webpack快速入门教程

1、了解Webpack相关

- 什么是webpack
  - Webpack是一个模块打包器(bundler)。
  - 在Webpack看来, 前端的所有资源文件(js/json/css/img/less/...)都会作为模块处理
  - 它将根据模块的依赖关系进行静态分析，生成对应的静态资源
- 五个核心概念
  - Entry：入口起点(entry point)指示 webpack 应该使用哪个模块，来作为构建其内部依赖图的开始。
  - Output：output 属性告诉 webpack 在哪里输出它所创建的 bundles，以及如何命名这些文件，默认值为 ./dist。
  - Loader：loader 让 webpack 能够去处理那些非 JavaScript 文件（webpack 自身只能解析 JavaScript）。
  - Plugins：插件则可以用于执行范围更广的任务。插件的范围包括，从打包优化和压缩，一直到重新定义环境中的变量等。
  - Mode：模式，有生产模式production和开发模式development
- 理解Loader
  - Webpack 本身只能加载JS/JSON模块，如果要加载其他类型的文件(模块)，就需要使用对应的loader 进行转换/加载
  - Loader 本身也是运行在 node.js 环境中的 JavaScript 模块
  - 它本身是一个函数，接受源文件作为参数，返回转换的结果
  - loader 一般以 xxx-loader 的方式命名，xxx 代表了这个 loader 要做的转换功能，比如 json-loader。
- 理解Plugins
  - 插件可以完成一些loader不能完成的功能。
  - 插件的使用一般是在 webpack 的配置信息 plugins 选项中指定。
- 配置文件(默认)
  - webpack.config.js : 是一个node模块，返回一个 json 格式的配置信息对象

2、开启项目

- 初始化项目：
  - 生成package.json文件

```
{
  "name": "webpack_test",
  "version": "1.0.0"
}
```

- 安装webpack
  - npm install webpack webpack-cli -g //全局安装,作为指令使用
  - npm install webpack webpack-cli -D //本地安装,作为本地依赖使用

3、编译打包应用

- 创建js文件
  - src/js/app.js
  - src/js/module1.js
  - src/js/module2.js
  - src/js/module3.js
- 创建json文件
  - src/json/data.json
- 创建主页面:
  - src/index.html
- 运行指令
  - 开发配置指令：webpack src/js/app.js -o dist/js/app.js --mode=development
    - 功能: webpack能够编译打包js和json文件，并且能将es6的模块化语法转换成浏览器能识别的语法
  - 生产配置指令：webpack src/js/app.js -o dist/js/app.js --mode=production
    - 功能: 在开发配置功能上加上一个压缩代码

- 结论：
  - webpack能够编译打包js和json文件
  - 能将es6的模块化语法转换成浏览器能识别的语法
  - 能压缩代码
- 缺点：
  - 不能编译打包css、img等文件
  - 不能将js的es6基本语法转化为es5以下语法
- 改善：使用webpack配置文件解决，自定义功能

H3 4、使用webpack配置文件

- 目的：在项目根目录定义配置文件，通过自定义配置文件，还原以上功能
- 文件名称：webpack.config.js
- 文件内容：

```
const { resolve } = require('path'); //node内置核心模块，用来设置路径。
module.exports = {
  entry: './src/js/app.js',    // 入口文件配置（简写）
  /*完整写法:
    entry:{
      main:'./src/js/app.js'
    }
  */
  output: {                      // 输出配置
    filename: './js/built.js',       // 输出文件名
    path: resolve(__dirname, 'build')    //输出文件路径配置
  },
  mode: 'development'    //开发环境(二选一)
  mode: 'production'    //生产环境(二选一)
};
```

- 运行指令： webpack

H3 5、打包less资源

- 概述：less文件webpack不能解析，需要借助loader编译解析
- 创建less文件

    - src/less/test1.less
    - src/less/test2.less
- 入口app.js文件
    - 引入less资源
- 安装loader
    - npm install css-loader style-loader less-loader less --save-dev
- 配置loader

```
{
    test: /\.less$/, // 检查文件是否以.less结尾（检查是否是less文件）
    use: [  // 数组中loader执行是从下到上，从右到左顺序执行
      'style-loader', // 创建style标签，添加上js中的css代码
      'css-loader', // 将css以commonjs方式整合到js文件中
      'less-loader'  // 将less文件解析成css文件
    ]
},
```

