Webpack-老

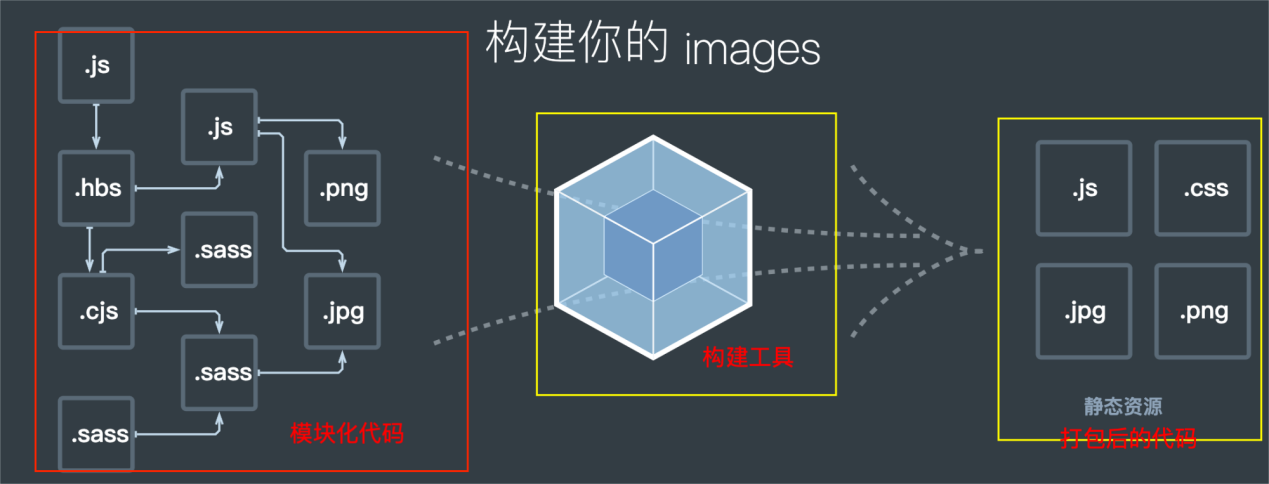
vite- 新

### 构建工具-webpack

-当我们习惯了在node中编写代码的方式后，再回到前端编写html、css、js这些东西会感到各种的不便。比如：不能放心的使用模块化规范（因为存在浏览器兼容性问题）、及时可以使用模块化规范也会面临模块过多时的加载问题。

-我们就迫切的希望有一款工具可以对代码进行打包，将多个模块打包成一个文件。这样一来即解决了兼容性问题，又能解决模块过多的问题。

-构建工具就起到这样一个作用，通过构建工具可以将使用ESM规范编写的代码转换为旧的JS语法，这样可以使得所有的浏览器都可以支持代码；（解决兼容性问题）



开发依赖：devDependencies (只有开发的时候需要 -DD)

项目依赖：dependencies（网页中需要）

src文件里的都是前端运行的；

除此之外都是在node中运行的

//注意区分前端模块还是 node模块，也就是commonJS还是EMSJS，二者导出模块和引入模块的方式是不一样的

index.js--主文件/入口文件

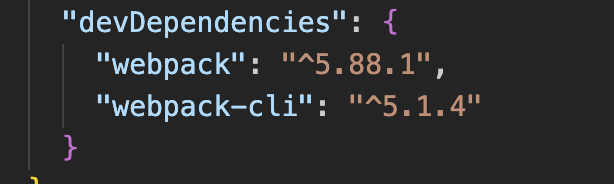
Wabpack介绍<https://www.webpackjs.com/concepts/>

#### webpack安装

1、初始化项目 yarn init -y (全局安装yarn： sudo npm install -g yarn)

2、安装webpack 、webpack-cli (（可以在命令行运行webpack,也就是端口，命令行工具）

npm add -D webpack webpack-cli (加-D的含义是 webpack在开发的时候使用，它只是一个打包工具)



3、在项目中创建src目录，然后编写代码（src/index.js）--源码

4、执行 yarn webpack/yarn webpack--watch(自动build)来对代码进行打包（打包后观察**dist目录/main.js**）--打包后的代码;

