Vue+iView

# **1、vue 的watch**

用于监听对象的属性时不建议使用deep：true, 因为会执行两次， 建议在conputed属性里进行转换，在到watch里监听返回的某个值即可，或者直接监听对象的某个属性

# 2、// 去掉有边框

/deep/ .ivu-table:after

width 0

// 去掉下边框

/deep/ .ivu-table:before

height 0

// 去掉上、左边框

.ivu-table-wrapper

border-top none

border-left none

// 去掉最后一个tr的下边框

/deep/ .ivu-table-row

&:nth-last-of-type(1)

td

&:nth-of-type(1)

border-bottom none

# 3. iView在使用模态框时会在body上面加overflow：hidden属性，

页面就无法滚动了，解决是Modal的将scrollable设为true。

# **4.v-if在使用时**，

有时状态值改变了，页面却没有变化，用this.$forceUpdate（）强制更新界面

# 5 vue-router的active-class：

router-link 默认情况下的路由是模糊匹配，例如当前路径是 /article/1 那么也会激活 <router-link to="/article">，所以当设置 exact-active-class ，或者在路由中配置linkExactActiveClass: 'active',以后，这个 router-link 只有在当前路由被全包含匹配时才会被激活 exact-active-class 中的class。

# 6 使用iview表单验证的时候，

对于自定义的验证，判断语句应当有if,else 需要有一个判断进行callback(),表示验证通过

# 7. v-model 本质上不过是语法糖。

它负责监听用户的输入事件以更新数据，并对一些极端场景进行一些特殊处理。，例如可以在父组件v-model某值，在子组件this.$emit('input'， value),父组件v-model类似于监听input,并绑定value

# 8.VUE热更新内存溢出的解决办法:

packagejson里添加这两个插件：

"dependencies": {

"increase-memory-limit": "^1.0.3",

"cross-env": "^5.0.5"

}

执行npm install 来安装组件

然后再添加如下：

"scripts": {

"fix-memory-limit": "cross-env LIMIT=3072 increase-memory-limit"

}

LIMIT是你想分配的内存大小，然后执行npm run fix-memory-limit

只需要执行一次即可，然后重新启动项目，热更新就不会再内存溢出了。

或

package.json文件：

"scripts": {

"dev": "node --max\_old\_space\_size=4096 node\_modules/webpack-dev-server/bin/webpack-dev-server.js --inline --progress --config build/webpack.dev.conf.js",

"start": "npm run dev",

"build": "node --max\_old\_space\_size=4096 build/build.js"

},

————————————————

版权声明：本文为CSDN博主「倾听雨落~」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/weixin\_43292447/article/details/102628425

# 9. .sync修饰符

在有些情况下，我们可能需要对一个 prop 进行“双向绑定”。不幸的是，真正的双向绑定会带来维护上的问题，因为子组件可以修改父组件，且在父组件和子组件都没有明显的改动来源。这也是为什么我们推荐以 update:myPropName 的模式触发事件取而代之。举个例子，在一个包含 title prop 的假设的组件中，我们可以用以下方法表达对其赋新值的意图：子组件：this.$emit('update:title', newTitle)，父组件：v-bind:title="doc.title"

v-on:update:title="doc.title = $event"，父组件简写： v-bind:title.sync="doc.title"

# 10 你可能有很多次想要在一个组件的根元素上直接监听一个原生事件。

这时，你可以使用 v-on 的 .native 修饰符： <base-input v-on:focus.native="onFocus"></base-input>

# 11 在动态组件上使用 keep-alive

我们之前曾经在一个多标签的界面中使用 is 特性来切换不同的组件： <component v-bind:is="currentTabComponent"></component> 当在这些组件之间切换的时候，你有时会想保持这些组件的状态，以避免反复重渲染导致的性能问题。 重新创建动态组件的行为通常是非常有用的，但是在这个案例中，我们更希望那些标签的组件实例能够被在它们第一次被创建的时候缓存下来。为了解决这个问题，我们可以用一个 <keep-alive> 元素将其动态组件包裹起来。

