EBook Gratuito

APPRENDIMENTO webpack

Free unaffiliated eBook created from
Stack Overflow contributors.

#webpack

Sommario

Di

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

Capitolo 1: Iniziare con il webpack

Osservazioni

Webpack è un bundle di moduli che legge moduli con dipendenze e produce asset statici che rappresentano quei moduli.

È dotato di un *sistema di caricamento* estensibile che consente ai pacchetti di includere non solo risorse Javascript, ma anche CSS, immagini, HTML e molto altro.

Ad esempio, utilizzando il caricatore di Javascript integrato, css-loader e 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
```

Diventerebbe un singolo file in bundle:

```
// 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 =
"
```

Le dipendenze possono essere definite in uno qualsiasi degli stili di modulo più comuni (CommonJS e AMD).

Versioni

Versione	Data di rilascio
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

Versione	Data di rilascio
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

Installazione

Prerequisiti:

NodeJS e npm

Esistono due modi per installare Webpack: globalmente o per progetto. È meglio avere la dipendenza installata per progetto, in quanto ciò consentirà di utilizzare diverse versioni di webpack per ciascun progetto e non richiedere all'utente di aver installato il webpack a livello globale.

Installazione per progetto

Esegui il seguente comando dalla cartella principale del tuo progetto:

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

È quindi possibile eseguire l'eseguibile del `node_modules` installato su `node_modules` :

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

O creare uno script NPM nel `package.json` file, dove è possibile omettere la `node_modules` parte - NPM è intelligente sufficienti per includere tale cartella sul suo cammino.

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

// from terminal:
npm start
```

Installazione a livello globale

Esegui il seguente comando al prompt:

```
npm install webpack -g
```

Esempio di webpack.config.js con babel

dipendenze

```
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')
    }],
  }
};
```

Esempio di Javascript + CSS + tipi di carattere + immagini

Moduli richiesti

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

Struttura delle cartelle

```
.
└── 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/**/*.*')` richiederà tutti i file nella cartella images come voce.

`ExtractTextPlugin` acquisirà l'output generato e creerà un file `css` bundle.

Esempio semplice di Webpack

Il minimo richiesto per utilizzare Webpack è il seguente comando:

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

// this is equivalent to:

webpack source-file destination-file
```

Il pacchetto Web prenderà il file sorgente, compilerà la destinazione di output e risolverà qualsiasi dipendenza nei file di origine.

Webpack, React JSX, Babel, ES6, configurazione semplice

Assicurati di installare le corrette dipendenze di npm (babel ha deciso di suddividersi in un gruppo di pacchetti, qualcosa che ha a che fare con le "dipendenze dei peer"):

```
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` è una funzione che esegue la scansione dei tuoi `node_modules` e garantisce

che non siano transpiled e raggruppati insieme al tuo codice front-end, sebbene assicuri che il bundle mantenga il riferimento ad essi. Questo aiuta con una transpilazione più veloce, dal momento che non stai ri-codificando le librerie.

Configurazione semplice del webpack con Node.js

Struttura delle cartelle

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

package.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
    }
};
```

webserver.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

eseguire ***npm i --save-dev*** per installare le dipendenze

eseguire il ***nodo. \ node_modules \ webpack \ bin \ webpack.js*** una volta installate le dipendenze

eseguire il ***nodo webserver.js*** per avviare il server

Leggi Iniziare con il webpack online: https://riptutorial.com/it/webpack/topic/924/iniziare-con-il-webpack

Capitolo 2: Agitazione dell'albero

Examples

Agitazione di alberi ES2015

webpack 2 introduce il tree shaking che può rimuovere il codice inutilizzato quando i moduli ES2015 vengono utilizzati per importare ed esportare il codice.

Installare

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

uso

in .babelrc:

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

Leggi Agitazione dell'albero online: https://riptutorial.com/it/webpack/topic/6466/agitazione-dell-albero

Capitolo 3: caricatori

Osservazioni

I caricatori di Webpack possono essere configurati come "preLoaders", "loaders" e "postLoaders". Sebbene non debbano esserlo, le configurazioni che utilizzano il linting o altre operazioni imperative o seriali possono trarre vantaggio da queste fasi di costruzione nella pipeline.

La chiave per comprendere i caricatori e il loro utilizzo è che Webpack eseguirà ciascun modulo nel grafico dei requisiti tramite il sistema del caricatore. Seguendo l'esempio sopra, questo significa che quando Webpack inizia a scansionare le importazioni della tua applicazione, identifica i file richiesti e usa una semplice regex, determinerà quale tipo di file o file richiede quale loader o serie di caricatori.

