**VUE+wepack基础环境安装快速搭建开发环境**

**目录结构**

**第一步：**

下载并安装node.js

node –v 检测node.js的版本

npm –v 检测npa的版本

**第二步：**

执行 npm init 创建webpack.json文件

在执行第三步骤前 npm install

**第三步：**

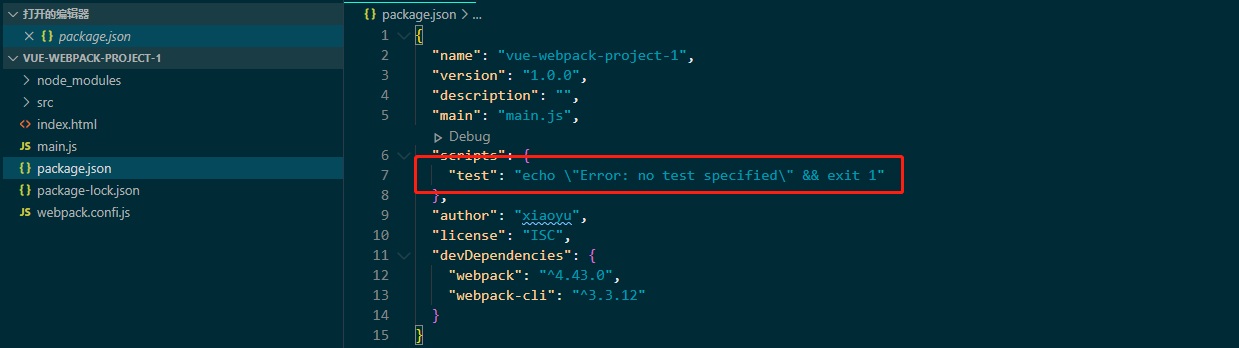
执行npm install webpack webpack-cli --save-dev 安装webpack

**第四步：**

配置package.json文件

1. webpack --mode development 为开发阶段打包
2. webpack --mode production 为生产阶段打包

其中dev和build是我们命名的名字，当我们执行npm run dev时就会执行后面的命令



    "dev": "webpack --mode development",

    "build": "webpack --mode production"

**第五步：**

配置webpack.config.js文件（配置的入口文件）

打包是执行命令： npm run dev

**注：**如果存在webpack.config.js文件，则打包是会默认用此文件中的命令进行打包

**二、本地测试服务器安装**

本地测试服务安装

第一步：

执行命令：npm install --save-dev webpack-dev-server 安装webpack-dev-server

第二步：配置webpack.config.js文件

    devServer:{

        contentBase:'./dist',*//设置服务器访问的基本目录*

        host:'localhost',*//配置服务器的IP地址*

        port:8080,*//端口*

        open:true*//自动打开页面*

    }

第三步：在进入到package.json中写入脚本命令start

  "start": "webpack-dev-server --mode development"

然后运行npm run dev 先打包一次

执行npm run start 开启start

完整代码：

const path = require('path'); //引入path的依赖

const webpack = require('webpack');//引入第三方插件库

const HtmlWebpackPlugin = require('html-webpack-plugin');//引入生成html文件的插件

module.exports = {

    entry:'./src/main.js',//配置入口信息

    output:{

        path:path.resolve(\_\_dirname,'dist'),//配置出口信息,即文件打包后的存放路径

        filename:'bundle.js'//存放文件的名称

    },

    // 配置本地服务器

    devServer:{

        contentBase:'./dist',//设置服务器访问的基本目录

        host:'localhost',//配置服务器的IP地址

        port:8080,//端口

        open:true//自动打开页面

},

}

**三、加载CSS的loader安装**

第一步：在入口JS文件中引入CSS文件

import './src/publickCss/Publickstyle.css';

第二步：安装css的loader

**即：执行命令：**npm install style-loader css-loader --save-dev

第三步：配置webpack.config.js中的loader

    module:{

        rules:[*//配置CSS的loader*

            {

                test:/\.css$/,

                use:['style-loader','css-loader']

            }

        ]

