前端知识总结

BMAN

2020 年 6 月 30 日

目录

1 Webpack

Webpack 是打包工具，生产力工具，前端项目的可定制自动化机床。

安装命令：npm install webpack webpack-cli

1.1 基本工作流程

webpack 将各种资源（可以是 js、jsx、css、ts、图片等）都视为模块，从一个起点出发，通过各种 loader 翻译依赖关系，使用图算法遍历，将所有 module 模块都打包成一个 bundle 包。

开发者在开发时可以用 dev-server 将浏览器与后端 webpack 进程联系起来，源文件

1.2 配置文件解读

一个常见的 Webpack 配置文件如下所示

```
{
  context: '',
  entry: {
      demo: '',
      vendor: []
  },
  output: {
    path: '',
    filename: '',
    chunkFilename: '',
    hotUpdateChunkFilename: '',
    publicPath: ''
  },
  module: {
    noParse: '',
    rules: []
  },
  resolve: {
    alias: {},
    extensions: []
  },
  target:'',
  devServer: {},
  plugins: [],
  devtool: ''
}
```

下面对其进行逐一解读。

1.2.1 context

基础工作目录的绝对路径，可用于解析 entry 与 loader 中的相对路径，默认使用当前目录（CWD）。

1.2.2 entry

传入应用程序的起点，从这个起点开始 webpack 开始进行打包，如果传入多个或一个数组，则 webpack 会按序批量工作，一般地，这些工作它们互不干扰。每一个入口所代表的数据流在 webpack 中都被称为一个 chunk（数据块）。

```
// 以下配置产生demo1,2,3...各种chunk
{
  entry: {
    demo1: '',
    demo2: '',
    demo3: '',
    demos: ['', '', ....],
  }
}
```

1.2.3 output

output 是 webpack 工作流的出口，它有两个常用属性，一个是 filename，用于指定输出的文件名，一个是 path，用于指定输出的路径。

如果入口只有一个，则可以将输出的文件名指定为固定的名称；

```
// 单入口
{
  entry: {
    app: '...',
  },
  output: {
    filename: 'bundle.js',
    path: '...',
  }
}
```

如果有多个入口，则需要占位符（substitutions）来确保每个入口的输出都有唯一的名称。

```
// 多入口
{
  entry: {
    app1: '...',
    app2: '...',
    ...
  },
  output: {
    fielname: '[name].js',
    path: '...'
  }
}
```

常用的占位符有:

- [name]，表示使用入口的名称；
- [id]，表示使用 chunk 的 id，该 id 由 webpack 根据顺序自动生成；
- [hash]，表示使用每次构建过程中的生成的唯一 hash，长度可通过后接":length" 来指定，如 [hash:20]；
- [chunkhash]，表示使用根据 chunk 的内容生成的 hash，长度可后接":length" 来指定，如 [chunkhash:16]；
- [query]，模块的 query，例如文件名? 后面的字符串；

占用符不仅可以用于指定文件名，还可以用于指定文件夹名称，比如

```
{
  entry: {
    filename:'/js/[name]/bundle.js'
  }
}
```

output 中还可以指定 publicPath，该属性会给所有的输出都加上一个公共路径，可以通过 publicPath 来生成 CDN 网络地址，如下所示：

```
output: {
  publicPath: 'http://cdn.example.com/assets/[hash]'
}
```

如果需要动态地指定每一个文件的 publicPath，则可以在配置文件中留空 publicPath，然后在入口起点处设置 __webpack_public_path__，如下所示：

```
__webpack_public_path__ = myRuntimePublicPath
// 其他入口内容
```

1.2.4 module

在模块化编程中，开发者将程序分解成离散功能块（discrete chunks of functionality），并称之为模块。模块化编程让校验、调试、测试轻而易举。Node.js 从最一开始就支持模块化编程，然而现今的多种支持 JavaScript 的模块化工具各有优劣，webpack 取长补短，将模块的概念应用于项目中的任何文件。

Node.js 中的常用模块语法（表达依赖关系）如下：

- ES2015 import 语句
- CommonJs require() 语句
- AMD define 和 require 语句
- css/sass/scss/less 文件中的 @import 语句
- css 中的 url() 或 HTML 中 <img src=...> 图片链接语句

webpack 可以通过 loader 来处理 Node.js 中各种表示模块关系的语法，社区已经为常用的各种流行语言和语言处理器构建了 loader，常用的有：

