2021年10月25日 星期一

Webpack 打包优化

**target file** ：可以使用自己的**vue3**的项目做实验

打包css文件

在在**mian.js**中**import**的**css**文件是被打包到独立的文件里面吗？

是的，以**link**标签的形式引入到**html**文件里面

以**link**标签引入的**css**是由**css**解析器解析的吗？

在.vue文件中的css是怎样处理的。

Css-loader 按照commonjs规范，将css 文件输出到 js中

**style-loader** 将 **js**中的样式挂载到**<style>**标签中，在**vue**开发中，打开调试工具，检查**element** 在**title**下面的**<style>**标签中的**css**样式就是 **style-loader**处理后的样式

为啥有的文件的解析规则要放在**module** 里面配置？？ **import ‘./css’** 会将这个**css**文件看做一个模块所以需要配置模块的解析规则

抽取**CSS**的插件

插件：

**Mini-css-extract-plugin**

**Const miniCssExtractPlugin = require(‘mini-css-extract-plugin**

**’)**

用**miniCssExtractPlugin.loader**取代**style-loader**

压缩**css**

开启**CSS**压缩，压缩的是**build**目录下面的**css**

**Autoprefixer**

**1: Npm I postcss-loader autoprefixer**

**2:Config in webbpack.config.js**

**Use: [‘css-loader’,’postcss-loader’]**

**3: setup postcss.config.js**

**module.exports = {**

**plugins: [require(‘autoprefixer’)]**

**}**

**4: config target browser**

关于**node**里面的模块作用域：

每个**js**文件都是一个模块有自己的作用域那种情况下被引入的文件不能正常访问呢？？

**Case1**：在**Index.html** 使用被打包的**JS**文件里面的变量

**Assets modules:**

**File-loader** ：将文件输出到指定目录用于对字体文件的处理

**Url-loader** ：处理图片的**url; base64**

**Raw-loader :** 将文件导入为字符串

若使用**webpack5** 上述的**loader** 不在维护

**module.exports = {**

**Rules: [**

**{**

**test: / \.(e0t | svg| ttf| woff)$/I,**

**type:assets,**

**parser: {**

**dataUrlCondition: {**

**maxSize: 8 \* 1024**

**}**

**},**

**generator: {**

**filename: ‘fonts/[name][ext]’**

**}**

**]**

**}**

当声明 **type** 为 **assets**时，会分为以下两种情形

**1**： **assets/resource** 文件大于**8kb** 外联

**2**： **assets/inline** 小于**8kb inline base64** 加载资源

**Tips :**

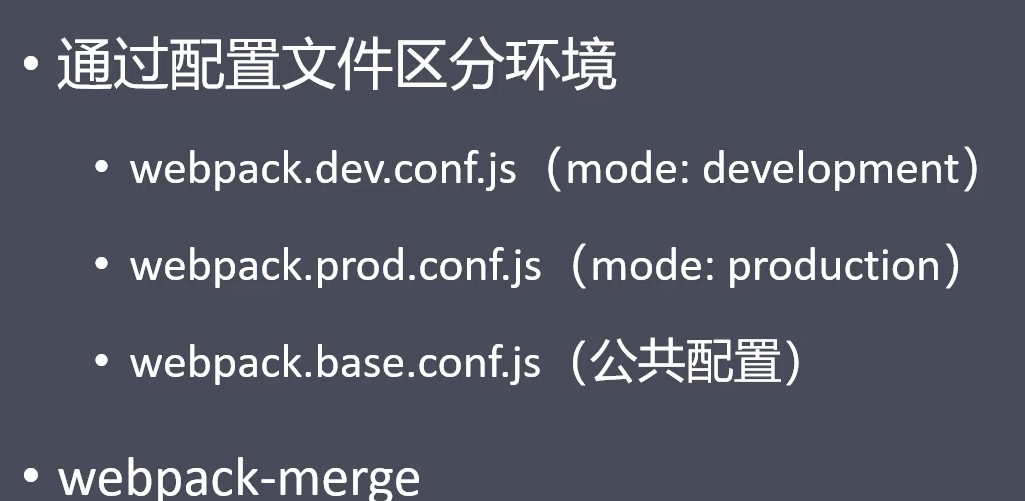
当配置了**webpack.output** 后 **,** 默认会将所有的文件输出到**output**文件夹下面配置**loader** 处理文件的输出位置时只需要指明这类资源输出的文件夹即可如上面的示例： **filename**： **’font/[name]’** 会将对应资源放到 **webpack**指定的**output**文件夹下面

