# 什么是webpack

概念：

webpack是前端项目工程化的具体解决方案

主要功能：

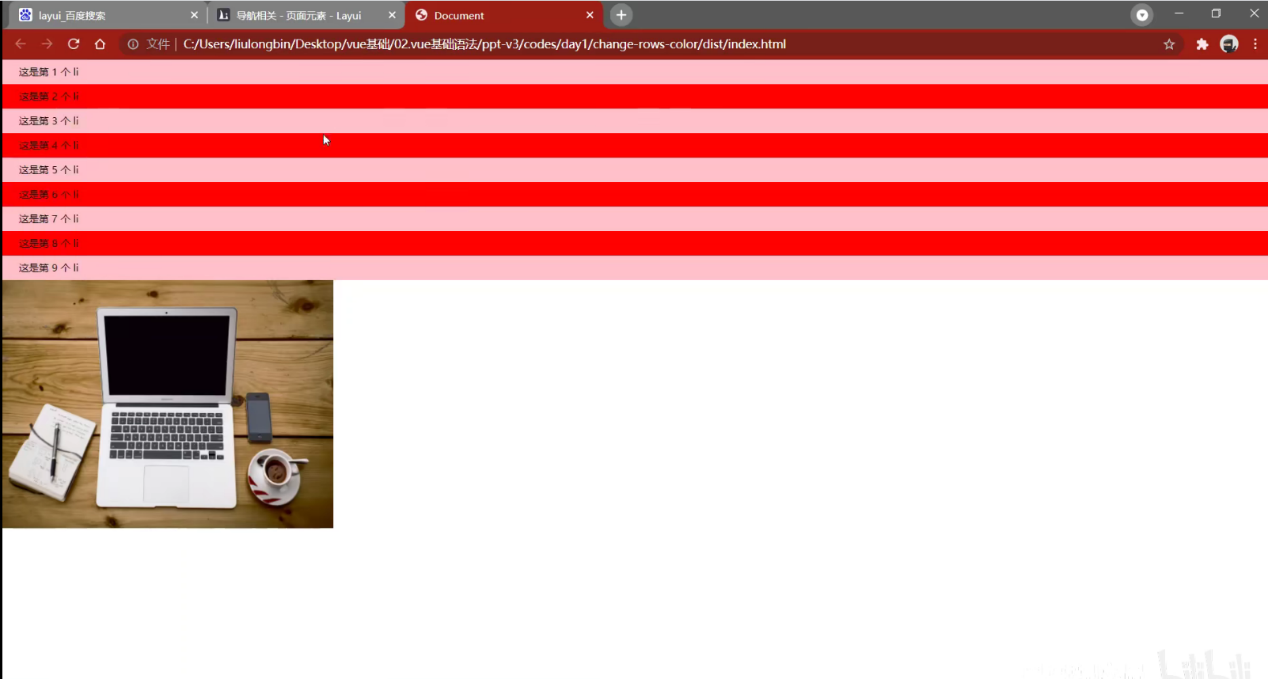
它提供了友好的前端模块开发支持，以及代码压缩混淆、处理浏览器Javascrpit的兼容性、性能优化等强大的功能

