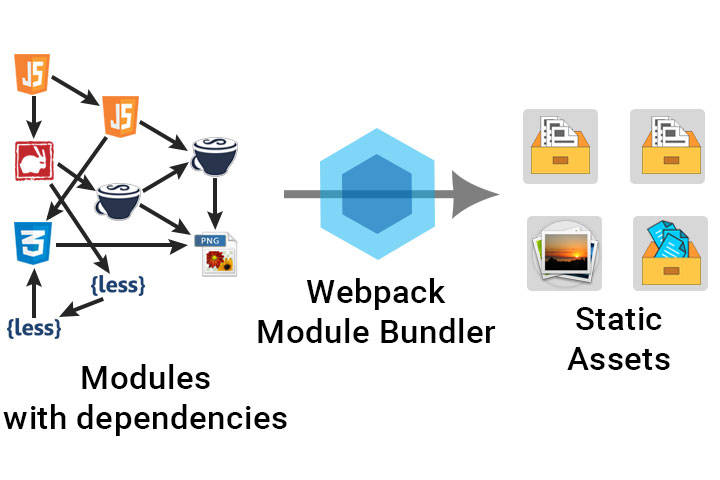
Webpack - Course Introduction

Welcome to the course on **Webpack**. In this course you will learn,

* How to install webpack
* How to use Webpack to bundle your large projects using webpack
* How to use Webpack loaders
* How to use Webpack plugins

##### What is Webpack?



**Webpack** is an open source module bundler for modern JavaScript applications. It takes modules with dependencies and generates static assets representing those modules.

##### When do you need Webpack?

Webpack is **good fit** for a complex front-end application with non-code static assets like style sheets, images etc.

Webpack is not recommended for small web applications.

##### Why do you need Webpack?

Based on the **dependency**, Webpack creates a dependency graph so as to give a modular approach to the web applications.

***Code Splitting*** (detailed later in the course) is an exciting feature that keeps Webpack distinct from all other module bundlers.

Goals of Webpack

The Webpack goals are not limited to:

* Spliting the dependency tree into chunks loaded on demand
* Reducing initial load time
* Every static asset should be able to be a module
* Integrating Third party libraries as modules
* Ensuring each part of the module is customizable

***Webpack is very much suitable for large projects***

##### Setup the environment

**Prerequisites:** Node and npm should be installed on your device. **1.** Create a directory - webpack\_practice **2.** Navigate to the newly created directory, and run npm init - Keep entering to keep the default configuration - This will create a package.json file for your project

##### Install webpack using npm

**3.** Install Webpack using npm

cmd >> npm install webpack --save-dev

Saving webpack only as dev dependency **This will create a node modules folder.**

Creating Source and Destination Folders

**4.** Create 2 folders

* ***src***: To keep all our working files.
* ***dist:*** Where all the compiled/bundled files will go.

Also create index.html in root and app.js in /src folder.

##### Sample html and js files

Here is the code of index.html and app.js

**index.html**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Webpack Demo</title>

<script src="dist/bundle.js" charset="utf-8"></script>

</head>

<body>

</body>

</html>

**app.js**

document.write('Hello FrescoPlayers');

##### Let us Compile

Run this code,

./node\_modules/.bin/webpack src/app.js dist/bundle.js

Open index.html in browser to ensure the page loads fine.

##### Bundle.js in Webpack

Did you note, ***bundle.js*** is loaded in our html file rather than ***app.js*** ? But, there is no bundle.js.

You might wonder **How will the file be created?**

Well, the answer is, **Webpack** creates the ***bundle.js*** file!

##### Automate the Compilation

Let us write some small script which does the compilation for us automatically.

Add "build": "webpack ./src/app.js ./dist/bundle.js" in script section of package.json.

**package.json**

{

"name": "webpack\_practice",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts":

{

"test": "echo \"Error: no test specified\" && exit 1",

"build": "webpack ./src/app.js ./dist/bundle.js"

},

"author": "",

"license": "ISC",

"devDependencies":

{

"webpack": "^3.8.1A"

}

}

Building the Package

1. Delete bundle.js first, and then run the build script from package.json using the following command from your folder.

webpack\_practice >> `npm run build`

This essentially does the same. It creates bundle.js file in ***/dist*** folder.