打包环境变量****

终端里面切换**webpack** 在不同环境下**build**

**Webpack —env production**

**Webpack —env development**

配置文件区分打包环境****

**HTML-webpack-plugin**

作用：

**1:** 以指定的**template (x.html)**文件为模板，复制一个**x.html**文件，并将打包后的文件夹（**output**）下面的**js /css** 以 **link** 或者 **script** 形式引入**x.html**文件中。

**Notice :** 这个插件在不额外配置的情况下会自己生成一个**index.html**文件，但是只是最简单的内容，无法做额外的配置

**2:** 设置**html**变量

**3:** 压缩**html**

**babel**

**babel-preset-env** 所有最新转换规则的预设

**@babel/preset-env** 只能转译基本语法 **let const**

**@babel/polyfill** 转译所有**js**新语法会使打包文件体积变大

**Core-js** 按需转译**js**新语法就不需要使用 **polyfill**

**Babelrc.js**

**UseBuiltIns:usage,**

**Corejs: 3**

**targets**：**{}** 目标浏览器

开启**eslint**自动在打包时候修复**js**格式错误

使用 **eslintPlugin**插件

**New eslintPlugin({ fix:true})**

**Plugin:**

生命周期： **webpack**运行过程中的关键节点

钩子：

**1**：提前在可能增加功能的地方，预设好一个函数

**2 :** 标记当前所处位置

**3:** 钩子是按照顺序执行的****

详细用法，参考官网

**Compilation :**当前的模块对象

**loader**：

****

定义：是一个**es**模块，导出一个函数，在函数中对打包资源进行转换

**notice**！！！

**1**：自定义的**loader**必须引入

**2**：自定义的**loader**必须返回一段**js**代码因为**eval**（）函数会包裹 **loader**返回结果并执行；

将返回结果包装为**js**代码：

使用**json**转译解决字符串问题

**return ` module.exports = ‘${json.stringify(html}’ `**

**3**：多个**loader**只需要保证最后一个**loader**返回的是**js**代码即可。

**4**：若要在声明**loader**的函数中使用**this**；必须使用**function**声明**loader**函数

代码分割

**1**：多入口打包

**1-1**：**Entry**配置成对象

**1-2** 每个**html**文件，加载自己对应的**js**文件

**New HtmlWebpackPlugin({**

**chunk: [‘index’]**

**})**

**index.html** 加载**index.js**

这样的方式适合怎样的场景呢？？