好处：让程序员把工作的重心放到具体功能的实现上，提高了前端开发效率和项目的可维护性

注意：目前vue，react等前端项目，，就基本上都是基于webpack进行工程化开发

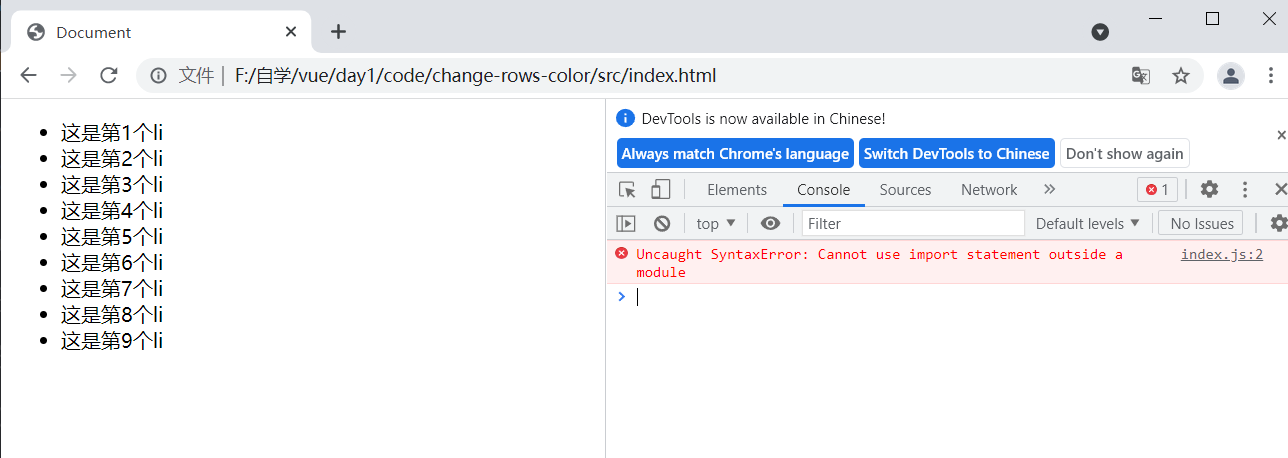
## Webpack的基本使用

### 创建列表隔行变色项目



1. 新建项目空白目录，并运行**npm init -y**命令。初始化包管理配置文件**packagejjson**
2. 新建**src**源代码目录
3. 新建**src->index.html**首页和**src->index.js**脚本文件
4. 初始化页面基本的结构
5. 运行 **npm install jquery -S** 命令，安装Jquery (-S表示把安装的模块显示再package文件的dependencies对象中，-S保存的模块再开发与上线阶段都是需要使用的，-S是 —save的简写)
6. 通过ES6模块化的方式导入JQuery，实现列表隔行变色效果

项目结果是会出错的，要使用webpack来解决兼容问题



### 2.在项目中安装webpack

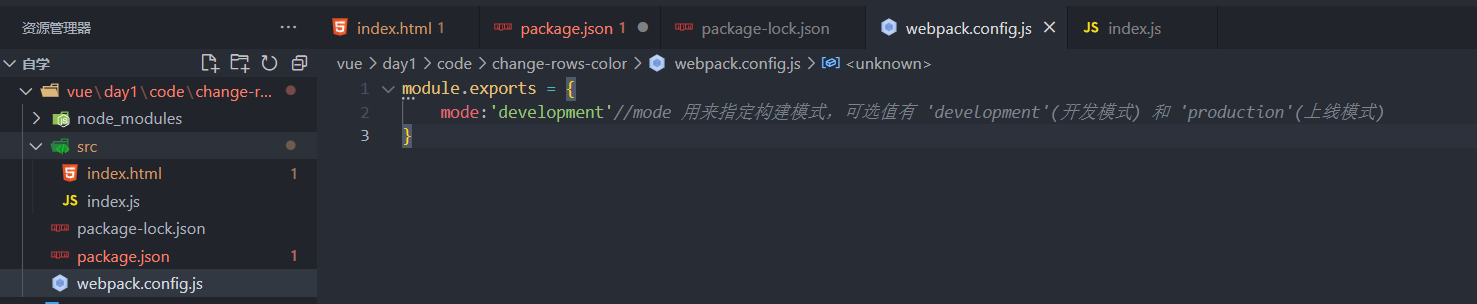
在终端运行如下两个程序，安装webpack相关的两个包

**npm install** [**webpack@5.42.1**](mailto:webpack@5.42.1)[**webpack-cli@4.7.2**](mailto:webpack-cli@4.7.2) **-D**

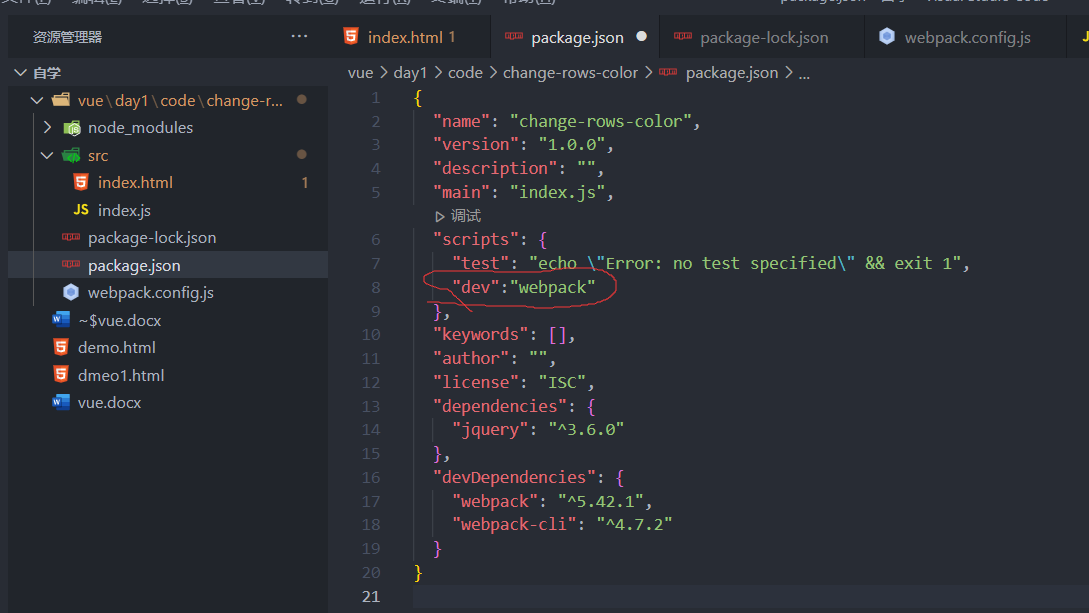
（-D的作用与-S类似，只是保存的对象不同，而且-D保存的模块只在开发阶段使用，上线阶段不使用，-D是—save-dev的简写）

