[vue2+3+webpack]

[2021.12.22]

国内较火的前端工程化框架之一

摘要

[前端框架]

[日期及邮编地址]

[ID名称]

[前端工程化 4](#_Toc23633)

[webpack 4](#_Toc15138)

[初始化项目 4](#_Toc25090)

[在项目中安装webpack 4](#_Toc31514)

[dependencies和devDependencies节点 4](#_Toc16280)

[在项目中配置webpack 4](#_Toc22531)

[webpack中的默认约定（即webpack.config.js文件中的初始默认配置） 5](#_Toc23938)

[自定义打包的入口与出口 5](#_Toc8053)

[webpack插件 5](#_Toc26171)

[webpack-dev-server 5](#_Toc8251)

[安装webpack-dev-server 5](#_Toc27315)

[配置webpack-dev-server 5](#_Toc30607)

[webpack-dev-server生成的文件位置 5](#_Toc19235)

[devServer中常用选项 5](#_Toc25888)

[html-webpack-plugin 6](#_Toc3946)

[安装html-webpakc-plugin 6](#_Toc17533)

[配置html-webpakc-plugin 6](#_Toc29536)

[打包处理css文件 7](#_Toc32404)

[处理流程 7](#_Toc20169)

[打包处理样式表中与url路径相关的文件 8](#_Toc7271)

[base64与普通image的对比 8](#_Toc32201)

[打包处理js高级语法 8](#_Toc11298)

[配置webpack的打包发布 9](#_Toc22023)

[优化各类文件存放路径 9](#_Toc29756)

[js文件归类设置 9](#_Toc21987)

[img文件归类设置 9](#_Toc3146)

[配置webpack每次发布自动先删除旧dist 9](#_Toc26223)

[Source Map 10](#_Toc29092)

[开发环境下 10](#_Toc3589)

[生产环境下 10](#_Toc7704)

[修改webpack.config.js配置文件后需要重启webpack 10](#_Toc8364)

[VUE 10](#_Toc6454)

[vue简介 11](#_Toc9401)

[vue特性 11](#_Toc252)

[数据驱动视图 11](#_Toc5443)

[双向数据绑定 11](#_Toc11500)

[vue原理简介 11](#_Toc14692)

[MVVM 11](#_Toc26325)

[MVVM工作原理 12](#_Toc15969)

[vue基本使用 12](#_Toc20179)

[指令 14](#_Toc29738)

[内容渲染指令v-text、{{ }}、v-html 14](#_Toc27727)

[v-text 14](#_Toc18135)

[{{ }}插值表达式 14](#_Toc15367)

[v-html 14](#_Toc9484)

[属性绑定指令v-bind（:） 15](#_Toc19311)

[上述模板渲染语法还支持js语法 15](#_Toc16995)

[事件绑定指令v-on（@） 15](#_Toc13279)

[访问数据源data内的数据：this 15](#_Toc24147)

[绑定函数的传参 16](#_Toc7276)

[事件event变量：$event 16](#_Toc15153)

[事件修饰符 16](#_Toc24421)

[按键修饰符keyup，keydown 17](#_Toc3634)

[双向绑定指令v-model 17](#_Toc6337)

[v-model指令的修饰符 18](#_Toc14497)

[条件渲染指令v-if、v-show 18](#_Toc19721)

[列表渲染指令v-for 18](#_Toc23463)

[:key属性绑定 19](#_Toc5248)

[过滤器（vue3内移除） 19](#_Toc9035)

[声明过滤器函数 19](#_Toc21654)

[js字符串string相关方法 20](#_Toc19106)

[私有过滤器和全局过滤器 20](#_Toc8681)

[dayjs 20](#_Toc10817)

[过滤器可以连续调用多个 20](#_Toc30564)

[过滤器可以接收参数 21](#_Toc27497)

[侦听器 21](#_Toc4923)

[watch侦听器 21](#_Toc3475)

