# webpack-examples (0.0.1)

Maksim Kostromin

Version 0.0.1, 2018-06-28 22:34:13 UTC

# Table of Contents

Travis CI status:

# Chapter 1. Introduction

Read reference documentation

webpack 4 examples:

- starter without config file
- basic starter
- add html-webpack-plugin
- add webpack-dev-server
- add css-loader and style-loader
- process html and using html / file loaders
- using `extract-text-webpack-plugin` (deprecated in order to `mini-css-extract-plugin`)
- production build: minify everything, extract css...
- using `optimize-css-assets-webpack-plugin`
- using webpack config as a function
- babel webpack config
- compression webpack config
- react basic webpack config
- webpack optimization and code splitting

# Chapter 2. Quickstart

## 2.1. no config

*using webpack without* `webpack.config.js` *file*

```
mkdir starter-no-config
cd starter-no-config/

mkdir src dist
echo "console.log('hey!');" >> src/index.js
echo "<script src='./main.js'></script>" >> dist/index.html

npm init -y
npm i -g webpack webpack-cli
#npm i -DE webpack webpack-cli

webpack --mode=development
webpack --mode production
```

## 2.2. basic config

*using webpack with basic config file:* `config/webpack.dev.js`

```
const { resolve } = require('path');

module.exports = {
  entry: {
    main: './src/index.js'
  },
  mode: 'development',
  output: {
    filename: '[name]-bundle.js',
    path: resolve(__dirname, '../dist'),
  },
};
```

## 2.3. add HTML plugin

*install html-webpack-plugin*

```
npm i -DE html-webpack-plugin
```

*update webpack condig*

```
const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  // ...
  plugins: [
    new HtmlWebpackPlugin({
      template: './src/index.html',
      favicon: './src/favicon.ico',
    }),
  ],
};
```

## 2.4. add webpack-dev-server

*update webpack condig*

```
const { resolve, join } = require('path');

module.exports = {
  // ...
  devServer: {
    contentBase: join(__dirname, '../dist'),
    // show / overlay errors in browser
    overlay: true,
  },
};
```

*install and run webpack-dev-server*

```
npm i -DE webpack-dev-server
webpack-dev-server --config config/webpack.dev.js
```

## 2.5. add css-loader and style-loader

*add some css styles*

```
body {
  margin: 0;
  background-color: #444;
}

/* lets centred content using flex */
#app {
  height: 100vh;
  display: flex;
  align-items: center;
  justify-content: center;
}

/* font styling */
h1 {
  color: white;
  font-size: 3em;
  font-family: Helvetica, sans-serif, 'DejaVu Sans', Arial;
  text-shadow: 0 0 25px white;
}
```

These loaders will apply in reverse order: first, `css-loader`, next: `style-loader`

*install css-loader and style-loader*

```
npm i -DE css-loader style-loader
```

*update webpack condig*

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

# 2.6. process html and using html / file loaders

*prepare* `./src/index.html`

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Webpack | Examples</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="shortcut icon" href="favicon.ico" type="image/x-icon">
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div id="app"></div>
  <script src="main-bundle.js"></script>
</body>
</html>
```

Few thinks are happening here: First, `html-loader` will do necessary linting of *.html files. Then `extract-loader` will tell webpack do not include it in result bundle.js file. And finally `file-loader` will put extracted content in output dir accordingly

*remove useless plugin and install requires loaders:* `html-loader`, `extract-loader` *and* `file-loader`

```
npm rm -DE html-webpack-plugin
npm i -DE html-loader extract-loader file-loader
```

*update webpack condig*

```
// const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  // ...
  /*
  plugins: [
    new HtmlWebpackPlugin({
      template: './src/index.html',
      favicon: './src/favicon.ico',
    }),
  ],
  module: {
    rules: [
      {
        test: /\.css$/i,
        use: [
          { loader: 'style-loader' },
          { loader: 'css-loader' },
```

