目录

[一、Vue起步 4](#_Toc797811)

[（一）每个 Vue 应用都需要通过实例化 Vue 来实现。 4](#_Toc797812)

[（二）插值 4](#_Toc797813)

[（三）过滤器：如果有runoob这么一条信息，我想将首字母变成大写 4](#_Toc797814)

[（四）缩写 5](#_Toc797815)

[（五）循环 5](#_Toc797816)

[（六）Vue.js 计算属性computed 5](#_Toc797817)

[（七）Vue.js 监听属性 6](#_Toc797818)

[1、加1计数器 6](#_Toc797819)

[2、千米和米转化 7](#_Toc797820)

[3、进入页面就监听： 7](#_Toc797821)

[4、深度监听 8](#_Toc797822)

[5、注销watch 9](#_Toc797823)

[（八）Vue.js 样式绑定 10](#_Toc797824)

[（九）Vue.js 事件处理器 11](#_Toc797825)

[（十）Vue.js 表单 13](#_Toc797826)

[二、Vue.js 组件 16](#_Toc797827)

[1、components 16](#_Toc797828)

[2、Prop 17](#_Toc797829)

[3、Vue.js 自定义指令 19](#_Toc797830)

[4、钩子 20](#_Toc797831)

[5、Vue.js 路由：Vue.js 路由允许我们通过不同的 URL 访问不同的内容。 21](#_Toc797832)

[三、Vue.js 过渡 & 动画 24](#_Toc797833)

[1、过渡： 24](#_Toc797834)

[2、CSS 过渡：通常我们都使用 CSS 过渡来实现效果。 25](#_Toc797835)

[3、CSS 动画： 25](#_Toc797836)

[4、自定义过渡的类名 26](#_Toc797837)

[5、同时使用过渡和动画 26](#_Toc797838)

[6、显性的过渡持续时间 27](#_Toc797839)

[7、JavaScript 钩子 27](#_Toc797840)

[四、Vue.js 混入 29](#_Toc797841)

[1、实例： 29](#_Toc797842)

[2、选项合并 29](#_Toc797843)

[3、全局混入 31](#_Toc797844)

[五、json只能get提交$http数据 32](#_Toc797845)

[六、post和get提交 33](#_Toc797846)

[七、passwordFlag：第一次进入文本框不显示错误信息 35](#_Toc797847)

[八、router之redirect使用 36](#_Toc797848)

[九、to={path''''}使用路由点击图片跳转到首页 37](#_Toc797849)

[十、router-link使用 38](#_Toc797850)

[十一、transparent实现三角形箭头 40](#_Toc797851)

[十二、props使用：子组件接收父组件传的数据 41](#_Toc797852)

[十三、lodash做多选组件 43](#_Toc797853)

[十四、keyup鼠标事件 46](#_Toc797854)

[十五、数量选择组件（有最大最小限制） 47](#_Toc797855)

[十六、选择数量之后将价格返回到界面上 50](#_Toc797856)

[十七、VScode快捷键 53](#_Toc797857)

[**主命令框** 53](#_Toc797858)

[**常用快捷键** 53](#_Toc797859)

[**其他** 54](#_Toc797860)

[**修改默认快捷键** 55](#_Toc797861)

[十八、点击购买显示购买购买弹窗包含一个购买信息表格 57](#_Toc797862)

[十九、vue-date-picker时间函数引入 61](#_Toc797863)

[二十、在json中读取数据存放到表格当中 62](#_Toc797864)

[二十一、this.$route.path点击对应文字切换图片this.$route.push点击关闭跳转页面 66](#_Toc797865)

[二十二、eventBus和stopPropagation阻止时间冒泡 67](#_Toc797866)

[一、一个selection组件的情况 67](#_Toc797867)

[二、两个selection组件的情况 67](#_Toc797868)

[二十三、Vue基本组件构成 69](#_Toc797869)

[二十四、Vue中引入jQuery 71](#_Toc797870)

[二十五、Vue之css样式 73](#_Toc797871)

[**椭圆半径** 73](#_Toc797872)

[二十六、前台直接跳转后台 76](#_Toc797873)

[二十七、1变一 11变十一 78](#_Toc797874)

[二十八、文字超长显示问题 81](#_Toc797875)

[二十九、返回主页面的时候保存页面上的纠违或者其他未保存的信息 81](#_Toc797876)

[三十、防止重复增加数据 88](#_Toc797877)

[三十一、手动增加按钮鼠标划过变小手 90](#_Toc797878)

[三十二、点击空白不可关闭组件 91](#_Toc797879)

[三十三、只在某个指定的页面显示某控件 91](#_Toc797880)

[三十四：样式子、孙css 92](#_Toc797881)

[三十五:字母和数字也自动换行 92](#_Toc797882)

[三十六：在一个vue页面的方法里面调用另外一个vue页面的方法 93](#_Toc797883)

[37、点击对应地方滚动条发生改变 95](#_Toc797884)

[38、js中的join(),reverse()与 split()函数用法解析 95](#_Toc797885)

[39、用key管理可复用元素 96](#_Toc797886)

[40、v-show  & v-if 97](#_Toc797887)

src文件夹：{

assets: 放置一些图片，如logo等。

components: 目录里面放了一个组件文件，可以不用。

App.vue: 项目入口文件，我们也可以直接将组件写这里，而不使用 components 目录。

main.js: 项目的核心文件

}

static 静态资源目录，如图片、字体等。

test 初始测试目录，可删除

index.html 首页入口文件，你可以添加一些 meta 信息或统计代码啥的。

package.json 项目配置文件。

README.md 项目的说明文档，markdown 格式

# 一、Vue起步

## （一）每个 Vue 应用都需要通过实例化 Vue 来实现。

语法格式如下：

var vm = new Vue({

// 选项

})