### 3.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：

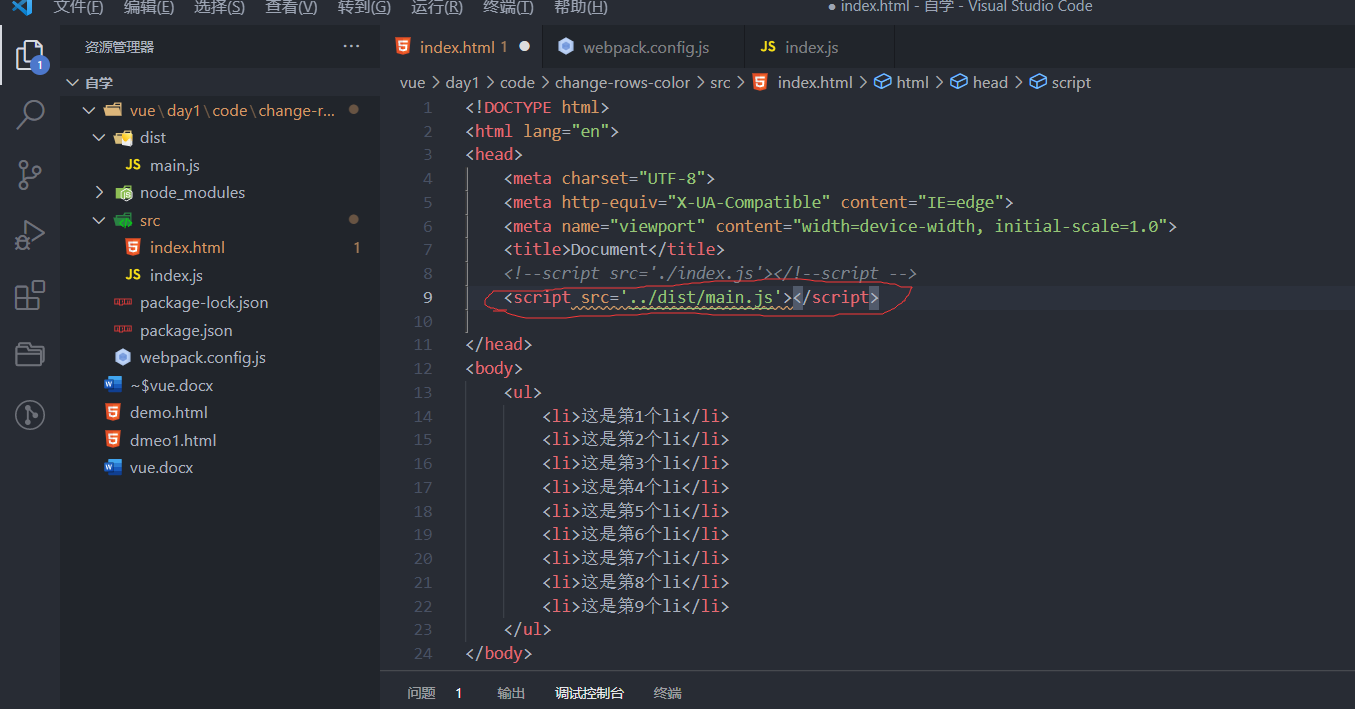


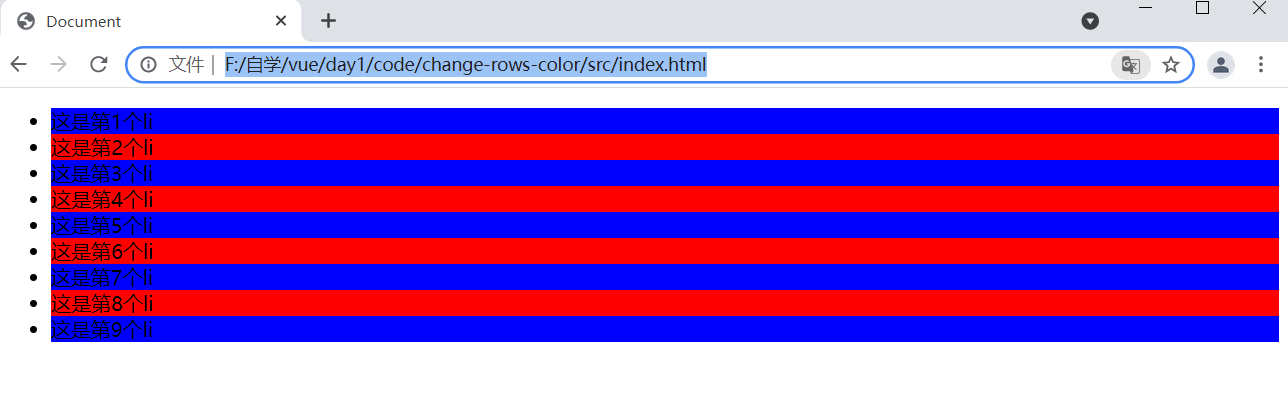
**2.**在 **package.json** 的**scrpits** 节点下，新增dev脚本如下：



Script节点下的脚本，可以通过 **‘npm run’** 执行，例如： **‘npm run dev’**

**3.**在终端中运行 **npm run dev** 命令，启动 webpack进行项目的打包构建，并且在引入时不引入index.js，而是引入webpack为我们在根目录下新建立的**dist**文件夹中的**main.js**





页面成功运行

Webpack会为我们自动修改js文件中的不兼容老版本浏览器的js语法（例如es6中的let，const定义，import引入），将它改为基本的js语法后放入根目录下的 **‘dist ’**  文件夹里的 **‘main.js ’** 中，在html页面引入js时引入这个 **‘main.js’** 文件即可

**4.webpack.config.js文件的作用**

webpack.config.js 是webpack的配置文件。webpack在真正开始打包构建之前，，会先读取这个配置文件，从而基于给定的配置，对项目进行打包。

注：由于webpack是基于node.js开发出来的打包工具，因此在它的配置文件中，支持使用node.js相关的语法和模块进行webpack的个性化配置。

**5.webpack中的默认约定**

在webpack 4.x和5.x的版本中，有如下的默认约定：

1. 默认的打包入口文件为src->index.js
2. 默认的输出文件路径为dist->main.js

注：可以在**webpack.config.js**中修改打包的默认约束

