**模块化开发**

**loader**

**www.webpackjs.com**

1. **css文件的打包**

**[style-loader](https://www.webpackjs.com/loaders/style-loader)：将模块的导出作为样式添加到 DOM 中**

**[css-loader](https://www.webpackjs.com/loaders/css-loader)：解析 CSS 文件，使用 import 加载，并且返回 CSS 代码**

**css-loader、style-loader安装：**

**npm install --save-dev css-loader**

**npm install --save-dev style-loader**

**webpack.config.js配置：//在之前的配置文件里加上module模块**

**module.exports = {**

**module: {**

**rules: [ //规则**

**{**

**test: /\.css$/, //正则表达式，规定打包css文件**

**use: [ 'style-loader', 'css-loader' ]//从右到左，先解析，再添加**

**}**

**]**

**}**

**}**

1. **less文件的打包（scss、stylus也可以写样式）**

[less-loader](https://www.webpackjs.com/loaders/less-loader)：**加载和转译 LESS 文件**

**less-loader安装 + less转译css**

**npm install --save-dev less-loader less**

**webpack.config.js配置：//在之前的配置文件里加上module模块**

**module.exports = {**

**module: {**

**rules: [{**

**test: /\.less$/,**

**use: [{**

**loader: "style-loader" // creates style nodes from JS strings**

**}, {**

**loader: "css-loader" // translates CSS into CommonJS**

**}, {**

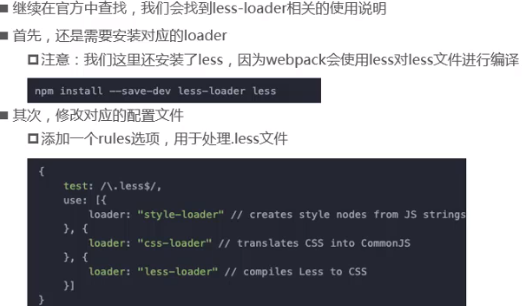
**loader: "less-loader" // compiles Less to CSS**

**}]**

**}]**

**}**

**}**



1. **图片文件的处理**

[url-loader](https://www.webpackjs.com/loaders/url-loader)：**像 file loader 一样工作，但如果文件小于限制，可以返回 [data URL](https://tools.ietf.org/html/rfc2397)**

**url-loader、file-loader安装**

**npm install --save-dev url-loader**

**npm install file-loader --save-dev**

**webpack.config.js配置**

**{**

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

**use: [**

**{**

**loader: 'url-loader',**

**options: {**

**limit: 8192 //图片文件的大小限制，单位是字节（B），可以修改**

**}}]}**

**注意：**

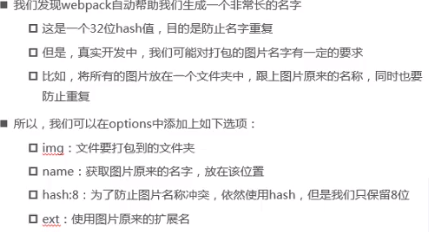
**当加载的图片小于limit时，会将图片编译成base64 字符串格式（不会生成 文件），浏览器可以正常显示**

**当加载的图片大于limit时，需要使用file-loader模块进行加载，但是加载后 的图片存放在dist文件夹中（这种方式会将图片打包成文件，与bundle.js 存放在同一个目录下），但此时我们的index.html还在dist目录外，bundle.js 里的图片路径却直接是图片文件名，没有” dist/ ”，导致浏览器显示错 误，需要在webpack.config.js配置文件里的output对象里添加：**