1、Vue 构造器中有一个el 参数，它是 DOM 元素中的 id。

2、methods 用于定义的函数，可以通过 return 来返回函数值。

## （二）插值

1、文本插值：数据绑定最常见的形式就是使用 {{...}}（双大括号）的文本插值：

2、使用 v-html 指令用于输出 html 代码：

反转字符转：message.split('').reverse().join('') RUNOOB=>BOONUR

## （三）过滤器：如果有runoob这么一条信息，我想将首字母变成大写

<div id="app">

{{ message | capitalize }}

</div>

data: {

message: 'runoob1'

},

filters: {

capitalize: function (value) {

if (!value) return ''

value = value.toString()

alert(value.slice(1))

return value.charAt(0).toUpperCase() + value.slice(1)

}

}

过滤器可以串联：

{{ message | filterA | filterB }}

过滤器是 JavaScript 函数，因此可以接受参数：

{{ message | filterA('arg1', arg2) }}

## （四）缩写

v-bind 缩写 ：

v-on 缩写 @

## （五）循环

循环集合：(item,index) in list {{item.name}} {{item.value}} {{index}}

list:[

{name:"1",value:"a1"},

{name:"2",value:"a2"},

{name:"3",value:"a3"}

]

循环对象三种格式：

0、 <li v-for="value in object"> {{ value }}

1、<li v-for="(value, key) in object"> {{ key }} : {{ value }}

2、<li v-for="(value, key, index) in object"> {{ index }}. {{ key }} : {{ value }}

object: {

name: '菜鸟教程',

url: 'http://www.runoob.com',

slogan: '学的不仅是技术，更是梦想！'

}

v-for 迭代整数

<li v-for="n in 10"> {{ n }}

## （六）Vue.js 计算属性computed

<p>原始字符串: {{ message }}</p>

<p>计算后反转字符串: {{ reversedMessage }}</p>

message: 'Runoob!'

1、computed getter: {

// 计算属性的 getter

reversedMessage: function () {

// `this` 指向 vm 实例

return this.message.split('').reverse().join('')

}

computed vs methods

我们可以使用 methods 来替代 computed，效果上两个都是一样的，但是 computed 是基于它的依赖缓存，

只有相关依赖发生改变时才会重新取值。而使用 methods ，在重新渲染的时候，函数总会重新调用执行。

可以说使用 computed 性能会更好，但是如果你不希望缓存，你可以使用 methods 属性。

2、computed setter：

data: {

name: 'Google',

url: 'http://www.google.com'

},

computed: {

site: {

// getter

get: function () {

return this.name + ' ' + this.url

},

// setter

set: function (newValue) {

var names = newValue.split(' ')

this.name = names[0]

this.url = names[names.length - 1]

}

})

// 调用 setter， vm.name 和 vm.url 也会被对应更新

vm.site = '菜鸟教程 http://www.runoob.com';

document.write('name: ' + vm.name);

document.write('<br>');

document.write('url: ' + vm.url);

## （七）Vue.js 监听属性

watch，我们可以通过 watch 来响应数据的变化。

### 1、加1计数器

<p style = "font-size:25px;">计数器: {{ counter }}</p>

<button @click = "counter++" style = "font-size:25px;">点我</button>

data: {

counter: 1

}

vm.$watch('counter', function(nval, oval) {

alert('计数器值的变化 :' + oval + ' 变为 ' + nval + '!');

});

oval:以前的值 nval:当前的值

### 2、千米和米转化

<div id = "computed\_props">

千米 : <input type = "text" v-model = "kilometers">

米 : <input type = "text" v-model = "meters">

</div>

<p id="info"></p>

data: {

kilometers : 0,

meters:0

},

watch : {

kilometers:function(val) {

this.kilometers = val;

this.meters = val \* 1000;

},

meters : function (val) {

this.kilometers = val/ 1000;

this.meters = val;

}

}

});

// $watch 是一个实例方法

vm.$watch('kilometers', function (newValue, oldValue) {

// 这个回调将在 vm.kilometers 改变后调用

document.getElementById ("info").innerHTML = "修改前值为: " + oldValue + "，修改后值为: " + newValue;

})

### 3、进入页面就监听：

<template>

<div class="hello">

<button @click="clickButton($event)" class="hahahah">点击</button>

<p>FullName: {{fullName}}</p>

<p>FirstName: <input type="text" v-model="firstName"></p>

</div>

</template>

<script>

export default {

name: 'HelloWorld',

data () {

return {

msg: 'Welcome to Your Vue.js App',

firstName: 'Dawei',

lastName: 'Lou',

fullName: ''

}

},

methods: {

clickButton(event){

console.log(event.path[0]);

}

},

watch: {

firstName: {

handler(newName, oldName) {

this.fullName = newName + ' ' + this.lastName;

},

// 代表在wacth里声明了firstName这个方法之后立即先去执行handler方法

immediate: true

}

}

}

</script>

<style scoped>

</style>

### 4、深度监听

<template>

<div class="hello">

<p>obj.a: {{obj.a}}</p>

<p>obj.a: <input type="text" v-model="obj.a"></p>

</div>

</template>

<script>

export default {

name: 'HelloWorld',

// mounted() {

// this.obj = {

// a:'090'

// }

// },

data () {

return {

obj: {

a: 123

}

}

},

methods: {

clickButton(event){

console.log(event.path[0]);

}

},

watch: {

'obj.a': {

handler(newName, oldName) {

console.log('obj.a changed');

},

immediate: true,

// deep: true （用字符串指定要要监听的内容或者deep）

deep的意思就是深入观察，[监听器](https://www.baidu.com/s?wd=%E7%9B%91%E5%90%AC%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)会一层层的往下遍历，给对象的所有属性都加上这个监听器，但是这样性能开销就会非常大了，任何修改obj里面任何一个属性都会触发这个监听器里的 handler。