**6.自定义打包的入口与出口**

在 **‘webpack.config.js‘**  配置文件中通过 ’**entry‘** 节点指定打包的入口。通过 **’output‘** 节点指定打包的出口 ，在打包出口中用 ‘**filename’** 属性来指定打包完成后的文件名，’**plugin’** 属性用来指定调用的插件（用数组表示），示例代码如下：



### 4.安装webpack插件

**webpack插件的作用：**

通过安装和配置第三方的插件，可以扩展webpack的能里，从而让webpack用起来更方便。最常用的webpack插件有如下两个：

1. **webpack-dev-server**

●类似于node.js阶段用到的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

1. **html-webpack-plugin**

**●**webpack中的HTML插件（蕾仕于一个模板引擎插件）

**●**可以通过次插件自定制index.html页面的内容

#### 1.webpack-dev-server

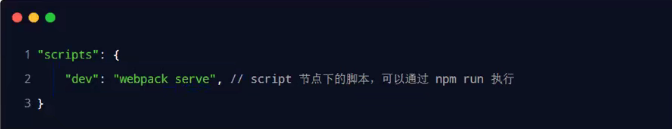
**安装webpack-dev-server**

运行如下的命令，即可在项目中安装此插件

**npm install** [**webpack-dev-server@3.11.2**](mailto:webpack-dev-server@3.11.2) **-D**

**配置webpack-dev-server**

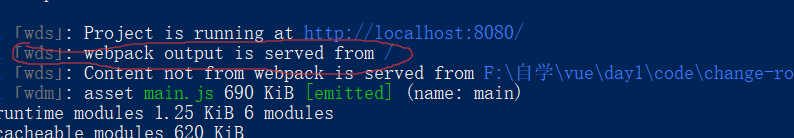
●修改package.json->script中的dev命令如下：



●再次运行npm run dev命令，重新进行项目的打包

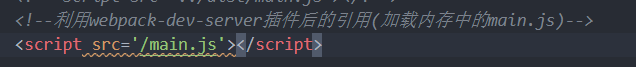
●在游览器中访问<http://localhost:8080>地址，查看自动打包效果