- CoffeScript
- TypeScript
- ESNext(Babel)
- Sass
- Less
- Stylus

webpack 是通过图的方式处理模块间依赖关系的（Dependency Graph），webpack 从入口起点开始，递归地构建一个依赖图，这个依赖图中包含应用程序所需的每个模块，通过各种图算法将所有模块打包为少量的 bundle。

注：对于 HTTP/1.1（串行加载），考虑一个多页面应用，如果将每个页面的内容都各自打包成一个 bundle，每个页面自身的加载时间确实是最小的（单文件的压缩与响应总比多文件要快），但是这样做，页面每次跳转都需要加载一整个应用；

如果我们适当地切分应用，将公共部分提取出来，公共部分一次加载即缓存，页面再跳转时只需要加载非公共部分即可，用户体验会更好；

注：对于 HTTP/2（能够并行加载多个文件），有文章webpack & HTTP/2分析，我们的处理策略应当与处理 HTTP/1 一致；

(1) module.noParse

给定正则表达式，指定 webpack 忽略与正则表达式相匹配的文件，忽略大型的库可以提高构建性能。webpack3.0 还支持将此字段设置为函数，函数参数为文件名，返回 true 表示忽略该文件。

```
noParse: /jquery|lodash/

// webpack3.0+
noParse: function(name) {
  return /jquery|lodash/.test(content)
}
```

(2) module.rules

模块可以有很多类型，所以需要定义处理不同模块的不同规则（Rule）。每个规则（Rule）都可以被分为三部分：条件（condition）、结果（result）和嵌套规则（nested rule）。

Condition

条件（condition）有两种输入值，一是请求文件（issuer）的绝对路径，二是被请求资源（the requested resource）的绝对路径。例如：从 app.js 中导入'style.css'，那么请求文件就是'/path/to/app.js'，被请求资源就是'/path/to/style.css'。webpack 使用 test、include、exclude、resource 属性对被请求资

源进行匹配；使用 issuer 属性对 issuer 匹配。范例如下：

```
{
  module: {
    rules: [    // 规则数组
      {
        // Condition
        // 以下输入值均为被请求资源的绝对路径
        test: [Condition]   // 匹配正则表达式
        resourceQuery: [Condition] // 正则表达式（匹配文
          件?后的字符串）
        include: [Condition] // 字符串或字符串数组
        exclude: [Condition] // 字符串或字符串数组
        and: [Condition]  // 正则或字符串数组
        or: [Condition]   // 正则或字符串数组
        not: [Condition]  // 正则、字符串或数组
        // 以下输入值为请求文件的绝对路径
        issuer: ? // issuer属性暂时不知是否可用

        // Result
        // ...

        // Nested Rules
        // ...
      }
    ]
  }
}
```

Result

结果（result）指定如何处理被条件（condition）匹配到的模块，它有两类：一是 loader；二是 parser。

loader　webpack 的 loader 可以被视为一种“工厂”，它接受指定类型的原料，将其加工后产出开发者想要的产品。例如，loader 可以将 TypeScript 转换为 JavaScript，或者将图像转换为 data URL，或者处理引入的 CSS 文件。常用的 loader 有 css-loader、img-loader，它们都以独立插件的形式提供给开发者。

loaders 属性（别名为 use 属性）可以识别字符串或数组或对象，用来指定和配置所用的 loader，范例如下：

```
{
  module: {
    rules: [    // 规则数组
      {
        // Condition
        // ...

        // Result
        loaders: [            // 表示用以下loader依次处理
          'style-loader',
          {
            loader: 'css-loader',    // loader属性指定单个
               loader
            options: {    // 通过options属性来配置loader
              importLoaders: 1,
            }
          }
        ]

        // Nested Rules
        // ...
      }
    ]
  }
}
```

parser webpack 可通过 parser 选项配置默认解析器或插件。例如可以配置默认解析器禁用哪些语法，范例如下：

```
{
  module: {
    rules: [    // 规则数组
      // Condition
      // ...

      // Result
      parser: { // webpack将忽略被禁用了的语法
        amd: false,    // 禁用AMD
        commonjs: false,  // 禁用CommonJs
        ...
      }

      // Nested Rules
      // ...
    ]
```

```
  }
}
```