<!-- 失活的组件将会被缓存！-->

<keep-alive>

<component v-bind:is="currentTabComponent"></component>

</keep-alive>

# 12 插槽 slot分发

会替换slot内容， iview的某些组件中也应用了slot。

（1）有时为一个插槽设置具体的后备 (也就是默认的) 内容是很有用的，它只会在没有提供内容的时候被渲染。例如在一个 <submit-button> 组件中：

<button type="submit">

<slot></slot>

</button>

我们可能希望这个 <button> 内绝大多数情况下都渲染文本“Submit”。为了将“Submit”作为后备内容，我们可以将它放在 <slot> 标签内：

<button type="submit">

<slot>Submit</slot>

</button>

现在当我在一个父级组件中使用 <submit-button> 并且不提供任何插槽内容时：

<submit-button></submit-button>

后备内容“Submit”将会被渲染：

<button type="submit">

Submit

</button>

但是如果我们提供内容：

<submit-button>

Save

</submit-button>

则这个提供的内容将会被渲染从而取代后备内容：

<button type="submit">

Save

</button>

（2）有时我们需要多个插槽。对于这样的情况，<slot> 元素有一个特殊的特性：name。这个特性可以用来定义额外的插槽：

<div class="container">

<header>

<slot name="header"></slot>

</header>

<main>

<slot></slot>

</main>

<footer>

<slot name="footer"></slot>

</footer>

</div>

一个不带 name 的 <slot> 出口会带有隐含的名字“default”。

在向具名插槽提供内容的时候，我们可以在一个 <template> 元素上使用 v-slot 指令，并以 v-slot 的参数的形式提供其名称：

<base-layout>

<template v-slot:header>

<h1>Here might be a page title</h1>

</template>

<template v-slot:default>

<p>A paragraph for the main content.</p>

<p>And another one.</p>

</template>

<template v-slot:footer>

<p>Here's some contact info</p>

</template>

</base-layout>

（3）作用域插槽

有时让插槽内容能够访问子组件中才有的数据是很有用的。例如，设想一个带有如下模板的 <current-user> 组件：

span>

<slot>{{ user.lastName }}</slot>

</span>

我们想让它的后备内容显示用户的名，以取代正常情况下用户的姓，如下：

<current-user>

{{ user.firstName }}

</current-user>

然而上述代码不会正常工作，因为只有 <current-user> 组件可以访问到 user 而我们提供的内容是在父级渲染的。

为了让 user 在父级的插槽内容中可用，我们可以将 user 作为 <slot> 元素的一个特性绑定上去：

<span>

<slot v-bind:user="user">

{{ user.lastName }}