    }

第四步：将dist中的bundle.js文件引入到index.html中

<script src="./dist/bundle.js" type="text/javascript" charset="utf-8"></script>

完整代码

const path = require('path'); *//引入path的依赖*

module.exports = {

    entry:'./src/main.js',*//配置入口信息*

    output:{

        path:path.resolve(\_\_dirname,'dist'),*//配置出口信息,即文件打包后的存放路径*

        filename:'bundle.js'*//存放文件的名称*

    },

*// 配置本地服务器*

    devServer:{

        contentBase:'./dist',*//设置服务器访问的基本目录*

        host:'localhost',*//配置服务器的IP地址*

        port:8080,*//端口*

        open:true*//自动打开页面*

    },

    module:{

        rules:[*//配置CSS的loader*

            {

                test:/\.css$/,

                use:['style-loader','css-loader']

            }

        ]

    }

}

安装完css的loader后，就可以在入口文件中引入不同的css文件

**四、加载图片的loader安装**

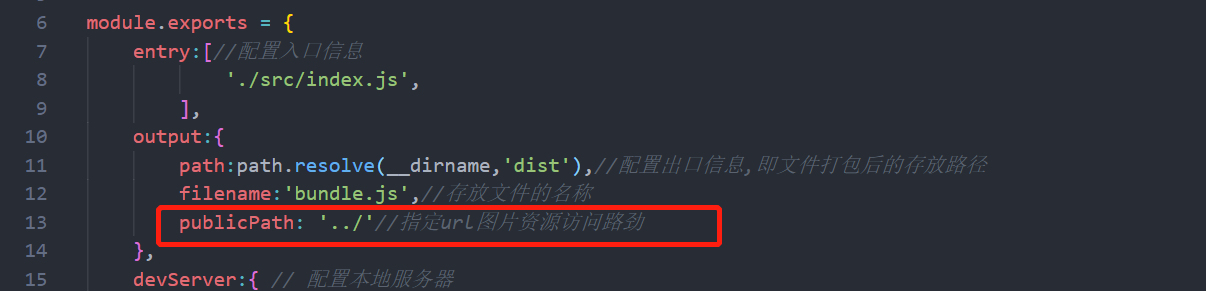
第一步：安装file-loader

执行指令：npm install file-loader --save-dev 安装file-loader

执行指令：npm install url-loader --save-dev 安装url-loader

**容易采坑的地方1**，没有在入口文件中指定url访问路径，这个路径并非固定，需要更具项目目录结构来做适配，在实际应用中使用url-loader即可，同时引入两个会冲突

        publicPath: '../'*//指定url图片资源访问路劲*



同时url-loader和file-loader具有相同的功能，都能够处理图片，但url只能识别css中引入的图片，优点能够将图片转化成base值，减少二次请求。同时也可以更具需求对图片上限大小做一个限定，避免base值过大，造成文件过大，如果要在页面中直接引入图片,还要在安装用file-loader

**容易采坑的地方2；**

                            name:'timg.jpg',

如果这样设置图片名称，则无论你放入多少张图片，都只能打包出一张

改进

name:'[name].[ext]?[hash].jpg',

配置rules:

**标准的做法是：将出口路径该为**

        publicPath: '/'//指定url图片资源访问路劲

**只使用url-loader,并设置limit属性**

            {//配置样式引入的photo解析器loader

                test:/\.(png|svg|jpg|gif|woff|woff2|svg|eot|ttf)$/,

                use:[

                    {

                        loader:'url-loader',

                        options:{//配置photo的选项

                            name:'[name].[ext]?[hash:8].jpg',

                            limit: 81920,

                            // name:'[path]timg.jpg',//图片名称  //[path]为打包后会自动生成文件夹存放图片

                            // name:'[hash]timg.jpg',// 设置[hash]

                            // context:'/',// 设置context，即是配置打包后图片的存放路径

                            // publicPath:'www.baidu.com',// 设置publicPath，即将图片发布到某个网站

                            outputPath:'images',// 设置outputPath，即将图片存放到XXXX文件夹

                        }

                    }

                ]