}

},

}

### 5、注销watch

但是，如果我们使用下面这样的方式写 watch，那么就要手动注销了，这种注销其实也很简单

1. const unWatch = app.$watch('text', (newVal, oldVal) => {
2. console.log(`${newVal} : ${oldVal}`);
3. })
5. unWatch(); *// 手动注销watch*

app.$watch调用后会返回一个值，就是unWatch方法，你要注销 watch 只要调用unWatch方法就可以了。

## （八）Vue.js 样式绑定

1、class 属性绑定

a、单一绑定

<div v-bind:class="{ active: isActive }"></div>

b、在对象中传入更多属性用来动态切换多个 class 。

v-bind:class="{ active: isActive, 'text-danger': hasError }">

c、直接绑定数据里的一个对象：

<div v-bind:class="classObject"></div>

classObject: {

active: true,

'text-danger': false

}

d、绑定返回对象的计算属性。这是一个常用且强大的模式：

<div v-bind:class="classObject"></div>

data: {

isActive: true,

error: null

},

computed: {

classObject: function () {

return {

active: this.isActive && !this.error,

'text-danger': this.error && this.error.type === 'fatal',

}

e、可以把一个数组传给 v-bind:class

<div v-bind:class="[activeClass, errorClass]"></div>

data: {

activeClass: 'active',

errorClass: 'text-danger'

}

f、errorClass 是始终存在的，isActive 为 true 时添加 activeClass 类：

<div v-bind:class="[errorClass ,isActive ? activeClass : '']"></div>

data: {

isActive: true,

activeClass: 'active',

errorClass: 'text-danger'

}

2、Vue.js style(内联样式)

a、 v-bind:style 直接设置样式

<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }">菜鸟教程</div>

data: {

activeColor: 'green',

fontSize: 30

}

b、直接绑定到一个样式对象，让模板更清晰：

<div v-bind:style="styleObject">菜鸟教程</div>

data: {

styleObject: {

color: 'green',

fontSize: '30px'

}

}

c、使用数组将多个样式对象应用到一个元素上：

<div v-bind:style="[baseStyles, overridingStyles]">菜鸟教程</div>

data: {

baseStyles: {

color: 'green',

fontSize: '30px'

},

overridingStyles: {

'font-weight': 'bold' （加粗）

}

}

d、使按钮失效

<button v-bind:disabled="someDynamicCondition">Button</button>

someDynamicCondition：（失效） true '1' 'xdd'

（有效） false

## （九）Vue.js 事件处理器

1、事件监听可以使用 v-on 指令：

<button v-on:click="counter += 1">增加 1</button>

<p>这个按钮被点击了 {{ counter }} 次。</p>

data: {

counter: 0

}

2、v-on 可以接收一个定义的方法来调用。

<button v-on:click="greet">Greet</button>

data: {

name: 'Vue.js'

},

// 在 `methods` 对象中定义方法

methods: {

greet: function (event) {

// `this` 在方法里指当前 Vue 实例

alert('Hello ' + this.name + '!')

// `event` 是原生 DOM 事件

if (event) {

alert(event.target.tagName) alert(BUTTON)组件的名称

}

// 也可以用 JavaScript 直接调用方法

app.greet() // -> 'Hello Vue.js!'

3、用内联 JavaScript 语句：

<button v-on:click="say('hi')">Say hi</button>

<button v-on:click="say('what')">Say what</button>

methods: {

say: function (message) {

alert(message)

}

4、事件修饰符：Vue.js 为 v-on 提供了事件修饰符来处理 DOM 事件细节

<!-- 阻止单击事件冒泡 -->

<a v-on:click.stop="doThis"></a>

<!-- 提交事件不再重载页面 -->

<form v-on:submit.prevent="onSubmit"></form>

<!-- 修饰符可以串联 -->

<a v-on:click.stop.prevent="doThat"></a>

<!-- 只有修饰符 -->

<form v-on:submit.prevent></form>

<!-- 添加事件侦听器时使用事件捕获模式 -->

<div v-on:click.capture="doThis">...</div>

<!-- 只当事件在该元素本身（而不是子元素）触发时触发回调 -->

<div v-on:click.self="doThat">...</div>

<!-- click 事件只能点击一次，2.1.4版本新增 -->

<a v-on:click.once="doThis"></a>

5、按键修饰符：Vue 允许为 v-on 在监听键盘事件时添加按键修饰符：

（1）、<!-- 只有在 keyCode 是 13 时调用 vm.submit() -->

<input v-on:keyup.13="submit">

（2）、记住所有的 keyCode 比较困难，所以 Vue 为最常用的按键提供了别名：

<input v-on:keyup.enter="submit">

<!-- 缩写语法 -->

<input @keyup.enter="submit">

（3）、全部的按键别名：

.enter

.tab

.delete (捕获 "删除" 和 "退格" 键)

.esc

.space

.up

.down

.left

.right

.ctrl

.alt

.shift

.meta

实例：

<p><!-- Alt + C -->

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

<!-- Ctrl + Click -->

<div @click.ctrl="doSomething">Do something</div>

## （十）Vue.js 表单

（1）、用 v-model 指令在表单控件元素上创建双向数据绑定。

1、input框：<input v-model="message" placeholder="编辑我……">

2、textarea: <textarea v-model="message2" placeholder="多行文本输入……"></textarea>

3、复选框

a、单个复选框

<input type="checkbox" id="checkbox" v-model="checked"> 没选中为false,选中为true

b、多个复选框

<input type="checkbox" id="runoob" value="Runoob" v-model="checkedNames">

<label for="runoob">Runoob</label>

<input type="checkbox" id="google" value="Google" v-model="checkedNames">

<label for="google">Google</label>

<input type="checkbox" id="taobao" value="Taobao" v-model="checkedNames">

<label for="taobao">taobao</label>

data: {

checked : false,

checkedNames: []

}

c、单选按钮

<input type="radio" id="runoob" value="Runoob1" v-model="picked">

<label for="runoob">Runoob</label>

<input type="radio" id="google" value="Google" v-model="picked">

<label for="google">Google</label>

<span>选中值为: {{ picked }}</span>

data: {

picked : 'Runoob1' 对应的value

}

d、select 列表

<select v-model="selected" name="fruit">

<option value="">选择一个网站</option>

<option value="www.runoob.com">Runoob</option>

<option value="www.google.com">Google</option>

</select>

data: {

selected: ''