- 运行指令：webpack

6、js语法检查

- 概述：对js基本语法错误/隐患，进行提前检查
- 安装loader
    - npm install eslint-loader eslint --save-dev
- 备注1：在：eslint.org网站 -> userGuide -> Configuring ESLint 查看如何配置
- 备注2：在：eslint.org网站 -> userGuide -> Rules 查看所有规则
- 配置loader

```
module: {
  rules: [
    {
      test: /\.js$/,  //只检测js文件
      exclude: /node_modules/,  //排除node_modules文件夹
      enforce: "pre",  //提前加载使用
      use: { //使用eslint-loader解析
        loader: "eslint-loader"
      }
    }
  ]
}
```

- 修改package.json（需要删除注释才能生效）

```
"eslintConfig": {
    "parserOptions": {
      "ecmaVersion": 6,          // 支持es6
      "sourceType": "module"     // 使用es6模块化
    },
    "env": { // 设置环境
      "browser": true,    // 支持浏览器环境：  能够使用window上的全局变量
      "node": true        // 支持服务器环境：  能够使用node上global的全局
变量
    },
    "globals": {    // 声明使用的全局变量，这样即使没有定义也不会报错了
      "$": "readonly"    // $ 只读变量
    },
    "rules": {  // eslint检查的规则  0 忽略 1 警告 2 错误
      "no-console": 0,  // 不检查console
      "eqeqeq": 2,  // 用==而不用===就报错
      "no-alert": 2 // 不能使用alert
    },
    "extends": "eslint:recommended" // 使用eslint推荐的默认规则
https://cn.eslint.org/docs/rules/
},
```

- 运行指令：webpack

H3 7、js语法转换

- 概述：将浏览器不能识别的新语法转换成原来识别的旧语法，做浏览器兼容性处理
- 安装loader

    - npm install babel-loader @babel/core @babel/preset-env --save-dev
- 配置loader

```
module: {
  rules: [
    {
      test: /\.js$/,
      exclude: /node_modules/,
      use: {
        loader: "babel-loader",
        options: {
          presets: ['@babel/preset-env']
        }
      }
    }
  ]
}
```

- 运行指令：webpack

8、 js兼容性处理

####第一种方法：使用经典的polyfill

- 安装包
    - npm install @babel/polyfill
- 使用

```
- app.js

    import '@babel/polyfill'; // 包含ES6的高级语法的转换
```

- 优点：解决babel只能转换部分低级语法的问题(如：let/const/解构赋值...)，引入polyfill可以转换高级语法(如:Promise...)
- 缺点：将所有高级语法都进行了转换，但实际上可能只使用一部分
- 解决：需要按需加载（使用了什么高级语法，就转换什么，而其他的不转换）

####第二种方法：借助按需引入core-js按需引入

- 安装包
  - npm install core-js
- 配置loader

```
{
    test: /\.js$/,
    exclude: /(node_modules)/,
    use: {
      loader: 'babel-loader',
      options: {
        presets: [
          [
            '@babel/preset-env',
            {
              useBuiltIns: 'usage',  // 按需引入需要使用polyfill
              corejs: { version: 3 }, // 解决warn
              targets: { // 指定兼容性处理哪些浏览器
                "chrome": "58",
                "ie": "9",
              }
            }
          ]
        ],
        cacheDirectory: true, // 开启babel缓存
      }
    }
},
```

9、打包样式文件中的图片资源

- 概述：图片文件webpack不能解析，需要借助loader编译解析
- 添加2张图片:
  - 小图, 小于8kb: src/images/vue.png
  - 大图, 大于8kb: src/images/react.jpg
- 在less文件中通过背景图的方式引入图片
- 安装loader
  - npm install file-loader url-loader --save-dev
  - 补充：url-loader是对象file-loader的上层封装，使用时需配合file-loader使用。
- 配置loader

```
{
    test: /\.(png|jpg|gif)$/,
    use: {
      loader: 'url-loader',
      options: {
        limit: 8192, // 8kb ⟶ 8kb以下的图片会base64处理
        outputPath: 'images', // 决定文件本地输出路径
        publicPath: '../dist/images',  // 决定图片的url路径
        name: '[hash:5].[ext]' // 修改文件名称 [hash:5] hash值取5位
[ext] 文件扩展名
      }
    }
},
```