</slot>

</span>

绑定在 <slot> 元素上的特性被称为插槽 prop。现在在父级作用域中，我们可以给 v-slot 带一个值来定义我们提供的插槽 prop 的名字：

<current-user>

<template v-slot:default="slotProps">

{{ slotProps.user.firstName }}

</template>

</current-user>

# 13 在每个 new Vue 实例的子组件中，其根实例可以通过 $root 属性进行访问。

this.$root 访问父级组件实例 this.$parent 访问子组件实例或子元素 this.$refs.xx

# 14 通过 $on(eventName, eventHandler) 侦听一个事件

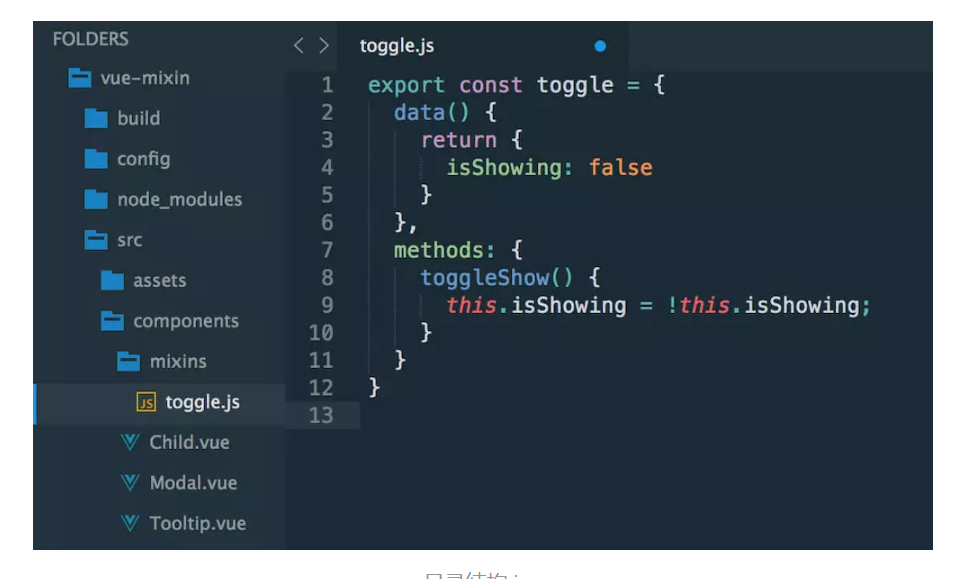
通过 $once(eventName, eventHandler) 一次性侦听一个事件

通过 $off(eventName, eventHandler) 停止侦听一个事件

# 15 vue - mixin

组件在引用之后相当于在父组件内开辟了一块单独的空间，来根据父组件props过来的值进行相应的操作，单本质上两者还是泾渭分明，相对独立。

而mixins则是在引入组件之后，则是将组件内部的内容如data等方法、method等属性与父组件相应内容进行合并。相当于在引入后，父组件的各种属性方法都被扩充了。



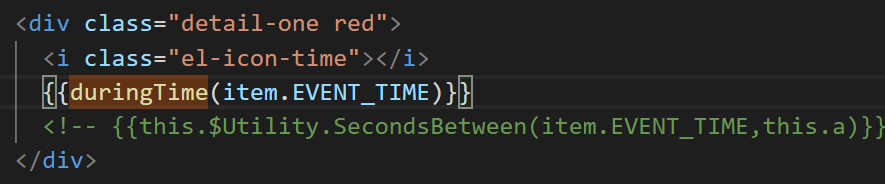


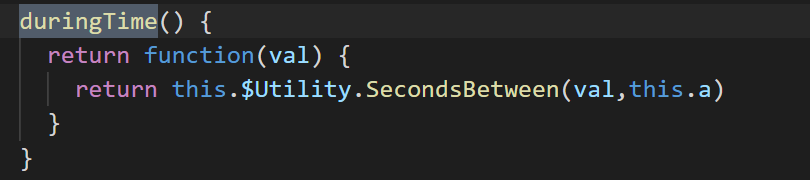
# 16. Iview 表格可编辑

<https://run.iviewui.com/E95OdGNQ>

**vue-easytable**

# 17. vue计算属性传参





# 18. vue计算属性缓存

# 19. 在每次调用方法前先解绑事件( bus.$off )，然后在重新绑定( bus.$on ),否则会多次触发方法

# 20. vue项目版本更新后文件存在缓存问题解决方案(vue-cli2.0/vue-cli3.0)

## 1 vue-cli2.0

**1.配置****webpack.prod.conf.js**

 配置output加t=[chunkhash:8]相当于加时间戳，用于清缓存，即改变build后生成的文件，时间戳不同文件不同，不会产生缓存

1. output: {
2. path: config.build.assetsRoot,
3. filename: utils.assetsPath('js/[name].[chunkhash].js?t=[chunkhash:8]'),
4. chunkFilename: utils.assetsPath('js/[id].[chunkhash].js?t=[chunkhash:8]')
5. }

实际上webpack.prod.conf.js 里面已经配置了，所以不用加，没有的话加上

const Version = new Date().getTime()

output: {

    path: config.build.assetsRoot,

    filename: utils.assetsPath('js/[name].[chunkhash].' + Version + '.js'),

    chunkFilename: utils.assetsPath('js/[id].[chunkhash].' + Version + '.js')

  },

**2.在static静态目录下新建version.json每次发版更改里面的版本号**

{

  "version": "V2020.10.29.14.29.02"