[方法类型的声明语法： 21](#_Toc4304)

[方法类型侦听器的缺点： 22](#_Toc6459)

[侦听对象属性的声明语法： 22](#_Toc20170)

[对象类型的声明语法： 22](#_Toc29209)

[immediate选项 22](#_Toc27369)

[deep选项 22](#_Toc3123)

[计算属性 22](#_Toc10578)

前端工程化

webpack

webpack是前端工程化的具体解决方案

他提供了友好的前端模块化开发支持，以及代码压缩混淆、处理浏览器端JavaScript的兼容性、性能优化等强大功能

需要安装nodejs才能使用

初始化项目



注：-S表示将安装的包名和版本号记录在package.json文件的dependencies节点下，不写的 话也会记录。-S是--save的简写

install可以简写为i

在项目中安装webpack

npm install webpack@5.42.1 webpack-cli@4.7.2 -D

注：-D表示将安装的包名和版本号记录到devDependencies节点下

-D是--save-dev的简写

@用来指定版本号

dependencies和devDependencies节点

dependencies：开发和生产环境用到的包记录在该目录下

devDependencies：只有开发阶段会被用到的包记录在该目录下

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

module.exports = {

mode: ’development’, //mode用来指定构建模式，有development和production

}

该配置控制webpack做不做代码压缩，development不压缩，production压缩

1. 在package.json的scripts节点下，新增dev脚本

“scripts”: {

“dev”: “webpack”, //scripts节点下的脚本可以通过npm run执行，如npm run dev

}

1. 在终端中运行npm run dev命令，启动webpack进行项目的打包构建

打包完成后会生成dist文件夹，里面有生成的main.js文件，该文件就是webpack打包的综合性js文件，包含用户自己写的代码和用到的工具包代码，可以解决低版本浏览器不兼容高级代码的问题

webpack中的默认约定（即webpack.config.js文件中的初始默认配置）

在webpack4.x 和5.x中

1. 默认打包的入口文件为src/index.js
2. 默认的输出文件路径为dist/main.js

这两种配置可以在webpack.config.js内修改

自定义打包的入口与出口

通过配置webpack.config.js文件中的entry节点指定打包的入口，通过output节点指定打包的出口，示例代码如下

const path = require(‘path’); //导入nodejs中专门操作路径的模块

module.exports = {

entry: path.join(\_\_dirname, ‘./src/index.js’);

output: {

path: path.join(\_\_dirname, ‘./dist’), //输出文件的存放路径

filename: ‘main.js’, //输出的文件名

}

webpack插件

通过安装和排至第三方的插件，可以拓展webpack的能力，从而让webpack用起来更方便。最常用的webpack插件有两个：

webpack-dev-server

类似于nodejs中的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

安装webpack-dev-server

npm install webpack-dev-server@3.11.2 -D

配置webpack-dev-server

1. 修改package.json文件中scripts的dev命令如下

“scripts” : {

“dev”: “webpack serve”,

}

1. 再次运行npm run dev重新进行项目的打包
2. 浏览器中访问localhost:8080可以查看打包效果

webpack-dev-server生成的文件位置

webpack-dev-server生成的文件不会直接保存在硬盘上（照顾硬盘的读写寿命），所以在资源管理器上看不到它，生成的文件被保存在内存的根目录下，即/，引用路径直接写/xx.js即可

devServer中常用选项

devServer: {

open: true, //首次打包成功后自动打开浏览器

port: 80, //指定端口号

host: ‘127.0.0.1’, //指定主机地址

}

html-webpack-plugin

webpack中的html插件（类似于一个模板引擎插件），配合webpack-dev-server使用

1.可以通过此插件自定制index.html页面的内容，本质是把src内的index.html复制到根目录下（内存中），如此一来访问localhost:8080时会直接加载index.html

