Migrating to Webpack 2

Published on Oct 21, 2016 - Edit this page on GitHub

Webpack is on the verge of having its latest major version released, and it's expected to drop very soon. However, the main thing holding the release back is documentation, and the code is mostly written. I recently took the time to update our work project from Webpack 1 to 2, and thought I'd document the steps taken for anyone else who wants to make the move.

You can also check out the Migrating from V1 to V2 guide on the Webpack documentation.

Install Webpack 2

The first thing to do is install the latest version. Because it's not a stable release, you have to specify the exact beta version you'd like. At the time of writing it's 2.1.0-beta.25:

```
npm install --save-dev webpack@2.1.0-beta.25
```

If you're using any other Webpack plugins, be aware that they might need updating. For example, the Extract Text Plugin has a v2 in beta also:

```
npm install --save-dev extract-text-webpack-plugin@2.0.0-beta.4
```

`module.loaders` => `module.rules`

This is not a breaking change because `module.loaders` will continue to be supported, but in the future it will be deprecated in favour of `module.rules`. This is just an easy renaming step.

```
// before
modules: {
  loaders: [...]
}

// after
modules: {
  rules: [...]
}
```

`resolve.modulesDirectories` => `resolve.modules`

Another renaming step, the `resolve` options have been renamed:

```
// before
resolve: {
  modulesDirectories: [...],
}

// after
resolve: {
  modules: [...],
}
```

No `webpack.optimize.OccurenceOrderPlugin`

It's now included by default, so there is no need to have this in our config.

Configuring loaders

At work we're using postcss and [postcss-loader](#) to load our CSS through Webpack. The loader used to expect a top level `postcss` key in the Webpack config. As of Webpack 2 this is no longer allowed; we can instead define an `options` key when we configure the loader. This replaces the `query` option from Webpack 1. Any plugin that looked for top level configuration will have to be swapped to this style.

```
// before, in Webpack top level
postcss: {
  plugins: ...
}

// after
module: {
  rules: [{
    test: /\.scss$/,
    use: [
      {
        loader: 'postcss-loader',
        options: {
          plugins: ...
        }
      },
      'sass-loader'
    ]
  }]
}
```

ExtractTextPlugin changes

The above change to loader configuration also makes it way easier to configure multiple loaders; previously it would only be possible to pass an array of loaders in string form to some plugins, such as `ExtractTextPlugin`:

```
// Webpack 1
ExtractTextPlugin.extract(
```

```
  'style-loader',
  'css-loader!postcss-loader!sass-loader'
);
```

This quickly got very hard to work with if you had to pass options:

```
// Webpack 1
ExtractTextPlugin.extract(
  'style-loader',
  'css-loader?modules-true!postcss-loader!sass-loader'
);
```

But now Webpack 2 can deal with arrays of objects to configure loaders. We can replace the above with:

```
// Webpack 2
var loaders = [
  {
    loader: 'css-loader',
    options: {
      modules: true
    }
  },
  {
    loader: 'postcss-loader'
  },
  {
    loader: 'sass-loader'
  }
]
```

Whereas in Webpack 1 we used the key `query` for configuring loaders, we now use `options`. `ExtractTextPlugin` can now take this array, rather than only allowing the string form:

```
// Webpack 2
ExtractTextPlugin.extract({
```

```
  fallbackLoader: 'style-loader',
  loader: loaders,
})
```

Stop Babel from compiling ES2015 modules

Webpack 1 wasn't able to parse ES2015 modules, so Babel would convert them into CommonJS. Webpack 2 can parse ES2015 modules, and is able to eliminate dead code based on which modules are used, so it's recommended that you tell Babel not to convert modules into CommonJS. You can do this by changing your `.babelrc`:

```
// before
"presets": ["es2015"]

// after
"presets": [
  ["es2015", { "modules": false }]
]
```

We've seen a good file size saving by doing this, and hopefully this will continue to improve in the future!

Fin

Webpack 2 offers better performance, improved bundling and a much nicer experience when configuring it. Given that the code is so stable, despite its beta status, I highly recommend giving it a go on your projects when you can.

Thank you for visiting the JavaScript Playground and reading this article. If you have any questions, please feel free to tweet me, or ask on GitHub. If you enjoyed this article please feel free to share it on Twitter.

Subscribe to keep up to date with the latest content.

Join the JS Playground mailing list to be kept up to date with the latest tutorials, screencasts and more. You won't be spammed and you can unsubscribe at any time.

email address

SIGN UP

Jack is a Senior JavaScript Developer at Songkick in London where he builds JavaScript applications using Angular, React and more. He's a keen Elm enthusiast and a Google Developer Expert.

Follow @Jack_Franklin

Info icon by Jürgen Bauer from the Noun Project