**2**：公共文件提取第三方库

开启**webpack**的**optimization: splitChunks: chunks:all**

**3:** 动态导入

**1**：懒加载

使用时候加载

**1**：在使用该文件的时候采用如下写法

**import(/\* webpackChunkName: ‘util ‘ \*/‘./utils’).then((file) => {**

**file.say()**

**})**

使用场景？？？

和用户交互的文件；自定义触发条件

**2**：预加载

**import(/\* webpackChunkName:‘util ‘**， **webpackPrefetch:true \*/‘./utils’).then((file) => {**

**file.say()**

**})**

文件会预先拉取但是不会执行

**Tree Shaking**

一、**usedExports**

生产环境默认开启

**1:** 前提：

使用**ESMdoule**规范的模块才能执行 **tree shaking**

原因： **es** 具有静态语法分析功能可以提前知道模块间的依赖关系

配置：

**Optimization.usedExports(webpack.config.js)** 标记**dead code**

**Terser-webpack-plugin** 删除**dead code**

**optimization.minimize**：**true**

**optimization.minimizer: [new TerserPlugin()]**

**question**？？？

**Webpack** 采用的是**common js** 规范在一些配置文件中会使用**require**，开启了这个选项后会对这些文件**treeshaking**吗？

**Keypoint**

**Tree shaking source map** 存在兼容性问题

当

**Devtools: source-map | inline-source-map | hidden-source-map | nosources-source-map**

时，

**treeshaking**才能正常运行

二、 **sideEffects**

作用：将于无副作用的模块未使用的代码不会被打包****

副作用文件举例：

**1**：定义了一个**js**文件，可能的代码有给**window** 或者数组对象的原型上面添加了属性；

**2**：没有使用**export**

**3**：只在入口文件引入

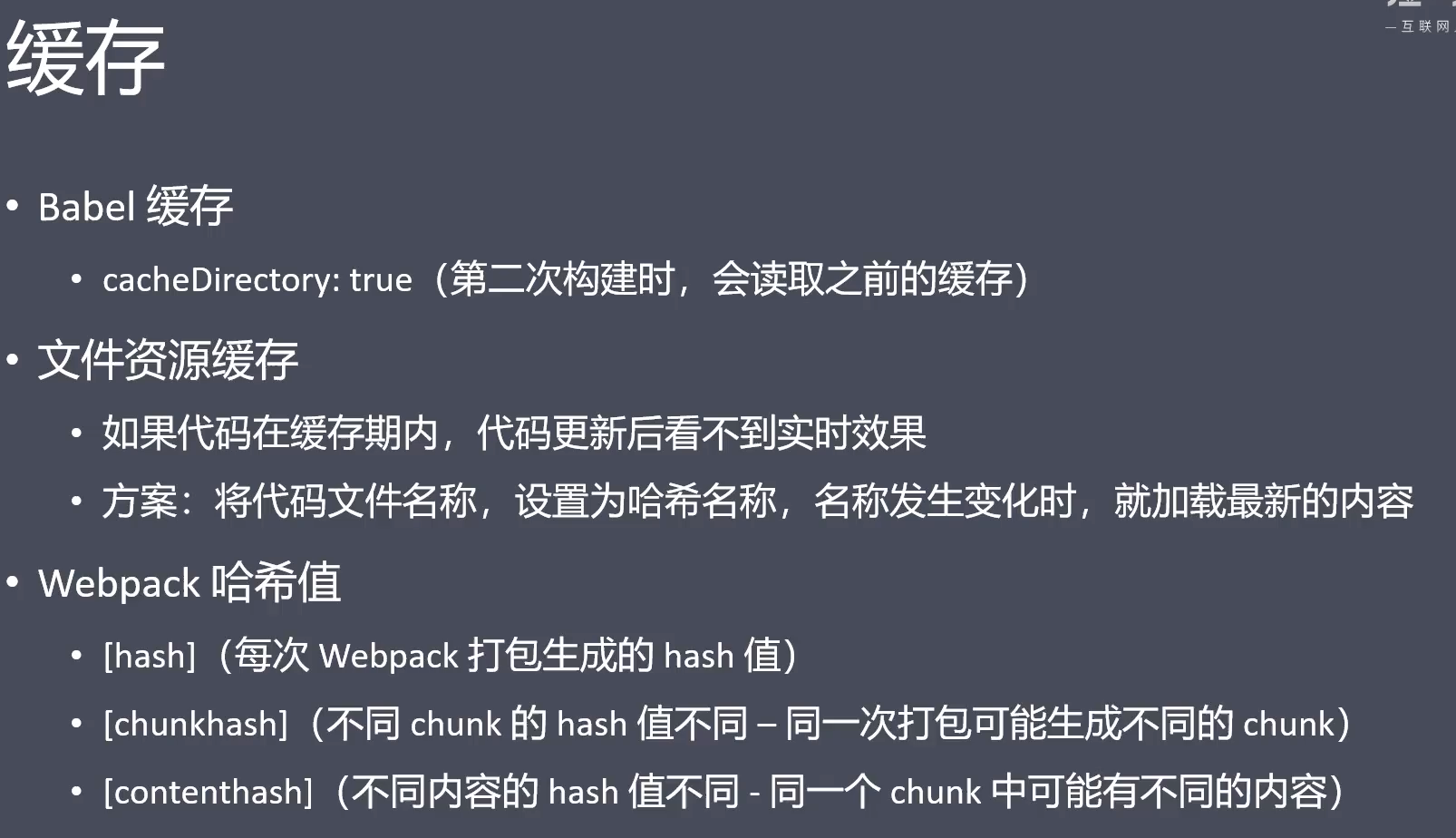
在现在的工程化开发模式中；若使用**exports/export default** 等形式定义的文件是一个模块有自己的作用域不会对全局的文件产生影响；若直接 **import ‘./index.js’** 并且这个**js**文件没有使用**export** ，那这个文件的作用域就是全局，可能会对全局的代码安全产生影响。

