重点难点不是业务开发、性能要求并不是很高，不会做在线ps一样的应用

最重要的是前端工程化的问题。

**前端的价值**

1. 搭建前端工程
2. 网络优化

减少http请求

压缩静态资源文件

使用浏览器强缓存使浏览器的流量变更小、加载速度更快

1. API定制
2. nodejs层

# 2-1 vue-loader+webpack项目配置

新建文件夹，在VS code里打开，终端运行 npm init ，初始化成一个npm项目



一些选项可默认

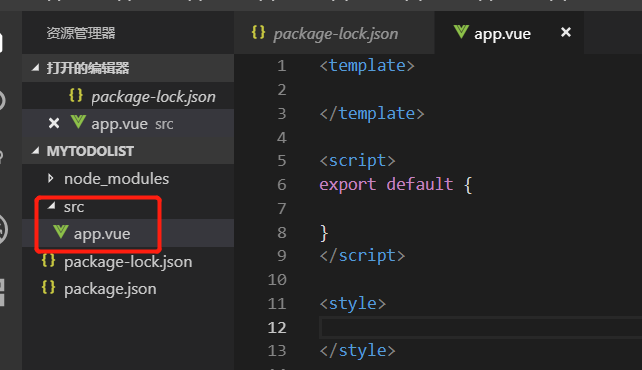
生成package.json文件，然后可安装所需要的包

配置vue 和wepack



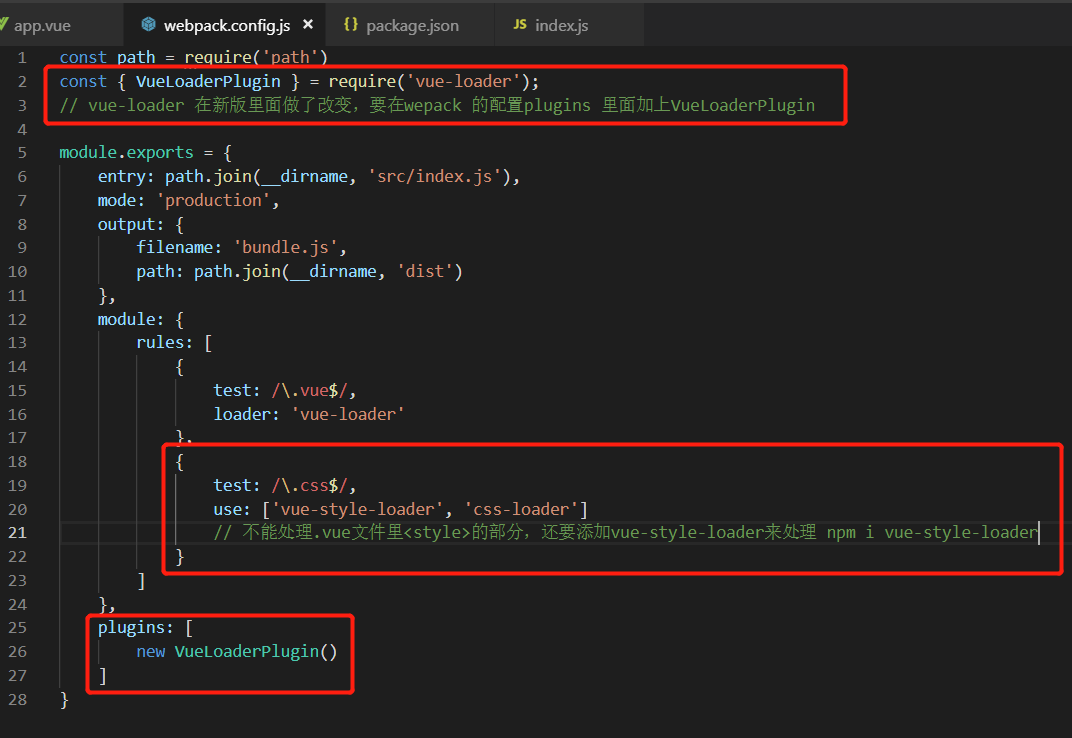
根据警告，继续配置css-loader和 vue-template-compiler