}

（2）、修饰符

1、.lazy

<!-- 在 "change" 而不是 "input" 事件中更新 -->

<input v-model.lazy="msg" > 文本框的值输入完成鼠标退出才有效

2、.number

如果想自动将用户的输入值转为 Number 类型（如果原值的转换结果为 NaN 则返回原值），

可以添加一个修饰符 number 给 v-model 来处理输入值：

<input v-model.number="age" type="number"> 只能输入number类型，其他类型不可输入

3、.trim 如果要自动过滤用户输入的首尾空格，可以添加 trim 修饰符到 v-model 上过滤输入：

<input v-model.trim="msg">

4、全选与取消全选（练习）

<input type="checkbox" id="checkbox" v-model="checked" @change="changeAllChecked()">

<label for="checkbox">

{{checked}}

</label>

<p>

多个复选框：

</p>

<input type="checkbox" id="runoob" value="Runoob" v-model="checkedNames">

<label for="runoob">

Runoob

</label>

<input type="checkbox" id="google" value="Google" v-model="checkedNames">

<label for="google">

Google

</label>

<input type="checkbox" id="taobao" value="Taobao" v-model="checkedNames">

<label for="taobao">

taobao

</label>

<br>

<span>

选择的值为:{{checkedNames}}

</span>

data: {

checked: false,

checkedNames: [],

checkedArr: ["Runoob", "Taobao", "Google"]

},

methods: {

changeAllChecked: function() {

if (this.checked) {

this.checkedNames = this.checkedArr

} else {

this.checkedNames = []

}

}

},

# 二、Vue.js 组件

组件可以扩展 HTML 元素，封装可重用的代码。

注册一个全局组件语法格式如下：Vue.component(tagName, options)

tagName 为组件名，options 为配置选项。注册后，我们可以使用以下方式来调用组件：<tagName></tagName>

## 1、components

（1）、全局组件:所有实例都能用全局组件。

<div id="app">

<runoob></runoob>

</div>

<script>

// 注册

Vue.component('runoob', {

template: '<h1>自定义组件!</h1>'

})

// 创建根实例

new Vue({

el: '#app'

})

</script>

（2）、局部组件：我们也可以在实例选项中注册局部组件，这样组件只能在这个实例中使用：

<div id="app">

<runoob></runoob>

</div>

<script>

var Child = {

template: '<h1>自定义组件!</h1>'

}

// 创建根实例

new Vue({

el: '#app',

components: {

// <runoob> 将只在父模板可用

'runoob': Child

}

})

</script>

## 2、Prop

prop 是父组件用来传递数据的一个自定义属性。父组件的数据需要通过 props 把数据传给子组件，子组件需要显式地用 props 选项声明 "prop"：

<div id="app">

<child message="hello!"></child>

</div>

// 注册

Vue.component('child', {

// 声明 props

props: ['message'],

// 同样也可以在 vm 实例中像 “this.message” 这样使用

template: '<span>{{ message }}</span>'

})

（1）、动态 Prop：类似于用 v-bind 绑定 HTML 特性到一个表达式，也可以用 v-bind 动态绑定 props 的值到父组件的数据中。

每当父组件的数据变化时，该变化也会传导给子组件：

a、<input v-model="parentMsg">

<child v-bind:message="parentMsg"></child>

// 注册

Vue.component('child', {

// 声明 props

props: ['message'],

// 同样也可以在 vm 实例中像 "this.message" 这样使用

template: '<span>{{ message }}</span>'

})

parentMsg: '父组件内容'

b、将 v-bind 指令将 todo 传到每一个重复的组件中：

<ol>

<todo-item v-for="item in sites" v-bind:todo="item"></todo-item>

</ol>

Vue.component('todo-item', {

props: ['todo'],

template: '<li>{{ todo.text }}</li>'

})

data: sites: [

{ text: 'Runoob' },

{ text: 'Google' },

{ text: 'Taobao' }

]

注意: prop 是单向绑定的：当父组件的属性变化时，将传导给子组件，但是不会反过来。

（2）、Prop 验证

组件可以为 props 指定验证要求。

prop 是一个对象而不是字符串数组时，它包含验证要求：

Vue.component('example', {

props: {

// 基础类型检测 （`null` 意思是任何类型都可以）

propA: Number,

// 多种类型

propB: [String, Number],

// 必传且是字符串

propC: {

type: String,

required: true

},

// 数字，有默认值

propD: {

type: Number,

default: 100

},

// 数组／对象的默认值应当由一个工厂函数返回

propE: {

type: Object,

default: function () {

return { message: 'hello' }

}

},

// 自定义验证函数

propF: {

validator: function (value) {

return value > 10

}

}

type 可以是下面原生构造器：

String

Number

Boolean

Function

Object

Array

type 也可以是一个自定义构造器，使用 instanceof 检测。

（3）、自定义事件

父组件是使用 props 传递数据给子组件，但如果子组件要把数据传递回去，就需要使用自定义事件！

我们可以使用 v-on 绑定自定义事件, 每个 Vue 实例都实现了事件接口(Events interface)，即：

使用 $on(eventName) 监听事件

使用 $emit(eventName) 触发事件

另外，父组件可以在使用子组件的地方直接用 v-on 来监听子组件触发的事件。

以下实例中子组件已经和它外部完全解耦了。它所做的只是触发一个父组件关心的内部事件。

<p>{{ total }}</p>

<button-counter v-on:increment="incrementTotal"></button-counter>

<button-counter v-on:increment="incrementTotal"></button-counter>（监听事件）

Vue.component('button-counter', {

template: '<button v-on:click="incrementHandler">{{ counter }}</button>',

data: function () {

return {

counter: 0

}

},

methods: {

incrementHandler: function () {

this.counter += 1

this.$emit('increment') （触发事件）

}

},

})

new Vue({

el: '#counter-event-example',

data: {

total: 0

},

methods: {

incrementTotal: function () {

this.total += 1

}

}

## 3、Vue.js 自定义指令

（1）、 a、注册全局指令 v-focus, 该指令的功能是在页面加载时，元素获得焦点

<input v-focus>

// 注册一个全局自定义指令 v-focus

Vue.directive('focus', {

// 当绑定元素插入到 DOM 中。

inserted: function (el) {

// 聚焦元素

el.focus()