或者使用 yarn build来对代码进行打包，但是注意使用yarn build的时候一定是提前在package.json中已经配置好了，如下图：



#### webpack的按需打包

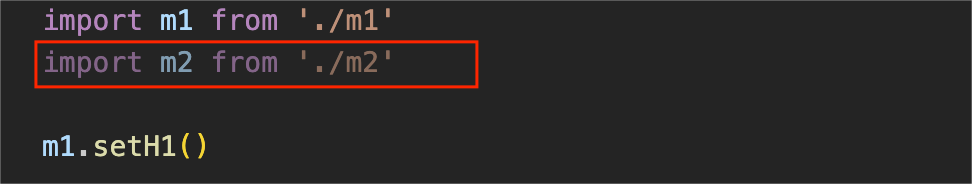
webpack是**按需打包**，

他会判断是否需要打包，当模块需要使用的时候，就会打包；

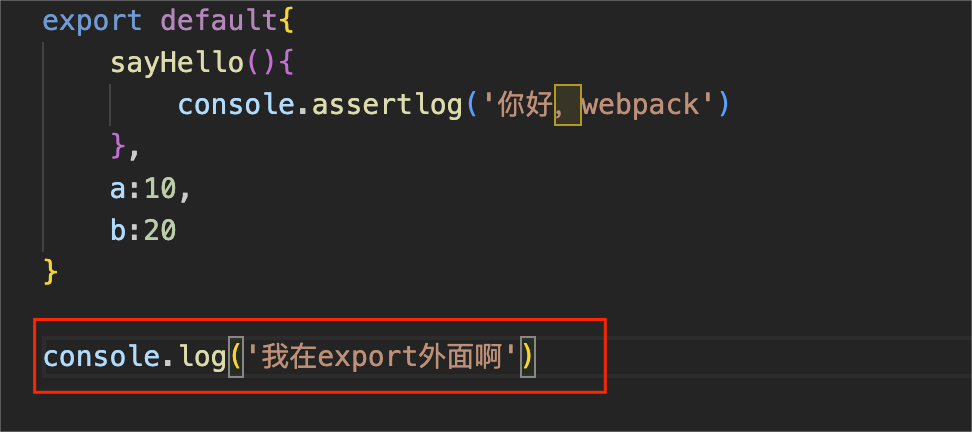
不需要使用的时候就不会打包；

但是，当一个js文件中有未包含在export default {}中的语句的时候，此时webpack无法判断其是否使用，所以他就会将该语句打包（没有在export {}），如下图所示：（前提在index.js文件中引入该模块，但是未使用）

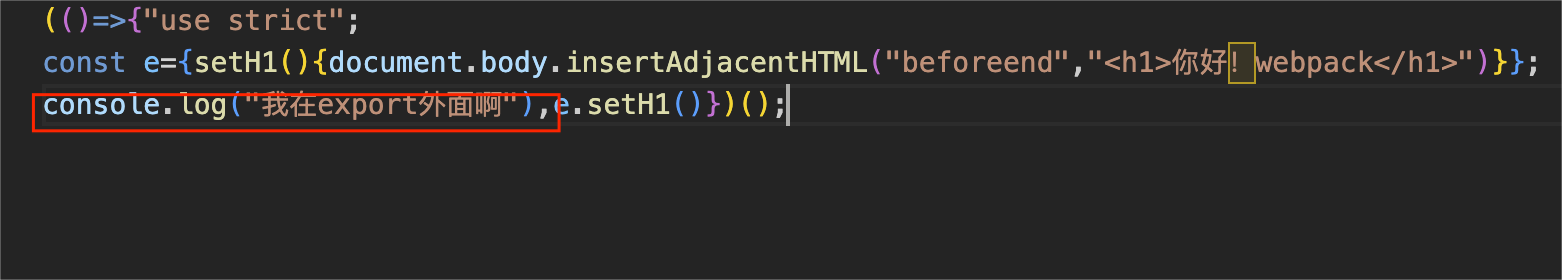
src文件中的index.js文件



src文件中的m2.js模块



dist（打包后的文件）中的main.js文件



#### webpack的配置：

##### model：

mode:"production", //设置打包模式，production表示生产模式， development开发模式

##### entry:

entry:"./src/index.js", //用来指定打包时的主文件， 默认 ./src/index.js 即：entry:''，一般不修改

// entry:["./src/a.js", "./src/b.js",] //可以打包多个文件，并且将他们打包到一个main.js中

// entry:{ //分别打包多个文件，并且文件名为entry中的键名，比如hello和b

// hello:"./src/a.js",

// b:"./src/b.js",

// },

##### output

output:{ //配置代码打包后的地址

path:path.resolve(\_\_dirname, "hello"), //指定打包的目录，必须要绝对路径,表示路径配置到当前目录下的 hello文件中；