注：webpack-dev-server会启动一个实时打包的http服务器，打包完成后的文件不会存放到物理磁盘上（即电脑中），它会把打包好的文件放到内存中，储存的位置为下图所示：



上图中储存位置为 ‘/’ ，打包完成后的文件名为 ’main.js’

引用的路径即为：’/main.js’



在webpack.config.js配置文件中在导出时可以使用 ’devServer’ 属性对webpack-dev-server插件进行更多的配置，例：



注：修改了config文件或修改了package.json文件需要重新打包服务器

#### 2.html-webpack-plugin

运行如下指令进行安装：

npm install [html-webpack-plugin@5.3.2](mailto:html-webpack-plugin@5.3.2) -D

插件作用：

帮助我们把src下的index主页复制到根目录下，这样开启webpack-dev-server服务后打开本地服务器8080端口地址时就会自动显示index.html首页的内容了

配置html-webpack-plugin:



解惑html-webpack-plugin：

通过HTNL插件复制到项目根目录中的index.html页面，也被放到了内存中

HTML插件在生成的index.html页面，自动注入了打包的index.js文件

### 5.webpack中的loader（加载器）

Loader概述：

在实际开发过程中，webpack默认只能打包处理以.js后缀名结尾的模块。其他非.js后缀名结尾的模块，webpack默认处理不了，需要调用loader加载起才可以正常进行打包，否则会报错！

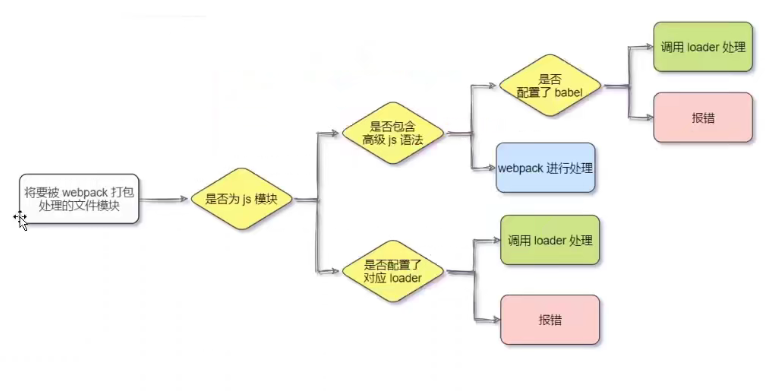
Loader加载器的作用：协助webpack打包处理特定的文件模块，例如：

css-loader 可以打包处理.css相关的文件，

less-loader可以打包处理.less相关的文件，

babel-loader可以打包处理webpack无法处理的高级JS语法

loader的调用过程：



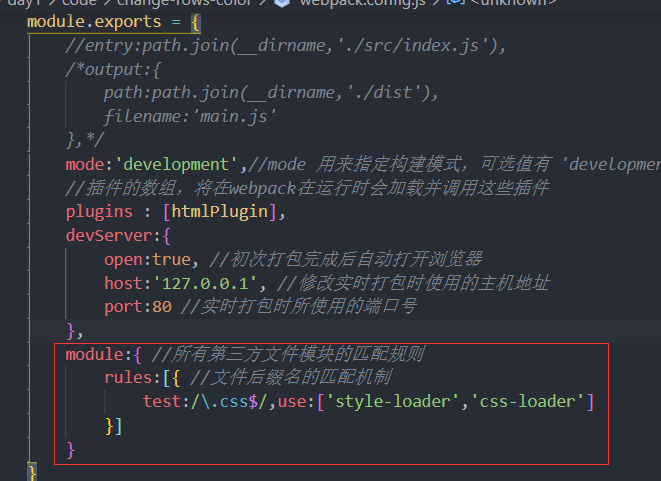
一般css，less文件都会在js文件中进行引入

#### 使用css-loader对css文件进行打包

1. 安装css-loader：

npm install [style-loader@3.0.0](mailto:style-loader@3.0.0) [css-loader@5.2.6](mailto:css-loader@5.2.6) -D