More on Installing Webpack

In this video you will learn how to install webpack and make use of the out the box features it comes with:

* Module bundling
* Automatic bundling
* JavaScript minification

Essentially, this video will explain how to install webpack, set up a configuration file, and create a bundle for production.

##### Webpack Installation Handson

Now, try installing webpack on your own and configure the dev server.

For this, use the playground: https://www.katacoda.com/courses/nodejs/playground

##### Why do you need a Dev server?

Did you note, index.html file which opened in the browser earlier has a file protocol file:///home/ram/work/webpack\_practice/index.html, meaning the file is being loaded from our local machine.

The main challenge with file protocol based approach is that, we are not sure if all the assets are actually being downloaded from a server and served.

Here Is The Solution

* To address the issue of running the app using http protocol, you can use webpack-dev-server.
  + The webpack-dev-server is a little Node.js Express server, which uses the webpack-dev-middleware to serve a webpack bundle
  + The server listens for any change in your project and then injects that change into the browser

Installing Dev server

To ***install*** webpack dev server,

webpack\_practice >> `npm install webpack-dev-server --save-dev`

***Modify*** the build script in package.json as follows

"build": "webpack-dev-server --entry ./src/app.js --output-filename ./dist/bundle.js"

* Now type npm run build

Let Us Work With Dev Server

* Open index.html(http://localhost:8080) and check network tab in developers console. You should see assets being downloaded.

##### Assets From Memory

Webpack dev server prevents webpack from emitting the resulting files to the disk. Instead it keeps and serves the resulting files from memory.

- Delete dist folder

- Run `npm run build`

- open index.html in browser.

We will still get the same output without bundle.js getting created.

##### Core concepts of Webpack

Webpack includes four core concepts

1. Entry

2. Output

3. Module loaders

4. Plugins.

##### Entry and Output

**Entry:** The starting point of a dependency graph is called an Entry point, it tells webpack where to start bundling.

**Output :** It tells the Webpack where to bundle the modules of the application and how to treat a bundled code.

##### Defining Entry and Output

**Create** webpack.config.js in the root of our working directory.

### **webpack.config.js**

// Nodejs module for absolute path resolving

var path = require('path');

module.exports = {

// Entry point of webpack

entry: './src/app.js',

output: {

// Need to be resolved to absolute path

path: path.resolve(\_\_dirname, 'dist'),

// bundled/compiled file name

filename: './bundle.js',

// For webpack dev server to identify the full path.

publicPath: '/dist'

}

};

**Modify** the script for webpack dev server by removing the entry and outputs in package.json **package.json**

{

"name": "webpack\_practice",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts":

{

"test": "echo \"Error: no test specified\" && exit 1",

"build": "webpack-dev-server"

},

"author": "",

"license": "ISC",

"devDependencies":

{

"webpack": "^2.4.1"

}

}

Now run the build

webpack\_practice >> npm run build

Open localhost:8080 in browser, our js loads as expected.

Why do we need Loaders?

* Webpack can handle JavaScript natively
* Loaders are like **tasks** in other build tools, and provide a powerful way to handle frontend build steps
* Loaders can:
  + Transform files from a different language like, CoffeeScript to JavaScript or inline images as data URLs
  + Loaders also allow you to include css files( require() ) right in JavaScript!

There is wide variety of loaders available which includes: - conversion on ES6 code to ES5 code using babel. - scss loader which converts SASS code to css

Loading CSS

To load CSS, you need to use

1. `css-loader` : For processing css files.

2. `style-loader` : injects the styling through a style element.

* Install css and style loaders

webpack\_practice >> npm install css-loader style-loader --save-dev

* create css folder in /src and include main.css in it.

##### Create your HTML file

Modify the index.html file

**index.html**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Webpack Demo</title>

<script src="dist/bundle.js" charset="utf-8"></script>

</head>

<body>

<button type="button" name="button" id="redButton">I'm Red</button>

<button type="button" name="button" id="greenButton">I'm Green</button>

<button type="button" name="button" id="blueButton">I'm Blue</button>

</body>

</html>

##### Create your StyleSheet

Modify the main.css file

**main.css**

body {

background-color: aquamarine;

}

button {

border: 1px solid black;

padding: 5px 10px;

font-size: 2rem;

color: ghostwhite;

border-radius: 10px;

}

#redButton {

background-color: red;

}

#greenButton {

background-color: forestgreen;

}

#blueButton {

background-color: blue;

}

##### Configure your webpack

Modify the app.js file

**app.js**

require("./css/main.css");

Add loader modules in webpack.config.js

**webpack.config.js**

var path = require('path');

module.exports = {

entry: './src/app.js',

output: {

path: path.resolve(\_\_dirname, 'dist'),

filename: './bundle.js',

publicPath: '/dist'

},

module: {

loaders: [

{ test : /\.css$/, loader: "style-loader!css-loader" }

]

}

};

##### Run your CSS Loader

Type npm run build in terminal. It opens http://localhost:8080 in a browser. Check to ensure that all styles are being applied.