// 一般打包到dist文件

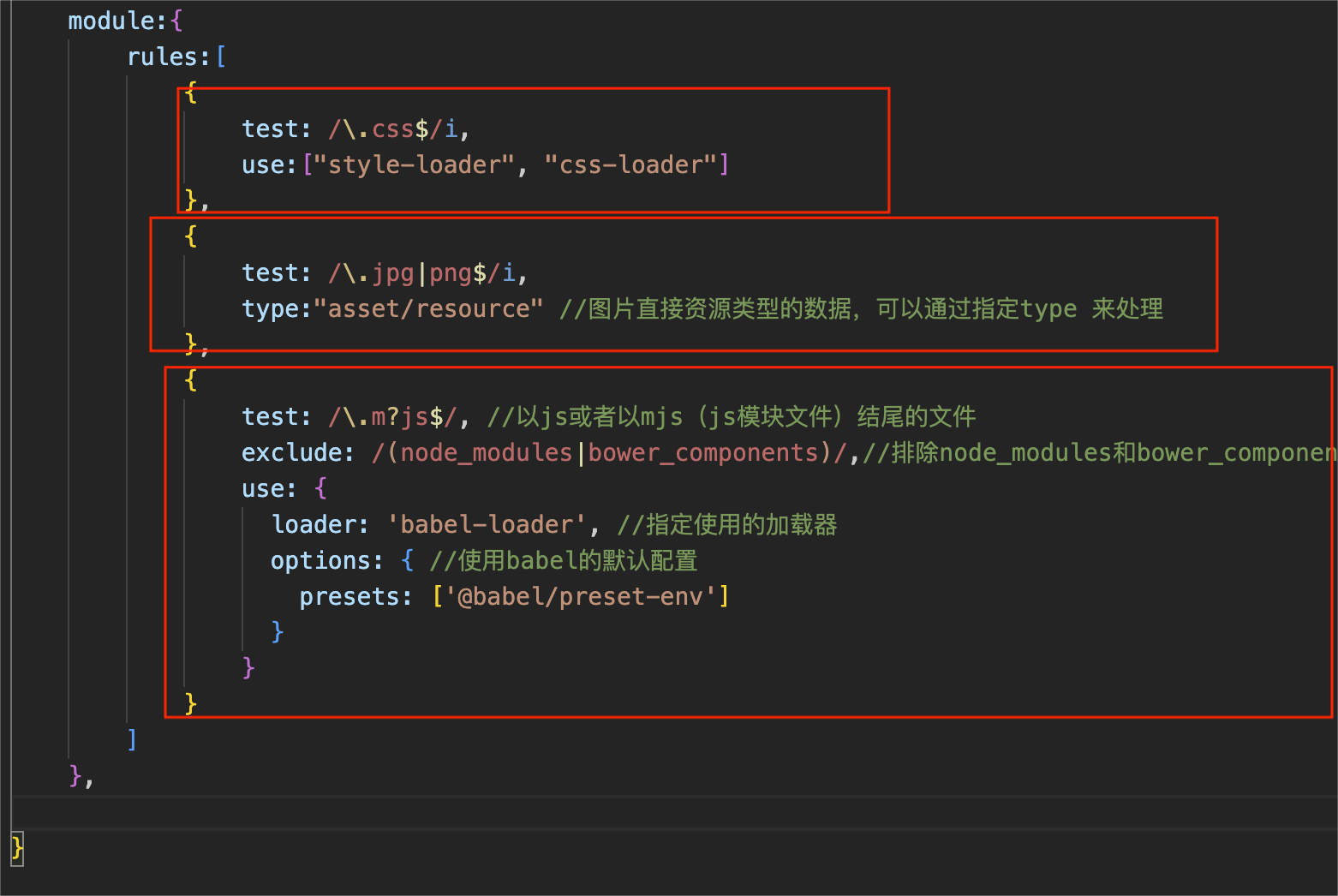
filename:"bundle.js", //打包后的文件名 ， 前后文件名不一样时，会保留二者

