目录

[ vue指令 3](#_Toc28379)

[1. 常用指令： 3](#_Toc25948)

[2. ：src 在data内定义的图片路径写法(require) 4](#_Toc26348)

[3. 自定义日期格式化过滤器 4](#_Toc26164)

[ 生命周期 8](#_Toc742)

[ 数组更新 9](#_Toc10718)

[4. 数组跟新-改变原数组 9](#_Toc6078)

[5. 数组更新-不改变原数组 9](#_Toc22521)

[ 方法与事件 9](#_Toc8997)

[6. 特殊变量$event，用于访问原生DOM 9](#_Toc25018)

[7. 事件修饰符-阻止默认事件、事件串联和只触发一次等passive 10](#_Toc15009)

[8. 按键修饰符 10](#_Toc21287)

[ v-modal表单双向数据绑定 11](#_Toc15441)

[9. v-modal修饰符.laze .number .trim 11](#_Toc21331)

[ 组件通信props/$emit/v-model 11](#_Toc5823)

[10. Props传递数据验证(必须传入，自定义验证函数)： 11](#_Toc29523)

[11. 子向父通信$emit传递参数 13](#_Toc26747)

[12. 组件自定义事件传递参数v-model 14](#_Toc10757)

[13. 组件非父子通信、兄弟通信$emit--$on 16](#_Toc32081)

[14. 父链通信—子组件通过this.$parent直接访问该父组件实例 18](#_Toc2783)

[15. 子组件索引—this.$children 19](#_Toc7761)

[ 动态组件和异步组件 20](#_Toc30812)

[16. 动态组件 20](#_Toc4872)

[17. 异步组件、延迟加载组件 22](#_Toc22219)

[18. $nextTick 下一个vue事件循环时，才开始创建 23](#_Toc26013)

[19. X-Templates-在script标签内写template内容 24](#_Toc10884)

[20. 自定义指令directive 25](#_Toc18603)

[ +Vue-cli项目构建 25](#_Toc9422)

[21. Vue-cli项目构建 25](#_Toc13582)

[22. Vue-cli安装配置css预处理器less 25](#_Toc32001)

[23. Vue-cli安装配置css预处理器stylus 27](#_Toc18177)

[ 路由 28](#_Toc8102)

[24. 路由配置： 28](#_Toc23867)

[25. 路由跳转： 29](#_Toc19360)

[26. 路由激活样式：active 29](#_Toc20977)

[27. 路由页面过渡和缓存 30](#_Toc13740)

[28. 路由嵌套 30](#_Toc9445)

[29. 路由守卫 31](#_Toc19630)

[next('/') 或者 next({ path: '/' }): 跳转到一个不同的地址。当前的导航被中断，然后进行一个新的导航。你可以向 next 传递任意位置对象，且允许设置诸如 replace: true、name: 'home' 之类的选项以及任何用在 router-link 的 to prop 或 router.push 中的选项。 31](#_Toc26344)

[ Vue-Resource （Ajax） 33](#_Toc19883)

[30. 异步请求数据- Vue-Resource 33](#_Toc31356)

[ 参数传递 34](#_Toc3175)

[ 常见问题 34](#_Toc6779)

[31. webpack vue-loader was used without the corresponding plugin. Make sure to include VueLoaderPlugin 34](#_Toc15858)

[解决 34](#_Toc24293)

[32. Vue警告：Duplicate keys Detected: ‘0’ 35](#_Toc15537)

[33. 图片路径(路径拼接、路径引用) 35](#_Toc31054)

[34. Json-server 后台数据模拟 36](#_Toc14760)

[35. 最新的vue没有dev-server.js文件，如何进行后台数据模拟？ 36](#_Toc28027)

[36. vue.js学习之 如何在手机上查看vue-cli构建的项目 42](#_Toc11531)

[37. Mint-UI修改.babelIrc报错-- Error: Couldn't find preset "es2015" relative to directory "E:/Project\_Documents/Test/js" 43](#_Toc28724)

[38. 使用mint-ui Navbar遇到的一个小坑 首次加载tab-container不显示 43](#_Toc10336)

[39. this dependency was not found---组件引入报错 43](#_Toc9556)

[ VUE2.0不可忽视的很多变化 43](#_Toc32448)

[ 防止刷新vuex的state丢失 45](#_Toc32663)

# vue指令

## 常用指令：

v-if / v-else-if / v-else v-show

v-for: 2.0需绑定 :key = obj.name

v-html: 插入html文本

v-pre: {{该内容不会被编译}}

v-bind:

v-on、v-dbclick、v-keyup、 v-mousemove、 v-blur

v-cloak: 消除网速慢时加载{{message}}字样

v-once: 值渲染一次

## ：src 在data内定义的图片路径写法(require)

data () {  
 **return** {  
 src: require('../assets/slideShow/pic.jpg')