Qui sopra puoi vedere che tutti i file ".js" o ".jsx" verranno laccati da eslint-loader nella fase di preLoader. Altri tipi di file `js|x` verranno eseguiti anche attraverso il babel-loader nella fase di caricamento principale. Inoltre, nella stessa fase, tutti i file `.scss` verranno caricati nel sass-loader . Questo ti permette di importare file Sass nei tuoi moduli JS e di farli emettere sul bundle JS risultante o anche su un altro bundle CSS autonomo separato (usando un plugin).

Nota: l' importazione di file `.scss` funzionerà solo con Webpack e un caricatore appropriato. Il nodo non capirà questo tipo di importazione senza un pre-processore o un transpiler.

Inoltre, nell'esempio `.scss` è possibile `.scss` i "loader" con il `.scss !` punto esclamativo come un "tubo" tra caricatori diversi. L'esempio sopra collega l'output del "sass-loader" nel "css-loader" e infine nello "style-loader" Questo potrebbe anche essere eseguito con una serie di `loaders: ['style-loader', 'css-loader', 'sass-loader']` . Diverse opzioni sono anche disponibili per le definizioni caricatore in linea e seguono la sintassi dei parametri di query che si trova comunemente negli URL.

Vedi anche: https://webpack.github.io/docs/loaders.html

Examples

Config usando preLoader per eslint, babel per jsx e css loader.

La seguente configurazione può essere utilizzata come configurazione di base per raggruppare il progetto come libreria. Si noti come la configurazione del modulo contenga un elenco di pre-caricatori e caricatori.

```
// 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'),
    }
};
```

Leggi caricatori online: https://riptutorial.com/it/webpack/topic/6534/caricatori

Capitolo 4: Caricatori e plugin

Osservazioni

Caricatori e plugin costituiscono gli elementi costitutivi di Webpack.

I caricatori sono in genere delegati a una singola attività e tipo di file. Sono più facili da configurare e di solito richiedono un codice di codice inferiore.

I plugin, d'altra parte, hanno accesso al sistema di build interno di Webpack tramite hook e sono quindi più potenti. I plugin possono modificare l'ambiente Webpack completamente configurato e possono eseguire azioni personalizzate durante il processo di compilazione.

Quando si gestiscono i nostri file CSS, ad esempio, è possibile utilizzare un caricatore per aggiungere automaticamente i prefissi dei fornitori alle proprietà, mentre un plug-in può essere utilizzato per produrre un foglio di stile minificato nel processo di creazione del pacchetto.

Examples

Iniziare con i caricatori

Per iniziare, `npm install` i caricatori desiderati per il tuo progetto.

All'interno dell'oggetto di configurazione che viene esportato in `webpack.config.js` , una proprietà del `module` manterrà tutti i caricatori.

```
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']
          }
        ]
  },
};
```

Nella configurazione sopra, stiamo usando tre impostazioni di base per i nostri caricatori:

- **test:** è qui che associamo i caricatori a estensioni specifiche mediante RegExp. Il primo set

di caricatori viene eseguito su tutti i file .js e .jsx. Il secondo set è in esecuzione su tutti i file .css e .scss.

- **include:** questa è la directory in cui vogliamo che i nostri programmi di caricamento vengano eseguiti. Facoltativamente, potremmo altrettanto facilmente utilizzare la proprietà `exclude` per definire le directory che non vogliamo includere.
- **loader:** questo è un elenco di tutti i caricatori che vogliamo eseguire sui file specificati in `test` e `include` .

È importante notare che i caricatori vengono eseguiti da destra a sinistra in ciascun array di caricatori e dal basso verso l'alto nelle singole definizioni. Il codice seguente eseguirà i caricatori nel seguente ordine: `sass, autoprefixer, css, style` .

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

Questa è una fonte comune di confusione e bug per gli sviluppatori che sono nuovi al webpack. Ad esempio, quando si utilizza la sintassi JSX ed ES6, si desidera filtrare quel codice, *non* filtrare il codice compilato fornito dal nostro caricatore babel. Pertanto, il nostro caricatore eslint è posizionato a destra del nostro caricatore babel.

Il suffisso `-loader` è facoltativo quando si elencano i nostri caricatori.

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

... è equivalente a:

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

In alternativa, è possibile utilizzare la proprietà `loader` (singolare) insieme a una stringa che separa l'elenco di caricatori da `!` personaggio.