}

})

b、使用 directives 选项来注册局部指令

<input v-focus>

new Vue({

el: '#app',

directives: {

// 注册一个局部的自定义指令 v-focus

focus: {

// 指令的定义

inserted: function (el) {

// 聚焦元素

el.focus()

}

}

## 4、钩子

（1）、钩子函数

指令定义函数提供了几个钩子函数（可选）：

bind: 只调用一次，指令第一次绑定到元素时调用，用这个钩子函数可以定义一个在绑定时执行一次的初始化动作。

inserted: 被绑定元素插入父节点时调用（父节点存在即可调用，不必存在于 document 中）。

update: 被绑定元素所在的模板更新时调用，而不论绑定值是否变化。通过比较更新前后的绑定值，可以忽略不必要的模板更新（详细的钩子函数参数见下）。

componentUpdated: 被绑定元素所在模板完成一次更新周期时调用。

unbind: 只调用一次， 指令与元素解绑时调用。

例子1：

<div id="app" v-runoob:hello.a.b="message">

Vue.directive('runoob', {

bind: function (el, binding, vnode) {

var s = JSON.stringify

el.innerHTML =

'name: ' + s(binding.name) + '<br>' +

'value: ' + s(binding.value) + '<br>' +

'expression: ' + s(binding.expression) + '<br>' +

'argument: ' + s(binding.arg) + '<br>' +

'modifiers: ' + s(binding.modifiers) + '<br>' +

'vnode keys: ' + Object.keys(vnode).join(', ')

}

})

data: {

message: '菜鸟教程!'

}

结果：

name: "runoob"

value: "菜鸟教程!"

expression: "message"

argument: "hello"

modifiers: {"a":true,"b":true}

vnode keys: tag, data, children, text, elm, ns, context, functionalContext, key, componentOptions,

componentInstance, parent, raw, isStatic, isRootInsert, isComment, isCloned, isOnce

例子2：

Vue.directive('runoob', function (el, binding) {

// 设置指令的背景颜色

el.style.backgroundColor = binding.value.color

})

例子3：指令函数可接受所有合法的 JavaScript 表达式，以下实例传入了 JavaScript 对象：

<div v-runoob="{ color: 'green', text: '菜鸟教程!' }"></div>

Vue.directive('runoob', function (el, binding) {

// 简写方式设置文本及背景颜色

el.innerHTML = binding.value.text

el.style.backgroundColor = binding.value.color

})

## 5、Vue.js 路由：Vue.js 路由允许我们通过不同的 URL 访问不同的内容。

安装路由：npm install vue-router --save

(1)、简单实例

<!-- 使用 router-link 组件来导航. -->

<!-- 通过传入 `to` 属性指定链接. -->

<!-- <router-link> 默认会被渲染成一个 `<a>` 标签 -->

<router-link to="/foo">Go to Foo</router-link>

<router-link to="/bar">Go to Bar</router-link>

<!-- 路由出口 -->

<!-- 路由匹配到的组件将渲染在这里 -->

<router-view></router-view>

// 0. 如果使用模块化机制编程，導入Vue和VueRouter，要调用 Vue.use(VueRouter)

​

// 1. 定义（路由）组件。

// 可以从其他文件 import 进来

const Foo = { template: '<div>foo</div>' }

const Bar = { template: '<div>bar</div>' }

​

// 2. 定义路由

// 每个路由应该映射一个组件。 其中"component" 可以是

// 通过 Vue.extend() 创建的组件构造器，

// 或者，只是一个组件配置对象。

// 我们晚点再讨论嵌套路由。

const routes = [

{ path: '/foo', component: Foo },

{ path: '/bar', component: Bar }

]

​

// 3. 创建 router 实例，然后传 `routes` 配置

// 你还可以传别的配置参数, 不过先这么简单着吧。

const router = new VueRouter({

routes // （缩写）相当于 routes: routes

})

​

// 4. 创建和挂载根实例。

// 记得要通过 router 配置参数注入路由，

// 从而让整个应用都有路由功能

const app = new Vue({

router

}).$mount('#app')

​

// 现在，应用已经启动了！

(2)、<router-link> 相关属性

1、to:表示目标路由的链接。 当被点击后，内部会立刻把 to 的值传到 router.push()，所以这个值可以是一个字符串或者是描述目标位置的对象。

<!-- 字符串 -->

<router-link to="home">Home</router-link>

<!-- 渲染结果 -->

<a href="home">Home</a>

<!-- 使用 v-bind 的 JS 表达式 -->

<router-link v-bind:to="'home'">Home</router-link>

<!-- 不写 v-bind 也可以，就像绑定别的属性一样 -->

<router-link :to="'home'">Home</router-link>

<!-- 同上 -->

<router-link :to="{ path: 'home' }">Home</router-link>

<!-- 命名的路由 -->

<router-link :to="{ name: 'user', params: { userId: 123 }}">User</router-link>

<!-- 带查询参数，下面的结果为 /register?plan=private -->

<router-link :to="{ path: 'register', query: { plan: 'private' }}">Register</router-link>

2、replace：设置 replace 属性的话，当点击时，会调用 router.replace() 而不是 router.push()，导航后不会留下 history 记录。

<router-link :to="{ path: '/abc'}" replace></router-link>

3、append：设置 append 属性后，则在当前 (相对) 路径前添加基路径。例如，我们从 /a 导航到一个相对路径 b，如果没有配置 append，则路径为 /b，如果配了，则为 /a/b

<router-link :to="{ path: 'relative/path'}" append></router-link>

4、tag:有时候想要 <router-link> 渲染成某种标签，例如 <li>。 于是我们使用 tag prop 类指定何种标签，同样它还是会监听点击，触发导航。

<router-link to="/foo" tag="li">foo</router-link>

<!-- 渲染结果 -->

<li>foo</li>

5、active-class：设置 链接激活时使用的 CSS 类名。可以通过以下代码来替代。

<style>

.\_active{

background-color : red;

