EBook Gratis

APRENDIZAJE webpack

Free unaffiliated eBook created from
Stack Overflow contributors.

#webpack

Tabla de contenido

Acerca de ... 1

Capítulo 1: Empezando con el webpack ... 2

Observaciones ... 2

Versiones ... 2

Examples ... 3

Instalación ... 3

Ejemplo de webpack.config.js con babel ... 4

Ejemplo de Javascript + CSS + Fuentes + Imágenes ... 5

Ejemplo simple de webpack ... 6

Webpack, React JSX, Babel, ES6, configuración simple ... 6

Configuración de webpack simple con Node.js ... 7

Capítulo 2: Cargadoras ... 10

Observaciones ... 10

Examples ... 10

Config. Utilizando preLoader para eslint, babel para jsx y css loader chaining ... 10

Capítulo 3: Cargadores y complementos ... 12

Observaciones ... 12

Examples ... 12

Empezando con los cargadores ... 12

cargando archivos mecanografiados ... 14

Capítulo 4: DllPlugin y DllReferencePlugin ... 15

Introducción ... 15

Sintaxis ... 15

Examples ... 15

Configuración del proveedor (DllPlugin) ... 15

Hacer referencia a un paquete Dll (DllReferencePlugin) ... 16

Capítulo 5: Reemplazo de módulo caliente ... 18

Observaciones ... 18

webpack-hot-middleware ... 18

Config ... 18

Examples ... 18
Usar con webpack-dev-middleware ... 18
Habilitar HMR para el módulo ... 19
Usar con webpack-dev-server ... 20
Capítulo 6: Sacudida de arbol ... 22
Examples ... 22
Sacudida de árboles ES2015 ... 22
Instalar ... 22
Uso ... 22
Capítulo 7: Servidor de desarrollo: webpack-dev-server ... 23
Examples ... 23
Instalación ... 23
Usando proxy ... 23
volver a escribir ... 24
filtrar ... 24
Capítulo 8: Uso de Webpack ... 26
Examples ... 26
Ejemplo de uso de módulos de CommonJS ... 26
Ejemplo de uso de módulos AMD ... 26
Ejemplo de uso de módulos ES6 (Babel) ... 27
Ejemplo de uso de módulos ES6 (Typescript) ... 28
Creditos ... 29

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: webpack

It is an unofficial and free webpack ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official webpack.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con el webpack

Observaciones

Webpack es un paquete de módulos que lee módulos con dependencias y produce activos estáticos que representan esos módulos.

Cuenta con un *sistema de cargador* extensible que permite que los paquetes incluyan no solo los recursos de Javascript, sino también CSS, imágenes, HTML y mucho más.

Por ejemplo, usando el cargador de Javascript incorporado, css-loader y url-loader :

```
require("./code.js") // Load Javascript dependency
var css = require("./styles.css"); // Load CSS as a string
var base64Image = require("./image.png"); // Load an image as a base64 string
```

Se convertiría en un único archivo empaquetado:

```
// From code.js
console.log("Hello, World");
// From styles.css
var css = "body { margin: 0; padding: 0; } h1 { color: #FF0000; }";
// From image.png
var base64Image =
"
```

Las dependencias se pueden definir en cualquiera de los estilos de módulos más comunes (CommonJS y AMD).

Versiones

Versión	Fecha de lanzamiento
3.0.0	2017-06-19
2.6.1	2017-05-25
2.6.0	2017-05-23
2.5.1	2017-05-07
2.5.0	2017-05-04
2.4.1	2017-04-14
2.4.0	2017-04-14

Versión	Fecha de lanzamiento
1.13	2016-04-17
1.12	2015-08-25
1.11	2015-08-06
1.10	2015-06-27
1.9	2015-05-10
1.8	2015-04-29
1.7	2015-03-11
1.6	2015-02-24
1.5	2015-01-21
1.4	2014-12-28
1.3	2014-08-25
1.2	2014-05-27
1.1	2014-05-17
1.0	2014-03-01
0.11	2013-12-31
0.10	2013-06-19
0.9	2013-03-19
0.8	2013-01-21

Examples

Instalación

Requisitos previos:

NodeJS y npm

Hay dos formas de instalar Webpack: globalmente o por proyecto. Es mejor tener la dependencia instalada por proyecto, ya que esto le permitirá usar diferentes versiones de webpack para cada proyecto y no requiere que el usuario tenga instalado webpack globalmente.

