

# Angular + webpack

如何使用 webpack 构建 Angular.js 项目

黄俊亮 (@JLHwung)

# Why webpack

- js 模块化
  - 清晰定义每个模块的输入与输出: `import / export`
  - 不再通过全局空间来调用方法: ~~`window.foo = function() {}`~~
- 用 ES2017 来写 js, 即便浏览器还不支持: `async / await`,  
`Promise`, `Symbol` ...
- 不再理会 CSS 的浏览器前缀, 即便浏览器还不支持: `flex`,  
`:fullscreen`
- Even more...

# Basic Concept - Entry & Output

A minimal webpack configuration

```
module.exports = {
  entry: './path/to/my/entry/file.js',
  output: {
    path: '/path/to/dist',
    filename: 'my-first-webpack.bundle.js'
  }
}
```

./path/to/my/entry/file.js

```
import $ from 'jquery'
// 业务入口
```

# Basic Concept - Module & Chunk

## Module

- 每一个被 webpack 解析到的文件

## Chunk

- 每一个被 webpack 生成的最终的静态文件

# Basic Concept - Loaders & Plugins

An slightly enhanced version

```
const webpack = require('webpack')

module.exports = {
  entry: './path/to/my/entry/file.js',
  output: {
    path: '/path/to/dist',
    filename: 'my-first-webpack.bundle.js'
  },
  module: {
    rules: { test: /\.js$/, use: 'babel-loader' }
  },
  plugins: [
    new webpack.optimize.UglifyJsPlugin()
  ]
}
```

# Environments

## 开发环境

- 代码改动即时重载
- 只需要解析依赖拼接模块，不需要最小化
- 所有模块全部拼在一起，不做分离

## 生产环境

- 生成的静态文件必须带 hash
- 分离业务代码与框架代码
- 分离 css 与 js 文件
- 更保守的浏览器兼容性处理

# 开发环境: webpack-dev-server

```
{  
  devServer: {  
    port: 1339,  
    proxy: {  
      '**': 'http://localhost:8080' // 将 API 请求代理到后端  
    }  
  }  
}
```

- 将所有 chunk 保存在内存里，内部使用 express 提供静态文件服务
- 内建 WebSocket 服务器，当新的 chunk 生成时往客户端推送通知进行页面重载

# Get your hands dirty

克隆镜像: Toy Example

```
git clone https://github.com/easyops-cn/webpack-transition-toy-
```

Fast-cheat: 跳过大部分的安装时间

```
git checkout webpack  
npm i
```

*# or if you have yarn installed*  
yarn

# Get your hands dirty (cont.)

## 安装基础依赖

```
# 如果已经运行过 Fast-cheat, 可以略过这一步  
npm i webpack webpack-dev-server --save-dev
```

```
# or if you have yarn installed  
yarn add --dev webpack webpack-dev-server
```

# Get your hands dirty (cont.)

设置启动脚本，基本 webpack 配置

```
# 如果已经运行过 Fast-cheat, 可以略过这一步  
yarn add --dev babel-loader babel-core babel-preset-env
```

```
# 查看这一步我们都做了什么  
git checkout cea55cb  
git show
```

# Pause and Ponder

## 项目特点

- 有 `bower_components` 与 `node_modules` 两套依赖体系
- 业务文件直接假定了 `angular` 和 `jquery` 全局存在
- 样式文件与业务文件是分离的（注意：webpack 只能处理 js 文件）

# Target and Conquer

## 两套依赖体系

使用 `resolve` 指定依赖体系

```
{  
  resolve: {  
    // 包的解析路径  
    modules: [  
      path.resolve(process.cwd(), './node_modules'),  
      path.resolve(process.cwd(), './app/bower_components')  
    ],  
    // 包的信息文件  
    descriptionFiles: ['package.json', 'bower.json']  
  }  
}
```

