[vue2+3+webpack]

[2021.12.22]

国内较火的前端工程化框架之一

摘要

[前端框架]

[日期及邮编地址]

[ID名称]

[vue2 3](#_Toc31083)

[webpack 3](#_Toc5309)

[初始化项目 3](#_Toc30568)

[在项目中安装webpack 3](#_Toc30071)

[dependencies和devDependencies节点 3](#_Toc32076)

[在项目中配置webpack 3](#_Toc17206)

[webpack中的默认约定（即webpack.config.js文件中的初始默认配置） 4](#_Toc28394)

[自定义打包的入口与出口 4](#_Toc29286)

[webpack插件 4](#_Toc5981)

[webpack-dev-server 4](#_Toc4262)

[安装webpack-dev-server 4](#_Toc18669)

[配置webpack-dev-server 4](#_Toc23970)

[webpack-dev-server生成的文件位置 4](#_Toc10516)

[devServer中常用选项 4](#_Toc9087)

[html-webpack-plugin 5](#_Toc29098)

[安装html-webpakc-plugin 5](#_Toc5075)

[配置html-webpakc-plugin 5](#_Toc30857)

[打包处理css文件 6](#_Toc21831)

[处理流程 6](#_Toc6431)

[打包处理样式表中与url路径相关的文件 7](#_Toc3918)

[base64与普通image的对比 7](#_Toc14284)

[打包处理js高级语法 7](#_Toc27804)

vue2

webpack

webpack是前端工程化的具体解决方案

他提供了友好的前端模块化开发支持，以及代码压缩混淆、处理浏览器端JavaScript的兼容性、性能优化等强大功能

需要安装nodejs才能使用

初始化项目



注：-S表示将安装的包名和版本号记录在package.json文件的dependencies节点下，不写的 话也会记录。-S是--save的简写

install可以简写为i

在项目中安装webpack

npm install webpack@5.42.1 webpack-cli@4.7.2 -D

注：-D表示将安装的包名和版本号记录到devDependencies节点下

-D是--save-dev的简写

@用来指定版本号

dependencies和devDependencies节点

dependencies：开发和生产环境用到的包记录在该目录下

devDependencies：只有开发阶段会被用到的包记录在该目录下

在项目中配置webpack

1. 在项目根目录创建名为webpack.config.js的webpack配置文件，并初始化如下的基本配置

module.exports = {

mode: ’development’, //mode用来指定构建模式，有development和production

}

该配置控制webpack做不做代码压缩，development不压缩，production压缩

1. 在package.json的scripts节点下，新增dev脚本

“scripts”: {

“dev”: “webpack”, //scripts节点下的脚本可以通过npm run执行，如npm run dev

}

1. 在终端中运行npm run dev命令，启动webpack进行项目的打包构建

打包完成后会生成dist文件夹，里面有生成的main.js文件，该文件就是webpack打包的综合性js文件，包含用户自己写的代码和用到的工具包代码，可以解决低版本浏览器不兼容高级代码的问题

webpack中的默认约定（即webpack.config.js文件中的初始默认配置）

在webpack4.x 和5.x中

1. 默认打包的入口文件为src/index.js
2. 默认的输出文件路径为dist/main.js

这两种配置可以在webpack.config.js内修改

自定义打包的入口与出口

通过配置webpack.config.js文件中的entry节点指定打包的入口，通过output节点指定打包的出口，示例代码如下

const path = require(‘path’); //导入nodejs中专门操作路径的模块

module.exports = {

entry: path.join(\_\_dirname, ‘./src/index.js’);

output: {

path: path.join(\_\_dirname, ‘./dist’), //输出文件的存放路径

filename: ‘main.js’, //输出的文件名

}

webpack插件

通过安装和排至第三方的插件，可以拓展webpack的能力，从而让webpack用起来更方便。最常用的webpack插件有两个：

webpack-dev-server

类似于nodejs中的nodemon工具

每当修改了源代码，webpack会自动进行项目的打包和构建

安装webpack-dev-server

npm install webpack-dev-server@3.11.2 -D

配置webpack-dev-server

1. 修改package.json文件中scripts的dev命令如下

“scripts” : {

“dev”: “webpack serve”,

}

1. 再次运行npm run dev重新进行项目的打包
2. 浏览器中访问localhost:8080可以查看打包效果

webpack-dev-server生成的文件位置

webpack-dev-server生成的文件不会直接保存在硬盘上（照顾硬盘的读写寿命），所以在资源管理器上看不到它，生成的文件被保存在内存的根目录下，即/，引用路径直接写/xx.js即可