}

**3.在src中新建 libs/versionUpdate.js文件**

import Vue from 'vue';

import axios from 'axios'

const vm = new Vue();

const isNewVersion = () => {

  let url = ''

  if(process.env.NODE\_ENV === 'production') {

    url = `http://${ window.location.host }/cdp/static/version.json`

  }else{

    url = `http://${ window.location.host }/static/version.json`

  }

  axios.get(url).then(res => {

    let version = res.data.version;

    let localVersion = localStorage.getItem('version');

    if (localVersion && localVersion != version) {

      if (confirm("系统检测到有新版本，是否更新？")) {

        localStorage.setItem('version', res.data.version);

        window.location.reload();

        return;

      }

    }else {

      localStorage.setItem('version', res.data.version);

    }

  });

};

export default {

  isNewVersion

}

**4.在全局路由拦截中写，只要每次版本号不同就重新加载页面配合第一步就可以清楚浏览器缓存**

1. import versionTood from '@/libs/versionUpdate'
3. router.beforeEach(( to, from, next ) => {
4. *//判断当前代码版本是否与服务器中代码版本一致，如不一致则刷新页面获取最新*
5. versionTood.isNewVersion();
6. }

## 2 vue-cli3.0

1. **在vue.config.js中配置output**
2. const Timestamp = new Date().getTime();
4. configureWebpack: {
5. output: {
6. filename: `js/[name].${Timestamp}.js`, *// 每次构建打包时给文件名加上时间戳，确保每次版本更新的文件名不一样*
7. chunkFilename: `js/[name].${Timestamp}.js`
8. }
9. }

**2.在public目录下新建version.json每次发版更改里面的版本号 (同vue-cli2.0)**

**3.在src中新建 libs/versionUpdate.js文件**

**4.在全局路由拦截中写，只要每次版本号不同就重新加载页面配合第一步就可以清楚浏览器缓存 (同vue-cli2.0)**

# 21. vue.js中获取文件路径时assets与static的区别

总结区别有以下几点:

　　1.打包时：（都是打包至dist目录下）

　　　　assets中的文件会被webpack中的file-loader打包，所以在打包的过程中assets路径中的文件会被压缩，耗时间但是节省了空间。而static中的文件则不会被打包，相比assect中的文件打包更快，但是打包完成后文件所占用空间比较大。

　　2.在文件中直接引用时，都是直接写入相对路径。但是在路径为动态绑定的时，由于webpack遵循commonJS规范，所以在给变量名赋值时需要使用require方法。

　　　　eg：assectImg：require（"../../assect/1.jpg"）,而static中则为：staticImg："../../../static/2.jpg"。然后才能在img标签下的src中使用之前定义好的路径。<img src="assectImg"/>   <img src="staticImg"/>

# 22. Vue.set(target,key,value) 响应式对象设置新属性，并确保这个新 property 同样是响应式的，且触发视图更新。

# 23. Vue.delete(target,key,value) 响应式对象删除属性，且触发视图更新。

# 24. 项目打包区分开发环境，生产环境和测试环境

vue cli3没有了和之前vue cli2一样的配置文件，   
但项目中 经常会有测试环境和生产环境，经常不同环境的域名是不一致的，因为build构建打包时，process.env.NODE\_ENV默认是production的生产环境，所以我们每次build都去手动更改请求域名，很麻烦。。。  
但我们又不能直接去更改process.env.NODE\_ENV等于test来区分是测试环境，这样打包出来的包结构和production打包出来的是不一样的

## 1. Vue-cli 3.0 、Vue-cli 4.0

根目录下创建 .env.production 配置

NODE\_ENV = production

VUE\_APP\_CURRENT = production

VUE\_APP\_OUTPUTDIR = dist

根目录下创建 .env.test配置

NODE\_ENV = production

VUE\_APP\_CURRENT = test

VUE\_APP\_OUTPUTDIR = test

注意 自定义key需要以 VUE\_APP\_ 开头

Package.josn设置

"serve": "vue-cli-service serve", // 开发环境

"build": "vue-cli-service build --mode production", // 生产环境

"test": "vue-cli-service build --mode test", // 测试环境

"publish": "npm run build && npm run test" // 同时打包生产环境和测试环境

Vue.config.js设置

outputDir: process.env.VUE\_APP\_OUTPUTDIR

## 2. Vue-cli 2.0

安装cross-env

Npm install cross-env –save

Package.json设置

"scripts": {

    "dev": "webpack-dev-server --inline --progress --config build/webpack.dev.conf.js", // 开发环境