}

}

过滤器

## 自定义日期格式化过滤器

（1）：

<div id="app">  
 {{ date | formatDate }}  
 <p v-html="link"></p>  
</div>  
<script src="https://unpkg.com/vue/dist/vue.min.js"></script>  
<script>  
 **var** padDate = **function** (val) {  
 **return** val < 10 ? '0' + val : val  
 }  
 **var** app = **new** Vue({  
 el: '#app',  
 data: {  
 date: **new** Date(),  
 link: '<a href = "#">这是一个链接</a>'  
 },  
 filters: {  
 formatDate (value) {  
 **var** date = **new** Date (value);  
 **var** year = date.getFullYear();  
 **var** month = padDate (date.getMonth() + 1);  
 **var** day = padDate(date.getDate());  
 **var** hours = padDate(date.getHours());  
 **var** minutes = padDate(date.getMinutes());  
 **var** seconds = padDate(date.getSeconds());  
 **return** year + '-' + month + '-' + day + ' ' + hours + ':' + minutes + ':' + seconds  
 }  
 },  
 mounted () {  
 **this**.timer = setInterval(() => {  
 **this**.date = **new** Date();  
 }, 1000)  
 },  
 beforeDestroy () {  
 **if** (**this**.timer) {  
 clearInterval(**this**.timer)  
 }  
 }  
 })  
 console.log(app.$data)

（2）注：过滤器也可以放在computed属性了（this.books.filter((book) => {})）

<div id="app">  
 <ul>  
 <template v-for="book in books">  
 <li>{{book.name}}</li>  
 <li>{{book.author}}</li>  
 </template>  
 </ul>  
</div>  
<script src="https://unpkg.com/vue/dist/vue.min.js"></script>  
<script>  
 **var** app = **new** Vue({  
 el: '#app',  
 data: {  
 books:[  
 {  
 name: 'vue.js实战',  
 author: 'yys'  
 },  
 {  
 name: 'java实战',  
 author: 'yys'  
 },  
 {  
 name: 'java实战',  
 author: 'yys'  
 }  
 ]  
 }  
 })  
 app.books = app.books.filter((item) => {  
 **return** item.name.match(/java/)  
 })

# 生命周期

**beforeCreate:** 在实例初始化之后，数据观测 (data observer) 和 event/watcher 事件配置之前被调用。

**created:** 在实例创建完成后被立即调用。在这一步，实例已完成以下的配置：数据观测 (data observer)，property 和方法的运算，watch/event 事件回调。然而，挂载阶段还没开始，$el property 目前尚不可用。

**beforeMount:** 在挂载开始之前被调用：相关的 render 函数首次被调用。

**mounted:** 实例被挂载后调用，这时 el 被新创建的 vm.$el 替换了。如果根实例挂载到了一个文档内的元素上，当 mounted 被调用时 vm.$el 也在文档内。

**beforeUpdate:** 数据更新时调用，发生在虚拟 DOM 打补丁之前。这里适合在更新之前访问现有的 DOM，比如手动移除已添加的事件监听器。

**updated:** 由于数据更改导致的虚拟 DOM 重新渲染和打补丁，在这之后会调用该钩子。

**activated:** 被 keep-alive 缓存的组件激活时调用。

**deactivated:** 被 keep-alive 缓存的组件停用时调用。

**beforeDestroy:** 实例销毁之前调用。在这一步，实例仍然完全可用。

destroyed: 实例销毁后调用。该钩子被调用后，对应 Vue 实例的所有指令都被解绑，所有的事件监听器被移除，所有的子实例也都被销毁。

**errorCaptured:** 当捕获一个来自子孙组件的错误时被调用。此钩子会收到三个参数：错误对象、发生错误的组件实例以及一个包含错误来源信息的字符串。此钩子可以返回 false 以阻止该错误继续向上传播。

# 数组更新

## 数组跟新-改变原数组

Push() pop() shift() unshift() splice() sort() reverse()

## 数组更新-不改变原数组

Filter() concat() slice()

# 方法与事件

## 特殊变量$event，用于访问原生DOM

<div id="app">  
 <a href="http://www.apple.com" @click="handleClick('禁止打开',$event)">打开链接</a>  
</div>  
<script src="https://unpkg.com/vue/dist/vue.min.js"></script>  
<script>  
 **var** app = **new** Vue({  
 el: '#app',  
 methods:{  
 handleClick (msg, event) {  
 event.preventDefault();  
 window.alert(msg)  
 }  
 }  
 })  
</script>

