1. 当在html文件中引入一个js文件，js文件里面又引入了其他的js文件，或者js文件里面又引入了其他的样式文件，这种语法浏览器是不认识的。这是我们为了做模块化而规定的语法，浏览器认识不到这种语法，这时候就需要webpack来将这些资源都处理一下，比如说less转换成css，js文件中引入的js文件直接整合成一个文件，再交给浏览器，浏览器才能认识。
2. Webpack五个核心概念：entry：入口(entry)指示webpack以哪个文件为入口起点开始打包，分析构建内部依赖图；output：输出(output)指示webpack打包后的资源bundles输出到哪里去，以及如何命名；loader：loader让webpack能够去处理那些非javascript文件(webpack自身只能理解JavaScript和json文件)；plugins：插件(plugins)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。意思就是loader只能打包翻译一些非js文件，而plugins的功能很强大。

3.1 运行指令:

开发环境：webpack ./src/index.js -o ./build/build.js --mode=development

webpack会以./src/index.js为入口文件开始打包，打包后输出到./build/build.js，整 体打包环境是开发环境。

生产环境：webpack ./src/index.js -o ./build/build.js --mode=production

webpack会以./src/index.js为入口文件开始打包，打包后输出到./build/build.js，整 体打包环境是生产环境。

3.2 结论：

1.webpack能处理js/json资源，不能处理css/img等其他资源

2.生产环境和开发环境将es6模块化编译成浏览器能识别的模块

3.生产环境比开发环境多一个压缩js代码。

4.打包样式资源

const {resolve} = require('path')

module.exports = {

  entry: './src/index.js',

  output: {

    filename: 'build.js',

    path: resolve(\_\_dirname, 'build')

  },

  module: {

    rules: [

      {

        test: /\.css$/,

        use: [

          'style-loader',

          'css-loader'

        ]

      },

      {

        test: /\.less$/,

        use: [

          'style-loader',

          'css-loader',

          'less-loader'

        ]

      }

    ]

  },

  mode: 'development'

}

less-loader将less文件转换成css文件

css-loader的作用是将css文件变成commonjs模块加载到js文件中，里面内容是样式字符串

style-loader的作用是创建style标签，将js中的样式资源内容字符串插入到html文件的head中。

use里面的数组执行顺序是从下到上后者说是从右到左的。

1. 使用HtmlWebpackPlugin

const {resolve} = require('path')

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

module.exports = {

  entry: './src/index.js',

  output: {

    filename: 'build.js',

    path: resolve(\_\_dirname, 'build')

  },

  plugins: [

    new HtmlWebpackPlugin({

      template: './src/index.html'

    })

  ],

  mode: 'development'

}

HtmlWebpackPlugin可以根据指定的模板路径，将打包生成的js文件挂载到这个文件当中。

1. 打包图片资源

{

        test: /\.(png|jpg)$/,

        use: [

          {

            loader: 'url-loader',

            options: {

              name: '[name].[ext]',

              limit: 20 \* 1024,

              outputPath: 'images/',

esModule: false

            }

          }

        ]

      },

url-loader: 在一个js文件中引入css的样式文件，如果这个css的样式文件中有图片的路径就需要用url-loader来打包图片资源。上图limit参数，如果图片资源小于20kb图片就会被编码成base64放到生成的js文件中（生成的base64字符串大小比原来的图片大，但是可以减少向服务器请求的数量），如果大于20kb则与file-loader进行的处理一样。

因为url-loader默认使用es6模块化解析，而html-loader引入图片是commonjs，解析时会出现问题：[object Module]，所以要关闭url-loader的es6模块化，使用commonjs解析。

{

        test: /\.html$/,

        loader: 'html-loader'

}

html-loader: 可以将html模板中引用到的图片资源交给url-loader来进行处理

1. 打包其他资源

      {

        exclude: /\.(css|js|html|less|png)$/,

        loader: 'file-loader',

        options: {

          name: '[name].[ext]',

          outputPath: 'font'

        }

      }

打包其他资源可以用file-loader来进行打包，file-loader不会对资源进行相关处理，它会直接将资源复制到输出目录。exclude表示不包括，这里注意如果要使用url-loader对图片进行打包操作，在flie-loader中一定要不包括相关的图片资源，否则会引发错误。

1. Webpack-dev-server

  devServer: {

    contentBase: resolve(\_\_dirname, 'build'),

    open: true,

    port: 3000,

    compress: true // 启动gzip压缩

  },

首先要安装webpack-dev-server，然后配置上图所示，然后使用命令npx webpack-dev-server来开启开发环境服务器。开发服务器用来自动编译，自动打开浏览器，自动刷新浏览器，只会在内存中编译打包，不会有任何输出。

生产环境的配置

前面都是开发环境的一些配置，开发环境配置的目的就是为了我们方便进行项目的开发，接下来是生产环境的配置，生产环境的配置主要解决三大问题。1：开发环境中的css全部都在js文件中了，加载js的时候才会创建style标签会导致闪屏的问题。2：进行代码的压缩。3：进行css的兼容性处理。为什在开发环境不处理上述问题呢，因为开发环境我们只是为了更好的开发，如果将这些都处理之后打包就会变得很慢，所以一般都是最后在生产环境中配置这些，为了发布更好更简洁的项目。