Instalación por proyecto

Ejecute el siguiente comando desde la carpeta raíz de su proyecto:

```
npm install webpack --save-dev
```

A continuación, puede ejecutar el ejecutable webpack instalado en `node_modules` :

```
./node_modules/.bin/webpack
```

O cree un script NPM en su archivo `package.json` , donde puede omitir la parte `node_modules` : npm es lo suficientemente inteligente como para incluir esa carpeta en su PATH.

```
// in package.json:
{
  ...
  "scripts": {
    "start": "webpack"
  },
  ...
}

// from terminal:
npm start
```

Instalando globalmente

Ejecute el siguiente comando cuando se le solicite:

```
npm install webpack -g
```

Ejemplo de webpack.config.js con babel

Dependencias

```
npm i -D webpack babel-loader
```

webpack.config.js

```
const path = require('path');

module.exports = {
  entry: {
    app: ['babel-polyfill', './src/'],
  },
  output: {
    path: __dirname,
    filename: './dist/[name].js',
  },
  resolve: {
    extensions: ['', '.js'],
  },
```

```
  module: {
    loaders: [{
      test: /\.js$/,
      loaders: ['babel-loader'],
      include: path.resolve(__dirname, 'src')
    }],
  }
};
```

Ejemplo de Javascript + CSS + Fuentes + Imágenes

Módulos requeridos

```
npm install --save-dev webpack extract-text-webpack-plugin file-loader css-loader style-loader
```

Estructura de la carpeta

```
.
└── assets
    ├── css
    ├── images
    └── js
```

webpack.config.js

```
const webpack = require('webpack');
const ExtractTextPlugin = require('extract-text-webpack-plugin');
const path = require('path');
const glob = require('glob');

module.exports = {
  entry: {
    script: path.resolve(__dirname, './assets/js/app.js'),
    style: path.resolve(__dirname, './assets/css/app.css'),
    images: glob.sync(path.resolve(__dirname, './assets/images/**/*.*')),
  },
  context: __dirname,
  output: {
    path: path.resolve('./dist/assets'),
    publicPath: '/dist/assets',
    filename: '[name].js',
  },
  module: {
    loaders: [
      {
        test: /\.css$/,
        loader: ExtractTextPlugin.extract({
            fallback: 'style-loader',
            use: 'css-loader'
        }),
      },
      {
        test: /(\.woff2?|\.woff|\.ttf|\.eot|\.svg)(\?v=\d+\.\d+\.\d+)?$/,
        loader: 'file-loader?name=[name]-[hash:6].[ext]',
      },
      {
```

```
        test: /\.(png|jpe?g|gif|ico)$/,
        loader: 'file-loader?name=[name].[ext]',
      },
    ],
  },
  plugins: [
    new ExtractTextPlugin('app.css' /* optional: , { allChunks: true } */),
  ],
};
```

`glob.sync('./assets/images/**/*.*')` requerirá todos los archivos en la carpeta de imágenes como entrada.

`ExtractTextPlugin` tomará el resultado generado y creará un archivo `css` empaquetado.

Ejemplo simple de webpack

El mínimo requerido para usar Webpack es el siguiente comando:

```
webpack ./src/index.js ./dist/bundle.js

// this is equivalent to:

webpack source-file destination-file
```

El paquete web tomará el archivo de origen, compilará el destino de salida y resolverá las dependencias en los archivos de origen.

Webpack, React JSX, Babel, ES6, configuración simple

Asegúrese de instalar las dependencias npm correctas (babel decidió dividirse en un montón de paquetes, algo que ver con las "dependencias de pares"):

```
npm install webpack webpack-node-externals babel-core babel-loader babel-preset-react babel-preset-latest --save
```

`webpack.config.js` :

```
module.exports = {
    context: __dirname, // sets the relative dot (optional)
    entry: "./index.jsx",
    output: {
        filename: "./index-transpiled.js"
    },
    module: {
        loaders: [{
            test: /\.jsx$/,
            loader: "babel?presets[]=react,presets[]=latest" // avoid .babelrc
        }]
    }, // may need libraryTarget: umd if exporting as a module
    externals: [require("webpack-node-externals")()], // probably not required
    devtool: "inline-source-map"
};
```

`webpack-node-externals` es una función que escanea sus `node_modules` y garantiza que no se compilen y agrupen junto con su código de front-end, aunque asegura que el paquete conserva la referencia a ellos. Esto ayuda a una transpilación más rápida, ya que no está recodificando bibliotecas.