##### Setting up ES6 Loader

To use ES6 in our application with webpack, we need

- `babel-core`: Transpiles es6 code to es5 so that it works in any browser.

- `babel-loader`:Helps in using babel with webpack

- `babel-preset-es2015`: unlocks all the es6 features to use in our application.

npm install babel-core babel-loader babel-preset-es2015 --save-dev

Add operations.js in /src folder with the following content **operations.js**

function sum(a, b) {

return a+b;

}

function difference(a, b) {

return a-b;

}

export { sum, difference }

##### Configure your webpack

Modify app.js as follows

**app.js**

require("./css/main.css");

import {sum, difference} from './operations.js';

document.write("Sum of 2 and 3 is "+sum(2, 3)); // 5

document.write("Difference of 5 and 5 is "+difference(5, 2)); // 3

**But this will not work unless we include the babel loader configuration in webpack.config.js**

**webpack.config.js**

var path = require('path');

module.exports = {

entry: './src/app.js',

output: {

path: path.resolve(\_\_dirname, 'dist'),

filename: './bundle.js',

publicPath: '/dist'

},

module: {

loaders: [

{ test : /\.css$/, loader: "style-loader!css-loader" },

{

test: /\.js$/,

use: [

{

loader: 'babel-loader',

options: { presets: ['es2015'] }

}

]

}

]

}

};

##### Run your ES6 Loader

Try running npm run build and open **localhost:8080** to see the changes in browser.

Plugins

* In Webpack, the developers are not confined to the ***predefined build callbacks***. They can use ***customized build callback***s using plugins.
* Plugins are the backbone of Webpack as it fully exposes the potential of webpack.
* Plugins are additional node modules that usually work on the resulting bundle.

***For Example:***

* ***HtmlWebpackPlugin*** simplifies the creation of HTML files to serve Webpack bundles.
* ***ExtractTextWebpackPlugin*** moves all CSS modules in entry chunks into a separate CSS file. So that the styles are no longer inlined into the JS bundle, but are saved in a separate CSS file (styles.css).

##### HTMLWebpackPlugin

First install plugin as dev dependency:

npm install html-webpack-plugin --save-dev

To make it work, modify the webpack.config.js as follows:

**webpack.config.js**

var path = require('path');

var HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {

entry: './src/app.js',

output: {

path: path.resolve(\_\_dirname, 'dist'),

filename: 'bundle.js'

},

module: {

loaders: [

{ test : /\.css$/, loader: "style-loader!css-loader" },

{

test: /\.js$/,

use: [

{

loader: 'babel-loader',

options: { presets: ['es2015'] }

}

]

}

]

},

plugins: [ new HtmlWebpackPlugin() ]

};

##### Run the build

This will generate a **dist/index.html** file, which has the following content.

## **dist/index.html**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Webpack App</title>

</head>

<body>

// output assets

<script src="bundle.js" charset="utf-8"></script>

</body>

</html>

webpack\_practice >> npm run build

##### Modifying webpack.config.js

If you notice, all the buttons that are missing from the final index.html, will by default serve just js and css compiled assets.

We can also give specifications for the template to generate, rather than using the default template.

Modify the **webpack.config.js** as follows:

**webpack.config.js**

var path = require('path');

var HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {

entry: './src/app.js',

output: {

path: path.resolve(\_\_dirname, 'dist'),

filename: './bundle.js',

publicPath: '/dist'

},

module: {

loaders: [

{ test : /\.css$/, loader: "style-loader!css-loader" },

{

test: /\.js$/,

use: [

{

loader: 'babel-loader',

options: { presets: ['es2015'] }

}

]

}

]

},

plugins: [

new HtmlWebpackPlugin({

template: 'index.html' // we are providing our index.html which has all our code.