}

</style>

<p>

<router-link v-bind:to = "{ path: '/route1'}" active-class = "\_active">Router Link 1</router-link> 注意这里 class 使用 active\_class="\_active"。

<router-link v-bind:to = "{ path: '/route2'}" tag = "span">Router Link 2</router-link>

</p>

6、exact-active-class：配置当链接被精确匹配的时候应该激活的 class。可以通过以下代码来替代。

<p>

<router-link v-bind:to = "{ path: '/route1'}" exact-active-class = "\_active">Router Link 1</router-link>

<router-link v-bind:to = "{ path: '/route2'}" tag = "span">Router Link 2</router-link>

</p>

7、event：声明可以用来触发导航的事件。可以是一个字符串或是一个包含字符串的数组。

<router-link v-bind:to = "{ path: '/route1'}" event = "mouseover">Router Link 1</router-link>

以上代码设置了 event 为 mouseover ，及在鼠标移动到 Router Link 1 上时导航的 HTML 内容会发生改变。

# 三、Vue.js 过渡 & 动画

1、过渡：Vue 在插入、更新或者移除 DOM 时，提供多种不同方式的应用过渡效果。Vue 提供了内置的过渡封装组件，该组件用于包裹要实现过渡效果的组件。

（1）语法格式

<transition name = "nameoftransition">

<div></div>

</transition>

例子：

<button v-on:click = "show = !show">点我</button>

<transition name = "fade">

<p v-show = "show" v-bind:style = "styleobj">动画实例</p>

</transition>

<style>

/\* 可以设置不同的进入和离开动画 \*/

/\* 设置持续时间和动画函数 \*/

.fade-enter-active, .fade-leave-active {

transition: opacity 2s

}

.fade-enter, .fade-leave-to /\* .fade-leave-active, 2.1.8 版本以下 \*/ {

opacity: 0

}

</style>

data: {

show:true,

styleobj :{

fontSize:'30px',

color:'red'

}

},

（2）过渡其实就是一个淡入淡出的效果。Vue在元素显示与隐藏的过渡中，提供了 6 个 class 来切换：

v-enter：定义进入过渡的开始状态。在元素被插入之前生效，在元素被插入之后的下一帧移除。opacity:0

v-enter-active：定义进入过渡生效时的状态。在整个进入过渡的阶段中应用，在元素被插入之前生效，在过渡/动画完成之后移除。这个类可以被用来定义进入过渡的过程时间，延迟和曲线函数。

v-enter-to: 2.1.8版及以上 定义进入过渡的结束状态。在元素被插入之后下一帧生效 (与此同时 v-enter 被移除)，在过渡/动画完成之后移除。 opacity:1

v-leave: 定义离开过渡的开始状态。在离开过渡被触发时立刻生效，下一帧被移除。 opacity:1

v-leave-active：定义离开过渡生效时的状态。在整个离开过渡的阶段中应用，在离开过渡被触发时立刻生效，在过渡/动画完成之后移除。这个类可以被用来定义离开过渡的过程时间，延迟和曲线函数。

v-leave-to: 2.1.8版及以上 定义离开过渡的结束状态。在离开过渡被触发之后下一帧生效 (与此同时 v-leave 被删除)，在过渡/动画完成之后移除。 opacity:0

v-enter-active 和 v-leave-active 可以控制进入/离开过渡的不同的缓和曲线。

对于这些在过渡中切换的类名来说，如果你使用一个没有名字的 <transition>，则 v- 是这些类名的默认前缀。如果你使用了 <transition name="my-transition">，那么 v-enter 会替换为 my-transition-enter。

2、CSS 过渡：通常我们都使用 CSS 过渡来实现效果。

<button v-on:click = "show = !show">点我</button>

<transition name="slide-fade">

<p v-if="show">菜鸟教程</p>

</transition>

<style>

/\* 可以设置不同的进入和离开动画 \*/

/\* 设置持续时间和动画函数 \*/

.slide-fade-enter-active {

transition: all .3s ease;

}

.slide-fade-leave-active {

transition: all .8s cubic-bezier(1.0, 0.5, 0.8, 1.0);

}

.slide-fade-enter, .slide-fade-leave-to

/\* .slide-fade-leave-active 用于 2.1.8 以下版本 \*/ {

transform: translateX(10px);

opacity: 0;

}

</style>

## 3、CSS 动画：

CSS 动画用法类似 CSS 过渡，但是在动画中 v-enter 类名在节点插入 DOM 后不会立即删除，而是在 animationend 事件触发时删除。

<button v-on:click = "show = !show">点我</button>

<transition name="bounce">

<p v-if="show">菜鸟教程 -- 学的不仅是技术，更是梦想！！！</p>

</transition>

<style>

.bounce-enter-active {

animation: bounce-in .5s;

}

.bounce-leave-active {

animation: bounce-in .5s reverse;

}

@keyframes bounce-in {

0% {

transform: scale(0);

}

50% {

transform: scale(1.5);

}

100% {

transform: scale(1);

}

}

</style>

## 4、自定义过渡的类名

自定义过渡的类名

我们可以通过以下特性来自定义过渡类名：

enter-class

enter-active-class

enter-to-class (2.1.8+)

leave-class

leave-active-class

leave-to-class (2.1.8+)

自定义过渡的类名优先级高于普通的类名，这样就能很好的与第三方（如：animate.css）的动画库结合使用。

例子：

<link href="https://cdn.jsdelivr.net/npm/animate.css@3.5.1" rel="stylesheet" type="text/css">

<button v-on:click = "show = !show">点我</button>

<transition

name="custom-classes-transition"

enter-active-class="animated tada"

leave-active-class="animated bounceOutRight"

>

<p v-if="show">菜鸟教程 -- 学的不仅是技术，更是梦想！！！</p>

</transition>

## 5、同时使用过渡和动画

Vue 为了知道过渡的完成，必须设置相应的事件监听器。它可以是 transitionend 或 animationend ，这取决于给元素应用的 CSS 规则。如果你使用其中任何一种，Vue 能自动识别类型并设置监听。

