

Webpack

Learn and lunch

November 2017

**Back in the day in the front-end
environment**

“Script tag stack” era

```
body>  
<script src="http://cdn.jsdelivr.net/jquery"></script>  
<script src="http://cdn.jsdelivr.net/jquery.slider"></script>  
<script src="http://cdn.jsdelivr.net/jquery.date-picker"></script>  
<script src="http://cdn.jsdelivr.net/jquery.slider"></script>  
<script src="http://cdn.jsdelivr.net/jquery.lava"></script>  
<script src="http://cdn.jsdelivr.net/jquery.parallax"></script>  
<script src="http://cdn.jsdelivr.net/lodash"></script>  
<script src="http://cdn.jsdelivr.net/backbone"></script>  
  
<script src="utils.js"></script>  
<script src="component-a.js"></script>  
<script src="component-b.js"></script>  
<script src="component-c.js"></script>  
<script src="component-d.js"></script>  
<script src="component-e.js"></script>  
<script src="my-app.js"></script>  
body>
```

“Script tag stack” problems

Multiple HTTP requests

Order is important

Scripts can have interdependencies

all.js/script concatenation era

```
ay>
```

```
<script src="all.js"></script>
```

```
ody>
```

all.js/script concatenation problems

~~Multiple HTTP requests~~

Order is **still** important

Scripts can **still** have interdependencies

Unnecessary chunks of scripts are loaded

Node.js

2009

Based on Google's Chrome's JS V8

Event loop

Javascript on server side

Modules

Node.js - module

- Function/variable/class reusable easily
- Maintainability
- Imported with keyword “require”
- Exported with keyword `modules.exports`
- scoped => no mainspace pollution

Node.js - npm

Node package manager

Has a ton of packages for almost everything

Twitter's Bower 2012

Pkg manager for front-end (img, js...)

“Deprecated”

Browserify 2011

Allows “require” in the browser

Allows node_modules in the browser*

Transforms your javascript files

Loads **synchronously** modules

Bundles your js file

* Not all pkg are usable in the browser

Bundle ?

Bundling

Create a file containing multiple modules

Process non-javascript code/assets :

- Optimize image
- Transpile templates into js / non-js to js

Single Page Application (SPA) era (20XX)

BackboneJS

Angular / AngularJS

Aurelia

Ember.js

VueJS

React

...

Single Page Application (SPA) era

- Websites are dead, long life to applications
- Everything is in the javascript
- Images/css/templates are loaded in the js

Browserify

Does not handle by
default all front-end assets...

So Webpack came

<https://webpack.js.org>

Bundle manager for front-end

Webpack

- Created in 2012
- Two major versions this year : 2 and 3

Webpack

- Bundles javascript

Webpack

- Bundles javascript
- Bundles html, images, css and more natively*
- Philosophy: Convention over configuration**
- (Can) Loads modules **asynchronously** natively
- Allows ES6 modules before native browser support

<https://webpack.github.io/docs/motivation.html>

* Requires specific loaders

** https://en.wikipedia.org/wiki/Convention_over_configuration

Let's use it

- npm install -D webpack / yarn install -D webpack
- webpack <entry> <output>

That's it

Webpack - cli

Has a lot of options* for compilation :

- -p : build for production
- -d : build for development
- --watch, -w : watch file for changes
- --help, -h : list all options
- [...]
- --config : build source using a config file

* <https://webpack.js.org/api/cli/>

Webpack - config file

- Named webpack.config.js by default
- Can **inherit** from another file
- More user-friendly than cli
- Must return an object

* <https://webpack.js.org/api/cli/>

Example

<https://github.com/DanYellow/misc-tests/tree/master/webpack-presentation-examples/webpack-samples/basic>

Webpack - config file's anatomy*

```
1  const path = require('path');
2  const HtmlWebpackPlugin = require('html-webpack-plugin');
3  const CleanWebpackPlugin = require('clean-webpack-plugin')
4
5  module.exports = {
6    entry: './src/main.js', // Entry point
7    output: {
8      path: path.resolve(__dirname, 'dist'), // Path for output MUST BE ABSOLUTE
9      filename: '[name].[hash].js' // name of the output
10   },
11   plugins: [ // List of plugins
12     new HtmlWebpackPlugin(),
13     new CleanWebpackPlugin(['dist']),
14   ],
15   module: { // List of loaders
16     rules: [
17       { test: /\.js$/, exclude: /(node_modules)/, use: { loader: 'babel-loader' } }
18     ]
19   }
20 };
```