```
        ],
      },
    ],
  },
  */
  module: {
    rules: [
      {
        test: /\.css$/i,
        use: [
          {
            loader: 'file-loader',
            options: {
              name: '[name].[ext]',
            },
          },
          { loader: 'extract-loader' },
          { loader: 'css-loader' },
        ],
      },
      {
        test: /\.html$/i,
        use: [
          {
            loader: 'file-loader',
            options: {
              name: '[name].[ext]',
            },
          },
          { loader: 'extract-loader' },
          { loader: 'html-loader' },
        ],
      },
      {
        test: /\.(ico)$/i,
        use: [
          {
            loader: 'file-loader',
            options: {
              name: '[name].[ext]',
            },
          },
        ],
      },
    ],
  },
};
```

*finally combine all together in `./src/index.js`*

```
require('./index.html');
require('./styles.css');
require('./favicon.ico');
```

I wont use that approach, I prefer `HtmlWebpackPlugin` and `ExtractTextWebpackPlugin`

## 2.7. using extract-text-webpack-plugin

*install*

```
npm rm -ED style-loader
npm i -DE extract-text-webpack-plugin@next css-loader
# also add less support
npm i -DE less-loader less
```

*update webpack condig*

```
const ExtractTextPlugin = require('extract-text-webpack-plugin');
const cssContent = new ExtractTextPlugin('[name]-[hash:8].css');
const lessContent = new ExtractTextPlugin('[name].less-[hash:8].css');

module.exports = {
  // ...
  module: {
    rules: [
      {
        test: /\.css$/i,
        use: cssContent.extract([
          'css-loader',
        ]),
      },
      {
        test: /\.less$/i,
        use: cssContent.extract([
          'css-loader',
          'less-loader',
        ]),
      },
    ],
  },
  plugins: [
    cssContent,
    lessContent,
  ],
};
```

There are better option available for using instead of extract text plugin: `mini-css-extract-plugin`

## 2.8. using mini-css-extract-plugin

*install*

```
npm rm extract-text-webpack-plugin
npm i -DE mini-css-extract-plugin
```

*update webpack condig*

```
const mode = process.env.NODE_ENV || 'production';
const isProduction = mode === 'production';
const MiniCssExtractPlugin = require('mini-css-extract-plugin');

module.exports = {
  mode,
  // ...
  module: {
    rules: [
      {
        test: /\.css$/i,
        use: [
          isProduction ? MiniCssExtractPlugin.loader : 'style-loader',
          {
            loader: 'css-loader',
            options: {
              importLoaders: 1,
              minimize: isProduction,
            },
          },
          // 'postcss-loader',
          {
            loader: 'postcss-loader',
            options: {
              ident: 'postcss',
            },
          },
        ],
      },
    ],
  },
  plugins: [
    new MiniCssExtractPlugin({
      filename: '[name]-[hash:8].css',
    }),
  ],
};
```

*update package.json*

```
{
  "scripts": {
    "dev": "cross-env NODE_ENV=development webpack",
    "build": "cross-env NODE_ENV=production webpack",
    "start": "cross-env NODE_ENV=development webpack-dev-server --open"
  }
}
```

## 2.9. using optimize-css-assets-webpack-plugin

THis plugin solves extract-text-webpack-plugin CSS duplication problem

*install*

```
npm i -DE optimize-css-assets-webpack-plugin
```

*update webpack condig*

```
const OptimizeCssPlugin = require('optimize-css-assets-webpack-plugin');

module.exports = {
  // ...
  plugins: [
    isProduction ? new OptimizeCssPlugin() : undefined,
  ].filter(p => !!p),
};
```

## 2.10. using webpack config as a function

*remove useless packages*

```
npm rm cross-env
```

*update webpack condig*

```
const safety = env => env || process.env || {};
const mode = env => safety(env).NODE_ENV || 'production';
const isProduction = env => 'production' === mode(env);

module.exports = env => ({
  mode: mode(env),
  module: {
    rules: [
      {
        test: /\.css$/i,
        use: [
          isProduction(env) ? MiniCssExtractPlugin.loader : 'style-loader',
          {
            loader: 'css-loader',
            options: { importLoaders: 1 },
          },
          'postcss-loader',
          {
            loader: 'postcss-loader',
            options: {
              ident: 'postcss',
            },
          },
        ],
      },
    ],
  },
  plugins: [
    isProduction(env) ? new OptimizeCssPlugin() : undefined,
    new webpack.DefinePlugin({
      'process.env': {
        NODE_ENV: JSON.stringify(mode(env)),
      },
    })
  ].filter(p => !!p),
});
```

## 2.11. babel setup (compare babel minify with uglifyjs)

*install required packages*

