

04 使用 webpack-cli 体验零配置打包

更新时间：2019-07-25 11:11:52



“

人生太短，要干的事太多，我要争分夺秒。

——爱迪生

”

下面通过搭建一个使用 Webpack 打包的项目，来体验下 Webpack 的零配置打包。

初始化项目

首先是创建项目，创建一个名字为 `zero-config` 的文件夹，并且进入文件夹使用 `npm init` 进行初始化：

```
# 创建目录并且进入
mkdir zero-config && cd $_
# 初始化
npm init -y
# 安装 webpack 和 webpack-cli 到开发依赖
npm i webpack --save-dev
npm i webpack-cli --save-dev
```

创建目录

新建 `src` 目录，放置我们的源码，目录结构如下：

```
├─ src
│  ├── hello.js
│  ├── index.js
│  └─ world.js
```

`hello.js`、`world.js` 和 `index.js` 三个文件内容如下：

```
// hello
module.exports = 'hello';
// world
module.exports = 'world';
// index
const hello = require('./hello');
const world = require('./world');

console.log(`${hello} ${world}`);
```

这时候我们使用 `webpack` 命令（如果全局安装 `webpack-cli` 包的话）可以直接体验 Webpack 打包效果。

TIPS: `npx` 是一个方便开发者访问 `node_modules` 内的 `bin` 命令行的小工具，`npx webpack -v` 相当于执行了 `node ./node_modules/webpack/bin/webpack -v`，`npx` 在开发中非常方便，推荐安装：`npm install -g npx`。

npm scripts

下面，我们打开 `package.json` 文件，然后添加 `scripts` 字段：

```
"scripts": {
  "build": "webpack"
}
```

添加完保存后，在项目目录下执行 `npm run build`，看到如下执行结果：

```
Hash: f6316f74d848d93943f6
Version: webpack 4.29.6
Time: 108ms
Built at: 2019-03-25 21:24:47
    Asset      Size  Chunks             Chunk Names
main.js  1.02 KiB       0  [emitted]  main
Entrypoint main = main.js
[0] ./src/index.js 103 bytes {0} [built]
[1] ./src/hello.js 26 bytes {0} [built]
[2] ./src/world.js 26 bytes {0} [built]

WARNING in configuration
The 'mode' option has not been set, webpack will fallback to
'production' for this value. Set 'mode' option to 'development'
or 'production' to enable defaults for each environment.
You can also set it to 'none'
to disable any default behavior. Learn more:
https://webpack.js.org/concepts/mode/
```

执行成功，`index.js` 文件被打包到了 `dist` 文件夹下了，同时提示我们默认使用了 `production mode`，即生产环境，打开 `dist/main.js`，里面的代码的确是压缩的，说明是生产环境打包；下面继续修改 `package.json` 添加 `scripts`：

```
"scripts": {
  "dev": "webpack --mode development",
  "build": "webpack --mode production"
}
```

这时候在执行 `npm run dev`，结果如下：

```
Hash: 4ac0aa49ec6b0edb5545
Version: webpack 4.29.6
Time: 86ms
Built at: 2019-03-25 23:27:44
    Asset      Size  Chunks             Chunk Names
main.js  4.49 KiB       0  [emitted]  main
Entrypoint main = main.js
[./src/hello.js] 26 bytes {main} [built]
[./src/index.js] 103 bytes {main} [built]
[./src/world.js] 26 bytes {main} [built]
```

嗯，这次没有压缩，而且也不报 **warning** 了！

Tips: Webpack 的打包环境有 **production** 和 **development** 两种，分别对应生产环境和开发环境，生产环境默认配置包括压缩等常用的配置。

修改打包输出目录

通过上面的演示，我们得到下面的结论：

1. Webpack 默认的入口文件是 **src/index.js**；
2. Webpack 的默认输出目录是 **dist/main.js**。

我们如果要修改 Webpack 的默认输出目录，需要用到 Webpack 命令的 **--output**，我们将上面的 **npm scripts** 做下修改：

```
"scripts": {
  "dev": "webpack --mode development --output ./output/main.js",
  "build": "webpack --mode production --output ./output/main.js"
}
```

这时候在执行 **npm run build**，则 **webpack** 会将打包之后的文件输出到 **output/main.js** 路径了：

```
> webpack --mode production --output ./output/main.js

Hash: 9ee30ef43d177459d4fa
Version: webpack 4.29.6
Time: 84ms
Built at: 2019-03-24 15:29:07
    Asset      Size  Chunks             Chunk Names
main.js  1.02 KiB       0  [emitted]  main
Entrypoint main = main.js
[0] ./src/index.js 103 bytes {0} [built]
[1] ./src/hello.js 26 bytes {0} [built]
[2] ./src/world.js 26 bytes {0} [built]
```

