Webpack: from 0 to 2(.3.3)

Milano_JS

04.04.2017

Alessandro Bellini

frontend developer @ Spryker

- web: ilmente.com
- github: github.com/ilmente
- twitter: @ilmente

Agenda

- grunt, gulp and webpack
- webpack modules
- entry points
- loaders and plugins
- configuration - code overview
- code splitting - code overview
- tree shaking - code overview

in the beginning it was **grunt**

grunt

- focus on configurations
- built around a set of built-in commonly used tasks
- extension using plugins
- each plugin has its own custom configuration
- every task is an array of different plugin configurations, that simply get executed one after another, in a strictly independent, and sequential fashion

https://medium.com/@preslavrachev/gulp-vs-grunt-why-one-why-the-other-f5d3b398edc4

then came **gulp**

gulp

- focus on code
- micro-tasks connected to each other (agnostic about their nature)
- extension using plugins
- plugins use API to be programmed
- streams and pipelines

https://medium.com/@preslavrachev/gulp-vs-grunt-why-one-why-the-other-f5d3b398edc4

and now **webpack**

grunt

- focus on configurations
- built around a set of built-in commonly used tasks
- extension using plugins
- each plugin has its own custom configuration
- every task is an array of different plugin configurations, that simply get executed one after another, in a strictly independent, and sequential fashion

https://medium.com/@preslavrachev/gulp-vs-grunt-why-one-why-the-other-f5d3b398edc4

~~grunt~~ webpack

- focus on configurations
- built around a set of built-in commonly used tasks
- extension using plugins
- each plugin has its own custom configuration
- every task is an array of different plugin configurations, that simply get executed one after another, in a strictly independent, and sequential fashion

close, but it’s not exactly like this...

~~grunt~~ webpack

- focus on configurations
- built around a set of built-in commonly used ~~tasks~~ features
- extension using plugins and loaders
- each plugin/loader has its own custom configuration
- every ~~task~~ compile process (or step) is an array of different ~~plugin~~ loaders (eventually configured/extend using plugins) ~~configurations~~, that simply get executed one after another, in a ~~strictly independent, and~~ sequential fashion
- loaders can be chained together: this is helpful for applying multiple transformations to a file **in a pipeline**

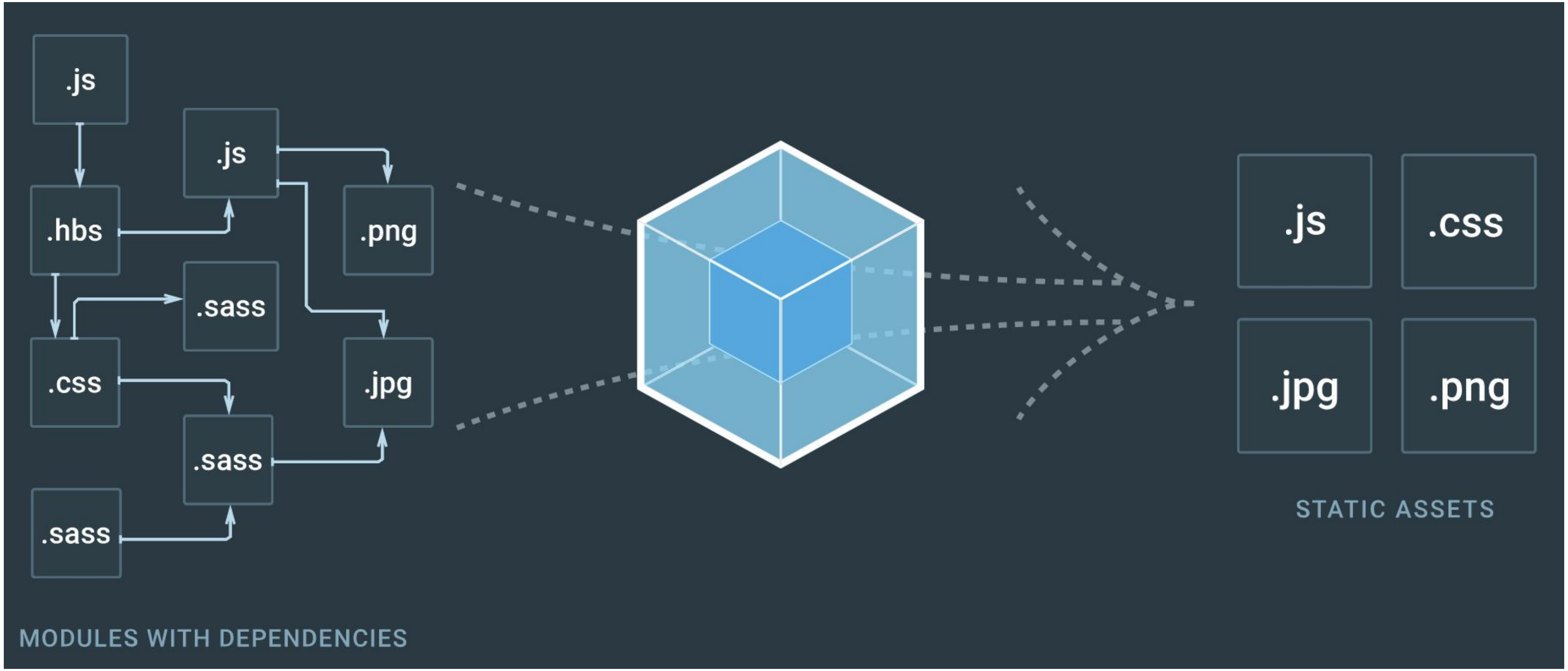
https://webpack.github.io/docs/loaders.html (webpack 1.x docs)