```
npm i -ED babel-core babel-loader babel-preset-env babel-preset-stage-0 babel-minify-
webpack-plugin babel-plugin-transform-runtime babel-polyfill uglifyjs-webpack-plugin
```

*prepare .babelrc*

```
{
  "presets": [
    "stage-0",
    [
      "env",
      {
        "debug": false,
        "targets": {
          "browsers": [
            "last 2 versions",
            "safari >= 7",
            "chrome >= 52",
            "firefox >= 48"
          ]
        }
      }
    ]
  ],
  "plugins": [
    "transform-runtime",
    "syntax-dynamic-import"
  ]
}
```

*update webpack condig*

```
const BabelMinifyPlugin = require('babel-minify-webpack-plugin');
// const UglifyJsPlugin = require('uglifyjs-webpack-plugin');

module.exports = env => ({
  entry: {
    main: [
      // 'babel-polyfill',
      './src/index.js',
    ],
  },
      {
        test: /\.css$/i,
        use: [
          isProduction(env) ? MiniCssExtractPlugin.loader : 'style-loader',
          {
            loader: 'css-loader',
            options: { importLoaders: 1 },
          },
          'postcss-loader',
          {
            loader: 'postcss-loader',
            options: {
              ident: 'postcss',
            },
          },
        ],
      },
    ],
  },
  plugins: [
    // you decide with one is better:
    isProduction(env) ? new BabelMinifyPlugin() : undefined,
    // isProduction(env) ? new UglifyJsPlugin() : undefined,
  ].filter(p => !!p),
});
```

## 2.12. compression files

*install required packages*

```
npm i -ED compression-webpack-plugin
npm i -ED brotli-webpack-plugin
```

*update webpack condig*

```
const CompressionPlugin = require('compression-webpack-plugin');
const BrotliPlugin = require('brotli-webpack-plugin');

module.exports = env => ({
  // ...
  plugins: [
    isProduction(env) ? new CompressionPlugin({ algorithm: 'gzip' }) : undefined,
    isProduction(env) ? new BrotliPlugin() : undefined,
  ].filter(p => !!p),
});
```

## 2.13. react (basic webpack setup)

*install required packages*

```
npm i -E react react-dom react-router react-router-dom
npm i -ED babel-preset-react babel-runtime babel-register webpack-hot-middleware
babel-plugin-syntax-dynamic-import
```

*prepare .babelrc*

```
{
  "presets": [
    "react",
    "stage-0",
    [
      "env",
      {
        "debug": false,
        "targets": {
          "browsers": [
            "last 2 versions",
            "safari >= 7",
            "chrome >= 52",
            "firefox >= 48"
          ]
        }
      }
    ]
  ],
  "plugins": [
    "transform-runtime",
    "syntax-dynamic-import"
  ]
}
```

*update webpack condig*

```
module.exports = env => ({
  module: {
    rules: [
      {
        test: /\.jsx?$/i,
        use: [
          'babel-loader',
        ],
        include: resolve(__dirname, 'src'),
      },
      {
        test: /\.(ico|jpe?g|png|gif)$/i,
        use: [
          {
            loader: 'file-loader',
            options: {
              name: isProduction(env) ? '[hash].[ext]' : '[path][name].[ext]',
            },
          },
        ],
      },
    ],
  },
  // ...
});
```

## 2.14. webpack optimization and code (chunks) splitting

*update webpack condig*

```
module.exports = env => ({
  optimization: {
    splitChunks: {
      chunks: 'all',
    }
  },
  // ...
});
```

# Chapter 3. Links

- Asciidoctor reference
- GitHub repo: lawwantsin/webpack-course
- GitHub
- Webpack documentation
- HTML webpack plugin documentation
- HTML webpack-dev-server documentation
- extract-loader
- extract-text-webpack-plugin
- optimize-css-assets-webpack-plugin