Configuración de webpack simple con Node.js

Estructura de la carpeta

```
.
├── lib
├── modules
|   ├── misc.js
|   ├── someFunctions.js
├── app.js
├── index.html
├── package.json
├── webpack.config.js
└── webserver.js
```

paquete.json

```
{
  "name": "webpack-example-with-nodejs",
  "version": "1.0.0",
  "description": "Example using webpack code-splitting with some Node.js to support the
example",
  "main": "webserver.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "@Gun",
  "license": "ISC",
  "devDependencies": {
    "body-parser": "^1.17.1",
    "express": "^4.15.2",
    "http": "0.0.0",
    "morgan": "^1.8.1",
    "multer": "^1.3.0",
    "webpack": "^2.4.1"
  }
}
```

webpack.config.js

```
var path = require('path'); // used to get context

module.exports = {
    context: path.join(__dirname, 'app'), // resolves entry below, must be absolute path
    entry: './app.js', // entry point or loader for the application
    output: {
        path: path.join(__dirname, 'app/lib'), // express static folder is at /app/lib
        filename: '[name].bundle.js', // the file name of the bundle to create.  [name] is
replaced by the name of the chunk (code-splitting)
        publicPath: 'static' // example uses express as the webserver
    }
```

```
};
```

servidor web.js

```
var express = require('express'),
    path = require('path'),
    bodyParser = require('body-parser'),
    multer = require('multer')()
    logger = require('morgan'),
    fs = require('fs'),
    http = require('http');

var app = express();
var port = 31416;

app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
app.use(logger('short'));
app.use('/jsBundles',express.static('lib'));
app.get('/', function(request, response){
    response.sendFile(__dirname + '/index.html');
});

var server = http.createServer(app).listen(port, function(){
    console.log("I feel a disturbance in the port:" + port);
});
```

index.html

```
<!DOCTYPE html>
<html>
    <body>
        <div id="someValue"><label for="num">Enter a number:</label><input id="num" /></div>
        <div class="buttonList">
            <ul>
                <li><button id="doubleIt">double it</button></li>
                <li><button id="tripleIt">triple it</button></li>
            </ul>
        </div>
        <div id="someOtherValue">
            And the count shall be: <span id="theCount"></span>
        </div>
        <script src="/jsBundles/main.bundle.js"></script>
    </body>
</html>
```

app.js

```
require(['./modules/someFunctions'],function(){
        window.onload = function(){
                var someFunctions  = require('./modules/someFunctions');
                document.getElementById('doubleIt').onclick = function(){
                        var num = document.getElementById('num').value;
                        document.getElementById('theCount').innerHTML =
someFunctions.Double(num);
                };
```

```
                document.getElementById('tripleIt').onclick = function(){
                        var num = document.getElementById('num').value;
                        document.getElementById('theCount').innerHTML =
someFunctions.Triple(num);
                };
        };
});
```

misc.js

```
var self = {};
self.isNumber = function(value){
    // http://stackoverflow.com/questions/9716468/is-there-any-function-like-isnumeric-in-
javascript-to-validate-numbers
    return !isNaN(parseFloat(value)) && isFinite(value);
};
module.exports = self;
```

someFunctions.js

```
require(['./misc'], function(){
    var misc= require('./misc');

    var self = {};
    self.Double = function(value){
        if(!misc.isNumber(value)){
            return 0;
        };
        return value*2;
    }

    self.Triple = function(value){
        if(!misc.isNumber(value)){
            return 0;
        };
        return value*3;
    }

    module.exports = self;
});
```

NOTA

ejecute ***npm i --save-dev*** para instalar dependencias

ejecutar el ***nodo. \ node_modules \ webpack \ bin \ webpack.js*** una vez que las dependencias estén instaladas

ejecute el ***nodo webserver.js*** para iniciar el servidor

Lea Empezando con el webpack en línea:
https://riptutorial.com/es/webpack/topic/924/empezando-con-el-webpack

Capítulo 2: Cargadoras

Observaciones

Los cargadores de paquetes web se pueden configurar como "preLoaders", "loaders" y "postLoaders". Aunque no tienen que serlo, las configuraciones que utilizan linting u otras operaciones imperativas o en serie pueden aprovechar estas etapas de compilación en la tubería.

La clave para comprender los cargadores y su uso es que Webpack ejecutará cada módulo en el gráfico de requisitos a través del sistema del cargador. Siguiendo el ejemplo anterior, esto significa que a medida que Webpack comience a rastrear a través de las importaciones de su aplicación, identificará los archivos requeridos y utilizando una expresión regular simple, determinará qué archivo o tipo de archivo requiere qué cargador o serie de cargadores.

Arriba puede ver que todos los archivos ".js" o ".jsx" serán esbozados por el eslint-loader en la fase preLoader. Otros tipos de archivos `js|x` también se ejecutarán a través del babel-loader en la fase de carga principal. Además, en la misma fase, cualquier archivo `.scss` se cargará en el sass-loader . Esto le permite importar archivos Sass en sus módulos JS y hacer que se envíen al paquete JS resultante o incluso a otro paquete CSS independiente (mediante un complemento).