**publicPath: ‘dist/’**

**webpack就会给新生成的图片文件路径前面加上dist/，这时index.html就能 找到图片路径了。可是在开发完后index.html也要存放在dist文件夹里，这时就要去掉：publicPath: ‘dist/’**

**修改生成的图片文件名称**



1. **ES6转ES5的处理---babel**

现在还有一些浏览器不支持ES6的语法，所以需要对ES6模块化

**babel-loader、babel-core、babel-preset-es2015安装**

**npm install --save-dev babel-loader@7 babel-core babel-preset-es2015**

**webpack.config.js配置:**

**{**

**test: /\.js$/,**

**exclude: /(node\_modules|bower\_components)/, //排除这两个文件,不用转化**

**use: {**

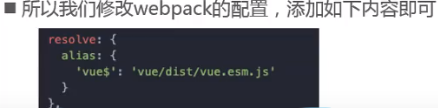
**loader: 'babel-loader',**

**options: {**

**presets: ['es2015'] //要修改为es2015**

**}}}**





1. **vue文件（组件）的封装处理**





**省略文件后缀**

**webpack.config.js配置：**

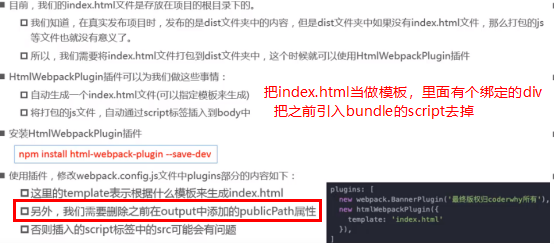
resolve: {

extensions: [‘ .js ’,’ .less ’,’ .vue ’],

}

**插件---plugin**

**打包HTML的plugin**



**除了输出bundle.js，其他打包新生成的文件路径（之前的图片文件，现在 这个新生成的index.html文件）都会加上dist/，所以要删除之前添加的 publicPath属性**

**压缩js的plugin（开发时不用，发布时再丑化）**



**vue-lazyload图片懒加载**

对于图片过多的页面，为了加速页面加载速度，所以很多时候我们需要将页面内未出现在可视区域内的图片先不做加载， 等到滚动到可视区域后再去加载。这样子对于页面加载性能上会有很大的提升，也提高了用户体验。

1. **安装插件**

**npm install vue-lazyload --save-dev**

1. **在main.js导入**

**import VueLazyLoad from 'vue-lazyload'**

1. **Vue.use()、或者添加自定义选项**

**Vue.use(VueLazyload [, options])**

**4.修改图片img的:src属性为v-lazy**

**github地址：<https://github.com/hilongjw/vue-lazyload>**

**postcss-px-to-viewport插件**

1. **安装**

**npm install postcss-px-to-viewport --save-dev**

1. **修改postcss.config.js文件**

**GitHub地址：<https://github.com/evrone/postcss-px-to-viewport>**

**类似的插件还有px2rem**

**自定义插件**

**Vue插件作用：为 Vue 添加全局功能**

所谓全局：

即不需要像组件那样，每次使用他前都需要先引入一次。对于插件只要在最开始引入一次，在任何组件就可以直接使用。（类似于我们在window上添加的方法属性那样，任何地方都可以用）将插件安装到Vue（在Vue.prototype原型里面添加），由于组件都是继承自Vue的，

所以组件都能使用Vue的功能

**插件原理：**

**vue插件其实就是一个简单的js对象而已。**在js文件中创建该插件对象，然后这个插件对象有一个公开属性 install ，他的值为一个函数，此函数接受两个参数。第一个参数是 Vue 构造器 , 第二个参数是一个可选的选项对象。最终导出此插件对象以供别人使用

1. **在js文件中创建插件对象**

let myplugin={ //myplugin是插件对象

install:function(Vue [,Options]){ //在这里写插件需要实现的功能

Vue.prototype.$myoption='插件的属性',//给Vue添加全局属性和方法

Vue.prototype.$myfn=function(){

console.log('插件的方法')

}

}

}

export default myplugin;

1. **安装、使用插件**

在入口文件main.js导入，再使用use方法安装插件，Vue就会执行插件的install函数为Vue添加全局功能

import myplugin from ‘.../文件名’

Vue.use(myplugin)

任何组件都可以使用添加的功能 this.$myoption、this.$myfn()

**toast弹框组件**

**之前都是将组件导入、注册、使用。如果很多地方都要使用这个组件，每次使用都要做这三步就很麻烦了。这时可以将组件封装到插件里面，再将插件安装到Vue全局中，只要在main.js里面new Vue之前导入、安装插件，那么哪个组件都可以使用该插件了。**

**vue-router、vuex也是这样的插件**

**axios没有install函数，不用安装，直接使用axios**

创建js文件，将组件导入，创建插件对象，给插件对象添加install函数，在函数里使用组件创建组件构造器，使用组件构造器创建一个组件对象，由于body里面没有可挂载的元素，所以要创建一个元素div，把组件对象挂载到这个div，那么组件对象的属性$el就是这个div了，再将这个div追加到body里面，最后给Vue添加全局属性$toast，指向组件对象。由于组件对象里面有模板，所以会替换被挂载的div，每个组件都使用$toast属性（toast组件）

import Toast from ‘.../vue文件’ //导入

const obj = {

install: function(Vue) { //默认将Vue传过来

//创建组件构造器

const toastTonstructor = Vue.extend(Toast);

//创建组件对象

const toast = new toastConstructor();

//将组件对象挂载到元素上

toast.$mount(document.createElement(‘div’));

//将元素添加到body

document.body.appendChild(toast.$el);

//添加一个指向组件对象的全局属性$toast

Vue.prototype.$toast = toast;

}

}

export default obj; //导出，main.js导入、安装