})

]

};

Again, run npm run build. You will notice that all buttons will be included.

##### More on Plugins

CSS loaders for webpack inline's the CSS rather than placing it within a separate CSS file. This is good for page speed. However, when the amount of CSS is high, inline CSS can slow down the site since it can not be cached. In this video, you will learn how to use the extract-text-webpack-plugin, combined with webpack, to export the CSS into a separate file.

Dependency Graph

* In Webpack, whenever a file refers to another, it is called a dependency
* Webpack starts from the entry point, finds each dependency and creates the dependency graph.
* It considers all the non-code assets such as images and stylesheets and adds them to the graph.
* Based on the graph, it bundles the files together, hence the overhead of loading all the distinct files is eliminated.

What is Code Splitting

In this video you will learn about,

* Need to split the code
* How code spliting makes the application dynamic
* How to use AggressiveSplittingPlugin to split the module

##### More on Bundling a web application

Module bundling while creating specific scripts for specific pages is possible by having multiple entry points that compile into multiple bundles. In this video, you will learn how to implement this using web pack.

Optimization

Now that you have successfully bundled your application, **What next**?

* The bundled application is not directly suitable for production because of the lengthy junk comments and spaces.
* You should optimize it using a webpack command to make it ready for production.

webpack -p

This will do the minification for you.

##### Minification

Modify the package.json file like this

package.json

---------------

{

"name": "webpack\_practice",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"build": "webpack ./src/app.js ./dist/bundle.js",

"prod": "webpack –p"

},

"author": "",

"license": "ISC",

"devDependencies": {

"webpack": "^3.8.1"

}

}

Run it using the following command:

Cmd>> npm run prod

Congratulations! You are all done now!

Webpack - Course Summary

Hurrah! You have successfully completed a course on Webpack. In this course you have learnt,

* How to install webpack
* How to use Webpack to bundle your large projects using webpack
* How to use Webpack loaders
* How to use Webpack plugins

Quiz:

1. Executing the following command will automatically create a file named: webpack\_practice >> npm init PACKAGE.JSON
2. Webpack supports third party libraries T
3. The latest version of Webpack is 2X
4. Webpack is not recommended for large web applications F
5. Which of these softwares is necessary to install webpack NODE.JS
6. The distinct feature of webpack is CODE-SPLITTING
7. Which of the following webpack feature enables Code on Demand CODE-SPLITTING
8. Which of the following code can help us make the compilation automatically BUILD...
9. Which helps in using babel with webpack BABEL-LOADER
10. Which of the following tells webpack how to treat a Bundled Code OUTPUT
11. Which of the following code can be used to install the style loader? NPM....--SAVE
12. Which of the following loader can be used to process style sheets CSS-LOADER
13. Webpack understands only \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ JAVA SCRIPT
14. file:///home/ram/work/webpack\_practice/index.html means that the file gets loaded from LOCAL MACHINE
15. Which of the following tells webpack how to treat a Bundled Code OUTPUT
16. Which of the following features, considered the backbone of Webpack PLUGIN
17. Which of the following code snippet instantiates the plugin FrescoPlugin REQUIRE.........NEW
18. \_\_\_\_\_\_\_\_\_\_\_\_\_ simplies the creation of HTML files to serve webpack bundles HTML WEBPACKPLUGIN
19. Plugins are instantiable objects T
20. Webpack can be optimized with the following command WEBPACK--OPTIMIZE-MINMISE
21. Webpack is written in JAVA SCRIPT
22. Webpack is a/an \_\_\_\_\_\_\_\_\_\_\_\_\_\_ tool OPEN SOURCE
23. A plugin can be instantiated with the keyword NEW
24. Which of the following is a Node.js express server? WEBPACK-DEV-SERVER
25. Which of the following makes Webpack extensible Entry
26. In Webpack, we can use None of the options
27. Custom plugins can be installed via \_\_\_\_\_\_\_\_\_\_\_ NPM
28. Webpack starts building the dependency graph from It is Context dependent
29. Which of the following, can be used to define an entry point? module.exports = {entry: './path/to/my/entry/file.js'};
30. CoffeeScript can be transformed to JavaScript with the help of LOADER
31. Which of the following code can help us make the compilation automatically build": "webpack ./src/app.js ./dist/bundle.js
32. Which of these softwares is necessary to install webpack Node.js
33. Which of the following is a Node.js express server? webpack-dev-server