## 事件修饰符-阻止默认事件、事件串联和只触发一次等passive

* .stop // 阻止冒泡
* .prevent //阻止默认行为
* .capture //添加事件监听器时使用事件捕获模式
* .self //触发该元素本身而不是子元素或父元素
* .once //事件只触发一次

<div @click.once=”handleClick”>只触发一次</div>

Vue 还对应 **[addEventListener 中的 passive 选项](https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener" \l "Parameters" \t "_blank)**提供了 .passive 修饰符。

|  |
| --- |
| <!-- 滚动事件的默认行为 (即滚动行为) 将会立即触发 --> <!-- 而不会等待 `onScroll` 完成 --> <!-- 这其中包含 `event.preventDefault()` 的情况 --> <div v-on:scroll.passive="onScroll">...</div> |

这个 .passive 修饰符尤其能够提升移动端的性能。

## 按键修饰符

.enter .tab .delete(捕获“”删除”或“退格”) .esc .space .up .down .left .right

.ctrl .alt .shift .meta(在 Mac 系统键盘上，meta 对应 command 键 (⌘)。在 Windows 系统键盘 meta 对应 Windows 徽标键 (⊞))

**组合使用：**

<input @keyup.alt.67="clear">

**自定义按键修饰符：**

Vue.config.keyCods.f1 = 112

# v-modal表单双向数据绑定

## v-modal修饰符.laze .number .trim

v-model.lazy //在失去焦点或按回车时才更新数据

v-model.number //输入的数字转为Number类型

v-model.trim //自动过滤前后空格

# 组件通信props/$emit/v-model

## Props传递数据验证(必须传入，自定义验证函数)：

Vue.component('my', {  
 props: {  
 propA: Number,  
 propB: [Number, String],  
 propC: {  
 type: Number,  
 **default**: **true** },  
 // 数字类型，必传  
 propD: {  
 type: Number,  
 required: **true** //必传  
 },  
 propE: {  
 type: Array,  
 **default** () {  
 **return** []  
 }  
 },  
 // 自定义一个验证函数  
 propF: {  
 myVal (value) {  
 **return** value > 10  
 }  
 }  
 }  
})

## 子向父通信$emit传递参数

**(注意：自定义事件名不能使用驼峰命名，如@getTotal无效，需写@gettotal)**

<div id="app">  
 <p>总数：{{total}}</p>  
 <my @gettotal="handleGetTotal"></my>  
</div>  
<script src="https://unpkg.com/vue/dist/vue.min.js"></script>  
<script>  
 Vue.component('my', {  
 template: `  
 <div>  
 <button @click="handelClick">+1</button>  
 </div>  
 `,  
 data () { **return** { counter: 0 } },  
 methods: {  
 handelClick () {  
 **this**.counter ++  
 **this**.$emit('gettotal', **this**.counter)  
 },  
 }  
 })  
 **let** app = **new** Vue({  
 el: '#app',  
 data: { total: 1 },  
 methods: {  
 handleGetTotal (total) {**this**.total = total}  
 }  
 })  
</script>

## 组件自定义事件传递参数v-model

<div id="app">  
 <p>总数：{{total}}</p>  
 <my v-model="total"></my>  
 <button @click="handleReduce">-1</button>  
</div>  
<script src="https://unpkg.com/vue/dist/vue.min.js"></script>  
<script>  
 Vue.component('my', {  
 props: ['value'],  
 template: `  
 <div>  
 <input type="text" :value="value" @input="updateValue">  
 </div>  
 `,  
 data () {  
 **return** { counter: 0 }  
 },  
 methods: {  
 updateValue (event) {  
 **this**.$emit('input', event.target.value)  
 },  
 }  
 })  
 **let** app = **new** Vue({  
 el: '#app',  
 data: { total: 0 },  
 methods: {  
 handleReduce () { **this**.total -- }  
 }  
 })  
</script>

## 组件非父子通信、兄弟通信$emit--$on

参考文档：<https://www.cnblogs.com/xk-one/p/8520714.html>

<div id="app">  
 {{message}}  
 <component-a></component-a>  
</div>  
<script src="https://unpkg.com/vue/dist/vue.min.js"></script>  
<script>