// filename:"[name].js" //表示继承源码的名字 [name]中name表示变量， 可以查看webpack 文档

clean:true, //自动清理打包目录，你的path是什么就清理什么

},//配置代码打包后的地址

##### module:



###### loader-其他文件类型

1-源码 不能写在 打包后的文件中；

也就是 webpack打包后的dist文件中不能有手写文件，所有的文件都应该能被webpack管理。

2-webpack默认只能 打包.js文件

1. 如果想让webpack处理其他类型的文件，需要不同类型的loader，并且需要在源码index.js文件中引入css文件

-以.css文件为例：  
 -使用步骤：

-安装：yarn add css-loader -D和yarn add style-loader -D (使样式生效)

-配置：

module:{

rules:[{

test: /\.css$/i,

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

}],

},

//style-loader负责用，css-loader负责引的，所以他们有顺序，先引用，再转化，也就是先css-loader,后style-loader,而且【】规定倒叙。

###### loader-照片

{

test: /\.jpg|png$/i,

type:"asset/resource" //图片直接资源类型的数据，可以通过指定type 来处理

},

###### \_babel:

- 在编写js代码的时，经常需要使用一些js中的新特性，而新特性在旧的浏览器中兼容性不好。此时就导致我们无法使用一些新的特性。

- 但是我们现在希望能够使用新的特性，我们可以采用折中的方案。依然使用新特性码编写代码，但是代码编写完成时，我们可以通过一些工具将新代码转换为旧代码；

- babel就是这样一个工具，可以将新的js语法转换为旧的js，**以提高代码的兼容性；**

- 如果我们希望在webpack支持babel，则需要向webpack中引入babel的loader。

- 使用步骤：

1. 安装 npm install -D babel-loader @babel/core @babel/preset-env
2. 配置

module: {

rules: [

{

test: /\.m?js$/,//处理以js或者以mjs（js模块文件）结尾的文件

exclude: /(node\_modules|bower\_components)/,

//排除node\_modules和bower\_components文件

use: {

loader: 'babel-loader',//指定使用的加载器

options: {//使用babel的默认配置

presets: ['@babel/preset-env']

}

}

}

]}

1. 可以通过 package.json 文件中设置兼容列表：

"browserslist": [

"defaults"

]

浏览器参考博客：

<https://juejin.cn/post/6844903669524086797>

#### 插件（plugin）

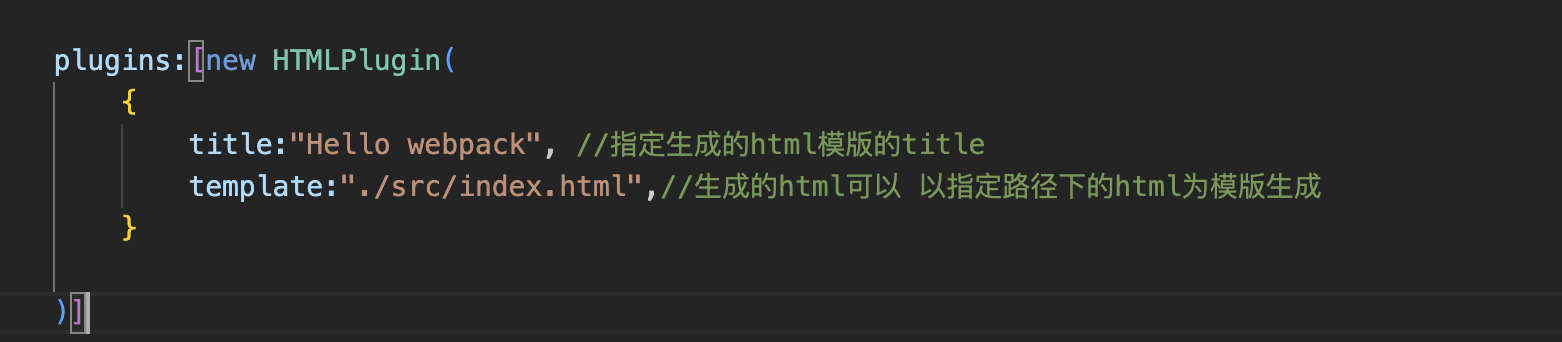
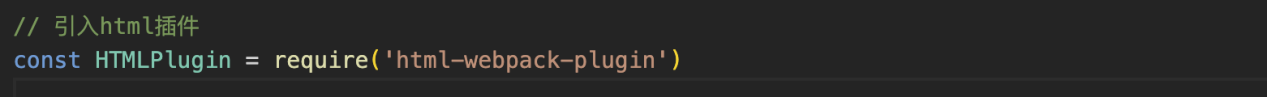
- 插件用来为webpack来扩展功能（对代码本事不进行任何操作）