            },

            {//配置页面引入的photo解析器loader

                test: /\.(png|svg|jpg|gif|woff|woff2|svg|eot|ttf)$/,

                use: [

                    {

                        loader: 'file-loader',

                        options: {

                            name: '[path][name][hash:8].[ext]',

                            publicPath: 'assets/',//为文件配置自定义 public 发布目录

                            outputPath: 'images/',//为文件配置自定义 output 输出目录

                            useRelativePath: true,//为每个文件生成一个相对 url 的 context 时，应该将其设置为 true

                        }

                    }

                ]

            }

**目的是兼顾页面中引入和样式中引入的路径保持一致**

**五、ES6的babel编译**

1. 执行命令：npm install babel-loader @babel/core @babel/preset-env --save-dev
2. 配置config文件

            {*//配置ES6解析器babel*

                test:/\.js$/,

                exclude:/node\_modules/,

                use:'babel-loader'

            },

1. 新建文件：.babelrc并写入转换规则

{

*"presets"*:["@babel/preset-env"]

}

**六、自动生成HTML模板**

**1、执行命令**：npm install html-webpack-plugin --save-dev

**2、引入插件：**const HtmlWebpackPlugin = require('html-webpack-plugin');

const webpack = require('webpack');

const HtmlWebpackPlugin = require('html-webpack-plugin');*//引入生成html文件的插件*

3、Config代码中的plugins文件

    plugins:[*//配置生成html文件*

        new HtmlWebpackPlugin({

            template:'./index.html',

            filename:'index.html',

                minify:{

                minimize:true,*//是否打包为最小值*

                removeAttributeQuotes:true,*//去除引号*

                removeComments:true,*//去除注释*

*// collapseWhitespace:true,//去除空格*

                minifyCSS:true,*//压缩html内的样式*

                minifyJS:true,*//压缩html内的JS*

*// removeEmptyElements:true,//清理内容为空的元素*

            },

            hash:true*//引入产出资源的时候加上哈希避免缓存*

        })

    ]

**七、样式分离**

1、执行命令：npm install --save-dev mini-css-extract-plugin

安装MiniCssExtractPlugin插件

2、引入插件：

const MiniCssExtractPlugin = require('mini-css-extract-plugin'); *//分离CSS*

3、配置插件：rules

            {

                test:/\.css$/,

                use:[MiniCssExtractPlugin.loader,'css-loader']

            },

注意：如果已经有CSS模块，则只需要在原模块上加以下代码即可

                use:[MiniCssExtractPlugin.loader,'css-loader']*//分离CSS*

4、配置插件：plugins

        new MiniCssExtractPlugin(*//分离CSS*

            {

                filename:'styleCss/main.css'

            }

        ),

完整代码参考

**八、样式压缩与优化**

1、执行命令：npm install --save-dev optimize-css-assets-webpack-plugin 安装插件

2、引入插件：

const OptimizeCSSAssetsPlugin = require('optimize-css-assets-webpack-plugin');

3、设置plugin

        new OptimizeCSSAssetsPlugin({*// 压缩和优化CSS文件*

            assetNameRegExp:/\.css$/g,

            cssProcessor:require('cssnano'),

            cssProcessorPluginOptions:{

                preset:['default',{discardComments:{removeAll:true}}]

            },

            canPrint:true

        }),

**完整参考代码**

const path = require('path'); *//引入path的依赖*

const HtmlWebpackPlugin = require('html-webpack-plugin');*//引入生成html文件的插件*

const MiniCssExtractPlugin = require('mini-css-extract-plugin');*//分离CSS*

const OptimizeCSSAssetsPlugin = require('optimize-css-assets-webpack-plugin');*//样式优化与压缩*