// 创建一个名为bus的空的Vue实例，作中间商把信息在组件间传递

**let** bus = **new** Vue () ;  
 Vue.component('component-a', {  
 template: `  
 <button @click="handelEvent">传递事件</button>  
 `,  
 methods: {  
 handelEvent () {  
 bus.$emit('on-message', '来自组件component-a 的内容')  
 }  
 }  
 })  
 **let** app = **new** Vue({  
 el: '#app',  
 data: {  
 message: ''  
 },  
 mounted () {  
 bus.$on('on-message', (msg) => {  
 **this**.message = msg  
 })  
 }  
 })  
</script>

## 父链通信—子组件通过this.$parent直接访问该父组件实例

<div id="app">  
 {{message}}  
 <component-a></component-a>  
</div>  
<script src="https://unpkg.com/vue/dist/vue.min.js"></script>  
<script>  
 Vue.component('component-a',{  
 template: `  
 <button @click="handelEvent">通过父链直接修改数据</button>  
 `,  
 methods: {  
 handelEvent () {  
 **this**.$parent.message = '来自组件"component-a的内容"'  
 }  
 }  
 })  
 **var** app = **new** Vue({  
 el: '#app',  
 data: {message: ''}  
 })  
</script>

## 子组件索引—this.$children

<div id="app">  
 <button @click="handleRef">通过ref获取子组件的实例</button>  
 <component-a ref = 'comA'></component-a>  
</div>  
<script src="https://unpkg.com/vue/dist/vue.min.js"></script>  
<script>  
 Vue.component('component-a', {  
 template: `<div>子组件</div>`,  
 data () {  
 **return** {  
 message: '子组件的内容'  
 }  
 }  
 })  
 **var** app = **new** Vue({  
 el: '#app',  
 methods: {  
 handleRef () {  
 // 通过$refs来访问指定的实例  
 **let** msg = **this**.$refs.comA.message  
 console.log(msg);  
 }  
 }  
 })  
</script>

# 动态组件和异步组件

## 动态组件

* <div id="app">  
   <component :is="showComponent"></component>  
   <button @click="handelChange('A')">切换到A</button>  
   <button @click="handelChange('B')">切换到B</button>  
   <button @click="handelChange('C')">切换到C</button>  
  </div>  
  <script src="https://unpkg.com/vue/dist/vue.min.js"></script>  
  <script>  
   **var** app = **new** Vue({  
   el: '#app',  
   data: {  
   showComponent: "comA"  
   },  
   components: {  
   comA: {template: '<div>comA</div>'},  
   comB: {template: '<div>comB</div>'},  
   comC: {template: '<div>comC</div>'}  
   },  
   methods: {  
   handelChange (component) {  
   **this**.showComponent = 'com' + component  
   }  
   }  
   })  
  </script>

## 异步组件、延迟加载组件

<div id="app">  
 <child-component></child-component>  
</div>  
<script src="vue.js"></script>  
<script>  
 // 工厂函数，接收一个resolve回调，也可以调用reject(reason)指示加载失败  
 Vue.component('child-component', **function** (resolve, reject) {  
 // setTimeout只是为了演示异步，也可以是Ajax请求等  
 setTimeout(() => {  
 resolve({  
 template: '<div>我是异步渲染的</div>'  
 })  
 },2000)  
 })  
 **var** app = **new** Vue({  
 el: '#app',  
 })  
</script>

## $nextTick 下一个vue事件循环时，才开始创建

<div id="app">  
 <div id="div" v-if="showDiv">这是一段文本</div>  
 <button @click="getText">获取div内容</button>  
</div>  
<script src="https://unpkg.com/vue/dist/vue.min.js"></script>  
<script>  
 **var** app = **new** Vue({  
 el: '#app',  
 data: {showDiv: **false**},  
 methods: {  
 getText () {  
 **this**. showDiv = **true  
 this**.$nextTick(() => {  
 **var** text =document.getElementById('div').innerHTML;  
 console.log(text);  
 })  
 }  
 }  
 })  
</script>

## X-Templates-在script标签内写template内容

<div id="app">  
 <my-component></my-component>  
</div>  
<script type="text/x-template" id="my-component">  
 <div>这是组件内容</div>  
</script>  
<script src="https://unpkg.com/vue/dist/vue.min.js"></script>  
<script>  
 Vue.component('my-component',{  
 template: '#my-component'  
 })  
 **var** app = **new** Vue({  
 el: '#app'  
 })  
</script>

## 自定义指令directive

[https://cn.vuejs.org/v2/guide/custom-directive.html#ad](https://cn.vuejs.org/v2/guide/custom-directive.html" \l "ad)

# +Vue-cli项目构建

## Vue-cli项目构建

# 全局安装 vue-cli  
$ npm install --global vue-cli  
# 创建一个基于 webpack 模板的新项目  
$ vue init webpack my-project  
# 安装依赖，走你  
$ cd my-project  
$ npm install  
$ npm run dev