2.会自动添加一条script标签，把web-dev-server生成的js文件引入到index.html中

安装html-webpakc-plugin

npm install html-webpakc-plugin@5.3.2 -D

配置html-webpakc-plugin

在webpack.config.js文件内配置

//导入插件，得到一个构造函数

const HtmlPlugin = require(‘html-webpack-plugin’)

//创建插件实例对象

const htmlPlugin = new HtmlPlugin({

template: ‘./src/index.html’, //指定原文件的存放路径

filename: ‘./index.html’, //指定生成的文件的存放路径

})

module.exports = {

mode: ‘development’,

plugins: [htmlPlugin], //通过plugins节点，使htmlPlugin插件生效

}

webpack中的loader

loader概述

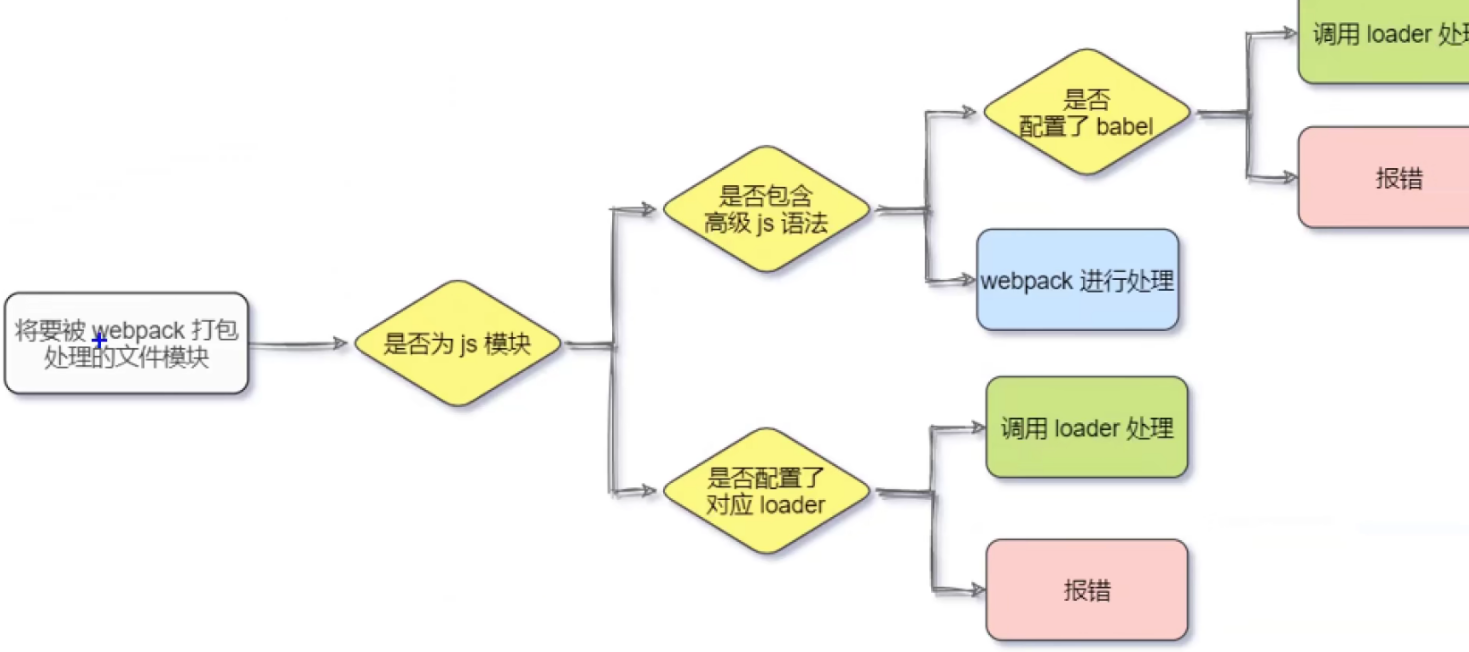
webpack默认只能打包处理.js后缀名的文件，其他非.js后缀名的模块默认处理不了，需要调用loader加载器才能正常打包，否则会报错