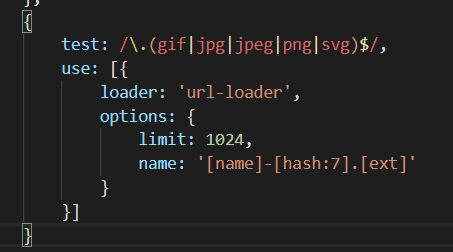


新建文件夹和app.vue

# 2-2webpack配置项目加载各种静态资源及css预处理器

配置webpack.config.js



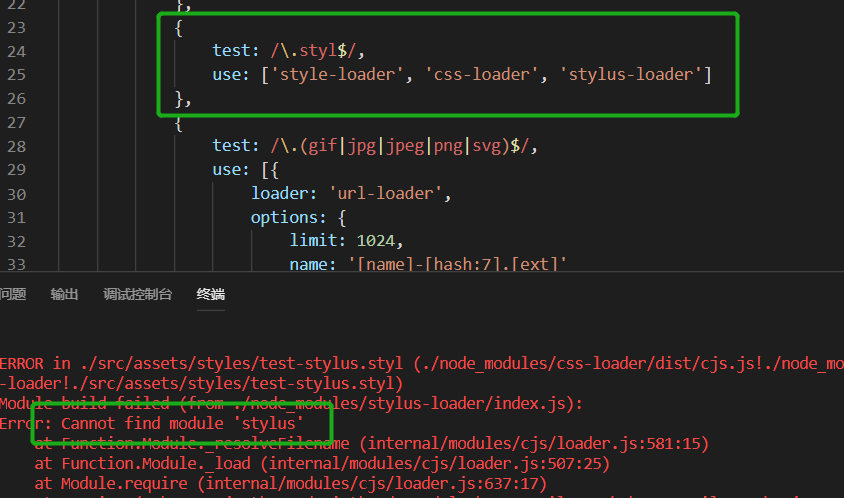


图片处理

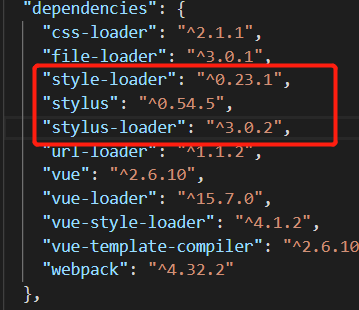
url-loader 依赖于 file-loader

所以要一起安装





这个报错，少装了个依赖

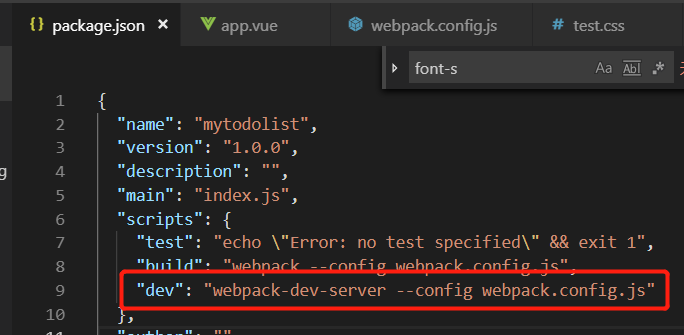


# 2-3 webpack-dev-server的配置和使用

首先安排webpack-dev-server的包

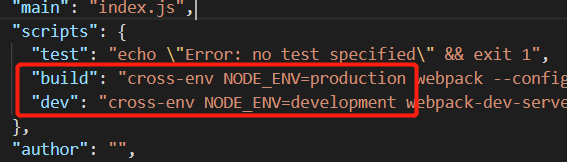


配置



应用于开发环境

安装包 cross-env



安装webpack插件，引入，将打包的文件自动引入html中



然后在webpack.config.js里配置

const HtmlPlugin = require('html-webpack-plugin')

plugins: [

new webpack.DefinePlugin({

'procss.env': {

NODE\_ENV: isDev ? '"devolopment"' : '"production"'

}

}),

new VueLoaderPlugin(),

new HtmlPlugin()

]