* <https://webpack.js.org/configuration/>

Webpack - loaders*

- Equivalent of browserify's transforms
- Process non-JavaScript modules as dependancies for bundles
- Loaded under "module.rules" key in a config file

* <https://webpack.js.org/concepts/loaders/>

Example

<https://github.com/DanYellow/misc-tests/tree/master/webpack-presentation-examples/webpack-samples/loaders>

What we saw until now

- Script loading were painful until browserify
- Browserify allow developers to bundle js
- webpack's loaders are browserify's transforms

What we saw until now

- Script loading were painful until browserify
- Browserify allow developers to bundle js
- webpack's loaders are browserify's transforms
- webpack and browserify do the same thing

Plugins

<https://webpack.js.org/plugins/>

Webpack - plugins

- Plugin does what a loader can't
- Most of the time they are applied **after** loaders
- Customizes build process of webpack

<https://webpack.js.org/configuration/plugins/>

Webpack – plugins examples*

- Define env vars
- Copy file
- Compress files
- and more and yours

* <https://webpack.js.org/plugins>

Example

<https://github.com/DanYellow/misc-tests/tree/master/webpack-presentation-examples/webpack-samples/plugins>

Advanced webpack

Environments management

Goals :

- Execute specific code into specific environment
- Use inheritance for config files

Example :

<https://github.com/DanYellow/misc-tests/tree/master/webpack-presentation-examples/webpack-samples/environments>

Internationalization (i18n)

Goal :

- Create a specific bundle for each localization

Example :

<https://github.com/DanYellow/misc-tests/tree/master/webpack-presentation-examples/webpack-samples/modules>

Single Page App – CSS in JS

Goal :

- Create a ReactJS application
- Load css and images in javascript file

Example :

<https://github.com/DanYellow/misc-tests/tree/master/webpack-presentation-examples/webpack-samples/assets-in-js>

Single Page App with (R)HMR*

Goals :

- Improve development production
- Reload only modules/css edited

Example :

<https://github.com/DanYellow/misc-tests/tree/master/webpack-presentation-examples/webpack-samples/hot-reload>

*(React) Hot Module Reload

Code Splitting

Use case :



PageA.html

lodash.js
luxon.js
pageA.js

pa.bundle.js



PageB.html

lodash.js
pageB.js

pb.bundle.js



PageC.html

lodash.js
react.js
pageC.js

pc.bundle.js

Code Splitting

Use case :



PageA.html

lodash.js

luxon.js

pageA.js

pa.bundle.js



PageB.html

lodash.js

pageB.js

pb.bundle.js



PageC.html

lodash.js

react.js

pageC.js

pc.bundle.js

Code Splitting

Lodash.js is loaded three times on three different pages! → Two loadings are useless

Code Splitting

Four ways to split code:

- Manual code splitting
- CommonsChunkPlugin
- Dynamic code splitting
- Lazy loading code splitting

<https://webpack.js.org/guides/code-splitting/>

<https://webpack.js.org/guides/lazy-loading/>

Code Splitting

Application (CommonsChunkPlugin) :



PageA.html

lodash.js

luxon.js

pageA.js

lodash.js (1st call)

pa.bundle.js



PageB.html

lodash.js

pageB.js

lodash.js (cached)

pb.bundle.js



PageC.html

lodash.js

react.js

pageC.js

lodash.js (cached)

pc.bundle.js

Code Splitting

Benefits:

- Decrease bundle size
- Decrease data download
- Decrease loading time

Code Splitting

Goal :

- Extract into a specific bundle every module in common

Example :

<https://github.com/DanYellow/misc-tests/tree/master/webpack-presentation-examples/webpack-samples/code-splitting>

Webpack

Pros

- Asynchronous and synchronous modules loading
- Code splitting
- Convention philosophy
- Handles by default all front-end assets
- Hot module reloading
- Ton of natives plugins / loaders

Cons

- Hard learning curve
- Complex to setup
- Could be overkill for non-SPA project
- Config file can be hard to read
- Limited to javascript

Summary of this presentation

Webpack is **convention**

Browserify is **configuration**

Webpack bundles all front-end assets **by default**

Webpack \sim gulp + browserify

Webpack's plugins = browserify's plugins

Webpack's loaders = browserify's transforms

Thank you for your attention

All examples and presentation:
<https://github.com/DanYellow/misc-tests/webpack-presentation-examples>

Questions ?

More sources

- Documentation: webpack.js.org
- Tree shaking explained:

https://medium.com/@Rich_Harris/tree-shaking-versus-dead-code-elimination-d3765df85c80

- <https://github.com/petehunt/webpack-howto>