- 运行指令：webpack

10、打包html文件

- 概述：html文件webpack不能解析，需要借助插件编译解析
- 添加html文件
  - src/index.html
  - 注意不要在html中引入任何css和js文件
- 安装插件Plugins
  - npm install html-webpack-plugin --save-dev
- 在webpack.config.js中引入插件（插件都需要手动引入，而loader会自动加载）
  - const HtmlWebpackPlugin = require('html-webpack-plugin')
- 配置插件Plugins

```
plugins: [
  new HtmlWebpackPlugin({
    template: './src/index.html', // 以当前文件为模板创建新的HtML(1.
结构和原来一样 2. 会自动引入打包的资源)
  }),
]
```

- 运行指令：webpack

11、打包html中图片资源

- 概述：html中的图片url-loader没法处理，它只能处理js中引入的图片 / 样式中图片，不能处理html中img标签，需要引入其他html-loader处理。
- 添加图片
  - 在src/index.html添加两个img标签
- 安装loader
  - npm install html-loader --save-dev
- 配置loader

```
{
  test: /\.(html)$/,
  use: {
    loader: 'html-loader'
  }
}
```

- 运行指令：webpack

12、打包其他资源

- 概述：其他资源webpack不能解析，需要借助loader编译解析
- 添加字体文件
  - src/media/iconfont.eot
  - src/media/iconfont.svg
  - src/media/iconfont.ttf
  - src/media/iconfont.woff
  - src/media/iconfont.woff2
- 修改样式

```
@font-face {
  font-family: 'iconfont';
  src: url('../media/iconfont.eot');
  src: url('../media/iconfont.eot?#iefix') format('embedded-
opentype'),
  url('../media/iconfont.woff2') format('woff2'),
  url('../media/iconfont.woff') format('woff'),
  url('../media/iconfont.ttf') format('truetype'),
  url('../media/iconfont.svg#iconfont') format('svg');
}
```

```
.iconfont {
  font-family: "iconfont" !important;
  font-size: 16px;
  font-style: normal;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}
```

- 修改html，添加字体
- 配置loader

```
{
    test: /\.(eot|svg|woff|woff2|ttf|mp3|mp4|avi)$/,  // 处理其他资源
    loader: 'file-loader',
    options: {
      outputPath: 'media',
      name: '[hash:8].[ext]'
    }
}
```

- 运行指令：webpack

13、自动编译打包运行

- 安装loader
  - npm install webpack-dev-server --save-dev
- 详细配置见官网：指南 -> 开发环境 -> 使用webpack-dev-server
- 修改webpack配置对象（注意不是loader中）

```
devServer: {
  open: true, // 自动打开浏览器
  compress: true, // 启动gzip压缩
  port: 3000, // 端口号
}
```

- 修改url-loader部分配置
  - 因为构建工具以build为根目录，不用再找build了
  - `publicPath: '../build/images/'` --> `publicPath: 'images/'`

- 修改package.json中scripts指令
  - "start": "webpack-dev-server",
- 运行指令：npm run start
  - 注意 `webpack-dev-server` 指令才能启动devServer配置，然后配置到package.json中才行

14、热模替换功能

- 概述：热模块替换（HMR）是webpack提供的最有用的功能之一。它允许在运行时更新所有类型的模块，而无需完全刷新（只更新变化的模块，不变的模块不更新）。
- 详细配置见官网：指南 -> 模块热替换
- 修改devServer配置

```
devServer: {
  contentBase: resolve(__dirname, 'build'), // 运行项目的目录
open: true, // 自动打开浏览器
compress: true, // 启动gzip压缩
port: 3000, // 端口号
hot: true // 开启热模替换功能 HMR
}
```

- 问题：html文件无法自动更新了，需要增加一个入口

```
entry: ['./src/js/app.js','./src/index.html']
```

15、devtool

- 概述： 一种将压缩/编译文件中的代码映射回源文件中的原始位置的技术，让我们调试代码不在困难
- 详细配置见官网：配置 -> devtool
- 介绍
  - cheap 只保留行, 编译速度快
  - eval webpack生成的代码（每个模块彼此分开，并使用模块名称进行注释）, 编译速度快

    - inline 以base64方式将source-map嵌入到代码中，缺点造成编译后代码体积很大
- 推荐使用：
    - 开发环境： cheap-module-eval-source-map
    - 生产环境： cheap-module-source-map