**Webpack**会删除无副作用并且没有导出任何成员的模块

定义了一个**index.js** 没有使用模块的形式导出

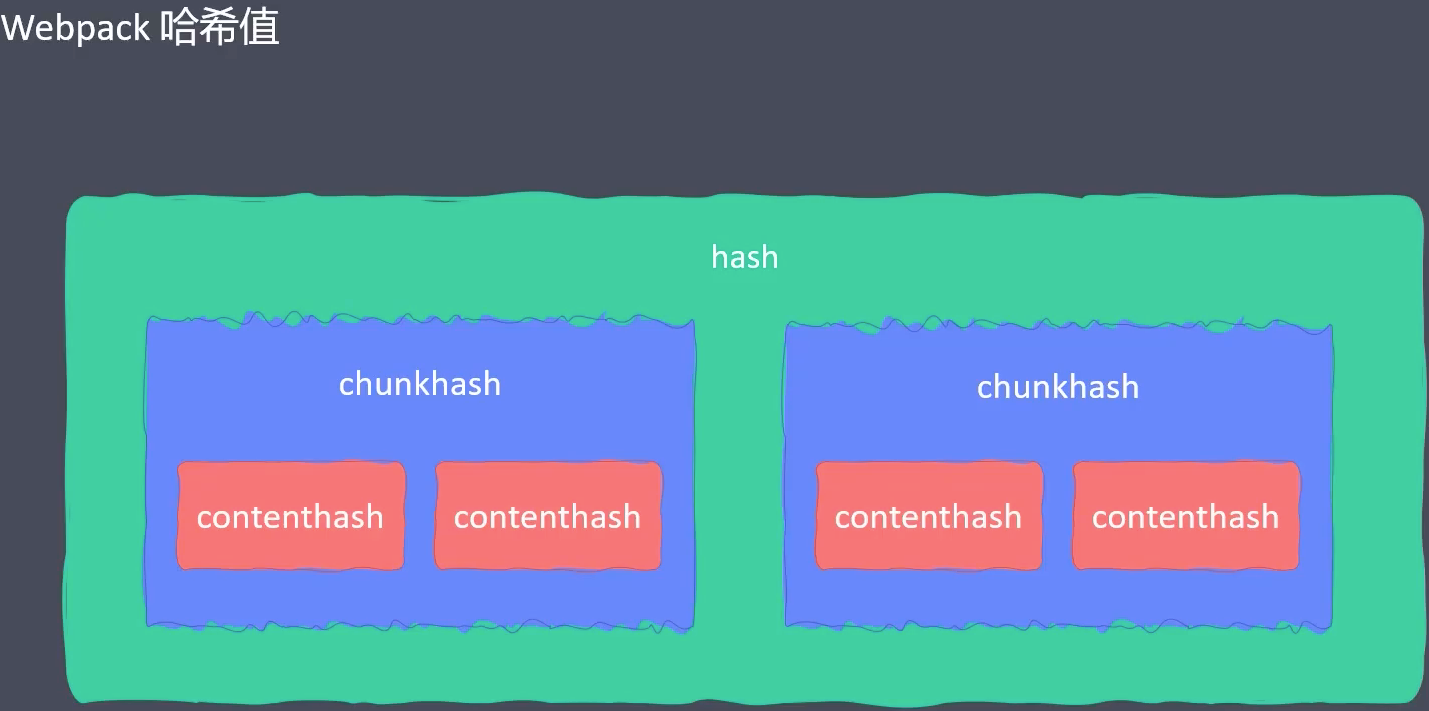
即使在入口文件中引入，当开启了 **sideEffects:true** 模式后 **webpack**不会打包该文件