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

... è equivalente a:

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

caricamento di file dattiloscritti

Per usare dattiloscritto con webpack è necessario installare `typescript` e `ts-loader`

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

Ora puoi configurare il webpack per usare i file dattiloscritti

```
// 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/
      }
    ]
  },
  ...
};
```

Leggi Caricatori e plugin online: https://riptutorial.com/it/webpack/topic/5651/caricatori-e-plugin

Capitolo 5: DllPlugin e DllReferencePlugin

introduzione

I plugin Dll e DllReference consentono di suddividere il codice in più pacchetti in modo che i bundle possano essere compilati in modo indipendente.

È possibile creare script di "fornitori" in una libreria che non è necessario compilare spesso (ad esempio: React, jQuery, Bootstrap, Fontawesome ...) e fare riferimento nel pacchetto di app che avrà bisogno di quegli script.

Il bundle dell'applicazione, quello che verrà costantemente modificato, sarà in una configurazione separata facendo semplicemente riferimento a un pacchetto "fornitore" già creato.

Sintassi

- new webpack.DllPlugin ({percorso: '[nome] -manifest.json', nome: '[nome] _ [hash]'})
- new webpack.DllReferencePlugin ({context: __dirname, manifest: require ('./ packname-manifest.json')})

Examples

Configurazione del fornitore (DllPlugin)

Nota: `output.library` e `name` (in DllPlugin) devono essere uguali.

```
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,
    }),
  ]),
};
```

Riferimento a un bundle di DLL (DllReferencePlugin)

Nota: `manifest` (in DllReferencePlugin) dovrebbe fare riferimento al `path` (definito in 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'));
```

Leggi DllPlugin e DllReferencePlugin online: https://riptutorial.com/it/webpack/topic/9508/dllplugin-e-dllreferenceplugin

Capitolo 6: Server di sviluppo: webpack-dev-server

Examples

Installazione

`webpack-dev-server` può essere installato tramite `npm`

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

ora puoi avviare il server

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

Per semplificare l'utilizzo è possibile aggiungere script a `package.json`

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

ora per eseguire il server è possibile utilizzare

```
npm run start
```

`webpack-dev-server` è configurato nel file `webpack.config.js` nella sezione `devServer` .

Per cambiare la directory base del contenuto del server è possibile utilizzare l'opzione `contentBase` . Esempio di impostazione della directory root di impostazione su `public_html` potrebbe essere simile

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

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

Utilizzo del proxy

`webpack-dev-server` può delegare alcune richieste ad altri server. Questo potrebbe essere utile per lo sviluppo di client API quando si desidera inviare richieste allo stesso dominio.

Il proxy è configurato tramite parametro `proxy` .

La configurazione di esempio del server dev che passa richieste a `/api` ad altri servizi di ascolto sulla porta 8080 potrebbe apparire come questa

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

riscrivere

È possibile riscrivere il percorso di destinazione usando `pathRewrite` opzione `pathRewrite` .

Supponendo di voler togliere `/api` prefisso `/api` dall'esempio precedente potrebbe essere la tua configurazione

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

Request `/api/user/256` verrà convertito in `http://localhost:8080/user/256` .

filtro

È possibile proxy solo alcune richieste. `bypass` consente di fornire una funzione il cui valore di ritorno determinerà se la richiesta deve essere inoltrata o meno.

Supponendo di voler solo proxy solo le richieste POST a `/api` e lasciare che il `webpack` gestisca il resto la configurazione potrebbe apparire come questa

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

Leggi Server di sviluppo: webpack-dev-server online:
https://riptutorial.com/it/webpack/topic/9877/server-di-sviluppo--webpack-dev-server

Capitolo 7: Sostituzione del modulo caldo

Osservazioni

webpack-hot-middleware

Utilizzare con `webpack-dev-middleware` , aggiungendo `webpack-hot-middleware/client` alla voce.

config

Aggiungi le configurazioni come stringa di query al percorso. Esempio:

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

Opzione	Descrizione
sentiero	Il percorso su cui il middleware sta servendo il flusso di eventi
tempo scaduto	Il tempo di aspettare dopo una disconnessione prima di tentare di riconnettersi
copertura	Impostare su false per disabilitare l'overlay lato client basato su DOM.
ricaricare	Impostare su true per ricaricare automaticamente la pagina quando il webpack si blocca.
NoInfo	Impostare su true per disabilitare la registrazione della console informativa.
silenzioso	Impostare su true per disabilitare tutte le registrazioni della console.
dynamicPublicPath	Impostare su true per utilizzare webpack publicPath come prefisso del percorso. (Possiamo impostare `__webpack_public_path__` dinamicamente in fase di runtime nel punto di ingresso, vedere la nota di `output.publicPath`)

Examples

Utilizzare con webpack-dev-middleware

1. Installa *webpack-dev-middleware* tramite npm

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

2. Modifica *webpack.config.js*

    - Aggiungi `webpack-hot-middleware/client` a ciascun elemento definito in `"entry"`
    - Aggiungi `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. Aggiungi questi 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,
}));
```