module.exports = {

    entry:[*//配置入口信息*

            './src/index.js',

        ],

    output:{

        path:path.resolve(\_\_dirname,'dist'),*//配置出口信息,即文件打包后的存放路径*

        filename:'bundle.js',*//存放文件的名称*

        publicPath: '../'*//指定url图片资源访问路劲*

    },

    devServer:{ *// 配置本地服务器*

        contentBase:'./dist',*//设置服务器访问的基本目录*

        host:'localhost',*//配置服务器的IP地址*

        port:9090,*//端口*

        open:true*//自动打开页面*

    },

    module:{*//模块*

        rules:[*//配置CSS的loader*

            {

                test:/\.css$/,

                use:['style-loader','css-loader'],

                use:[MiniCssExtractPlugin.loader,'css-loader']*//分离CSS*

            },

            {*//配置ES6解析器babel*

                test:/\.js$/,

                exclude:/node\_modules/,

                use:'babel-loader'

            },

            {*//配置photo解析器loader*

                test:/\.(png|jpg|gif|jpeg)$/,

                use:[

                    {

                        loader:'url-loader',

                        options:{*//配置photo的选项*

                            name:'timg.jpg',

*// name:'[path]timg.jpg',//图片名称  //[path]为打包后会自动生成文件夹存放图片*

*// name:'[hash]timg.jpg',// 设置[hash]*

*// context:'/',// 设置context，即是配置打包后图片的存放路径*

*// publicPath:'www.baidu.com',// 设置publicPath，即将图片发布到某个网站*

                            outputPath:'img',*// 设置outputPath，即将图片存放到XXXX文件夹*

                        }

                    }

                ]

            }

        ]

    },

    plugins:[*//插件    生产环境使用*

*new* HtmlWebpackPlugin({*//配置生成html文件,并压缩优化html文件*

            template:'./src/index.html',

            filename:'index.html',

            inject: 'head',*//指定打包后的JS文件放在头部里面*

                minify:{

*// minimize:true,//是否打包为最小值              // 生产环境使用*

*// removeAttributeQuotes:true,//去除引号              // 生产环境使用*

*// removeComments:true,//去除注释              // 生产环境使用*

*// collapseWhitespace:true,//去除空格              // 生产环境使用*

*// minifyCSS:true,//压缩html内的样式              // 生产环境使用*

*// minifyJS:true,//压缩html内的JS              // 生产环境使用*

*// removeEmptyElements:true,//清理内容为空的元素              // 生产环境使用*

            },

*// hash:true//引入产出资源的时候加上哈希避免缓存              // 生产环境使用*

        }),

*new* MiniCssExtractPlugin(*//分离CSS*

            {

                filename:'css/main.css'

            }

        ),

*/\**

*new OptimizeCSSAssetsPlugin({// 压缩和优化CSS文件      // 生产环境使用*

*assetNameRegExp:/\.css$/g,*

*cssProcessor:require('cssnano'),*

*cssProcessorPluginOptions:{*

*preset:['default',{discardComments:{removeAll:true}}]*

*},*

*canPrint:true*

*}),*

*\*/*

    ]

}

**第三方插件库处理**

1、在项目中下载依赖 npm install jquery --save-dev    当然用淘宝镜像 cnpm 也是可以的

注意：jquery 一定要小写 不然会提示  Please use 'jquery' (all lowercase)

2、找到build文件夹下的webpack.conf.js文件，打开，添加：

 var webpack=require('webpack')

在plugins:里输入：

        new webpack.ProvidePlugin({//引入jquery

            $:"jquery",

            jQuery:"jquery",

            "windows.jQuery":"jquery"

        }),

3、在入口文件main.js中输入：

         import $ from 'jquery'

4、结束项目进程，重新npm run dev. 注意一定要结束进程重新打开项目，不然还是会报错

**Webpack基础环境搭建完成**

**接下来是VUE环境构建**

1. Npm安装vue的命令 ： npm install vue -S
2. 安装vueloder 和vue编译器

执行：npm install vue-loader vue-template-compiler

分离的vue文件需要loader处理

然后配置config文件

var VueLoaderPlugin = require('vue-loader/lib/plugin');

配置plugin（插件）

 new VueLoaderPlugin()

在配置module（模块）

{

                test:/\.(vue)$/,

                use:'vue-loader'