1. 将css拆分成独立的文件

  module: {

    rules: [

      {

        test: /\.css$/,

        use: [

          MiniCssExtractPlugin.loader,

          'css-loader'

        ]

      }

    ]

  },

  plugins: [

    new HtmlWebpackPlugin({

      template: './src/index.html'

    }),

    new MiniCssExtractPlugin({

      filename: 'css/index.css'

    })

  ]

使用MiniCssExtractPlugin，可以指定css文件的输出路径，MiniCssExtractPlugin.loader替代style-loader就能实现拆分css文件。

1. css的兼容性处理，压缩css

  module: {

    rules: [

      {

        test: /\.css$/,

        use: [

          MiniCssExtractPlugin.loader,

          'css-loader',

          {

            loader: 'postcss-loader',

            options: {

              ident: 'postcss',

              plugins: () => [

                require('postcss-preset-env')()

              ]

            }

          }

        ]

      },

    ]

  },

  "browserslist": {

    "development": [

      "last 1 chrome version",

      "last 1 firefox version",

      "last 1 safari version"

    ],

    "production": [

      ">0.2%",

      "not dead",

      "not op\_mini all"

    ]

  }

process.env.NODE\_ENV = 'development'

new OptimizeCssAssetsWebpackPlugin()

使用postcss-loader和post-css-preset-env，在package.json文件中配置browserslist告诉post-css-preset-env处理css兼容性的标准。process.env.NODE\_ENV设置node的环境变量，development的话就根据browserslist的development配置做css兼容性处理；production的话就根据browserslist的production配置做css兼容性处理。

压缩css直接用optimize-css-assets-webpack-plugin就可以了。

1. eslint语法检查

      {

        test: /\.js$/,

        loader: 'eslint-loader',

        exclude: /node\_modules/,

        options: {

          fix: true //自动修复

        }

      }

  "eslintConfig": {

    "extends": "airbnb-base"

  }

语法检查：eslint-loader eslint。注意只检查自己写的源代码，第三方库是不用检查的。

package.json文件中配置检查规则为airbnb-base。

airbnb --> eslint-config-airbnb-base eslint-plugin-import

1. js兼容性处理方案具体看视频
2. js和html压缩

  mode: 'production'

    new HtmlWebpackPlugin({

      template: './src/index.html',

      minify: {

        // 移除空格

        collapseWhitespace: true,

        // 移除注释

        removeComments: true

      }

    })

js压缩直接将mode改为生产模式就行了，html压缩在htmlwebpackplugin里面配置minify参数。

性能优化

开发环境优化

1. HMR热模块替换

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

  devserver: {

    contentBase: resolve(\_\_dirname, 'build'),

    compress: true,

    port: 3000,

    open: true,

    hot: true // 开启HMR

  }

一个模块发生变化时只会重新打包这一个模块而不是打包所有模块，极大提升构建速度

样式文件：直接可以使用HMR功能，因为style-loader内部已经实现了

js文件：默认不能使用HMR功能 ----> 需要修改js代码，添加支持HMR功能的代码。注意HMR功能对js的处理只能处理非入口js文件。

Html文件：默认不能使用HMR功能，同时会导致html文件不能热更新了，可以修改entry入口，将html文件引入。Html不需要HMR功能。

1. Sourcemap

  devtool: 'eval-source-map'

一种提供源代码到构建后代码映射技术（如果构建后代码出错了，通过映射可以追踪源代码错误）。具体devtool的value值看视频讲解。

生产环境优化

1. Oneof

正常来讲，一个文件只能被一个loader处理，当一个文件要被多个loader处理，那么一定要指定loader执行的先后顺序。Oneof数组里面的loader只会执行一个，执行了一个后就会跳过，这样就节约了文件的检查时间。这样就可以避免每个文件都把所有loader都走一遍，节约了打包的时间。

1. 缓存

babel缓存：cacheDirectory：true。babel将es6以上的js代码编译成es5之后生成一个缓存，下一次编译的时候babel就会对比缓存中变化的代码，将变化的代码进行编译，没有变化的代码直接用内存中的代码，让第二次打包构建速度更快。

文件资源缓存：当服务器把文件发送给浏览器时可能会让浏览器强制缓存这些文件，但是当我们的源代码改了之后重新打包服务器重新加载，浏览器在请求的时候还会从缓存中加载文件，就这样改的代码就没有效果了。解决办法是每次打包生成的文件名不一样，这样浏览器再次请求的时候发现文件名字变了就不会从缓存中读取文件了。

Hash：webpack每次打包都会生成一个唯一的hash值。因为js和css同时使用一个hash值，如果重新打包，会导致所有缓存失效。（可能我只改了一个文件）

Chunkhash：根据chunk生成的hash值。如果打包来源于同一个chunk，那么hash值也就一样，还是会出现hash值会出现的问题，因为css是在js中被引入的，所以同属于一个chunk

Contenthash：根据文件的内容生成hahs值。不同文件hash值一定不一样，这个就可以解决上述问题。

1. Tree shaking

去掉没有使用到的代码。开启：js必须使用es6模块化，使用production环境。

Tree shaking可能会将css代码或者一些第三方模块摇掉，因为这些也是只引用了没调用的。

在package.json文中配置sideEffects: [“\*.css”,”\*.less”]可以避免css这类文件被摇掉。