但是，在一些场景中，你需要给同一个元素同时设置两种过渡动效，比如 animation 很快的被触发并完成了，而 transition 效果还没结束。在这种情况中，你就需要使用 type 特性并设置 animation 或 transition 来明确声明你需要 Vue 监听的类型。

## 6、显性的过渡持续时间

在很多情况下，Vue 可以自动得出过渡效果的完成时机。默认情况下，Vue 会等待其在过渡效果的根元素的第一个 transitionend 或 animationend 事件。然而也可以不这样设定——比如，我们可以拥有一个精心编排的一系列过渡效果，其中一些嵌套的内部元素相比于过渡效果的根元素有延迟的或更长的过渡效果。

在这种情况下你可以用 <transition> 组件上的 duration 属性定制一个显性的过渡持续时间 (以毫秒计)：

<transition :duration="1000">...</transition>

你也可以定制进入和移出的持续时间：

<transition :duration="{ enter: 500, leave: 800 }">...</transition>

## 7、JavaScript 钩子

<button v-on:click = "show = !show">点我</button>

<transition

v-on:before-enter="beforeEnter"

v-on:enter="enter"

v-on:leave="leave"

v-bind:css="false"

>

<p v-if="show">菜鸟教程 -- 学的不仅是技术，更是梦想！！！</p>

</transition>

data: {

show: false

},

methods: {

beforeEnter: function (el) {

el.style.opacity = 0

el.style.transformOrigin = 'left'

},

enter: function (el, done) {

Velocity(el, { opacity: 1, fontSize: '1.4em' }, { duration: 300 })

Velocity(el, { fontSize: '1em' }, { complete: done })

},

leave: function (el, done) {

Velocity(el, { translateX: '15px', rotateZ: '50deg' }, { duration: 600 })

Velocity(el, { rotateZ: '100deg' }, { loop: 2 })

Velocity(el, {

rotateZ: '45deg',

translateY: '30px',

translateX: '30px',

opacity: 0

}, { complete: done })

}

# 四、Vue.js 混入

混入 (mixins)定义了一部分可复用的方法或者计算属性。混入对象可以包含任意组件选项。当组件使用混入对象时，所有混入对象的选项将被混入该组件本身的选项。

## 1、实例：

<div id = "databinding"></div>

<script type = "text/javascript">

var vm = new Vue({

el: '#databinding',

data: {

},

methods : {

},

});

// 定义一个混入对象

var myMixin = {

created: function () {

this.startmixin()

},

methods: {

startmixin: function () {

document.write("欢迎来到混入实例");

}

}

};

var Component = Vue.extend({

mixins: [myMixin]

})

var component = new Component();

</script>

## 2、选项合并

当组件和混入对象含有同名选项时，这些选项将以恰当的方式混合。

比如，数据对象在内部会进行浅合并 (一层属性深度)，在和组件的数据发生冲突时以组件数据优先。

以下实例中，Vue 实例与混入对象包含了相同的方法。从输出结果可以看出两个选项合并了。

<div id = "databinding"></div>

<script type = "text/javascript">

var mixin = {

created: function () {

document.write('混入调用' + '<br>')

}

}

new Vue({

mixins: [mixin],

created: function () {

document.write('组件调用' + '<br>')

}

});

</script>

如果 methods 选项中有相同的函数名，则 Vue 实例优先级会较高。如下实例，Vue 实例与混入对象的 methods 选项都包含了相同的函数：

<div id = "databinding"></div>

var mixin = {

methods: {

hellworld: function () {

document.write('HelloWorld 方法' + '<br>');

},

samemethod: function () {

document.write('Mixin：相同方法名' + '<br>');

}

}

};

var vm = new Vue({

mixins: [mixin],

methods: {

start: function () {

document.write('start 方法' + '<br>');

},

samemethod: function () {

document.write('Main：相同方法名' + '<br>');

}

}

});

vm.hellworld();

vm.start();

vm.samemethod();

输出结果：

HelloWorld 方法

start 方法

Main：相同方法名

## 3、全局混入

也可以全局注册混入对象。注意使用！ 一旦使用全局混入对象，将会影响到 所有 之后创建的 Vue 实例。使用恰当时，可以为自定义对象注入处理逻辑。

<script type = "text/javascript">

// 为自定义的选项 'myOption' 注入一个处理器。

Vue.mixin({

created: function () {

var myOption = this.$options.myOption

if (myOption) {

document.write(myOption)

}

}

})

new Vue({

myOption: 'hello!'

})

// => "hello!"

</script>

# 五、json只能get提交$http数据

build\webpack.dev.conf.js:

var jsonServer=require('json-server')

var apiServer=jsonServer.create()

var apiRouter=jsonServer.router('db.json')

var middlewares=jsonServer.defaults()

apiServer.use(middlewares)

apiServer.use('/api',apiRouter)

apiServer.listen(port+1,function(){

console.log('JSON Server is running')

})

config\index.js里:

proxyTable: {

'/api': 'http://localhost:8081/#/'

},

index.Vue里：

export default {

created: function() {

this.$http.get('api/getNewsList')

.then((res)=>{

this.newsList=res.data

},(err)=>{

console.log(err)

})

},

# 六、post和get提交

webpack.dev.conf.js里面代码：：

var express = require('express')

var apiServer = express()

var bodyParser = require('body-parser')

apiServer.use(bodyParser.urlencoded({ extended: true }))

apiServer.use(bodyParser.json())

var apiRouter = express.Router()

var fs = require('fs')

apiRouter.get('/',function(req,res){

res.json({message:"hooray!welcome to your api!"})

});

apiRouter.route('/:apiName')

.all(function(req,res){

fs.readFile('./db.json','utf8',function(err,data){

if(err) throw err

var data=JSON.parse(data)

if(data[req.params.apiName]){

res.json(data[req.params.apiName])

}

else{

res.send("NO SUCH API NAME")

}

})