            },

完整代码参考

const path = require('path'); *//引入path的依赖*

const webpack = require('webpack');*//第三方插件处理*

const HtmlWebpackPlugin = require('html-webpack-plugin');*//引入生成html文件的插件*

const MiniCssExtractPlugin = require('mini-css-extract-plugin');*//分离CSS*

const OptimizeCSSAssetsPlugin = require('optimize-css-assets-webpack-plugin');*//样式优化与压缩*

var VueLoaderPlugin = require('vue-loader/lib/plugin');*//引入vue依赖loader*

module.exports = {

    entry:'./main.js',*//配置入口信息*

    output:{

        path:path.resolve(\_\_dirname,'dist'),*//配置出口信息,即文件打包后的存放路径*

        filename:'bundle.js',*//存放文件的名称*

        publicPath: '/',*//配置打包资源访路劲*

    },

*// 配置本地服务器*

    devServer:{

        contentBase:'./dist',*//设置服务器访问的基本目录*

        host:'localhost',*//配置服务器的IP地址*

        port:8080,*//端口*

        open:true*//自动打开页面*

    },

    module:{ *//模块配置*

        rules:[

            {*//配置CSS的loader*

                test:/\.css$/,

                use:['style-loader','css-loader'],

                use:[MiniCssExtractPlugin.loader,'css-loader']*//分离CSS*

            },

            {*//配置图片解析器loader*

                test: /\.(png|svg|jpg|gif|woff|woff2|svg|eot|ttf)$/,

                loader: 'url-loader?limit=8192&&name=src/imges/[name].[ext]'

            },

            {*//配置ES6解析器babel*

                test:/\.js$/,

                exclude:/node\_modules/,

                use:'babel-loader'

            },

            {*//配置VUE模块*

                test:/\.(vue)$/,

                use:'vue-loader'

            },

        ]

    },

    plugins:[*//配置生成html文件*

        new HtmlWebpackPlugin({

            template:'./index.html',

            filename:'index.html',

                minify:{

                minimize:true,*//是否打包为最小值*

                removeAttributeQuotes:true,*//去除引号*

                removeComments:true,*//去除注释*

*// collapseWhitespace:true,//去除空格*

                minifyCSS:true,*//压缩html内的样式*

                minifyJS:true,*//压缩html内的JS*

*// removeEmptyElements:true,//清理内容为空的元素*

            },

            hash:true*//引入产出资源的时候加上哈希避免缓存*

        }),

        new MiniCssExtractPlugin(*//分离CSS*

            {

                filename:'styleCss/main.css'

            }

        ),

        new OptimizeCSSAssetsPlugin({*// 压缩和优化CSS文件*

            assetNameRegExp:/\.css$/g,

            cssProcessor:require('cssnano'),

            cssProcessorPluginOptions:{

                preset:['default',{discardComments:{removeAll:true}}]

            },

            canPrint:true

        }),

        new VueLoaderPlugin(),*//配置VUE分离依赖*

    ]

}

**Webpack环境下的路由**

**单页面路由**（注意：在使用路由时不能再HTML中引入bundle.js文件）

安装路由：npm install vue-router -s

在入口文件中导入路由

import VueRouter from 'vue-router'*：*

Vue.use(VueRouter)

在根组件下使用路由

    <router-link to="/路由组件名"></router-link>

    <router-link to="/路由组件名"></router-link>

    <router-view></router-view>

在入口文件中导入路由并设置路由规则

import new\_file\_1 from './components/new\_file\_1.vue';

import new\_file\_2 from './components/new\_file\_2.vue';

*// 定义路由*

var router = new VueRouter({

    routes:[

        { path: '/new\_file\_1', component: new\_file\_1 },

        { path: '/new\_file\_2', component: new\_file\_2 }

    ]

})

在父组件中注册路由

var vm = new Vue({

    el:'#app\_vue',

    data:{

    },

    methods:{

    },

    render(createElements){

        return createElements(App)

    },

     router,

})

**注意：根组件调用子组件的时候文件名必须加后缀名，否则会报错**