缓存

****

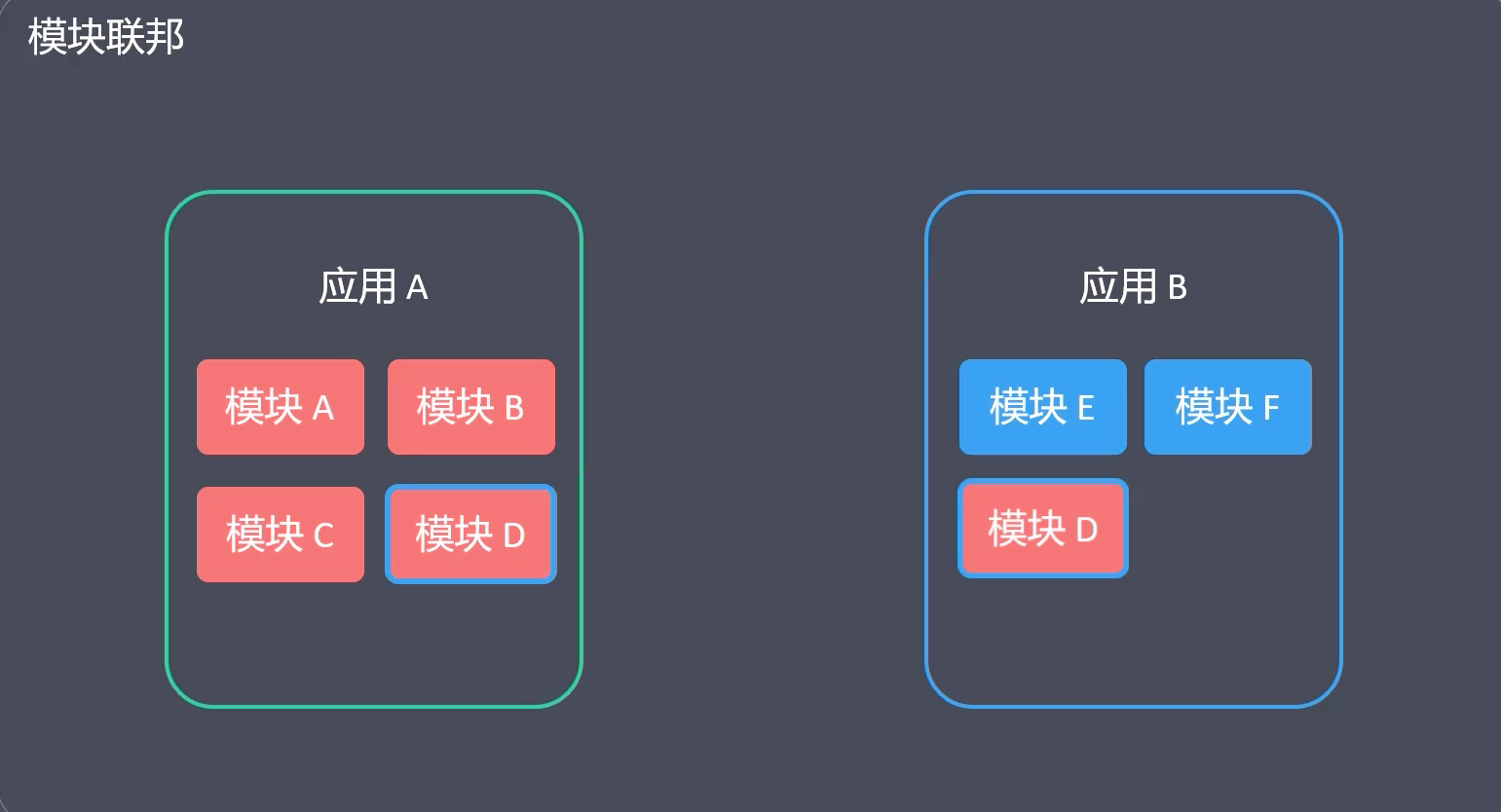
**Hash:**入口文件更改，所有入口文件加载的资源的**hash**都会更新

**ccontentHash**：根据文件类型打包，相同类型的文件，**hash**相同

****

模块解析：****

模块微联邦****

****

详细使用参考**micro-app** 里面的**demo**