loader作用：协助webpack打包处理特定的文件模块，如：

css-loader可以打包处理.css相关文件

less-loader可以打包处理.less相关文件

babel-loader可以打包处理webpack无法处理的高级JS语法



打包处理css文件

1. 运行*npm i style-loader@3.0.0 css-loader@5.2.6 -D* 安装处理css文件的loader
2. 在webpack.config.js的module->rules数组中，添加loader规则如下

module: {

rules: [

{test: /\.css$/, use: [‘style-loader’, ‘css-loader’]},

]

}

其中test表示匹配的文件类型，use表示对应要调用的loader

• use数组中指定指定的loader顺序是固定的

• 多个loader的调用顺序是从后往前调用

处理流程

webpack在发现了处理不了的模块时会查找webpack.config.,js的module内的rules数组， 是否配置了对应的loader加载器

若配置了，webpack把对应文件先交给数组最后一个loader进行处理，完毕后交给rules 数组内的下一个loader进行处理（倒序）

全都处理完毕后把结果转交给webpack，webpack把处理结果合并到/dist/main.js文件内， 最终生成打包好的文件。

打包处理less文件

1. 运行*npm i less-loader@10.0.1 less@4.1.1 -D*
2. 在webpack.config.js的rules数组中添加loader规则

module: {

rules: [

{test: /\.less$/, use: [‘style-loader’, ‘css-loader’, ‘less-loader’]},

]

}

less-loader依赖less运行，但less不写在rules数组中

打包处理样式表中与url路径相关的文件

1. 运行npm i [url-loader@4.1.1](mailto:url-loader@4.1.1) [file-loader@6.2.0](mailto:file-loader@6.2.0) -D
2. 在webpack.config.js中的rules数组配置规则

module: {

rules: [

{test: /\.jpg|png|gif$/, use: ‘url-loader?limit-22229’},

]

}

其中?之后的是loader的参数

limit用来指定图片的大小，单位是字节（byte）

只有小于等于limit大小的图片才会被转换成base64

base64与普通image的对比

base64将图片集成在html页面内，img标签src引用需要浏览器发送请求下载图片再加载

base64优点：减少图片网络请求

base64缺点：转换后的图片体积会变大一点

网络性能上base64具有优势，但大图片转换后编码很长占用内存较大，所以只推荐小规模，小体积的图片使用

打包处理js高级语法

webpack只能处理一部分js高级语法，对于webpack处理不了的js高级语法需要借助babel-loader进行打包处理

1. 运行npm i babel-loader@8.2.2 @babel/core@7.14.6 @babel/plugin-proposal-decorators@7.14.5 -D （后两个是依赖包,最后一个包针对装饰器语法）
2. 在webpack.config.js的rules数组中添加如下规则

module: {

rules: [

{test: /\.js$/, use: ‘babel-loader’, exclude: /node\_module/},

]

}

必须使用exclude排除指定项，因为node\_modules目录下的第三方包不需要被打包。因为.js会匹配到第三方js文件，所以要排除它们，只打包自己写的js文件即可

1. 配置babel-loader

在项目根目录下创建名为babel.config.js的配置文件，定义配置项如下

module.exports = {

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

}

配置webpack的打包发布

在package.json文件的scripts节点下，新增build命令

“scripts”: {

“dev”: “webpack serve”,

“build”: “webpack --mode produciton”,

}

--mode是一个参数，用来指定webpack的运行模式。porduction代表生产环境，进行代码压缩和性能优化

注：--mode指定的参数项会覆盖webpack.config.js中的mode选项，相当于把原来设置的development改成production

优化各类文件存放路径

将不同种类的文件分别装到对应的文件夹中，比如js文件夹里放.js文件，img文件夹放图片文件等

js文件归类设置

在webpack.config.js中

const path = require(‘path’); //导入nodejs中专门操作路径的模块

module.exports = {

entry: path.join(\_\_dirname, ‘./src/index.js’);

output: {

path: path.join(\_\_dirname, ‘./dist’), //输出文件的存放路径

filename: ‘js/main.js’, //输出的文件名

}

在filename的值内添加js/即可

img文件归类设置

在webpack.config.js中

module: {

rules: [

{test: /\.jpg|png|gif$/, use: ‘url-loader?limit-22229&Path=img’},

]