Nota: la importación de archivos `.scss` solo funcionará con Webpack y un cargador adecuado. Node no entenderá este tipo de importación sin un preprocesador o transpiler.

También es de destacar en el ejemplo `.scss` la capacidad de "encadenar" cargadores utilizando el `!` Signo de exclamación como "tubo" entre diferentes cargadores. El ejemplo anterior canaliza la salida del "sass-loader" al "css-loader" y finalmente al "style-loader" Esto también podría realizarse con una serie de `loaders: ['style-loader', 'css-loader', 'sass-loader']` . También hay diferentes opciones disponibles para las definiciones del cargador en línea y siguen la sintaxis de los parámetros de consulta que se encuentran comúnmente en las URL.

Consulte también: https://webpack.github.io/docs/loaders.html

Examples

Config. Utilizando preLoader para eslint, babel para jsx y css loader chaining.

La siguiente configuración se puede usar como configuración básica para agrupar su proyecto como una biblioteca. Observe cómo la configuración del módulo contiene una lista de preLoaders y cargadores.

```
// webpack.config.js

var path = require('path');

module.exports = {
    entry: path.join(__dirname, '..', 'src/index.js'),
    output: {
```

```
        path: path.join(__dirname, '..', '/lib'),
        filename: outputFile,
        library: 'myCoolBundle.js',
        libraryTarget: 'umd',
        umdNamedDefine: true
    },
    module: {
        preLoaders: [
            {
                test: /(\.jsx|\.js)$/,
                loader: "eslint-loader",
                exclude: /node_modules/,
            }
        ],
        loaders: [
            {
                test: /(\.jsx|\.js)$/,
                loader: ['babel'],
                exclude: /(node_modules)/,
                include: path.join(__dirname, '..'),
                query: {
                    presets: [ 'es2015', 'react']
                }
            },
            {
                test: /\.scss$/,
                loaders: ["style-loader", "css-loader!sass-loader"]
            }
        ]
    },
    resolve: {
        root: path.resolve(__dirname, '..', './src'),
        extensions: ['', '.js', '.jsx', '.scss'],
        fallback: path.join(__dirname, '../node_modules')
    },
    eslint: {
        configFile: path.resolve(__dirname, '..', '.eslintrc'),
    }
};
```

Lea Cargadoras en línea: https://riptutorial.com/es/webpack/topic/6534/cargadoras

Capítulo 3: Cargadores y complementos

Observaciones

Los cargadores y complementos constituyen los bloques de construcción de Webpack.

Los cargadores suelen estar delegados a una sola tarea y tipo de archivo. Son más fáciles de configurar y, por lo general, requieren menos código repetitivo.

Los complementos, por otro lado, tienen acceso al sistema de compilación interno de Webpack a través de enlaces y, por lo tanto, son más potentes. Los complementos pueden modificar el entorno de Webpack totalmente configurado y pueden realizar acciones personalizadas a lo largo del proceso de compilación.

Al tratar con nuestros archivos CSS, por ejemplo, un cargador puede usarse para agregar automáticamente prefijos de proveedores a las propiedades, mientras que un complemento se puede usar para producir una hoja de estilo minificada en el proceso de compilación del agrupador.

Examples

Empezando con los cargadores

Para comenzar, `npm install` los cargadores deseados para tu proyecto.

Dentro del objeto de configuración que se exporta en `webpack.config.js` , una propiedad de `module` contendrá todos sus cargadores.

```
const source = `${__dirname}/client/src/`;

module.exports = {
  // other settings here

  module: {
    loaders: [
          {
            test: /\.jsx?$/,
            include: source,
            loaders: ['babel?presets[]=es2015,presets[]=react', 'eslint']
          },
          {
            test: /\.s?css$/,
            include: source,
            loaders: ['style', 'css', 'autoprefixer', 'sass']
          }
        ]
  },
};
```

En la configuración anterior, estamos usando tres configuraciones básicas para nuestros

cargadores:

- **prueba: aquí** es donde unimos cargadores a extensiones específicas usando RegExp. El primer conjunto de cargadores se está ejecutando en todos los archivos .js y .jsx. El segundo conjunto se está ejecutando en todos los archivos .css y .scss.
- **include:** Este es el directorio en el que queremos que se ejecuten nuestros cargadores. Opcionalmente, podríamos usar la propiedad `exclude` la misma facilidad para definir directorios que no queremos que se incluyan.
- **cargadores:** esta es una lista de todos los cargadores que queremos ejecutar en los archivos especificados en `test` e `include` .