devServer中常用选项

devServer: {

open: true, //首次打包成功后自动打开浏览器

port: 80, //指定端口号

host: ‘127.0.0.1’, //指定主机地址

}

html-webpack-plugin

webpack中的html插件（类似于一个模板引擎插件），配合webpack-dev-server使用

1.可以通过此插件自定制index.html页面的内容，本质是把src内的index.html复制到根目录下（内存中），如此一来访问localhost:8080时会直接加载index.html

2.会自动添加一条script标签，把web-dev-server生成的js文件引入到index.html中

安装html-webpakc-plugin

npm install html-webpakc-plugin@5.3.2 -D

配置html-webpakc-plugin

在webpack.config.js文件内配置

//导入插件，得到一个构造函数

const HtmlPlugin = require(‘html-webpack-plugin’)

//创建插件实例对象

const htmlPlugin = new HtmlPlugin({

template: ‘./src/index.html’, //指定原文件的存放路径

filename: ‘./index.html’, //指定生成的文件的存放路径

})

module.exports = {

mode: ‘development’,

plugins: [htmlPlugin], //通过plugins节点，使htmlPlugin插件生效

}

webpack中的loader

loader概述

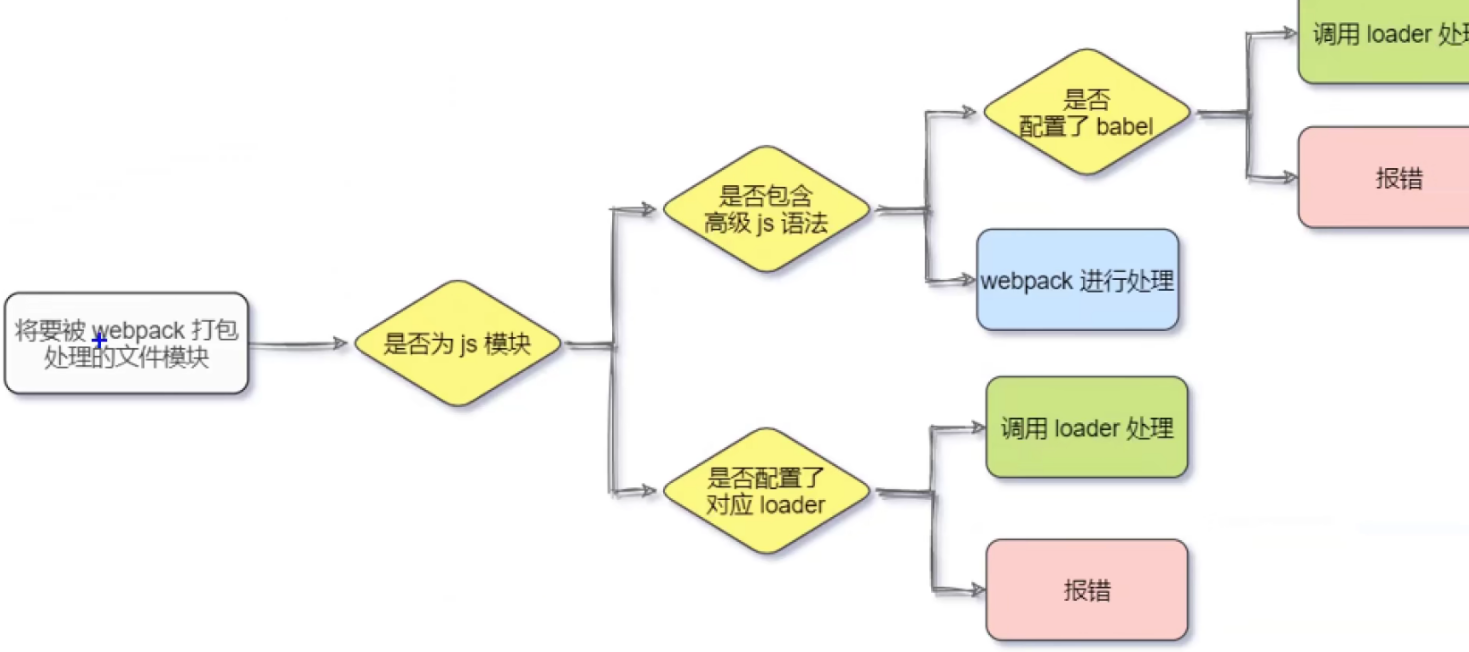
webpakc默认只能打包处理.js后缀名的文件，其他非.js后缀名的模块默认处理不了，需要调用loader加载器才能正常打包，否则会报错