}

在传递的参数里加上Path=xxx即可设置分类文件夹

配置webpack每次发布自动先删除旧dist

每次生成新的dist是不会覆盖旧dist的，相反会合并新旧dist容易造成误解。

clean-webpack-plugin

1. 安装

npm i --save-dev clean-webpack-plugin

1. 在webpack.config.js中配置

const { CleanWebpackPlugin } = require(‘clean-webpack-plugin’);

module.exports = {

mode: ‘development’,

plugins: [htmlPlugin, new CleanWebpackPlugin()],

}

在plugins节点内new一个CleanWebpackPlugin对象

Source Map

source map是一个文件，里面存储位置信息。source map存储着压缩混淆后的代码所对应的转换前的位置，出错的时候，除错工具将直接显示原始代码，而不是转换后的代码

开发环境下

推荐在webpack.config.js中添加如下配置，即可启用source map

module.exports = {

mode: ‘development’,

devtool: ‘eval-source-map’,

}

建议只在开发模式下使用

生产环境下

若代码在浏览器端报错，直接点击后面的链接会直接跳到源码，容易暴露源代码，建议如下：

1. 关闭source map，注释掉devtool就行了
2. 设置Source Map只定位行数，不暴露源码（推荐）

module.exports = {

mode: ‘development’,

devtool: onsource-source-map’,

}

更改devtool内的值即可，

webpack中的@路径查找方式

@代表一种路径查找方式，从外往里找，路径可以用@简写

在webpack.config.js中配置

resolve: {

alias: {

‘@’: path.join(\_\_dirname, ‘./src/’) //告诉webpack，@代表src这一层目录

}

}

平常的路径写法: ’../../../js/main.js’ 从里往外

@路径的写法: ‘@/main.js’ 从外往里

修改webpack.config.js配置文件后需要重启webpack

npm run dev

VUE

vue简介

一个用来构建用户界面的框架

构建用户界面：

用vue向html内填充数据非常简单

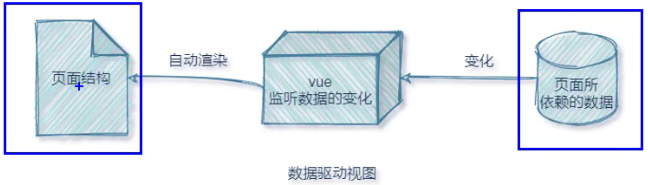
框架：

框架是一套现成的解决方案，包含vue的指令、组件（UI结构的复用）、路由、vuex、vue组件库

vue特性