    "start": "npm run dev",

    "build": "cross-env PATH\_ENV=production node build/build.js", // 打包生产环境

    "test": "cross-env PATH\_ENV=test node build/build.js", // 打包测试环境

    "publish": "npm run build && npm run test", // 打包开发和生产环境

    "fix-memory-limit": "cross-env LIMIT=4096 increase-memory-limit"

  },

在prod.env.js

'use strict'

let PATH\_ENV = '"test"'

if(process.env.PATH\_ENV === 'production') PATH\_ENV = '"production"'

module.exports = {

  NODE\_ENV: '"production"',

  PATH\_ENV

}

Config文件夹index.js

build: {

    env: require("./prod.env"),

    // Template for index.html

    index: process.env.PATH\_ENV === 'production' ? path.resolve(\_\_dirname, '../dist/index.html') : path.resolve(\_\_dirname, '../test/index.html'), // 区分开发环境和生产环境的打包路径

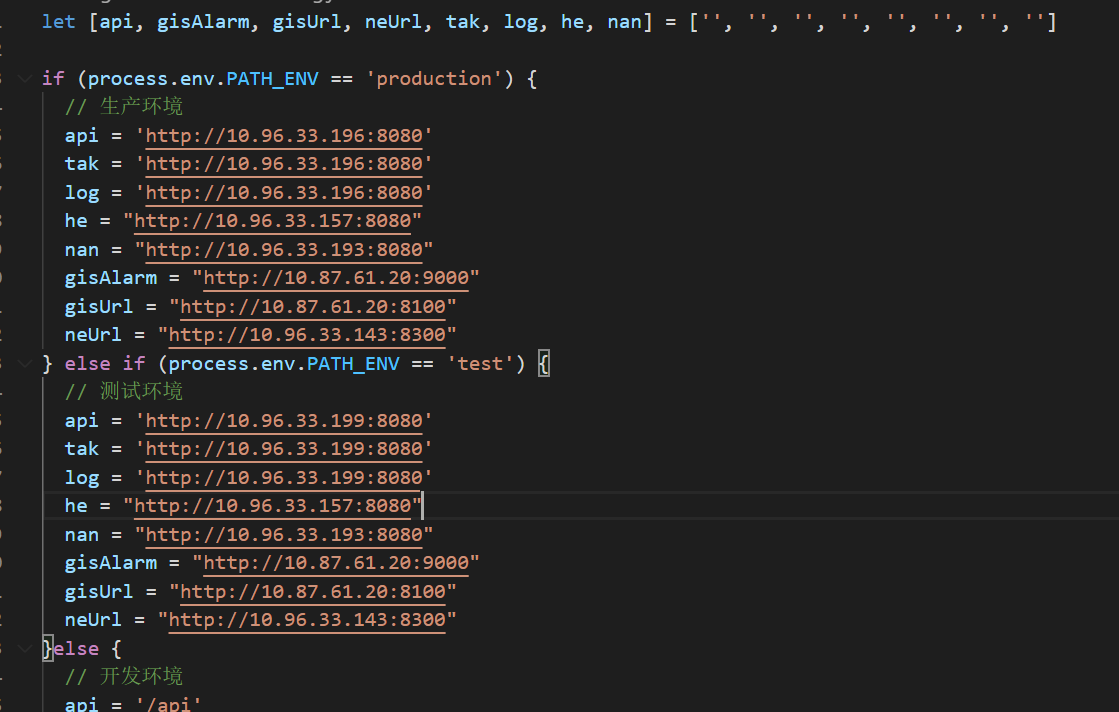
    // Paths

    assetsRoot: process.env.PATH\_ENV === 'production' ? path.resolve(\_\_dirname, '../dist') : path.resolve(\_\_dirname, '../test'),

    assetsSubDirectory: 'static',

    assetsPublicPath: './',

最后



其他build.js 、vue-loader.conf.js 、webpack.base.conf.js、webpack.dev.conf.js、webpack.prod.conf.js 是否更改配置视个人情况而定（非主要）