1. 在webpack.config.js的module->rules数组中，添加loader规则如下：

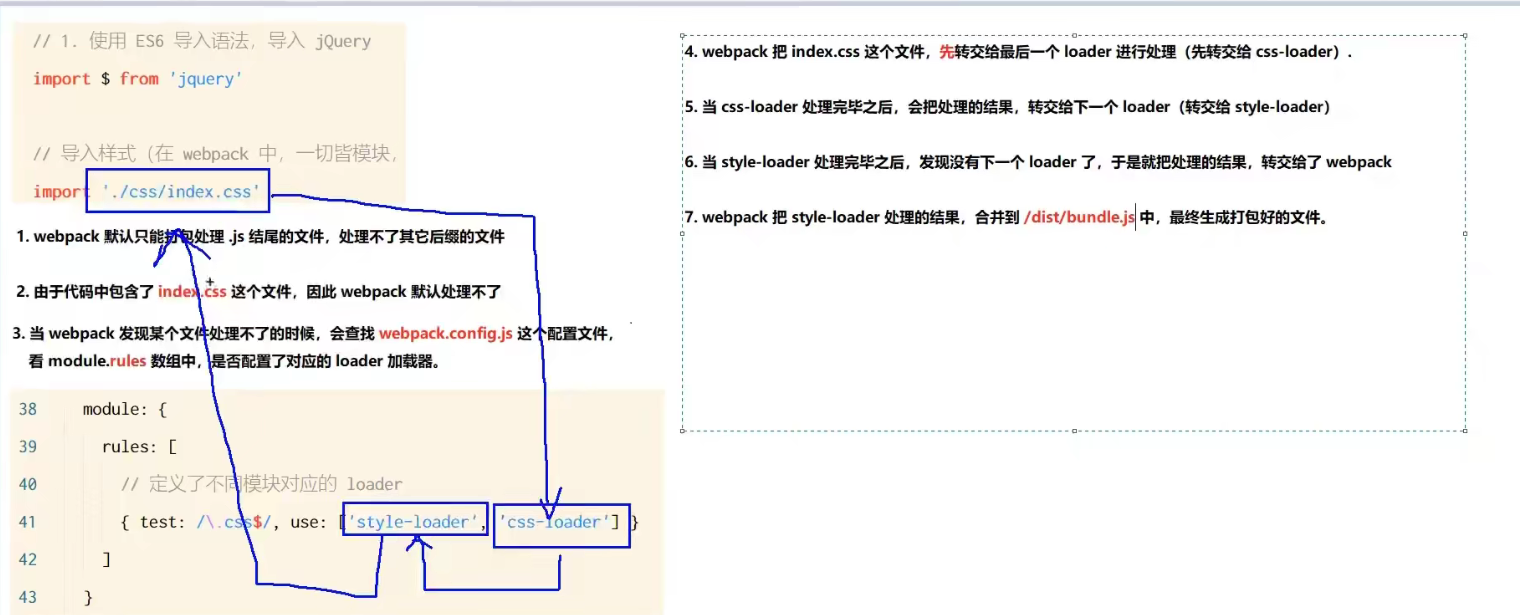


其中：test表示匹配的文件类型，use表示对应要调用的loader

注：

use数组中指定的loader顺序时固定的

多个loader的调用顺序是：从后往前调用



#### 使用less-loader对less文件进行打包

安装：

npm install [less-loader@10.0.1](mailto:less-loader@10.0.1) [less@4.1.1](mailto:less@4.1.1) -D

在webpack.config.js文件中添加规则：



#### 打包处理样式表中与url路径相关的文件（例如图片）

安装：

npm install [url-loader@4.1.1](mailto:url-loader@4.1.1) [file-loader@6.2.0](mailto:file-loader@6.2.0) -D

在webpack.config.js文件中的module->rules数组中添加规则：



其中？之后的是loader的参数项

注：

limit用来指定图片的大小，单位是字节（byte）

只有≤limit大小的图片，才会被转化为base64格式的图片

base64格式的优缺点： 可以防止发送一些不必要的网络请求，例如打开网页时的小图片，但转为base64格式后体积会增大一些（所以大体积图片不适合转为base64格式）

#### 打包处理js文件中的高级语法

webpack只能打包处理一部分高级javascript语法（let，const定义变量，箭头函数等）。对于那些webpack无法处理的高级js语法，需要借助于babel-loader进行打包处理。例如webpack无法处理下面的javascript代码：