Abilita HMR per modulo

Per rendere un modulo idoneo per la sostituzione di moduli caldi (HMR), il modo più semplice è aggiungere `module.hot.accept()` all'interno del modulo, in questo modo:

```
// ...

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

Utilizzare con webpack-dev-server

1. Installa *webpack-dev-server* tramite npm.

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

2. Configura `webpack-dev-server` aggiungendo *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. Modifica *webpack.config.js*

   - Aggiungi `webpack-dev-server/client` a ciascun elemento definito in `"entry"` .
   - Aggiungi `webpack/hot/only-dev-server` a ogni elemento definito in `"entry"` .
     - **NOTA:** cambia se necessario ...
     - Usa `webpack/hot/only-dev-server` per bloccare l'aggiornamento della pagina se HMR fallisce.
     - Usa `webpack/hot/dev-server` per aggiornare automaticamente la pagina se HMR fallisce.
   - Aggiungi `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. Aggiungi `hot: true` nella configurazione di `webpack-dev-server`

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

    // ... other configs
});
```

Leggi Sostituzione del modulo caldo online:
https://riptutorial.com/it/webpack/topic/4594/sostituzione-del-modulo-caldo

Capitolo 8: Utilizzo del Webpack

Examples

Esempio di moduli CommonJS di utilizzo

Creare una cartella. Aprilo nella riga di comando. Esegui `npm install webpack -g` . Crea 2 file:

cats.js:

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

app.js

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

Esegui nella riga di comando: `webpack ./app.js app.bundle.js`

Ora nella cartella sarà presente il file `app.bundle.js` . Puoi includerlo nella pagina index.html, aprirlo nel browser e vedere i risultati nella console.

Ma il modo più veloce è eseguito in linea di comando: `node app.bundle.js` e vedere immediatamente i risultati nella console:

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

Esempio di utilizzo dei moduli AMD

Creare una cartella. Aprilo nella riga di comando. Esegui `npm install webpack -g` . Crea 2 file:

cats.js:

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

app.js

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

Esegui nella riga di comando:

    webpack ./app.js app.bundle.js

Ora nella cartella sarà presente il file: `app.bundle.js` .

Crea il file index.html:

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

Aprilo nel browser e vedi i risultati nella console:

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

Esempio di utilizzo dei moduli ES6 (Babel)

come scritto in MDN a luglio 2016:

> Questa funzione non è implementata in nessun browser in questo momento. È implementato in molti transpilers, come il Traceur Compiler, Babel o Rollup.

Quindi ecco un esempio di Babel loader per Webpack:

Creare una cartella. Aggiungi qui il file package.json:

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

Apri la cartella nella riga di comando. Correre:

```
npm install .
```

Crea 2 file:

cats.js :

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

app.js :

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

Per un corretto utilizzo di babel-loader dovrebbe essere aggiunto il file **webpack.config.js** :

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

Esegui nella riga di comando:

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

Ora nella cartella sarà presente il file: `app.bundle.js` .

Crea il file **index.html** :

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

Aprilo nel browser e vedi i risultati nella console:

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

Esempio di utilizzo dei moduli ES6 (dattiloscritto)

come scritto in [MDN] [1] a luglio 2016:

> Questa funzione non è implementata in nessun browser in questo momento. È implementato in molti transpilers, come il Traceur Compiler, Babel o Rollup.

Quindi ecco un esempio con il caricatore di Typescript per Webpack:

//FARE

Crea una versione semplificata di questo articolo: http://www.jbrantly.com/typescript-and-webpack/ senza tsd e jquery.

Leggi Utilizzo del Webpack online: https://riptutorial.com/it/webpack/topic/6001/utilizzo-del-webpack

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con il webpack	BrunoLM, CodingIntrigue, Community, Everettss, Filip Dupanović, Gun, Ken Redler, m_callens, neaumusic, noah.muth, Pavlin, pongo, RamenChef, Ru Chern Chong, Toby
2	Agitazione dell'albero	RationalDev
3	caricatori	4m1r
4	Caricatori e plugin	jabacchetta, mleko
5	DllPlugin e DllReferencePlugin	BrunoLM
6	Server di sviluppo: webpack-dev-server	mleko
7	Sostituzione del modulo caldo	BrunoLM, Kevin Law
8	Utilizzo del Webpack	Rajab Shakirov