# Target and Conquer (cont.)

改了哪些配置？

```
git checkout 579214ad  
git show
```

看看效果

```
npm start
```

# Target and Conquer (cont.)

## 全局依赖

```
yarn add --dev imports-loader
```

使用 imports-loader 注入依赖

```
{
  module: {
    rules: [{
      test: /\.js$/,
      exclude: /node_modules|bower_components/,
      use: [{
        loader: 'imports-loader',
        options: 'angular' // use '$=jquery' for jquery imports
      }]
    }]
  }
}
```

## Target and Conquer (cont.)

```
git checkout b54b3afb  
git show
```

# Target and Conquer (cont.)

## 加载 CSS 文件

```
yarn add --dev style-loader css-loader
```

style-loader

将 CSS 文件通过 `<style>` 标签注入到 DOM 中

css-loader

把 CSS 文件中的 `@import/url()` 转换成 `import/require`

## 加载 CSS 文件 (cont.)

```
{  
  module: {  
    rules: [{  
      test: /\.css$/,  
      use: ['style-loader', 'css-loader']  
    }]  
  }  
}
```

```
git checkout 7f944a50  
git show
```

# Too big to debug

- 找不到设断点的地方
- 浏览器的调试器加载要半天
- 出错了都不知道是哪一行代码

# Sourcemap

源代码 <====> 生成代码

```
{  
  devtool: 'source-map'  
}
```

Sources | Content scripts | Snippets | ... | entry.bundle.js | entry.js | angular-route.js | app.js x

top

localhost:1339

bower\_components/html5-boilerplate

(index)

entry.bundle.js

(no domain)

webpack://

.

bower\_components

components/version

view1

view2

app.css

app.css?9ae3

app.js

entry.js

```
1  /** IMPORTS FROM imports-loader **/
2  var angular = require("angular");
3
4  'use strict';
5
6  // Declare app level module which depends on views
7  angular.module('myApp', [
8      'ngRoute',
9      'myApp.view1',
10     'myApp.view2',
11     'myApp.version'
12   ]).
13 config(['$locationProvider', '$routeProvider', fun
14   $locationProvider.hashPrefix('!');

16   $routeProvider.otherwise({redirectTo: '/view1'})
17 ]);

18
19
20
21
22 // WEBPACK FOOTER //
23 // ./app.js
```

{ } Line 12, Column 4 (source mapped from entry.bundle.js)

# Upshot & Take-away

- webpack
  - 从 entry 到 output 的构建工具
  - 每一个 module 最后经过组合、转换生成了最终的 chunk
  - loader 对 module 施加变换， plugin 对 chunk 施加变换
- 使用 `babel-loader` 和 `babel-preset-env` 加载业务 js 文件
- 使用 `imports-loader` 给业务文件注入依赖
- 使用 `style-loader` 和 `css-loader` 加载 css 文件
- 使用 `devtool: 'source-map'` 来生成 source-map

# 生产环境怎么办？

作为练习，你可以参考下面的内容

- [Guide](#): webpack 官方的生产环境构建指南
- [commons-chunk-plugin](#): 提取业务文件的公共模块放在一个 `vendor.js` 中
- [extract-text-webpack-plugin](#): 将不同地方导入的 CSS 抽取成一个文件
- [babili-webpack-plugin](#): 基于 babel 的 JavaScript 最小化工具
- [css-loader](#): 如何最小化 CSS
- [chunkhash](#): 如何让产出的静态资源带上 hash
- [html-webpack-plugin](#): 动态生成 HTML

# Open Problem

- 如何将不同地方导入的 CSS 文件根据规则抽取成多个文件，比如说 `vendor.css` , `app.css` ? [extract-text-webpack-plugin/#159](#)

# Advanced Topics

- [Code Splitting](#): 使用 `import()` 切分模块以实现按需加载
- [Tree Shaking](#): webpack 2 基于 ES2015 `import/export` 静态模块系统的代码消除功能
- [Hot Module Replacement](#): react 技术栈下的热重载方案
- [hard-source-webpack-plugin](#): 提供中间步骤 `cache` 以加速 webpack 构建
- [dll-plugin](#): 对框架库做封装以改善开发环境修改后的 rebuild time