（@info为装饰器，调用info函数，为person类添加info属性）

**安装babel-loader相关的包：**

npm i [babel-loader@8.2.2](mailto:babel-loader@8.2.2) @babel/core@7.14.6 @babel/plugin-proposal-decorators@7.14.5 -D

在webpack.config.js的module->数组中添加如下规则：



注意：必须使用 exclude 指定排除项，因为 node\_modludes 目录下的第三方包不需要被打包（第三方不用管兼容性问题，包已经自己解决了兼容性问题了）

**配置babel-loader：**

在根目录目录下，创建名为babel.config.js的配置文件，定义babel的配置项如下：

module.exports = {

//声明babel可用插件

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

}

### 6.打包发布

**1.为什么要打包发布**

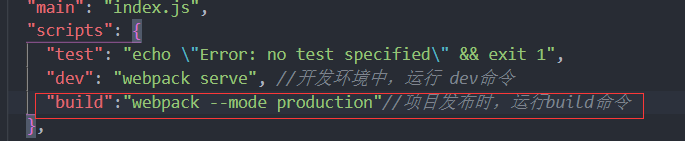
项目开发完成之后，需要使用webpack对项目进行打包发布，主要原因有以下两点:

1. 开发环境下，打包生成的文件存放在内存中，无法获取到最终打包生成的文件
2. 开发环境下，打包生成的文件不会进行代码压缩和性能优化

为了让项目能够在生产中高性能的运行，因此需要对项目进行打包发布

**2.配置webpack的打包发布**

在package.json文件的scrpits节点下，新增bulid命令如下：



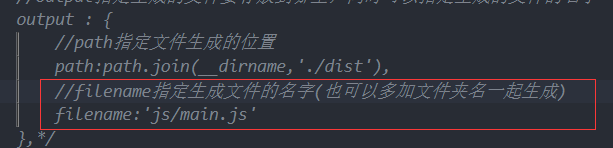
--model市一个参数项，用来指定webpack的运行模式。production代表生成环境，会对打包生成的文件进行代码压缩和性能优化。

注意：通过 –model指定的参数项，会覆盖webpack.config.js中的model选项

1. **优化图片和js文件的打包存放路径**

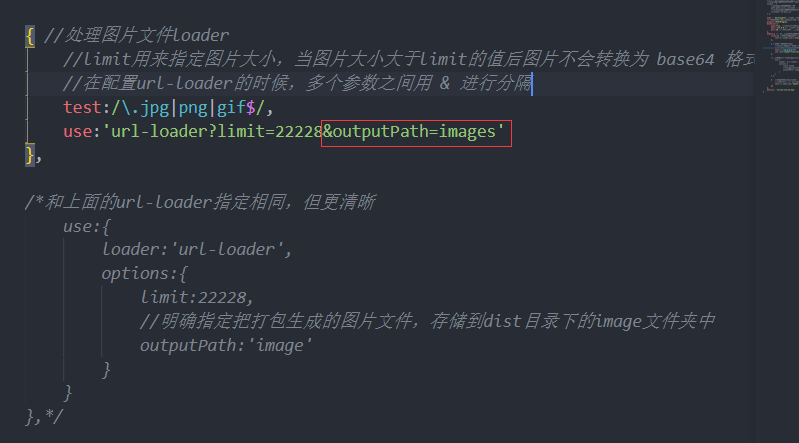
**把javascrpit文件统一生成到js目录中**

在webpack.config.js配置文件的output节点中，进行如下的配置：



**把图片文件统一生成到image目录中**

修改webpack.config.js中url-loader配置项，新增outputPath选项即可指定普通文件的输出路径：



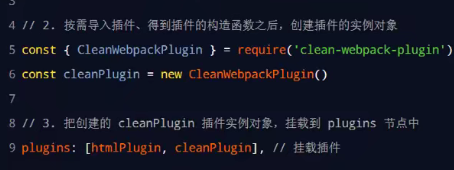
1. **配置和使用clean-webpack-plugin插件自动删除dist目录**