- html-webpack-plugin

-这个插件可以在打包代码后，自动在打包目录生成html页面

-使用步骤：

1. 安装依赖 npm add html-webpack-plugin
2. 配置插件



#### devtool:

devtool:"inline-source-map",//设置代码映射，运行的是的打包后的main.js，但是我们可以对源码进行调试

#### 开发服务器（webpack-dev-server）

安装：

-yarn add -D webpack -dev-server

-启动：yarn webpack serve --open

#### webpack.config.js文件中：

// 这是在node中运行的文件

const path = require('path')

module.exports ={

mode:"production", //设置打包模式，production表示生产模式， development开发模式

// entry:"./hello/hello.js", //用来指定打包时的主文件， 默认 ./src/index.js 即：entry:''，一般不修改

// entry:["./src/a.js", "./src/b.js",] //可以打包多个文件，并且将他们打包到一个main.js中

// entry:{ //分别打包多个文件，并且文件名为entry中的键名，比如hello和b

// hello:"./src/a.js",

// b:"./src/b.js",

// },

output:{

// path:path.resolve(\_\_dirname, "hello"), //指定打包的目录，必须要绝对路径,表示路径配置到当前目录下的 hello文件中；

// 一般打包到dist文件

// filename:"bundle.js", //打包后的文件名 ， 前后文件名不一样时，会保留二者

// filename:"[name].js" //表示继承源码的名字 [name]中name表示变量， 可以查看webpack 文档

clean:true, //自动清理打包目录，你的path是什么就清理什么

},//配置代码打包后的地址

module:{

rules:[

{

test: /\.css$/i,

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

},

{

test: /\.jpg|png$/i,

type:"asset/resource" //图片直接资源类型的数据，可以通过指定type 来处理

},

{

test: /\.m?js$/, //以js或者以mjs（js模块文件）结尾的文件

exclude: /(node\_modules|bower\_components)/,//排除node\_modules和bower\_components文件

use: {

loader: 'babel-loader', //指定使用的加载器

options: { //使用babel的默认配置

presets: ['@babel/preset-env']

}

}

}

]

},

plugins:[new HTMLPlugin(

{

title:"Hello webpack", //指定生成的html模版的title

template:"./src/index.html",//生成的html可以 以指定路径下的html为模版生成

}

)],

devtool:"inline-source-map",//设置代码映射，运行的是的打包后的main.js，但是我们可以对源码进行调试

}

### 构建工具-vite

-vite也是前端的构建工具

-相比较于webpack，vite采用了不同的运行方式：

-开发时，并不对代码打包，而是直接采用ESM的方式运行项目

-在项目部署时，再对项目进行打包

-除了速度外，vite使用起来更加方便

-基本使用：

1. 安装开发依赖 vite
2. vite的源码目录就是项目根目录
3. 开发命令：

-vite 启动开发服务器

-vite build 打包代码

-vite preview 预览打包后的代码

-使用命令构建

npm create vite@lastest

yarn create vite

pnpm create vite

-配置文件：vite.config.js

-格式：

import {defineConfig} from “vite”

import legacy from “@vitejs/plugin-legacy”

export default defineConfig({

plugins:[

legacy({

targets:[“defaults”, “ie 11”]

})

]

})