使用 Loader 来做 ES6 语法转换

现在，我们将之前的代码用的是 **ES6** 的 **Module** 语法进行重写，这在普通的浏览器执行起来可能会因为语法「太先进」而报错，所以我们需要使用 **Babel** 来对 **ES6** 语法的代码进行转换。

在 **src** 下新建一个 **es** 的文件夹，将 **ES6** 语法的代码放进去。

```
// hello.es.js
export default 'hello';

// world.es.js
export default 'world';
// index.es.js
import hello from './hello';
import world from './world';

console.log(`${hello} ${world}`);
```

这是需要修改下 `package.json` 的 `scripts` 字段，将默认打包的入口文件（`src/index.js`）修改为 `src/es/index.js` 地址：

```
"scripts": {
  "dev": "webpack --mode development ./src/es/index.js",
  "build": "webpack --mode production ./src/es/index.js"
}
```

这时候执行 `npm run dev` 的打包出来的是 ES6 语法的代码，在一些老点的浏览器或者移动终端浏览器中都会报错，这时候就需要 Babel 来转换 ES6 语法了。

通过 Babel 来转换 ES6 语法

Babel 是 JavaScript 的一个编译器，能够将 ES6+ 语法转换为 ES5 语法，保证浏览器的兼容性，后面的章节还会继续介绍 Babel 的用法，这里为了快速入门 webpack 所以从简介绍。

Loader 是 Webpack 中的一个重要概念，通过 Loader 我们可以做一些跟文件自身属性相关的一些操作，比如这里做的 ES6 语法转换，用到的是 `babel-loader`，这个 Loader 依赖 `@babel/core` 和 `@babel/preset-env`。

所以首先安装它们：

```
npm i @babel/core babel-loader @babel/preset-env --save-dev
```

然后在项目的根目录下，创建一个 `babel` 的配置文件 `.babelrc`，内容如下：

```
{
  "presets": ["@babel/preset-env"]
}
```

有了 `babel-loader`，可以使用 `webpack` 命令的 `--module-bind` 来指定对应的文件需要经过怎样的 Loader 处理，所以继续修改 `npm scripts`：

```
"scripts": {
  "dev": "webpack --mode development ./src/es/index.js --module-bind js=babel-loader",
  "build": "webpack --mode production ./src/es/index.js --module-bind js=babel-loader"
}
```

添加完毕，执行 `npm run build`，看下 `dist/main.js` 的文件，就是转换后的 ES5 语法的打包结果了。

```
Hash: a63a726e81c194b5c84c
Version: webpack 4.29.6
Time: 790ms
Built at: 2019-03-26 01:39:58
    Asset      Size  Chunks             Chunk Names
  main.js  978 bytes          0  [emitted]  main
Entrypoint main = main.js
[0] ./src/es/index.js + 2 modules 137 bytes {0} [built]
|   ./src/es/index.js 91 bytes [built]
|   ./src/es/hello.js 23 bytes [built]
|   ./src/es/world.js 23 bytes [built]
```

Tips:

下面介绍几个 `webpack-cli` 的小技巧:

1. 当项目逐渐变大或者使用生产环境打包的时候, `Webpack` 的编译时间会变长, 可以通过参数让编译的输出内容带有进度和颜色: `webpack --progress --colors`;
2. `Webpack` 的配置比较复杂, 很容出现错误, 如果出问题, 会打印一些简单的错误信息, 我们还可以通过参数 `--display-error-details` 来打印错误详情: `webpack --display-error-details`;
3. 如果不想每次修改模块后都重新编译, 那么可以启动监听模式, 开启监听模式后, 没有变化的模块会在编译后缓存到内存中, 而不会每次都被重新编译, 所以监听模式的整体速度是很快的: `webpack --watch`;
4. `webpack-cli` 支持两个快捷选项: `-d` 和 `-p`, 分别代表一些常用的开发环境和生产环境的打包。

小结

至此, 我们已经学习并且演示了使用 `webpack-cli` 命令来完成 `Webpack` 的零配置打包, 后面章节我们继续介绍 `Webpack` 的打包配置。`webpack-cli` 命令的选项比较多, 详细可以通过 [webpack-cli 的文档](#) 进行查阅, 这里总结我们日常用的最多的几个选项 (options):

- `--config`: 指定一个 `Webpack` 配置文件的路径;
- `--mode`: 指定打包环境的 `mode`, 取值为 `development` 和 `production`, 分别对应着开发环境和生产环境;
- `--json`: 输出 `Webpack` 打包的结果, 可以使用 `webpack --json > stats.json` 方式将打包结果输出到指定的文件;
- `--progress`: 显示 `Webpack` 打包进度;
- `--watch, -w`: `watch` 模式打包, 监控文件变化之后重新开始打包;
- `--color, --colors / --no-color, --no-colors`: 控制台输出的内容是否开启颜色;
- `--hot`: 开启 `Hot Module Replacement` 模式, 后面会详细介绍;
- `--profile`: 会详细的输出每个环节的用时 (时间), 方便排查打包速度瓶颈。