Nested Rules

嵌套规则（nested rules）表示在规则内部进行规则的嵌套，有两种属性可以定义嵌套的规则，分别是 rules 和 oneOf。范例如下：

```
{
  module: {
    rules: [
      // Condition
      // ...

      // Result
      // ...

      // Nested Rules
      rules: [], // 依次匹配数组内规则
      oneOf: []  // 仅使用第一个匹配到的规则
    ]
  }
}
```

1.2.5 resolve

webpack 的 resolver 本质上是一个库，enhanced-resolve被用于帮助找到模块的绝对路径，所以 webpack 的 resolve 配置实际上就是对这个插件进行配置。

通过 enhanced-resolve，webpack 能够解析三种文件路径：

- 绝对路径，resolver 会直接使用，不会对绝对路径进一步进行解析；
- 相对路径，resolver 会解析出绝对路径；
- 模块路径，如果直接给出模块名称，如‘react’，resolver 默认在 resolve.modules 中指定的所有目录内搜索；

resolver 在解析 loader 时与文件解析采用相同的策略，不过可以通过 resolver 默认会缓存对系统文件的访问（知道这个有什么用呢？）。
以下是对 resolve 常用选项的解析：

(1) resolve.alias

该选项可以将别名与文件夹或文件绑定在一起，这样就可以在模块引入语句中使用别名。

```
{
  resolve: {
    alias: {
      MyReact: path.resolve(__dirname, 'src/my-react'),
      MyVue$: path.resolve(__dirname, 'src/my-vue/index.js
         ') // 假名后加$表示精确匹配
    }
  }
}
```

(2) resolve.extensions

可以通过该选项设置 resolver 能够识别哪些扩展：

```
{
  resolve: {
    extensions: ['js', 'jsx']
  }
}
```

(3) resolve.resolveLoader

resolveLoader 配置选项为 loader 提供单独的解析规则。这个属性的值应当是一个对象，所有规则的属性名和内容都与 resolve 对象的相同。

1.2.6 mode

mode 属性的值有两种：

- development，该选项会将 process.env.NODE_ENV 的值设为 development。启用 NamedChunksPlugin 和 NamedModulesPlugin；
- production，该选项会将 process.env.NODE_ENV 的值设为 production。启用 FlagDependencyUsagePlugin，FlagIncludedChunksPlugin，ModuleConcatenationPlugin，NoEmitOnErrorsPlugin，OccurenceOrderPlugin，SideEffectsFlagPlugin 和 UglifyJsPlugin；

需要注意的是，手动设置 NODE_ENV 不会改变 webpack 的 mode。

1.2.7 target

Node 环境和浏览器环境都可以使用 JavaScript，但是这二者的语法是不同的，这也会影响到 webpack 编译的结果。

target 常用的值有：

- web，默认，编译为类浏览器环境里可用；
- node，编译为类 Node.js 环境可用（会使用 Node.js require 模块加载 chunk）；
- 函数，webpack 会传入 compiler 作为该函数的参数，通过 compiler.apply 函数使用自己想用的插件；

1.2.8 devServer

webpack 为了方便开发者开发，专门有插件 webpack-dev-server 可以负责监控文件变化从而自动刷新浏览器，devServer 属性就是配置该插件的。使用该选项，webpack 会生成一个类 Nginx 服务器来代理已编译打包好的文件，同时，webpack 会监视所有的模块，一旦某一模块发生改动，webpack 都会从相应入口（entry）开始重新编译打包，更新对应的包（bundle），然后通知浏览器进行更新（默认通过 websocket，也可以设置为轮询），浏览器默认会进行热更新，否则会刷新页面。

常用配置有：

- devServer.contentBase，webpack-dev-server 的工作目录，可以为数组（多个不重名目录）或 false（表示禁用），这些目录下的文件发生改变时，会触发 webpack 向浏览器发送消息；
- devServer.host，指定使用一个 host，默认为 localhost，如果希望外部访问，可以改为'0.0.0.0'；
- devServer.port，指定服务运行的端口；
- devServer.proxy，该选项非常常用，可以设置反向代理其他的后端服务，详细文档见http-proxy-middleware包；
- devServer.public，使用 inline 模式时，浏览器的 webpack 脚本默认使用 window.location 来连接 webpack 服务器，如果服务器被 Nginx 等