数据驱动视图

vue会监听数据变化，从而自动重新渲染页面结构

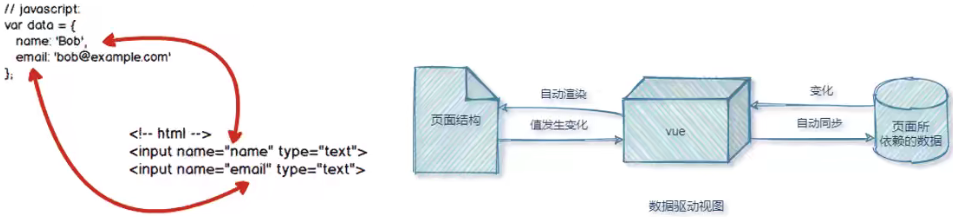


好处：当页面数据变化时，会自动重新渲染

注：数据驱动视图是单向数据绑定

双向数据绑定

在填写表单时，双向绑定数据可以辅助开发者在不操作DOM的前提下，自动把用户填写的内容同步到数据源中

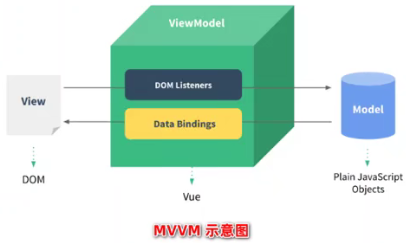


好处：开发者不再需要手动操作DOM来获取最新的值

vue原理简介

MVVM

mvvm是vue实现数据驱动视图和双向数据绑定的核心原理。MVVM指的是Model、View和ViewModel，它把每个html页面都拆成了3部分。



model表示当前页面渲染时所依赖的数据源

view表示当前页面所渲染的DOM结构

viewmodel表示vue的实例，它是mvvm的核心

MVVM工作原理

viewmodel作为mvvm的核心，把当前页面的数据源（model）和页面（view）连接在了一起



当数据源发生变化时，会被viewmodel监听到，vm会根据最新的数据源自动更新页面的结构

当表单元素的值发生变化时，也会被vm监听到，vm会把最新的值自动同步到model数据源中

vue基本使用

1. 导入vue.js的script脚本
2. 在页面中声明一个将要被vue所控制的DOM区域
3. 创建vm实例对象（vue实例对象）





指令

指令（directives）是vue为开发者提供的模板语法，用于辅助开发者渲染页面的基本结构，指令是vue开发中最基础、最常用、最简单的知识点

vue中的指令按照不同的用途可以分为6大类

内容渲染指令

属性绑定指令

事件绑定指令

双向绑定指令

条件渲染指令

列表渲染指令

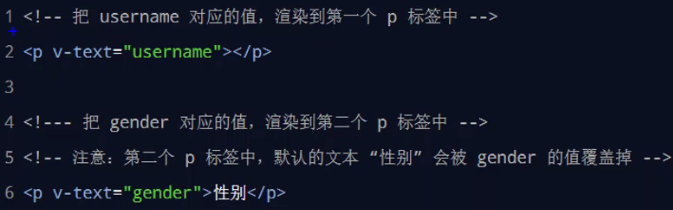
内容渲染指令v-text、{{ }}、v-html

辅助开发者渲染DOM元素的文本内容，常用有3个

v-text、{{ }}插值表达式、v-html

属于单向数据绑定

v-text



特性：

1. 会覆盖元素内原有的内容
2. 只能渲染纯文本内容

{{ }}插值表达式

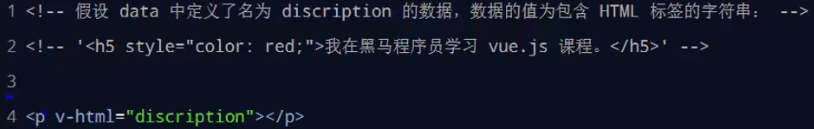
专门用来解决v-text指令的覆盖问题，只能渲染纯文本内容

特性：

只能用在内容节点，不能用在属性节点，如<input placeholder=”{{ xxx }}”>不行

v-html

可以渲染包含html标签的字符串



属性绑定指令v-bind（:）

给元素的属性动态绑定，需要v-bind属性绑定指令，v-bind: 属性=”数据键名”

<input v-bind: placeholder=”datakey”>

v-bind:可以简写为一个冒号:

