02

Configurando o Webpack para usar novos loaders

Transcrição

Devemos rodar o seguinte comando para instalar os loaders:

```
npm install css-loader@0.25.0 style-loader@0.13.1 --save-dev
```

COPIAR CÓDIGO

Vamos abrir o arquivo de configuração do webpack, no caso, o arquivo **alurapic/webpack.config.js**:

```
var path = require('path')
var webpack = require('webpack')

module.exports = {
  entry: './src/main.js',
  output: {
    path: path.resolve(__dirname, './dist'),
    publicPath: '/dist/',
    filename: 'build.js'
  },
  module: {
    rules: [
      {
        test: /\.vue$/,
        loader: 'vue-loader',
        options: {
          loaders: {
            // Since sass-loader (weirdly) has SCSS as its
```

```
      default parse mode, we map
            // the "scss" and "sass" values for the lang
      attribute to the right configs here.
            // other preprocessors should work out of the
      box, no loader config like this nessessary.
            'scss': 'vue-style-loader!css-loader!sass-
      loader',
            'sass': 'vue-style-loader!css-loader!sass-
      loader?indentedSyntax'
          }
          // other vue-loader options go here
        }
      },
      {
        test: /\.js$/,
        loader: 'babel-loader',
        exclude: /node_modules/
      },
      {
        test: /\.(png|jpg|gif|svg)$/,
        loader: 'file-loader',
        options: {
          name: '[name].[ext]?[hash]'
        }
      }
    ]
  },
  resolve: {
    alias: {
      'vue$': 'vue/dist/vue.common.js'
    }
  },
  devServer: {
    historyApiFallback: true,
    noInfo: true
  },
  performance: {
    hints: false
```

```
  },
  devtool: '#eval-source-map'
}

if (process.env.NODE_ENV === 'production') {
  module.exports.devtool = '#source-map'
  // http://vue-loader.vuejs.org/en/workflow/production.html
  module.exports.plugins = (module.exports.plugins ||
[]).concat([
    new webpack.DefinePlugin({
      'process.env': {
        NODE_ENV: '"production"'
      }
    }),
    new webpack.optimize.UglifyJsPlugin({
      sourceMap: true,
      compress: {
        warnings: false
      }
    }),
    new webpack.LoaderOptionsPlugin({
      minimize: true
    })
  ])
}
```

COPIAR CÓDIGO

Vejam o seguinte trecho:

```
  module: {
    rules: [
      {
```

COPIAR CÓDIGO

Temos a propriedade `rule` que recebe um array com várias regras.

Precisamos inserir uma nova regra que ensine que tudo que termina com `cs`

pode ser importado e adicionado em nosso bundle.

A nova regra que adicionaremos será:

```
{ test: /\.css$/, loader: 'style-loader!css-loader' }
```

COPIAR CÓDIGO

Nosso arquivo ficará assim:

```
var path = require('path')
var webpack = require('webpack')

module.exports = {
  entry: './src/main.js',
  output: {
    path: path.resolve(__dirname, './dist'),
    publicPath: '/dist/',
    filename: 'build.js'
  },
  module: {
    rules: [
      {
        test: /\.vue$/,
        loader: 'vue-loader',
        options: {
          loaders: {
            // Since sass-loader (weirdly) has SCSS as its
default parse mode, we map
            // the "scss" and "sass" values for the lang
attribute to the right configs here.
            // other preprocessors should work out of the
box, no loader config like this nessessary.
            'scss': 'vue-style-loader!css-loader!sass-
loader',
            'sass': 'vue-style-loader!css-loader!sass-
```

```
      loader?indentedSyntax'
          }
          // other vue-loader options go here
        }
      },
      {
        test: /\.js$/,
        loader: 'babel-loader',
        exclude: /node_modules/
      },
      {
        test: /\.(png|jpg|gif|svg)$/,
        loader: 'file-loader',
        options: {
          name: '[name].[ext]?[hash]'
        }
      },
      { test: /\.css$/, loader: 'style-loader!css-loader' }
    ]
  },
  resolve: {
    alias: {
      'vue$': 'vue/dist/vue.common.js'
    }
  },
  devServer: {
    historyApiFallback: true,
    noInfo: true
  },
  performance: {
    hints: false
  },
  devtool: '#eval-source-map'
}

if (process.env.NODE_ENV === 'production') {
  module.exports.devtool = '#source-map'
  // http://vue-loader.vuejs.org/en/workflow/production.html
```

```
  module.exports.plugins = (module.exports.plugins ||
[]).concat([
    new webpack.DefinePlugin({
      'process.env': {
        NODE_ENV: '"production"'
      }
    }),
    new webpack.optimize.UglifyJsPlugin({
      sourceMap: true,
      compress: {
        warnings: false
      }
    }),
    new webpack.LoaderOptionsPlugin({
      minimize: true
    })
  ])
}
```

COPIAR CÓDIGO

Vamos entender essa nova regra.