Es importante tener en cuenta que los cargadores se ejecutan de derecha a izquierda en cada matriz de cargadores y de abajo hacia arriba en las definiciones individuales. El siguiente código ejecutará los cargadores en el siguiente orden: `sass, autoprefixer, css, style` .

```
loaders: [
  {
    test: /\.s?css$/,
    include: source,
    loaders: ['style', 'css', 'autoprefixer']
  },
  {
    test: /\.s?css$/,
    include: source,
    loaders: ['sass']
  }
]
```

Esta es una fuente común de confusión y errores para los desarrolladores que son nuevos en el paquete web. Por ejemplo, cuando usamos la sintaxis JSX y ES6, queremos eliminar ese código, *no* agregar el código compilado que proporciona nuestro cargador de babel. Por lo tanto, nuestro cargador de eslint se coloca a la derecha de nuestro cargador de babel.

El sufijo `-loader` es opcional al listar nuestros cargadores.

```
loaders: ['sass']
```

... es equivalente a:

```
loaders: ['sass-loader']
```

Alternativamente, puede usar la propiedad del `loader` (singular) junto con una cadena que separa la lista de cargadores por el `!` personaje.

```
loaders: ['style', 'css']
```

... es equivalente a:

```
loader: "style!css"
```

cargando archivos mecanografiados

Para utilizar mecanografiado con un paquete web, necesita instalar `typescript` y `ts-loader`

```
npm --save-dev install typescript ts-loader
```

Ahora puedes configurar webpack para usar archivos mecanografiados

```
// webpack.config.js

module.exports = {
  ..
  resolve: {
    // .js is required for react imports.
    // .tsx is required for react tsx files.
    // .ts is optional, in case you will be importing any regular ts files.
    extensions: ['.js', '.ts', '.tsx']
  },
  module: {
    rules: [
      {
        // Set up ts-loader for .ts/.tsx files and exclude any imports from node_modules.
        test: /\.tsx?$/,
        loaders: isProduction ? ['ts-loader'] : ['react-hot-loader', 'ts-loader'],
        exclude: /node_modules/
      }
    ]
  },
  ...
};
```

Lea Cargadores y complementos en línea:
https://riptutorial.com/es/webpack/topic/5651/cargadores-y-complementos

Capítulo 4: DllPlugin y DllReferencePlugin

Introducción

Los complementos Dll y DllReference permiten que el código se divida en varios paquetes de una manera que los paquetes se pueden compilar de forma independiente.

Es posible crear scripts de "proveedores" en una biblioteca que no necesita compilarse a menudo (por ejemplo, React, jQuery, Bootstrap, Fontawesome ...) y hacer referencia a ellos en el paquete de aplicaciones que los necesitará.

El paquete de la aplicación, el que se cambiará constantemente, estará en una configuración separada que solo hará referencia a un paquete "vendedor" ya creado.

Sintaxis

- nuevo webpack.DllPlugin ({ruta: '[nombre] -manifest.json', nombre: '[nombre] _ [hash]'})
- nuevo webpack.DllReferencePlugin ({context: __dirname, manifest: require ('./ packname-manifest.json')})

Examples

Configuración del proveedor (DllPlugin)

Nota: La `output.library` y la `output.library` y el `name` (en DllPlugin) deben ser los mismos.

```
const path = require('path');
const webpack = require('webpack');
const ExtractTextPlugin = require('extract-text-webpack-plugin');
const extractCSS = new ExtractTextPlugin('vendor.css');
const isDevelopment = process.env.NODE_ENV !== 'production';

module.exports = {
  resolve: {
    extensions: ['.js'],
  },
  module: {
    rules: [
      { test: /\.(png|woff|woff2|eot|ttf|svg)$/, loader: 'url-loader?limit=100000' },
      { test: /\.s?css$/i, loader: extractCSS.extract(['css-loader?minimize', 'sass-loader'])
},
      { test: /\.json$/, loader: 'json-loader' },
    ],
  },
  entry: {
    vendor: [
      'babel-polyfill',
      'font-awesome/scss/font-awesome.scss',
      'bootstrap/scss/bootstrap.scss',
      'jquery',
```

```
      'history',
      'react',
      'react-dom',
      'redux',
      'react-redux',
      'react-router',
      'react-router-dom',
      'react-router-redux',
      'redux-thunk',
    ],
  },
  output: {
    path: path.resolve('./dist'),
    filename: '[name].js',
    library: '[name]_[hash]',
  },
  plugins: [
    extractCSS,
    new webpack.DllPlugin({
      path: path.join(__dirname, 'dist', '[name]-manifest.json'),
      name: '[name]_[hash]',
    })
  ].concat(isDevelopment ? [] : [
    new webpack.optimize.UglifyJsPlugin({
      beautify: false,
      comments: false,
    }),
  ]),
};
```