'"devolopment"' : '"production"'

里面一定要有双引号

if (isDev) {

config.devtool = '#cheap-module-eval-source-map'

config.devServer = {

port: 8000,

host: '0.0.0.0',

overlay: {

errors: true

},

open: true, // 运行npm run dev 自动打开浏览器

hot: true // 改组件代码，页面只重新渲染组件，不加载整个页面

}

config.plugins.push(

new webpack.HotModuleReplacementPlugin(), // hot: true 需要的插件

new webpack.NoEmitOnErrorsPlugin()

)

}

# 3-1 vue2的核心知识介绍

数据绑定

VUE文件开发方式

render方法

**API重点**

生命周期方法

computed 计算属性

# 3-2 配置VUE的jsx写法一级postcss

安装

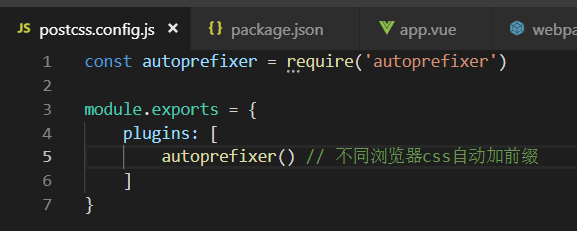


创建文件





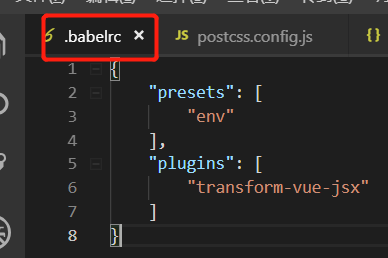
配置



安装



配置



补充安装

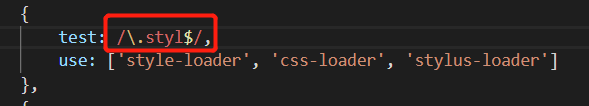


webpack.config.js增加配置



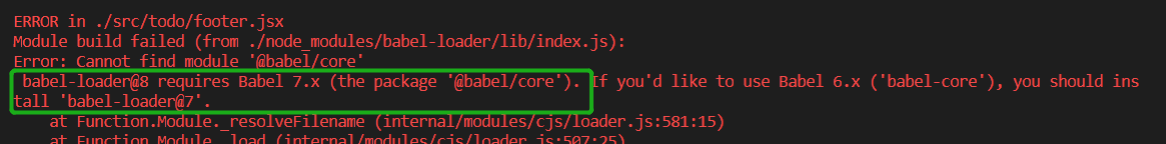
# 3-3 实现头都应用的界面

stylus不可以在.vue文件内使用时，修改webpack.config,js文件



不要$

错误



官方默认babel-loader | babel 对应的版本需要一致: 即babel-loader需要搭配最新版本babel

两种解决方案:

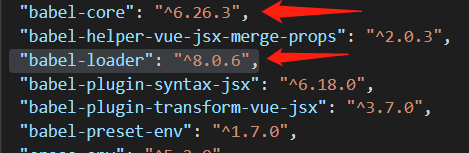
1、回退低版本

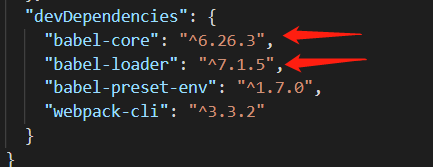
npm install -D babel-loader@7 babel-core babel-preset-env

2、更新到最高版本:

npm install -D babel-loader @babel/core @babel/preset-env webpack

参考： <https://www.cnblogs.com/soyxiaobi/p/9554565.html>





# 3-4 实现通道应用的业务逻辑

this.todos.unshift({...})

unshift() 将数据插入到数组的头部

value.trim()

trim()方法 去掉字符串两端的空格

deleteTodo(id) {

this.todos.splice(this.todos.findIndex(todo => todo.id === id), 1)

}

删掉数组中对应ID的那条数据

unFinishedTodoLength() {

return this.todos.filter(todo => !todo.comlpleted).length

}

返回数组中completed为false的个数

states: ['all', 'active', 'completed']