> 以上就是webpack开发环境的配置，可以在内存中自动打包所有类型文件并有自动编译运行、热更新等功能。

16、准备生产环境

- 创建文件夹config，将webpack.config.js复制两份
    - ./config/webpack.dev.js
    - ./config/webpack.prod.js
- 修改webpack.prod.js配置，删除webpack-dev-server配置

```
// / 代表根路径(等价于这个：http://localhost:5000/)，以后项目上线所有路径都以当前网址为根路径出发
module.exports = {
  output: {
    path: resolve(__dirname, '../build'), // 文件输出目录
    filename: './js/built.js', // 文件输出名称
    publicPath: '/'   // 所有输出资源在引入时的公共路径，若loader中也指定了publicPath，会以loader的为准。
  },
module: {
  rules: [
    {
        test: /\.(png|jpg|gif)$/,
        use: {
          loader: 'url-loader',
          options: {
            limit: 8192,
            outputPath: 'images',
            publicPath: '/images',   // 重写publicPath，需要在路径前面加上 /
            name: '[hash:8].[ext]'
          }
        }
      },
  ]
},
  mode: 'production',   //修改为生产环境
devtool: 'cheap-module-source-map' // 修改为生产环境的错误提示
```

```
// 删除devServer
}
```

- 修改package.json的指令
  - "start": "webpack-dev-server --config ./config/webpack.dev.js"
  - "dev": "webpack-dev-server --config ./config/webpack.dev.js"
  - "build": "webpack --config ./config/webpack.prod.js"
- 开发环境指令
  - npm start
  - npm run dev
- 生产环境指令
  - npm run build
  - 注意: 生产环境代码需要部署到服务器上才能运行 （serve这个库能帮助我们快速搭建一个静态资源服务器）
    - npm i serve -g
    - serve -s build -p 5000

17、清除打包文件目录

- 概述：每次打包生成了文件，都需要手动删除，引入插件帮助我们自动删除上一次的文件
- 安装插件
  - npm install clean-webpack-plugin --save-dev
- 引入插件
  - const { CleanWebpackPlugin } = require('clean-webpack-plugin'); // 注意要解构赋值！！！
- 配置插件
  - new CleanWebpackPlugin() // 自动清除output.path目录下的文件
- 运行指令：npm run build

18、提取css成单独文件

- 安装插件
  - npm install mini-css-extract-plugin --save-dev

- 引入插件
  - const MiniCssExtractPlugin = require("mini-css-extract-plugin");
- 配置loader

```
{
  test: /\.less$/,
  use: [
    MiniCssExtractPlugin.loader,
    'css-loader',
    'less-loader',
  ]
}
```

- 配置插件

```
new MiniCssExtractPlugin({
  filename: "css/[name].css",
})
```

- 运行指令vs
  - npm run build
  - serve -s build

19、添加css兼容

- 安装loader
  - npm install postcss-loader postcss-flexbugs-fixes postcss-preset-env postcss-normalize autoprefixer --save-dev
- 配置loader

```
{
  test: /\.less$/,
  use: [
    MiniCssExtractPlugin.loader,
    'css-loader',
    {
      loader: 'postcss-loader',
      options: {
        ident: 'postcss',
        plugins: () => [
          require('postcss-flexbugs-fixes'),
```

```
          require('postcss-preset-env')({
            autoprefixer: {
              flexbox: 'no-2009',
            },
            stage: 3,
          }),
          require('postcss-normalize')(),
        ],
        sourceMap: true,
      },
    },
    'less-loader',
  ]
}
```

- 添加配置文件: .browserslistrc

```
last 1 version
> 1%
IE 10 # sorry
```

- 运行指令：
  - npm run build
  - serve -s build

20、压缩css

- 安装插件
  - npm install optimize-css-assets-webpack-plugin --save-dev
- 引入插件
  - const OptimizeCssAssetsPlugin = require('optimize-css-assets-webpack-plugin');
- 配置插件

```
new OptimizeCssAssetsPlugin({
  cssProcessorPluginOptions: {
    preset: ['default', { discardComments: { removeAll: true } }],
  },
  cssProcessorOptions: { // 解决没有source map问题
    map: {
      inline: false,
      annotation: true,
    }
  }
})
```

- 运行指令：
  - npm run build
  - serve -s build

21、压缩html

- 修改插件配置

```
new HtmlWebpackPlugin({
  template: './src/index.html',
  minify: {
    removeComments: true,
    collapseWhitespace: true,
    removeRedundantAttributes: true,
    useShortDoctype: true,
    removeEmptyAttributes: true,
    removeStyleLinkTypeAttributes: true,
    keepClosingSlash: true,
    minifyJS: true,
    minifyCSS: true,
    minifyURLs: true,
  }
})
```

- 运行指令：
  - npm run build
  - serve -s dist

> 以上就是webpack生产环境的配置，可以生成打包后的文件。