Hacer referencia a un paquete Dll (DllReferencePlugin)

Nota: `manifest` (en DllReferencePlugin) debe hacer referencia a la `path` (definida en DllPlugin)

```
const webpack = require('webpack');
const path = require('path');
const isDevelopment = process.env.NODE_ENV !== 'production';

const ExtractTextPlugin = require('extract-text-webpack-plugin');
const extractCSS = new ExtractTextPlugin('app.css');

const merge = require('extendify')({ isDeep: true, arrays: 'concat' });

module.exports = merge({
  context: __dirname,
  entry: {
    app: (isDevelopment ? ['webpack-hot-middleware/client'] : []).concat(['./src/']),
  },
  output: {
    path: path.resolve('./dist'),
    publicPath: '/static',
    filename: '[name].js',
  },
  resolve: {
    extensions: ['.js', '.ts', '.tsx'],
  },
  module: {
    loaders: [
      {
```

```
        test: /\.tsx?$/,
        loader: 'babel-loader!awesome-typescript-loader?forkChecker=true',
        include: /src|spec/,
      },
      {
        test: /\.s?css$/,
        loader: extractCSS.extract(['css-loader?minimize', 'sass-loader']),
        include: /src/,
      },
    ],
  },
  plugins: [
    new webpack.DllReferencePlugin({
      context: __dirname,
      manifest: require('./dist/vendor-manifest.json'),
    }),
    new webpack.DefinePlugin({
      'process.env': {
        'ENV': JSON.stringify(process.env.NODE_ENV),
      },
    }),
    extractCSS,
  ],
}, isDevelopment ? require('./webpack.config.development') :
require('./webpack.config.production'));
```

Lea DllPlugin y DllReferencePlugin en línea:
https://riptutorial.com/es/webpack/topic/9508/dllplugin-y-dllreferenceplugin

Capítulo 5: Reemplazo de módulo caliente

Observaciones

webpack-hot-middleware

Use con `webpack-dev-middleware` , agregando `webpack-hot-middleware/client` a la entrada.

Config

Añadir configuraciones como cadena de consulta a la ruta. Ejemplo:

```
webpack-hot-middleware/client?path=/__what&timeout=2000&overlay=false
```

Opción	Descripción
camino	La ruta en la que el middleware está sirviendo el flujo de eventos
se acabó el tiempo	El tiempo de espera después de una desconexión antes de intentar reconectarse
cubrir	Establézcalo en falso para deshabilitar la superposición del lado del cliente basada en DOM.
recargar	Establézcalo en verdadero para volver a cargar automáticamente la página cuando el paquete web se atasque.
sin información	Establézcalo en verdadero para deshabilitar el registro de la consola informativa.
tranquilo	Establézcalo en verdadero para deshabilitar todo el registro de la consola.
dynamicPublicPath	Establézcalo en verdadero para usar webpack publicPath como prefijo de ruta. (Podemos establecer `__webpack_public_path__` dinámicamente en tiempo de ejecución en el punto de entrada, vea la nota de `output.publicPath`)

Examples

Usar con webpack-dev-middleware

1. Instalar *webpack-dev-middleware a* través de npm

```
npm i -D webpack-dev-middleware webpack-hot-middleware
```

2. Modificar *webpack.config.js*

   - Agregue `webpack-hot-middleware/client` a cada elemento definido en `"entry"`
   - Añadir `new webpack.HotModuleReplacementPlugin()` a `"plugins"`

```
module.exports = {
    entry: {
        js: [
            './index.js',
            'webpack-hot-
middleware/client?path=/__webpack_hmr&timeout=20000&reload=true'
        ]
    },
    plugins: [
        new webpack.HotModuleReplacementPlugin()
    ]
};
```

3. Añadir estos a *index.js*

```
var webpack = require('webpack');
var webpackDevMiddleware = require('webpack-dev-middleware');
var webpackHotMiddleware = require('webpack-hot-middleware');

var config = require('./webpack.config.js');
var compiler = webpack(config);

app.use(webpackDevMiddleware(compiler, {
    noInfo: true,
    publicPath: config.output.publicPath,
    stats: { colors: true },
    watchOptions: {
        aggregateTimeout: 300,
        poll: true
    },
}));

app.use(webpackHotMiddleware(compiler, {
    log: console.log,
}));
```

Habilitar HMR para el módulo

Para hacer que un módulo sea elegible para el Reemplazo de Módulo en Caliente (HMR), la forma más sencilla es agregar `module.hot.accept()` dentro del módulo, de esta manera:

```
// ...

if(module.hot) {
    module.hot.accept(); // This will make current module replaceable
}
```