有三个状态

filter: 'all'

Filter值在这三个状态来回切换

Filter为all时，显示所有数组，filter为active时，显示数组中completed值为false的那几条，filter为completed时，显示数组中completed值为true的那几条

filteredTodos() {

if (this.filter === 'all') {

return this.todos

}

const completed = this.filter ==='completed'

return this.todos.filter(todo => completed === todo.completed)

}

删掉数组completed的值为true的那几条

clearAllCompleted() {

this.todos = this.todos.filter(todo => !todo.completed)

}

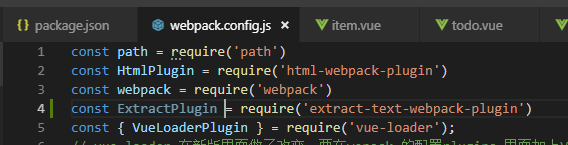
# 4-1 webpack配置css单独分离打包

将css单独分离打包

先安装插件



然后在config.js里引入



在开发环境中

// 开发环境

config.module.rules.push({

test: /\.styl/, // vue 文件里使用stylus写css时，文件名后面别加$

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

})

在正式环境中加入

// 正式环境 生产环境

config.output.filename = '[name].[chunkhash:8].js' // 在开发环境中绝对不要用chunkhash

config.module.rules.push({

test: /\.styl/, // vue 文件里使用stylus写css时，文件名后面别加$

use: ExtractPlugin.extract({

fallback: 'style-loader',

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

})

})

config.plugins.push(

new ExtractPlugin('styles.[hash:8].css')

)

# 4-2 webpack区分打包类库代码及hash优化

首先修改

// 正式环境 生产环境

config.entry = {

app: path.join(\_\_dirname, 'src/index.js'),

vendor: ['vue']

}

然后

config.plugins.push(

new ExtractPlugin('styles.[hash:8].css'),

new webpack.optimize.CommonsChunkPlugin({

name: 'vendor'

})

)

类库文件就会单独打包出来

webpack4更改了，不能向上面那么写了。。。

*//optimization与entry/plugins同级*

config.optimization = {

splitChunks: {

cacheGroups: {

commons: {

name: "vendor",

chunks: "initial",

minChunks: 2

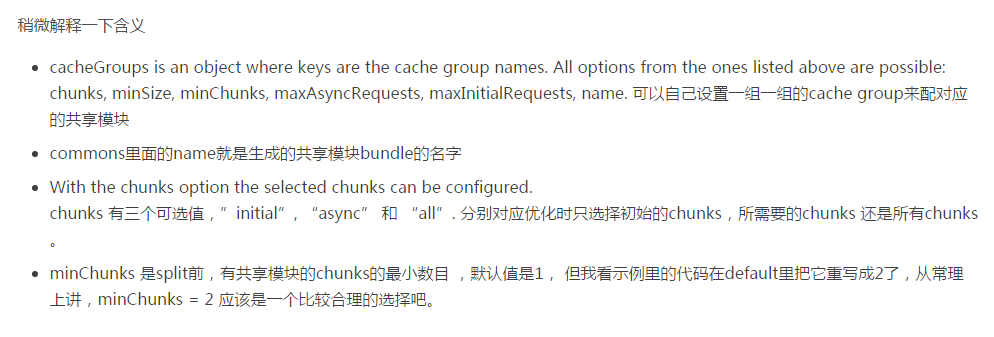
}

}

}

}

稍微解释一下含义



**cacheGroups** is an object where keys are the cache group names. All options from the ones listed above are possible: chunks, minSize, minChunks, maxAsyncRequests, maxInitialRequests, name. 可以自己设置一组一组的cache group来配对应的共享模块

**commons**里面的**name**就是生成的共享模块bundle的名字

With the **chunks** option the selected chunks can be configured.

chunks 有三个可选值，”initial”, “async” 和 “all”. 分别对应优化时只选择初始的chunks，所需要的chunks 还是所有chunks 。