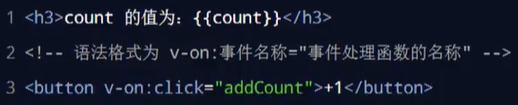
<input : placeholder=”datakey”>

上述模板渲染语法还支持js语法



事件绑定指令v-on（@）

给标签绑定事件用v-on:事件名，事件名和jquery里的一样



v-on可以简写为@，@click

<button @click=”addCount(x)”></button>

然后在vue实例对象中添加methods对象，并在其中定义处理函数

const vm = new Vue({

... //前置代码在此省略

methods: {

addCount: function () {

代码块

},

},

})

methods内的函数定义可以简写为addCount() {....}（去掉了冒号和function关键字）

访问数据源data内的数据：this

const vm = new Vue({

... //前置代码在此省略

data: {

count: 0,

},

methods: {

addCount: function () {

代码块

},

},

})

访问data里的count

1. 使用vm.count即可
2. 使用this.count也可（推荐）

绑定函数的传参

<button v-on:click=”addCount(x)”></button>

const vm = new Vue({

... //前置代码在此省略

methods: {

addCount: function (n) {

代码块

},

},

})

事件event变量：$event

若绑定的函数传参了，并且还要用到事件对象e，用vue固定的写法$event

<button @click=”addCount(x, $event)”></button>

const vm = new Vue({

... //前置代码在此省略

methods: {

addCount: function (n, e) {

代码块

},

},

})

事件修饰符

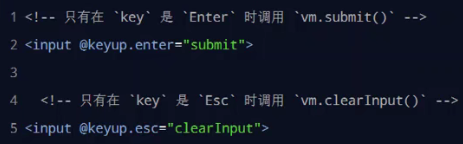
在事件处理函数中调用event.preventDefault()或event.stopPropagation()是非常常见的需求。因此vue提供了事件修饰符的概念，来辅助程序员更方便的对事件的触发进行控制



<button @click.prevent=”addCount(x)”></button>

按键修饰符keyup，keydown

在监听键盘事件时，我们经常需要判断详细的按键，此时可以添加按键修饰符



双向绑定指令v-model

v-model双向数据绑定指令，用来辅助开发者在不操作DOM的前提下，快速获取表单数据，v-model=”数据键名”



只有表单元素可以使用v-model

input、textarea、select

v-model指令的修饰符

为了方便对用户输入的内容进行处理，vue为v-model指令提供了3个修饰符

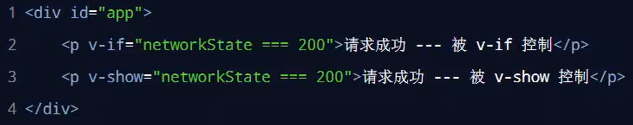


条件渲染指令v-if、v-show

用来辅助开发者按需控制DOM的显示与隐藏，v-if、v-show

v-if=”条件判断语句” v-else-if v-else

v-show=”条件判断语句”



v-show是通过display的方式控制元素的显示隐藏

v-if是通过创建删除来控制元素的显示隐藏

列表渲染指令v-for

v-for用来辅助开发者基于一个数组来循环渲染一个列表结构。

v-for=”x in list” //类似python中的for循环



v-for还有一个可选的参数，即当前项的索引。语法为v-for=”(item, index) in list”



并且在v-for所在标签也能访问到list内数据，如：

<li v-for=”(x, index) in list” :title=”x.name”>{{ index }}</li>

:key属性绑定

官方建议，只要用到了v-for，一定要绑定:key属性，尽量把id作为key的值（唯一性），为了配合后面.vue格式的文件

key属性的值类型有要求：数字或字符串。

设置了key既能提升性能，又能防止列表状态紊乱。

key的值最好是数据库内的主键，具有唯一性，index不具备唯一性。

过滤器（vue3内移除）

过滤器filter是vue为开发者提供的功能，常用于文本的格式化。过滤器可以用在两个地方：插值表达式和v-bind属性绑定。在js代码尾部通过管道符”|”进行调用，使用前必须要声明



声明过滤器函数

过滤器本质是一个函数，必须被定义到filters节点下

const vm = new Vue({

... //前置代码在此省略

data: {

count: 0,

},

fiters: {

//过滤器函数中的val是管道符前面的原值

xxx(val) {

....