Usar con webpack-dev-server

1. Instalar *webpack-dev-server a* través de npm.

```
npm i -D webpack-dev-server
```

2. Configure `webpack-dev-server` agregando *server.js* .

```
// server.js

var webpack = require("webpack");
var WebpackDevServer = require("webpack-dev-server");
var config = require("./webpack.dev.config");

var server = new WebpackDevServer(webpack(config), {
    // ...
});

server.listen(8080);
```

3. Modificar *webpack.config.js*

    - Agregue `webpack-dev-server/client` a cada ítem definido en `"entry"` .
    - Agregue `webpack/hot/only-dev-server` a cada ítem definido en `"entry"` .
        - **NOTA:** Cambie si es necesario ...
        - Use `webpack/hot/only-dev-server` para bloquear la actualización de la página si HMR falla.
        - Use `webpack/hot/dev-server` para actualizar automáticamente la página si HMR falla.
    - Añadir `new webpack.HotModuleReplacementPlugin()` a `"plugins"`

```
module.exports = {
    entry: {
        js: [
            'webpack-dev-server/client?http://localhost:8080'
            'webpack/hot/only-dev-server',
            './index.js'
        ]
    },
    plugins: [
        new webpack.HotModuleReplacementPlugin()
    ]
};
```

4. Agregue `hot: true` en la `webpack-dev-server`

```
var server = new WebpackDevServer(webpack(config), {
    hot: true

    // ... other configs
```

```
});
```

Lea Reemplazo de módulo caliente en línea:
https://riptutorial.com/es/webpack/topic/4594/reemplazo-de-modulo-caliente

Capítulo 6: Sacudida de arbol

Examples

Sacudida de árboles ES2015

webpack 2 introduce la sacudida de árbol que puede eliminar el código no utilizado cuando se utilizan los módulos ES2015 para importar y exportar código.

Instalar

```
npm install babel-preset-es2015-webpack --save-dev
```

Uso

en .babelrc:

```
{
    "presets": [
        "es2015-webpack"
    ]
}
```

Lea Sacudida de arbol en línea: https://riptutorial.com/es/webpack/topic/6466/sacudida-de-arbol

Capítulo 7: Servidor de desarrollo: webpack-dev-server

Examples

Instalación

`webpack-dev-server` se puede instalar a través de `npm`

```
npm --save-dev webpack-dev-server
```

ahora puedes iniciar el servidor

```
./node_modules/.bin/webpack-dev-server
```

Para simplificar el uso, puede agregar un script a `package.json`

```
// package.json
{
  ...
  "scripts": {
    "start": "webpack-dev-server"
  },
  ...
}
```

ahora para ejecutar el servidor puede utilizar

```
npm run start
```

`webpack-dev-server` está configurado en el archivo `webpack.config.js` en la sección `devServer` .

Para cambiar el directorio de base de contenido del servidor puede usar la opción `contentBase` . La configuración de ejemplo del directorio raíz a `public_html` podría verse como

```
let path = require("path");

module.exports = {
  ...
  devServer: {
    contentBase: path.resolve(__dirname, "public_html")
  },
  ...
}
```

Usando proxy

`webpack-dev-server`

puede `webpack-dev-server` algunas solicitudes a otros servidores. Esto puede ser útil para desarrollar el cliente API cuando desee enviar solicitudes al mismo dominio.

El proxy se configura a través del parámetro `proxy` .

La configuración de ejemplo del servidor dev que envía solicitudes a `/api` a otro servicio que escucha en el puerto 8080 podría tener este aspecto

```
// webpack.config.js
module.exports = {
  ...
  devServer: {
    proxy: {
      "/api": {
        target: "http://localhost:8080"
      }
    }
  }
...
}
```

volver a escribir

Es posible volver a escribir la ruta de destino usando la opción `pathRewrite` .

Suponiendo que desea eliminar el prefijo `/api` del ejemplo anterior, su configuración puede parecer

```
// webpack.config.js
  ...
  devServer: {
    proxy: {
      "/api": {
        target: "http://localhost:8080",
        pathRewrite: {"^/api" : ""}
      }
    }
  }
...
```

La solicitud `/api/user/256` se convertirá a `http://localhost:8080/user/256` .

filtrar

Es posible presentar solo algunas solicitudes. `bypass` permite proporcionar una función cuyo valor de retorno determinará si la solicitud debe ser proxy o no.