grunt - gulp - webpack

- grunt and gulp are **task runners**
- webpack is a **module bundler**
- gulp and webpack are not necessary enemies

You can use webpack with grunt/gulp:

- https://webpack.js.org/guides/task-test-runner/
  (almost useless page)
- http://webpack.github.io/docs/usage-with-grunt.html
- http://webpack.github.io/docs/usage-with-gulp.html


.js
.js
.hbs
.png
.sass
.css
.jpg
.sass
.sass
MODULES WITH DEPENDENCIES
.js
.css
.jpg
.png
STATIC ASSETS

webpack **modules**

not just modules

modules

- discrete chunk of functionality a program is broken into
  - abstraction
  - single responsability
  - reusability
  - better debugging and testing

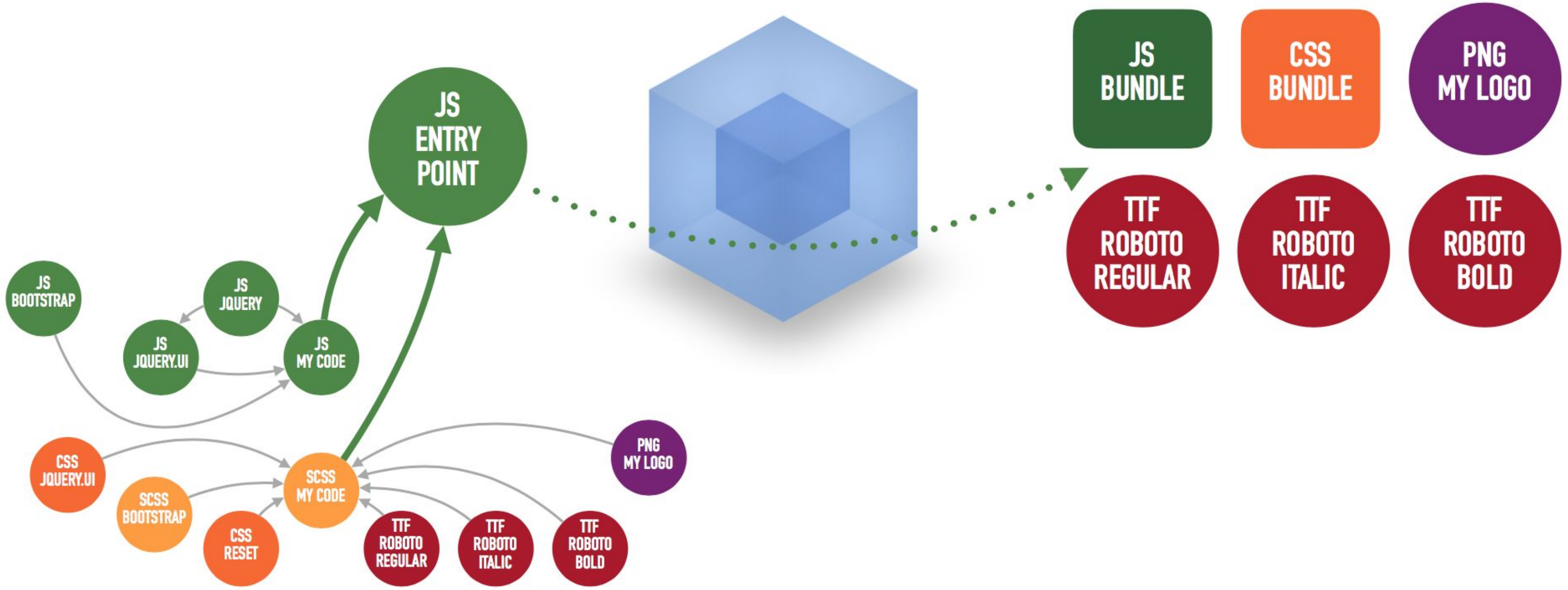
webpack modules

- a **dependency** expressed by:
  - **import** (es2015)
  - **require()** (CommonJS)
  - **define()** and **require()** (AMD)
  - **@import** (css/sass/less)
  - **url()** (stylesheet)
  - <img **src=**””> (html)

(!) Please refer to https://webpack.js.org/concepts: it’s well done and explained

entry points

the root of the tree


JS ENTRY POINT
JS BOOTSTRAP
JS JQUERY
JS JQUERY.UI
JS MY CODE
CSS JQUERY.UI
SCSS BOOTSTRAP
CSS RESET
SCSS MY CODE
PNG MY LOGO
TTF ROBOTO REGULAR
TTF ROBOTO ITALIC
TTF ROBOTO BOLD
JS BUNDLE
CSS BUNDLE
PNG MY LOGO
TTF ROBOTO REGULAR
TTF ROBOTO ITALIC
TTF ROBOTO BOLD

loaders and plugins

is this a kind of magic?

loaders

- functions
- transform the source code of a module
- synchronous or asynchronous
- can be configured with options
- can be chained
- can emit additional arbitrary files

https://webpack.js.org/concepts/loaders

plugins

- webpack **backbone**
- “anything else that a loader cannot do” (cit.)
  - it’s wizardry
  - can you do everything?
  - may I have a coffee?

- hooks into the whole webpack build system
- allow you to access, change, extend any phase of the build process, from resolving a module to bundling the output

https://webpack.github.io/docs/plugins.html

in a nutshell

- if you want to tell webpack how to load that specific type of resource: **use loaders**

- everything else: **use plugins**

in a nutshell

- if you want to tell webpack how to load that specific type of resource: **use loaders**

- ~~everything else: **use plugins**~~
- if you want to change how this module is resolved and compiled: **use plugins**