**minChunks** 是split前，有共享模块的chunks的最小数目 ，默认值是1， 但我看示例里的代码在default里把它重写成2了，从常理上讲，minChunks = 2 应该是一个比较合理的选择吧。



# **[webpack中hash与chunkhash区别和需要注意的问题](https://www.cnblogs.com/heyushuo/p/8543889.html)**

**项目发布时，为了解决缓存，需要进行md5签名，这时候就需要用到**hash**和**chunkhash**等。**

**问题一：hash问题**

* 使用 hash 对js和css进行签名时，每一次hash值都不一样，导致无法利用缓存
* 原因是因为, hash 字段是根据每次编译compilation的内容计算所得，也可以理解为项目总体文件的hash值，而不是针对每个具体文件的。(所以每一次编译都会有一个新的hash，并不适用)
* 解决:不用hash，而用 chunkhash (js和css要使用chunkhash)， chunkhash 的话每一个js的模块对应的值是不同的(根据js里的不同内容进行生成)

**问题二：图片和字体图标的chunkhash问题**

* 前面有提到，hash在js和css中不实用，所以在项目中所有的文件都准备用 chunkhash ，但是又有了新的问题-img和font等资源中，使用 chunkhash 会报错
* 解决:因为 chunkhash 只适用于js和css，img中是没有这种东西的，仍然需要用到 hash (这个hash有点区别，每一个资源本身有自己的hash)

**问题三：chunkhash重复问题**

* 打包时发现，js和js引入的css的 chunkhash 是相同的，导致无法区分css和js的更新，如下

　　index2-ddcf83c3b574d7c94a42.css

　　index2-ddcf83c3b574d7c94a42.js

* 原因是因为webpack的编译理念，webpack将css视为js的一部分，所以在计算chunkhash时，会把所有的js代码和css代码混合在一起计算 \*解决:css是使用 ExtractTextPlugin 插件引入的，这时候可以使用到这个插件提供的 contenthash ，如下(使用后css就有独立于js外的指纹了)，

//提取css文件new ExtractTextPlugin({

filename:'css/[name].[chunkhash:8].css' //提取chunkhash8位码

})

* 需要注意的是，在新版本中，我在webpack3中测试的是，修改css的内容并不会引起js中的 chunkhash 变动(原因估计是webpack内置的算法变为了只计算js chunk)，所以css请务必使用 contenthash ，否则修改后无法生成新的签名，而是会覆盖以前的资源

# **webpack：hash、chunkhash、contenthash三者区别**

#### 一、hash（所有文件哈希值相同，只要改变内容跟之前的不一致，所有哈希值都改变，没有做到缓存意义）

hash是跟整个项目的构建相关，构建生成的文件hash值都是一样的，所以hash计算是跟整个项目的构建相关，同一次构建过程中生成的hash都是一样的，只要项目里有文件更改，整个项目构建的hash值都会更改。

如果出口是hash，那么一旦针对项目中任何一个文件的修改，都会构建整个项目，重新获取hash值，缓存的目的将失效。

#### 二、chunkhash（同一个模块，就算将js和css分离，其哈希值也是相同的，修改一处，js和css哈希值都会变，同hash，没有做到缓存意义）

它根据不同的入口文件(Entry)进行依赖文件解析、构建对应的chunk，生成对应的hash值。我们在生产环境里把一些公共库和程序入口文件区分开，单独打包构建，接着我们采用chunkhash的方式生成hash值，那么只要我们不改动公共库的代码，就可以保证其hash值不会受影响。

由于采用chunkhash，所以项目主入口文件main.js及其对应的依赖文件main.css由于被打包在同一个模块，所以共用相同的chunkhash。  
这样就会有个问题，只要对应css或则js改变，与其关联的文件hash值也会改变，但其内容并没有改变，所以没有达到缓存意义。

#### 三、contenthash（只要文件内容不一样，产生的哈希值就不一样）

contenthash表示由文件内容产生的hash值，内容不同产生的contenthash值也不一样。在项目中，通常做法是把项目中css都抽离出对应的css文件来加以引用。

**所以css文件最好使用contenthash。**  
  
作者：codingQi  
链接：https://www.jianshu.com/p/e687aa625f71  
来源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。