Suponiendo que solo desea `webpack` proxy solo las solicitudes POST a `/api` y dejar que el `webpack` encargue del resto, su configuración podría tener este aspecto

```
// webpack.config.js
  ...
  devServer: {
    proxy: {
      "/api": {
        target: "http://localhost:8080",
        bypass: function(req, res, proxyOptions) {
          if(req.method != 'POST') return false;
        }
      }
    }
  }
...
```

Lea Servidor de desarrollo: webpack-dev-server en línea:
https://riptutorial.com/es/webpack/topic/9877/servidor-de-desarrollo--webpack-dev-server

Capítulo 8: Uso de Webpack

Examples

Ejemplo de uso de módulos de CommonJS

Crear carpeta. Ábrelo en la línea de comandos. Ejecute `npm install webpack -g` . Crea 2 archivos:

cats.js:

```
var cats = ['dave', 'henry', 'martha'];
module.exports = cats;
```

app.js

```
cats = require('./cats.js');
console.log(cats);
```

Ejecutar en la línea de comandos: `webpack ./app.js app.bundle.js`

Ahora en la carpeta será el archivo `app.bundle.js` . Puede incluirlo en la página index.html, abrirlo en el navegador y ver el resultado en la consola.

Pero la forma más rápida se ejecuta en la línea de comandos: `node app.bundle.js` y vea el resultado inmediatamente en la consola:

    ['dave', 'henry', 'martha']

Ejemplo de uso de módulos AMD

Crear carpeta. Ábrelo en la línea de comandos. Ejecute `npm install webpack -g` . Crea 2 archivos:

cats.js:

```
define(function(){
    return ['dave', 'henry', 'martha'];
});
```

app.js

```
require(['./cats'],function(cats){
    console.log(cats);
})
```

Ejecutar en línea de comandos:

    webpack ./app.js app.bundle.js

Ahora en la carpeta será archivo: `app.bundle.js` .

Crear archivo index.html:

```
<html>
    <body>
        <script src='app.bundle.js' type="text/javascript"></script>
    </body>
</html>
```

Ábralo en el navegador y vea el resultado en la consola:

> ['dave', 'henry', 'martha']

Ejemplo de uso de módulos ES6 (Babel)

como está escrito en MDN en julio de 2016:

> Esta característica no está implementada en ningún navegador de forma nativa en este momento. Se implementa en muchos transpilers, como Traceur Compiler, Babel o Rollup.

Así que aquí está el ejemplo con el cargador de Babel para Webpack:

Crear carpeta. Agregue el archivo package.json allí:

```
{
  "devDependencies": {
    "babel-core": "^6.11.4",
    "babel-loader": "^6.2.4",
    "babel-preset-es2015": "^6.9.0",
    "webpack": "^1.13.1"
  }
}
```

Abra la carpeta en la línea de comandos. Correr:

```
npm install .
```

Crea 2 archivos:

cats.js :

```
export var cats = ['dave', 'henry', 'martha'];
```

app.js :

```
import {cats} from "./cats.js";
console.log(cats);
```

Para el uso correcto del babel-loader se debe agregar el archivo **webpack.config.js** :

```
 module: {
  loaders: [
    {
      test: /\.js$/,
      exclude: /(node_modules|bower_components)/,
      loader: 'babel?presets[]=es2015'
    }
  ]
}
```

Ejecutar en línea de comandos:

> webpack ./app.js app.bundle.js

Ahora en la carpeta será archivo: `app.bundle.js` .

Crear archivo **index.html** :

```
<html>
    <body>
        <script src='app.bundle.js' type="text/javascript"></script>
    </body>
</html>
```

Ábralo en el navegador y vea el resultado en la consola:

> ['dave', 'henry', 'martha']

Ejemplo de uso de módulos ES6 (Typescript)

como está escrito en [MDN] [1] en julio de 2016:

> Esta característica no está implementada en ningún navegador de forma nativa en este momento. Se implementa en muchos transpilers, como Traceur Compiler, Babel o Rollup.

Así que aquí está el ejemplo con el cargador Typescript para Webpack:

//QUE HACER

Cree una versión simplificada de este artículo: http://www.jbrantly.com/typescript-and-webpack/ sin tsd y jquery.

Lea Uso de Webpack en línea: https://riptutorial.com/es/webpack/topic/6001/uso-de-webpack

Creditos

S. No	Capítulos	Contributors
1	Empezando con el webpack	BrunoLM, CodingIntrigue, Community, Everettss, Filip Dupanović, Gun, Ken Redler, m_callens, neaumusic, noah.muth, Pavlin, pongo, RamenChef, Ru Chern Chong, Toby
2	Cargadoras	4m1r
3	Cargadores y complementos	jabacchetta, mleko
4	DllPlugin y DllReferencePlugin	BrunoLM
5	Reemplazo de módulo caliente	BrunoLM, Kevin Law
6	Sacudida de arbol	RationalDev
7	Servidor de desarrollo: webpack-dev-server	mleko
8	Uso de Webpack	Rajab Shakirov