a real **configuration**

let’s switch to the code

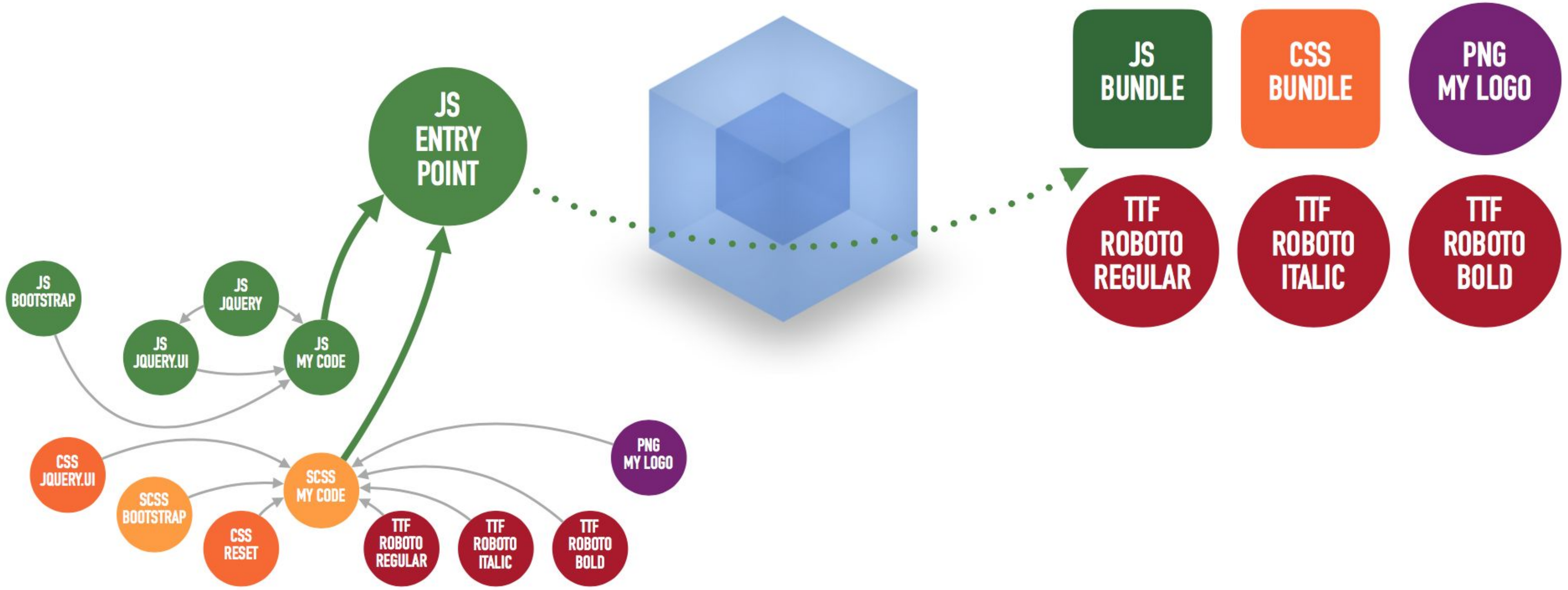
code **splitting**

introducing performance superheroes

code splitting

- it’s a features
- split your code into various bundles
  - smaller bundle size
  - resource load prioritization control
  - on-demand loading
- may have significant performance impact
  - on build/rebuild time
  - on application/site load time
- **you have the freedom to choose and implement your own strategy to optimise the assets management**

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


JS BOOTSTRAP
JS JQUERY
JS JQUERY.UI
JS MY CODE
JS ENTRY POINT
CSS JQUERY.UI
SCSS BOOTSTRAP
CSS RESET
SCSS MY CODE
TTF ROBOTO REGULAR
TTF ROBOTO ITALIC
TTF ROBOTO BOLD
PNG MY LOGO
JS BUNDLE
CSS BUNDLE
PNG MY LOGO
TTF ROBOTO REGULAR
TTF ROBOTO ITALIC
TTF ROBOTO BOLD

code splitting superheroes

- css
  - style-loader
  - ExtractTextWebpackPlugin
- vendor
  - CommonsChunkPlugin
- on-demand
  - import()
  - require.ensure()

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

tree shaking

bullshit, myth
or *shut-up-and-take-my-money* feature?

tree shaking

- positive selection of actual used code (*live code*)
- in opposition to *dead code removal* (UglifyJs plugin)
- introduced in bundlers world by github.com/rollup/rollup

Known limitations for webpack:

- static structure
- relies on es2015 module **import/export**

https://webpack.js.org/guides/tree-shaking

the code said…

- babel:
  - use preset *babel-preset-es2015* with { “modules”: false }
  - use plugin *babel-plugin-syntax-dynamic-import*
- limited to es2015 source code
- unable to shake objects or inside statements
- capable of dropping functions, but not classes
- uglifyJS plugin needed

https://webpack.js.org

it’s my main source

thanks

yes, it’s finally over…