loader作用：协助webpack打包处理特定的文件模块，如：

css-loader可以打包处理.css相关文件

less-loader可以打包处理.less相关文件

babel-loader可以打包处理webpack无法处理的高级JS语法



打包处理css文件

1. 运行*npm i style-loader@3.0.0 css-loader@5.2.6 -D* 安装处理css文件的loader
2. 在webpack.config.js的module->rules数组中，添加loader规则如下

module: {

rules: [

{test: /\.css$/, use: [‘style-loader’, ‘css-loader’]},

]

}

其中test表示匹配的文件类型，use表示对应要调用的loader

• use数组中指定指定的loader顺序是固定的

• 多个loader的调用顺序是从后往前调用

处理流程

webpack在发现了处理不了的模块时会查找webpack.config.,js的module内的rules数组， 是否配置了对应的loader加载器

若配置了，webpack把对应文件先交给数组最后一个loader进行处理，完毕后交给rules 数组内的下一个loader进行处理（倒序）

全都处理完毕后把结果转交给webpack，webpack把处理结果合并到/dist/main.js文件内， 最终生成打包好的文件。

打包处理less文件

1. 运行*npm i less-loader@10.0.1 less@4.1.1 -D*
2. 在webpack.config.js的rules数组中添加loader规则

module: {

rules: [

{test: /\.less$/, use: [‘style-loader’, ‘css-loader’, ‘less-loader’]},

]

}

less-loader依赖less运行，但less不写在rules数组中

打包处理样式表中与url路径相关的文件

1. 运行npm i [url-loader@4.1.1](mailto:url-loader@4.1.1) [file-loader@6.2.0](mailto:file-loader@6.2.0) -D
2. 在webpack.config.js中的rules数组配置规则

module: {

rules: [

{test: /\.jpg|png|gif$/, use: ‘url-loader?limit-22229’},

]

}

其中?之后的是loader的参数

limit用来指定图片的大小，单位是字节（byte）

只有小于等于limit大小的图片才会被转换成base64

base64与普通image的对比

base64将图片集成在html页面内，img标签src引用需要浏览器发送请求下载图片再加载

base64优点：减少图片网络请求

base64缺点：转换后的图片体积会变大一点

网络性能上base64具有优势，但大图片转换后编码很长占用内存较大，所以只推荐小规模，小体积的图片使用

打包处理js高级语法

webpack只能处理一部分js高级语法，对于webpack处理不了的js高级语法需要借助babel-loader进行打包处理

1. 运行npm i babel-loader@8.2.2 @babel/core@7.14.6 @babel/plugin-proposal-decorators@7.14.5 -D （后两个是依赖包,最后一个包针对装饰器语法）
2. 在webpack.config.js的rules数组中添加如下规则

module: {

rules: [

{test: /\.js$/, use: ‘babel-loader’, exclude: /node\_module/},

]

}

必须使用exclude排除指定项，因为node\_modules目录下的第三方包不需要被打包。因为.js会匹配到第三方js文件，所以要排除它们，只打包自己写的js文件即可

1. 配置babel-loader

在项目根目录下创建名为babel.config.js的配置文件，定义配置项如下

module.exports = {

plugins: [[‘@babel/plugin-proposal-decorators’, {legacy: true}],]

}