## Vue-cli安装配置css预处理器less

1. npm install less less-loader --save-dev
2. **在配置文件中配置**

　实际上如果我们通过vue-cli来构建项目，这一步是可以省略的。

　在webpack.dev.conf.js中，我们可以看到下面的代码：

module: {

rules: utils.styleLoaders({ sourceMap: config.dev.cssSourceMap })

},

　即webpack.dev.conf.js在合并了webpack.base.conf.js的基础上又添加了dev环境下的module。

**注意**：**在上面的代码中，我们还可以使用loaders来代替rules， 他们的含义是一样的。**

在build文件夹下有一个utils.js文件，这个文件提供了一些通用的方法，供webpack.dev.conf.js和webpack.prod.conf.js使用。 其中styleLoaders方法如下：

[复制代码](javascript:void(0);)

// Generate loaders for standalone style files (outside of .vue)

exports.styleLoaders = function (options) {

var output = []

var loaders = exports.cssLoaders(options)

for (var extension in loaders) {

var loader = loaders[extension]

output.push({

test: new RegExp('\\.' + extension + '$'),

use: loader

})

}

return output

}

[复制代码](javascript:void(0);)

　　通过这个方法可以对大多数css预处理进行了配置，具体配置在cssLoaders方法中。

3）：

<style lang="less">  
 @c:#f00;  
 #app {color: @c;}  
</style>

## Vue-cli安装配置css预处理器stylus

* cnpm install stylus --save-dev
* cnpm install stylus-loader --save-dev

# 路由

## 路由配置：

**import** Vue **from 'vue'  
import** Router **from 'vue-router'  
import** Apple **from '../components/apple'  
import** Banana **from '../components/banana'**Vue.use(Router)  
**export default new** Router({

linkActiveClass: 'active', //将激活的路由添加一个mui-active类名称

**mode**: **'history'**,

/\* [http://localhost:8080/#/banana](http://localhost:8080/" \l "/banana) =><http://localhost:8080/banana> \*/  
 **routes**: [

{  
  **path**: **'/'**,  
 **redirect: '/apple'** //首次不指定路径默认加载apple并高亮显示

},  
 {  
 **path**: **'/apple'**,  
 **component**: Apple  
 },  
 {  
 **path**: **'/banana'**,  
 **component**: Banana  
 }  
 ]  
})

## 路由跳转：

1、<**router-link to="/banana"**>to banana</**router-link**>

2、<**router-link ：to="{path:‘banana’}"**>to banana</**router-link**>

3、this.$router.push('/myLogin');

\* 注意： 如果使用3，首次加载导航条router-link 样式没激活需要用到重定向激活

{  
 path: '/myBought',  
 component: MyBought  
},  
{  
 path: '/bought',

//重定向  
 redirect: '/myBought'  
}

## 路由激活样式：active

**const** router = **new** VueRouter({  
 linkActiveClass: 'active', //将激活的路由添加一个mui-active类名称  
 routes // （缩写）相当于 routes: routes  
});

## 路由页面过渡和缓存

<**transition name="fade"**>  
 <**keep-alive**>  
 <**router-view**/>  
 </**keep-alive**>  
</**transition**>

## 路由嵌套

// 1、 “首页”组件中，还嵌套着 “登录”和 “注册”组件，那么URL对应就是home/login和/home/reg。

<template id="home">

<!-- 注意：组件只能有一个根元素，所以我们包装到这个div中 -->

<div>

<h2>首页</h2>

<router-link to="/home/login">登录</router-link>

<router-link to="/home/reg">注册</router-link>

<!-- 路由匹配到的组件将渲染在这里 -->

<router-view></router-view>

</div>

</template>

// 2. 定义路由

const routes = [

{ path: '/', redirect: '/home' },

{

path: '/home',

component: Home,

children:[

{ path: '/home/login', component: Login},

{ path: '/home/reg', component: Reg}

]

},

{ path: '/news', component: News}

]

## 路由守卫

**全局前置守卫 router.beforeEach(to, from, next)**

to: Route 即将进入的目标路由对象

from:Route 当前导航正要离开的路由

next: Function 一定要调用该方法来resolve这个钩子

next() 进行管道下一个钩子

next(false) 中断当前的导航。如果浏览器的 URL 改变了 (可能是用户手动或者浏览器后退按钮)，那么 URL 地址会重置到 from 路由对应的地址。