O objeto possui as propriedades `test` e `loader`. A primeira indica o padrão que deve ser encontrado para que o loader entre em ação. No caso usamos a expressão regular `/\.css$/` para consigerar tudo que termina com `css`. Excelente, justamente o que precisamos. Na segunda propriedade, estamos informando para o Webpack que ele deve usar os loaders `style-loader` e `css-loader`.

Será que terminamos? Quase! Experimente parar a aplicação e rodá-la novamente com o comando `npm run dev`. Você verá a seguinte mensagem de erro *assustadora** no console:

```
ERROR in ./~/bootstrap/dist/fonts/glyphicons-halflings-
```

```
regular.eot
Module parse failed:
/Users/flavioalmeida/Downloads/alurapic/node_modules/bootstrap/
halflings-regular.eot Unexpected character '' (1:0)
You may need an appropriate loader to handle this file type.
(Source code omitted for this binary file)
 @ ./~/css-loader!./~/bootstrap/dist/css/bootstrap.css
6:4445-4497 6:4520-4572
 @ ./~/bootstrap/dist/css/bootstrap.css
 @ ./src/main.js
 @ multi (webpack)-dev-server/client?http://localhost:8080
webpack/hot/dev-server ./src/main.js

ERROR in ./~/bootstrap/dist/fonts/glyphicons-halflings-
regular.woff2
Module parse failed:
/Users/flavioalmeida/Downloads/alurapic/node_modules/bootstrap/
halflings-regular.woff2 Unexpected character '' (1:4)
You may need an appropriate loader to handle this file type.
(Source code omitted for this binary file)
 @ ./~/css-loader!./~/bootstrap/dist/css/bootstrap.css
6:4622-4676
 @ ./~/bootstrap/dist/css/bootstrap.css
 @ ./src/main.js
 @ multi (webpack)-dev-server/client?http://localhost:8080
webpack/hot/dev-server ./src/main.js

ERROR in ./~/bootstrap/dist/fonts/glyphicons-halflings-
regular.ttf
Module parse failed:
/Users/flavioalmeida/Downloads/alurapic/node_modules/bootstrap/
halflings-regular.ttf Unexpected character '' (1:0)
You may need an appropriate loader to handle this file type.
(Source code omitted for this binary file)
 @ ./~/css-loader!./~/bootstrap/dist/css/bootstrap.css
6:4790-4842
 @ ./~/bootstrap/dist/css/bootstrap.css
 @ ./src/main.js
```

```
 @ multi (webpack)-dev-server/client?http://localhost:8080
webpack/hot/dev-server ./src/main.js

ERROR in ./~/bootstrap/dist/fonts/glyphicons-halflings-
regular.woff
Module parse failed:
/Users/flavioalmeida/Downloads/alurapic/node_modules/bootstrap/
halflings-regular.woff Unexpected character '' (1:4)
You may need an appropriate loader to handle this file type.
(Source code omitted for this binary file)
 @ ./~/css-loader!./~/bootstrap/dist/css/bootstrap.css
6:4707-4760
 @ ./~/bootstrap/dist/css/bootstrap.css
 @ ./src/main.js
 @ multi (webpack)-dev-server/client?http://localhost:8080
webpack/hot/dev-server ./src/main.js
```

COPIAR CÓDIGO

Resolvemos o problema da importação do `css`, mas como o `bootstrap` importa arquivos de fontes, o webpack detecta que o bootstrap tenta utilizá-los e nos avisa que não será possível o carregamento. E agora?

Precisamos ensinar o Webpack a lidar com elas. Para isso, precisamos de outro módulo, o **url-loader**. Vamos parar nosso terminal e baixar o loader.