二次代理，则连接会失败，此时需要将 public 设置为二次代理出来的地址，webpack 脚本就知道 webpack 后端服务在哪了；

- devServer.publicPath，webpack 服务器资源的公共前缀，比如服务运行在 https://localhost:8080/，如果 publicPath 设为'public'，则所有静态文件 URL 都为 https://localhost:8080/public/*；
- devServer.hot，值为 true（默认）或 false。开启时，当 webpack 的某一模块发生更改时（比如某 css），webpack-dev-server 会发送一个请求给浏览器，浏览器会用新编译好的模块替换掉旧模块（速度很快）；
- devServer.watchContentBase，该选项与 hot 互斥，如果该选项开启，则 server 会监视所有 contentBase 目录下的所有文件，一旦某文件发生改变，webpack 会通知浏览器刷新页面；
- devServer.watchOptions，server 默认使用文件系统（file system）对文件进行监视，很多时候会失效，可以设置 devServer.watchOptions.poll 为 true，然后通过 devServer.watchOptions.aggregatedTimeout 设置轮询时间（单位 ms），webpack 还提供了 devServer.watchOptions.ignored（值为正则表达式）来忽略不需要监听的文件；
- devServer.compress，是否开启 gzip 压缩，值为 true 或者 false；
- devServer.clientLogLevel，浏览器中 log 的打印级别，值为'none'、'error'、'warning'、'info'（默认）；
- devServer.headers，给所有文件响应数据包都添加 header 内容；
- devServer.historyApiFallback，React 路由和 Angular 的单页面应用会使用一些特殊的路由，该选项可以设置规则方便开发者，详细文档见connect-history-api-fallback包；
- devServer.inline，是否开启内联模式，默认开启，若开启则处理热加载的脚本会被插入到包（bundle）中，构建消息会出现在浏览器的 console 中。若关闭则会使用 iframe 模式，页面中会出现一个包含构建消息的 iframe；
- devServer.lazy，是否开启懒加载，如果开启，则只有在包（bundle）被请求的时候才会重新编译，该选项默认关闭；

- devServer.noInfo，如果设为 true，则在 server 启动和模块更新时不会打印消息，警告与错误仍会打印；
- devServer.quiet，如果设置为 true，则除了 server 初始启动信息之外的所有信息都不会被打印，即使是错误和警告；
- devServer.stats，可以自定义 server 端 log 打印级别，比如'errors-only'（仅错误）、'minimal'（错误或新的编译发生）、'none'（没有）、'normal'（默认）、'verbose'（全部）；
- devServer.openPage，server 启动时默认打开的浏览器网址；

(1) hmr

hot module reload，热加载，见上述。

(2) runtime

如果开启了 devServer，webpack-dev-server 在编译完成后会在浏览器端和 server 端留下一直运行的程序，这些程序统称为 runtime。他们负责通信、热加载等任务。

(3) manifest

webpack 的编译器在运行时就会保留所有模块的特征，这个数据集合被称为"Manifest"。webpack 编译打包完成后，所有的模块加载语句都变成了 ___webpack_require___ 方法，此方法使用的是模块标识（module identifier），通过 manifest，webpack 的 runtime 能够查询到模块标识符，检索出背后对应的模块。

开发者需要知道的是，每次构建都会让 runtime 和 manifest 发生改变，相应的 hash 值也会发生改变。

1.2.9 devtool

该选项可以调整已打包的文件与源文件之间的映射（即 SourceMap）的详细程度，映射内容越详细，包的体积会越大，编译和加载速度越慢。关于 SourceMap 的更多内容可见An Introduction to Source Maps。

关于该选项的更多设置见devtool documentation

1.2.10 plugins

插件是 webpack 的支柱。webpack 自身也是由各种插件构造出来的。插件目的在于解决 loader 无法实现的事情。

常见的插件范例如下：

```
{
  plugins: [
    new webpack.optimize.UglifyJsPlugin(),
    new HtmlWebpackPlugn({template: './src/index.html'})
  ]
}
```

可以看到，这些插件本质上是对象。插件开发者需要为这个对象编写 apply 属性函数，apply 函数会被 webpack 编译器在编译生命周期内反复调用，webpack 会传入一个 compiler 对象作为函数的参数，插件开发者就可以使用该 compiler 对象做一些工作，范例如下：

```
class MyOwnPlugin {
  apply(compiler) {
    // do some works here
    // ...
  }
}
```

常用的插件有 CommonsChunkPlugin 等，详细的插件列表可见webpack-plugins-list。

2 Vue

3 React

4 Nodejs