# next('/') 或者 next({ path: '/' }): 跳转到一个不同的地址。当前的导航被中断，然后进行一个新的导航。你可以向 next 传递任意位置对象，且允许设置诸如 replace: true、name: 'home' 之类的选项以及任何用在 [router-link 的 to prop](https://router.vuejs.org/zh/api/" \l "to) 或 [router.push](https://router.vuejs.org/zh/api/" \l "router-push) 中的选项。

**next(error)**: (2.4.0+) 如果传入 next 的参数是一个 Error 实例，则导航会被终止且该错误会被传递给 [router.onError()](https://router.vuejs.org/zh/api/" \l "router-onerror) 注册过的回调。

**全局解析守卫 router.beforeResolve(to, from, next)**

**全局后置钩子 router.afterEach(to, from)**

**路由独享守卫beforeEnter**

const router = new VueRouter({

  routes: [

    {

      path: '/foo',

      component: Foo,

      beforeEnter: (to, from, next) => {

        // ...

      }

    }

  ]

})

**组件内的守卫**

* **beforeRouteEnter**
* **beforeRouteUpdate (2.2 新增)**
* **beforeRouteLeave**

const Foo = {

  template: `...`,

  beforeRouteEnter (to, from, next) {

    // 在渲染该组件的对应路由被 confirm 前调用

    // 不！能！获取组件实例 `this`

    // 因为当守卫执行前，组件实例还没被创建

  },

  beforeRouteUpdate (to, from, next) {

    // 在当前路由改变，但是该组件被复用时调用

    // 举例来说，对于一个带有动态参数的路径 /foo/:id，在 /foo/1 和 /foo/2 之间跳转的时候，

    // 由于会渲染同样的 Foo 组件，因此组件实例会被复用。而这个钩子就会在这个情况下被调用。

    // 可以访问组件实例 `this`

  },

  beforeRouteLeave (to, from, next) {

    // 导航离开该组件的对应路由时调用

    // 可以访问组件实例 `this`

  }

}

# Vue-Resource （Ajax）

## 异步请求数据- Vue-Resource

* + npm install vue-resource
  + 引入vue-resource:

  *// GET /someUrl*

  this.$http.get('/someUrl').then(response => {

*// get body data*

    this.someData **=** response.body;

  }, response => {

*// error callback*

  });

}

# 参数传递

父传子：

# 常见问题

## webpack vue-loader was used without the corresponding plugin. Make sure to include VueLoaderPlugin

参考网站：<https://blog.csdn.net/cominglately/article/details/80555210>

### 解决

. 在webpack.config.js中加入

* 1
* 2

const VueLoaderPlugin = require('vue-loader/lib/plugin');

module.exports = {

devtool: "sourcemap",

entry: './js/entry.js', // 入口文件

output: {

filename: 'bundle.js' // 打包出来的wenjian

},

plugins: [

// make sure to include the plugin for the magic

new VueLoaderPlugin()

],

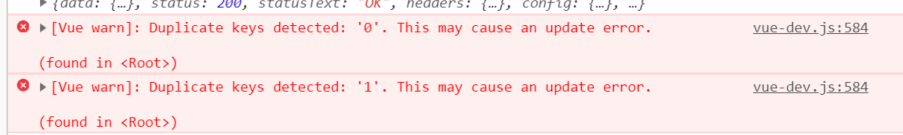
module : {

...

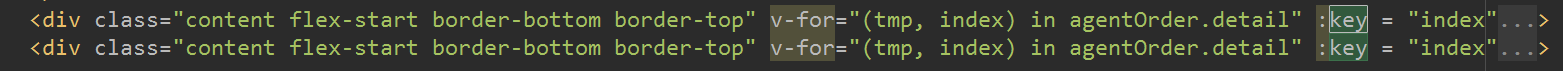
}

}

## Vue警告：Duplicate keys Detected: ‘0’



原因是重复了两个一样的for循环



去掉一个即可

## 图片路径(路径拼接、路径引用)

路径拼接：

<div class="swiper-slide" v-for="item in message">

　　<img v-bind:src="['xxx(需要拼接的字符串)'+item.picurl]" alt="" width="100%" height="245" />

</div>

路径引用：<http://blog.csdn.net/Letasian/article/details/76219889>

## Json-server 后台数据模拟

* 1、npm install json-server –save
* 2、在webpack.dev.conf.js下输入以下代码
* **const** jsonServer = require('json-server')  
  **const** apiServer = jsonServer.create()  
  **const** apiRouter = jsonServer.router('db.json') //db.json为我的jsons数据文件  
  **const** middlewares = jsonServer.defaults()  
  apiServer.use(middlewares)  
  apiServer.use(apiRouter)  
  apiServer.listen(8081, () => { //8081为json-server访问额端口  
   console.log('JSON Server is running')  
  })