# 25. Vue强制组件重新渲染方法

* 简单粗暴的方式：重新加载整个页面
* 不妥的方式：使用 v-if
* 较好的方法：使用Vue的内置forceUpdate方法
* 最好的方法：在组件上进行 key 更改

# 26. 无法得知父元素高度的情况下如何上下居中

* .level\_1{
* width: 100px;
* min-height: 30px;
* vertical-align: middle;
* &:before{
* content: " ";
* display: inline-block;
* height: 100%;
* width: 0;
* vertical-align: middle;
* }
* }

# 27. iview表单无法验证

表单的ref="..."是否有添加；  
表单的:model="..."是否有添加；  
表单的:rules="..."是否有添加；  
表单的ref="..."和:model="..." 是否一致；  
表单的FormItem的属性prop="..."是否与输入框（例如input）的v-moel="..."模版名称一致； 解决办法，formItem栏单独设置rules

# 28. Json2excel

<https://github.com/xxj95719/custom-json2excel>

# 29. axios.all

axios.all(info:Array).then(axios.spread((...data) => {

        console.log([...data])

      }))

循环调接口时

# 30. treeselect



# 31在vue文件的HTML结构里面使用moment方法报错

解决办法就是在data或者methods里面写个moment属性（方法）



# 32 css color: var(--message-text);

Body .dark{

--message-text: #3c3f56

}

# 33. vue 线上环境 开启 vue-devtools

<https://blog.csdn.net/weixin_44717473/article/details/110798112>

# 34. vue cli2项目中引入typescript

<https://blog.csdn.net/weixin_45323439/article/details/102851137>

# 35. vue cli3项目中引入typescript

<https://blog.csdn.net/qq_39216281/article/details/103851950>

# 36. vue v-html防止xss攻击

npm install xss *–save* 会把除了标签和内容之外的所有东西都给过滤掉，比如class，style这样过滤掉后富文本的样式就没有了，美观度一下就降低了。

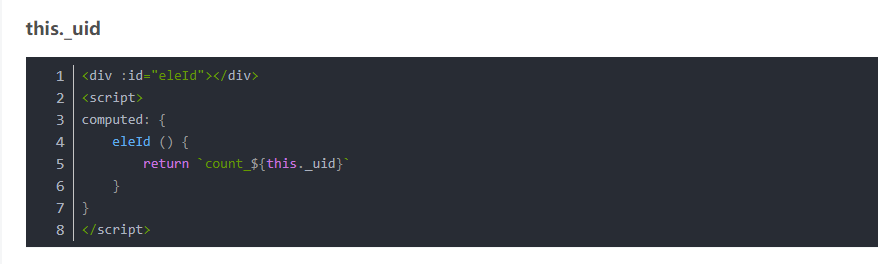
**vue-dompurify-html插件**

# 37. Provide和inject用法

通常，当我们需要从父组件向子组件传递数据时，我们使用 [**props**](https://v3.cn.vuejs.org/guide/component-props.html)。想象一下这样的结构：有一些深度嵌套的组件，而深层的子组件只需要父组件的部分内容。在这种情况下，如果仍然将 prop 沿着组件链逐级传递下去，可能会很麻烦。

对于这种情况，我们可以使用一对 provide 和 inject。无论组件层次结构有多深，父组件都可以作为其所有子组件的依赖提供者。这个特性有两个部分：父组件有一个 provide 选项来提供数据，子组件有一个 inject 选项来开始使用这些数据。

# 38. 每一个vue实例属性中都有一个this.\_uid, 每个组件中的this.\_uid值都不相同, 唯一的



# 39. vue 多个组件中同时使用window.onresize时，只有一个组件起作用时，解决办法

将 window.onresize 替换成 window.addEventListener(“resize”, () => { })

# 39. Object.assign有深拷贝和浅拷贝之分，深拷贝值改变不会影响原对象，浅拷贝会影响