//过滤器函数必须要返回值，否则会抹除数据

return 处理后的数据

}

},

})

js字符串string相关方法

str.charAt(index) //返回对应索引值的字符，从0开始

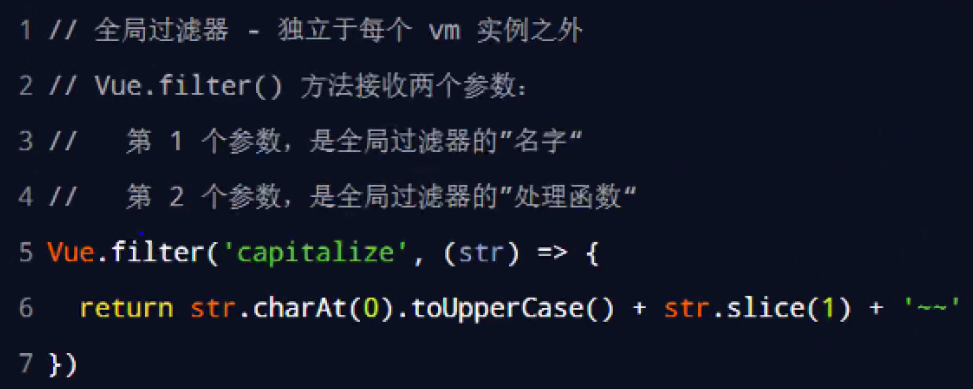
str.slice(index) //返回对应索引开始到末尾的字符串片段，从0开始

私有过滤器和全局过滤器

定义到vue实例中的过滤器是私有的，别的实例中调用不到

如果全局过滤器和私有过滤器名字冲突，按照就近原则调用，即优先调用私有过滤器

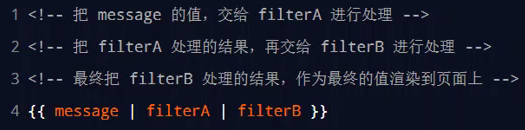
全局过滤器定义：



dayjs

处理时间的js轻量库

过滤器可以连续调用多个



过滤器可以接收参数

过滤器本质是一个js函数，可以接收参数



侦听器

监视数据变化并做出反应

watch侦听器

watch侦听器允许开发者监视数据的变化，从而针对数据的变化做特定的操作，想侦听谁就以谁为函数名，有两种声明方法：方法类型和对象类型。

方法类型的声明语法：



侦听器本质是个函数，要监听哪个数据的变化就把数据名作为函数名

方法类型侦听器的缺点：

1. 不可以在页面初始化后直接调用，对象类型声明可解决
2. 侦听的是一个对象的话，对象属性的改变不会触发侦听器，需要具体指明

侦听对象属性的声明语法：

watch: {

‘obj.xxx’() {...} //方法类型声明下，侦听xxx属性需具体指明

},

对象类型的声明语法：

watch: {

xxx: {

handler(newval, oldval) {...} //对象侦听器的处理函数

immediate: true //页面加载后直接出发一次，默认值为false

deep: true //开启深度侦听，对象中的任何变化都会触发侦听器

},

},

immediate选项

可以控制侦听器在页面加载完毕后是否自动触发一次

immediate: true //页面加载后直接出发一次，默认false

deep选项

深度侦听选项，控制侦听器是否侦听对象内部变化

deep: true //开启深度侦听，对象中的任何变化都会触发侦听器，默认false

计算属性

指的是通过一系列运算，最终得到一个属性值，这个动态计算出的属性值可以被模板结构（插值表达式、v-bind）或methods方法使用。

1. 所有的计算属性都要定义到computed节点下
2. 计算属性要定义成方法格式
3. 使用的时候当vue实例的属性使用：{{ xxx }}

代码如下：