Dentro da pasta `alurapic` vamos executar o comando:

```
npm install url-loader@0.5.7 --save-dev
```

COPIAR CÓDIGO

Prontinho. Com o novo `loader` baixado, vamos ensinar o Webpack através do loader a lidar com os diversos tipos de fonte acessados pelo bootstrap.

Nosso arquivo `alurapic/webpack.config.js` final fica assim:

```
var path = require('path')
var webpack = require('webpack')

module.exports = {
  entry: './src/main.js',
  output: {
    path: path.resolve(__dirname, './dist'),
    publicPath: '/dist/',
    filename: 'build.js'
  },
  module: {
    rules: [
      {
        test: /\.vue$/,
        loader: 'vue-loader',
        options: {
          loaders: {
            // Since sass-loader (weirdly) has SCSS as its
default parse mode, we map
            // the "scss" and "sass" values for the lang
attribute to the right configs here.
            // other preprocessors should work out of the
box, no loader config like this nessessary.
            'scss': 'vue-style-loader!css-loader!sass-
loader',
            'sass': 'vue-style-loader!css-loader!sass-
loader?indentedSyntax'
          }
          // other vue-loader options go here
        }
      },
      {
        test: /\.js$/,
        loader: 'babel-loader',
        exclude: /node_modules/
      },
      {
        test: /\.(png|jpg|gif|svg)$/,
```

```
        loader: 'file-loader',
        options: {
          name: '[name].[ext]?[hash]'
        }
      },
      { test: /\.css$/, loader: 'style-loader!css-loader' },
      { test: /\.(woff|woff2)(\?v=\d+\.\d+\.\d+)?$/, loader:
'url-loader?limit=10000&mimetype=application/font-woff' },
      { test: /\.ttf(\?v=\d+\.\d+\.\d+)?$/, loader: 'url-
loader?limit=10000&mimetype=application/octet-stream' },
      { test: /\.eot(\?v=\d+\.\d+\.\d+)?$/, loader: 'file-
loader' },
      { test: /\.svg(\?v=\d+\.\d+\.\d+)?$/, loader: 'url-
loader?limit=10000&mimetype=image/svg+xml' }
    ]
  },
  resolve: {
    alias: {
      'vue$': 'vue/dist/vue.common.js'
    }
  },
  devServer: {
    historyApiFallback: true,
    noInfo: true
  },
  performance: {
    hints: false
  },
  devtool: '#eval-source-map'
}

if (process.env.NODE_ENV === 'production') {
  module.exports.devtool = '#source-map'
  // http://vue-loader.vuejs.org/en/workflow/production.html
  module.exports.plugins = (module.exports.plugins ||
[]).concat([
    new webpack.DefinePlugin({
      'process.env': {
```

```
      NODE_ENV: '"production"'
    }
  }),
  new webpack.optimize.UglifyJsPlugin({
    sourceMap: true,
    compress: {
      warnings: false
    }
  }),
  new webpack.LoaderOptionsPlugin({
    minimize: true
  })
])
}
```

COPIAR CÓDIGO

Não é à toa que o nome do nosso módulo é `url-loader`. Ele identifica o tipo do arquivo que esta sendo acessado através de uma URL e faz o mapeamento correto.

Para a aplicação iniciando-a logo em seguida. Você verá que o layout da nossa aplicação mudou, pois o bootstrap foi carregado corretamente.

Como fica o build final do projeto?

Vamos executar o comando `npm run build` para empacotar nosso projeto para produção. Será que muda alguma coisa?

```
npm run build
```

COPIAR CÓDIGO

Visualizando o conteúdo da pasta `dist` temos:

```
├── 0.build.js
├── 0.build.js.map
├── 448c34a56d699c29117adc64c43affeb.woff2
├── 89889688147bd7575d6327160d64e760.svg
├── build.js
├── build.js.map
├── e18bbf611f2a2e43afc071aa2f4e1512.ttf
├── f4769f9bdb7466be65088239c12046d1.eot
├── fa2772327f55d8198301fdb8bcfc8158.woff
└── glyphicons-halflings-regular.svg
```

COPIAR CÓDIGO

Veja que além dos bundles que já vimos antes, as fontes usadas pelo bootstrap foram parar dentro da pasta `dist`. Agora, quando formos distribuir nossa aplicação, levaremos também os arquivos de fonte.