## 最新的vue没有dev-server.js文件，如何进行后台数据模拟？

最新的vue里dev-server.js被替换成了webpack-dev-conf.js

在模拟后台数据的时候直接在webpack-dev-conf.js文件中修改

第一步，；在const portfinder = require(‘portfinder’)后添加

//第一步

const express = require('express')

const app = express()//请求server

var appData = require('../data.json')//加载本地数据文件

var seller = appData.seller//获取对应的本地数据

var goods = appData.goods

var ratings = appData.ratings

var apiRoutes = express.Router()

app.use('/api', apiRoutes)//通过路由请求数据

第二步：找到devServer,在里面加上before（）方法

devServer: {

clientLogLevel: 'warning',

historyApiFallback: true,

hot: true,

compress: true,

host: HOST || config.dev.host,

port: PORT || config.dev.port,

open: config.dev.autoOpenBrowser,

overlay: config.dev.errorOverlay

? { warnings: false, errors: true }

: false,

publicPath: config.dev.assetsPublicPath,

proxy: config.dev.proxyTable,

quiet: true, // necessary for FriendlyErrorsPlugin

watchOptions: {

poll: config.dev.poll,

},

//第二步找到devServer,在里面添加

before(app) {

app.get('/api/seller', (req, res) => {

res.json({

errno: 0,

data: seller

})//接口返回json数据，上面配置的数据seller就赋值给data请求后调用

}),

app.get('/api/goods', (req, res) => {

res.json({

errno: 0,

data: goods

})

}),

app.get('/api/ratings', (req, res) => {

res.json({

errno: 0,

data: ratings

})

})

}

},

提供一个json.data数据

{

"seller": {

"name": "粥品香坊（回龙观）",

"description": "蜂鸟专送",

"deliveryTime": 38,

"score": 4.2,

"serviceScore": 4.1,

"foodScore": 4.3,

"rankRate": 69.2,

"minPrice": 20,

"deliveryPrice": 4,

"ratingCount": 24,

"sellCount": 90,

"bulletin": "粥品香坊其烹饪粥料的秘方源于中国千年古法，在融和现代制作工艺，由世界烹饪大师屈浩先生领衔研发。坚守纯天然、0添加的良心品质深得消费者青睐，发展至今成为粥类的引领品牌。是2008年奥运会和2013年园博会指定餐饮服务商。",

"supports": [

{

"type": 0,

"description": "在线支付满28减5"

},

{

"type": 1,

"description": "VC无限橙果汁全场8折"

},

{

"type": 2,

"description": "单人精彩套餐"

},

{

"type": 3,

"description": "该商家支持发票,请下单写好发票抬头"

},

{

"type": 4,

"description": "已加入“外卖保”计划,食品安全保障"

}

],

"avatar": "http://static.galileo.xiaojukeji.com/static/tms/seller\_avatar\_256px.jpg",

"pics": [

"http://fuss10.elemecdn.com/8/71/c5cf5715740998d5040dda6e66abfjpeg.jpeg?imageView2/1/w/180/h/180",

"http://fuss10.elemecdn.com/b/6c/75bd250e5ba69868f3b1178afbda3jpeg.jpeg?imageView2/1/w/180/h/180",

"http://fuss10.elemecdn.com/f/96/3d608c5811bc2d902fc9ab9a5baa7jpeg.jpeg?imageView2/1/w/180/h/180",

"http://fuss10.elemecdn.com/6/ad/779f8620ff49f701cd4c58f6448b6jpeg.jpeg?imageView2/1/w/180/h/180"

],

"infos": [

"该商家支持发票,请下单写好发票抬头",

"品类:其他菜系,包子粥店",

"北京市昌平区回龙观西大街龙观置业大厦底商B座102单元1340",

"营业时间:10:00-20:30"

]