作用：在每次打包发布时自动清理掉dist目录中的旧文件

安装：

npm i [clean-webpack-plugin@3.0.0](mailto:clean-webpack-plugin@3.0.0) -D

配置：



### source map

1.**什么是source map**

Source Map 就是一个信息文件，里面储存着位置信息。也就是说，Source Map 文件中存储着压缩混淆后的代码，所对应的转换前的位置。

有了它，出错的时候，除错工具将直接显示原始代码位置，而不是转换后的代码，能够极大的方便后期的调试

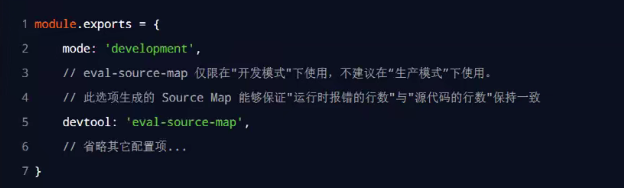
**2.默认 Source Map 问题**

开发环境下默认的Source Map，记录的是生成后的代码的位置。会导致运行时报错的行数与源代码的行数不一致的问题。示意图如下：



**3.解决默认Source Map的问题**

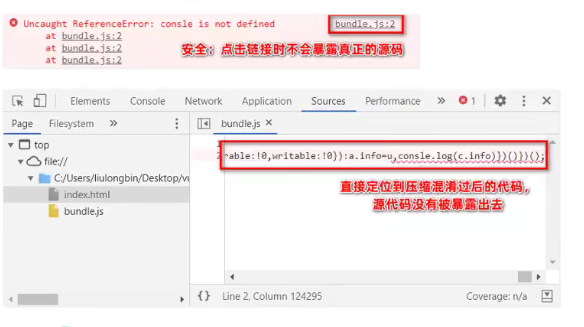
开发环境下，推荐在webpack.config.js中添加如下的配置，即可在运行时报错的行数与源代码的行数保持一致：



注意：在生产阶段需要使用另外的Source Map不然会有源代码暴露风险

**4.webpack生成环境（上线阶段）的Source Map**

在生产环境下，如果省略devtool选项，则最终生产的文件中不包含Source Map。这能够防止源代码通过Source Map的形式暴露给别人

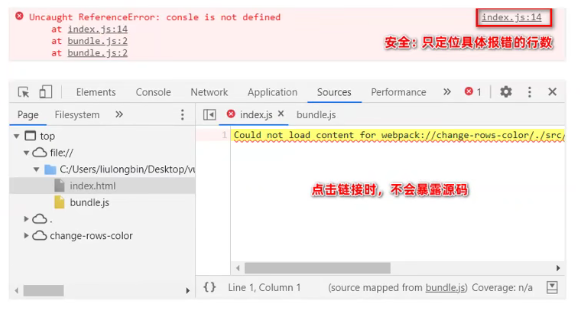


**5.解决在上线阶段实现报错后能定位到行数的方法：**

只定位行数不暴露源码

在生产环境下，如果只想定位报错的具体行数，且不想暴露源码。此时可以将devtool的值设置为：nosources-source-map

效果：



**6.定位行数且暴露源码**

在生产环境下，如果想在定位报错行数的同时，展示具体报错的源码。此时可以将devtool的值设置为：source-map

同时会生成对应的js.map文件

效果：



采用此选项后：你应该将你的服务器配置为不允许普通用户访问source map文件

**7.Source Map的最佳实践**

（1）开发环境：

●建议把devtool值设置为：eval-source-map

●好处：可以精确定位到具体的错误行

（2）生产环境：

●建议关闭Source Map 或将devtool的值设置为 nosources-source-map

●好处：防止源码泄露，提高网站的安全性

### 8.实际开发

在实际开发中是不需要我们配置webpack的

●在实际开发过程中会使用命令工具（俗称cli）一键生成带有webpack的项目

●开箱即用，所有webpack配置项都是现成的

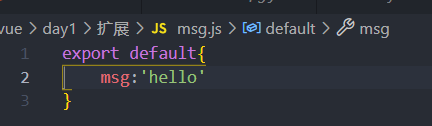
●我们只需要知道webpack中的基本概念即可

### 9.扩展

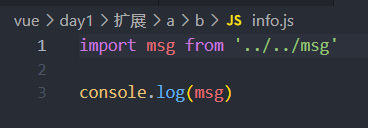
（1）es6中：export default {} 可以用来导出

在别的文件中使用import进行接收

例：在msg.js中导出一个msg属性



在info.js中接收：



（2）webpack中@的原理

在用@引入时会从src源代码目录，从src目录往内查找（例如在src/images/i.png可以直接写成@/1.png），直接引入则以文件为中心从内往外查找（就时平常的’./’或’../’），需要先在webpack.config.js中进行配置:



这样就能在引入时使用@来代表src目录了