}}

PS：

所有的修改配置都需要重新启动运行命令的：npm run dev才能生效（很重要，否则无法请求到数据）

## vue.js学习之 如何在手机上查看vue-cli构建的项目

一：找到config文件夹下的index.js文件，打开后，将host的值改为你本地的ip，保存后重启项目

二：输入ip和端口号打开项目

三：将地址复制，使用二维码转换工具（草料二维码）生成二维码后，使用手机扫描，就可以查看了。

注意：扫描后浏览器会有一个提示信息，点击访问原网页就行了

## Mint-UI修改.babelIrc报错-- Error: Couldn't find preset "es2015" relative to directory "E:/Project\_Documents/Test/js"

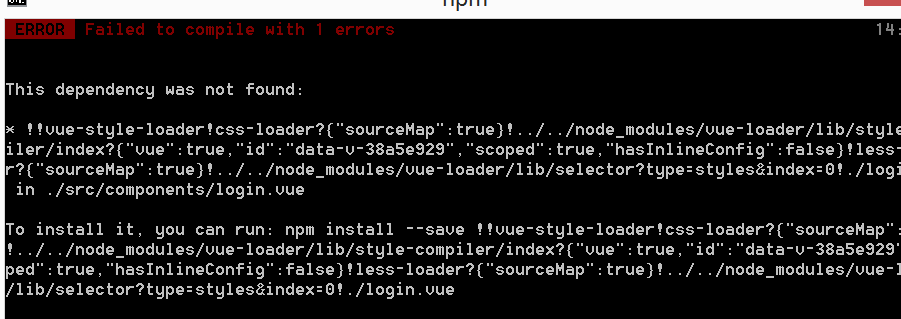
* npm install --save-dev babel-preset-es2015
* npm

## 使用mint-ui Navbar遇到的一个小坑 首次加载tab-container不显示

在绑定的id用v-bind(:)绑定到数据模型

<https://blog.csdn.net/lzzattack03x/article/details/79082382>

## this dependency was not found---组件引入报错



**解决**：未安装css预处理器-less

# [VUE2.0不可忽视的很多变化](http://www.cnblogs.com/dupd/p/5904109.html)

今天使用webpack-sample初始一个vue-cli项目，在app.vue文件中添加了个钩子函数ready，可是ready内的事件一直不执行，检查了webpack文件和package.json也没发现什么问题，浏览器也没报错或者提示，很令人捉急。然后去github看了webpack-simple源码，才发现原来vue init webpack-simple默认安装的vue是2.0版本。马上推测到是不是因为vue2.0废弃了ready的用法，果然不出所料，我真是太机智了。看了下vue2.0的英文的英文文档，发现相对于vue1.0有很多变化，而且变化不是一般的大。。。。如果一旦项目要升级到vue2.0，那么很多页面要整改，很多组件都不能用了，所以目前的项目打死不能升级。可以通过vue init webpack-simple#1.0安装基于vue1.0版本的vue-cli。

下边来分析下vue2.0几个重要的与自己目前项目相关的变化，纯粹是个人理解，权当了解.

　　详细变化请见官方文档分析[https://github.com/vuejs/vue/issues/2873](https://github.com/vuejs/vue/issues/2873" \t "_blank)

　　vue2.0中文官方文档已发布：[http://vuefe.cn/guide/](http://vuefe.cn/guide/" \t "_blank)

1.目前的项目几乎每个页面都用到了1.0的ready钩子函数，然而2.0已废弃不用，进而使用mounted替换，同时还新增了beforeMount、beforeUpdate、updated等，私以为越来越向react看齐了有木有。。

beforeUpdate的作用是在页面初始渲染视图之后，一旦监测到data发生变化，在变化的数据重新渲染视图之前会触发beforeUpdate钩子函数，这也是重新渲染之前最后修改数据的机会

updated的作用则是在data发生变化渲染更新视图之后触发。

2.同时废弃的还有events、$dispatch、$broadcast，官方推荐使用vuex或者全局的事件驱动，然而废弃的这些方法在vux UI框架中很多地方都有使用，无疑在项目中用到它的地方在2.0版本都会不起作用，甚至会报错。

3.v-ref、v-el 弃用 统一使用ref属性为元素或组件添加标记，然后通过this.$refs获取

例如<p ref="a"></p>   获取方法为this.$refs.a 对于自定义组件同样适用

4.$els 是用来获取元素DOM对象，这个也废弃不用，$refs可以起到替代性作用。

5.v-for循环中常用的$index、$key也已不支持使用，trackby被key属性替换。

6.自定义组件中的partial，弃用，这个一直没用到

7.新增 v-once指令

8.新增 propsData

9.新增 render

# 防止刷新vuex的state丢失

在app.vue写上

created() {  
 this.loadToken()  
 if (***sessionStorage***.getItem('store')) {  
 this.$store.replaceState(***Object***.assign({}, this.$store.state, ***JSON***.parse(***sessionStorage***.getItem('store'))))  
 ***sessionStorage***.removeItem('store')  
 }  
},  
mounted () {  
 ***window***.addEventListener('beforeunload', () => {  
 ***sessionStorage***.setItem('store', ***JSON***.stringify(this.$store.state))  
 })  
},