})

apiServer.use('/api',apiRouter)

port里面：

apiServer.listen(port+1,function(err){

if(err){

console.log(err)

return

}

console.log('Listening at http://localhost:' + (port + 1) + '\n')

})

config\index.js：

将proxyTable{}改为： proxyTable: {

'/api': 'http://localhost:8083'

},

index.Vue里：

export default {

created: function () {

this.$http.post('api/getNewsList')

.then((res) => {

this.newsList = res.data

}, (err) => {

console.log(err)

})

},

created: function () {

this.$http.post('api/getBoardList')

.then((res) => {

this.boardList = res.data

}, (err) => {

console.log(err)

})

},

# 七、passwordFlag：第一次进入文本框不显示错误信息

if(!this.passwordFlag){

errorText=''

this.passwordFlag=true

}

# 八、router之redirect使用

let router=new VueRouter({

mode:'history',

routes:[

{

path:'/',

component:IndexPage

},

{

path:'/detail',

component:DetailPage,

redirect:'/detail/analysis'，

**(页面地址输入：http://localhost:8082/detail)**

**（则自动跳转到：http://localhost:8082/detail/analysis）**

children:[

{

path:'analysis',

component:DetailAnaPage

},

{

path:'count',

component:DetailCouPage

},

{

path:'forecast',

component:DetailForPage

},

{

path:'publish',

component:DetailPubPage

}

]

}

]

})

# 九、to={path''''}使用路由点击图片跳转到首页

<router-link :to="{path:'/'}">

<img src="../assets/logo.png">

</router-link>

# 十、router-link使用

main.js:detail.vue为父组件，其他为子组件

import IndexPage from './pages/index.vue'

import DetailPage from './pages/detail.vue'

import DetailAnaPage from './pages/detail/analysis.vue'

import DetailCouPage from './pages/detail/count.vue'

import DetailForPage from './pages/detail/forecast.vue'

import DetailPubPage from './pages/detail/publish.vue'

let router=new VueRouter({

mode:'history',

routes:[

{

path:'/',

component:IndexPage

},

{

path:'/detail',

component:DetailPage,

children:[

{

path:'analysis',

component:DetailAnaPage

},

{

path:'count',

component:DetailCouPage

},

{

path:'forecast',

component:DetailForPage

},

{

path:'publish',

component:DetailPubPage

}

]

}

]

})

**detail.vue里面：**

<ul>

<router-link v-for="item in products" :to="{path :item.path}" tag="li" active-class="active">

{{item.name}}

</router-link>

</ul>

**1、使用router-link引入子组件**

**2、使用to来跳转到指定的path。path与相同。**

**3、tag的意思为自动生成的什么样式，因为外层是ul所以我们内层是li格式。**

**4、active-class如果没有，系统则自动生成一个router-link-active的名字的class,被激活时候的类名**

products: [

{

name: '数据统计',

path: 'count',

icon: require('../assets/images/1.png'),

active: false

},

{

name: '数据预测',

path: 'forecast',

active: false

},

# 十一、transparent实现三角形箭头

#box{

width: 0px;

height: 0px;

border-top: 40px solid red;

border-right:40px solid transparent;

border-bottom:40px solid transparent;

border-left:40px solid transparent;

}

transparent是设置透明度

# 十二、props使用：子组件接收父组件传的数据

子组件：

export default {

props: {

selections: {

type: Array, "类型是什么"

default: [{

label: 'test',

value: 0

}]

}

},

接收的属性是selections属性，类型是type，有默认值

父组件：

<v-selection :selections="productList" ></v-selection>

.

.

.

<script>

import VSelection from '../../components/selection.vue'

.

export default{

components:{

VSelection

},

数据：

productList: [

{

label: '入门版',

value: 0

},

{

label: '中级版',

value: 1

},

{

label: '高级版',

value: 2

}

]

子组件:

<span>{{ selections[nowIndex].label }}</span>

data () {

return {

isDrop: false,

nowIndex: 0

}

},

# 十三、lodash做多选组件

新建立: multiplyChooser.vue

<div class="chooser-component">

<ul class="chooser-list">

<li

v-for="(item, index) in selections"

@click="toggleSelection(index)"

:title="item.label"

:class="{active: checkActive(index)}" （数组里面如何验证index）

>{{ item.label }}</li>

</ul>

</div>

</div>

<script>

import \_ from 'lodash'（引入lodash的js工具库，缩写为下划线）

export default {

props: {

selections: {

type: Array,

default: [{

label: 'test',

value: 0

}]

}

},

data () {

return {

nowIndexes: [0] 为数组所以可以多项选择

}

},

methods: {

(此方法就是判断当前选择的序列是否在存在的序列里面，如果在就剔除之后加上，如果不在就push进去)

toggleSelection (index) {

if (this.nowIndexes.indexOf(index) === -1) { （===-1说明没有查找到，返回-1）

this.nowIndexes.push(index)

}

else { （lodash的remove方法，删除不在index里面的idx元素）

this.nowIndexes = \_.remove(this.nowIndexes, (idx) => { （引入lodash的js工具库）

return idx !== index

})

}

let nowObjArray = \_.map(this.nowIndexes, (idx) => {

return this.selections[idx]

})

this.$emit('on-change', nowObjArray)

},

checkActive (index) {

return this.nowIndexes.indexOf(index) !== -1

}

}

}

</script>

父组件：

import VMulChooser from '../../components/base/multiplyChooser'

export default {

components: {

VCounter,

VMulChooser

},

<v-mul-chooser :selections="versionList"></v-mul-chooser>

data () {

return {

versionList: [

{

label: '纸质报告',

value: 0

},

{

label: 'pdf',

value: 1

},

{

label: '页面报告',

value: 2

},

{

label: '邮件',

value: 3

}

]

}

}

# 十四、keyup鼠标事件

两个input的效果都是一样的 如果安13  也就是按键 enter   才会执行弹窗！！

@keyup.13 回车

@keyup.enter 回车

@keyup.left 左键

@keyup.right 右键

@keyup.up 上键

@keyup.down 下键

@keyup.delete 删除键

# 十五、数量选择组件（有最大最小限制）

count.vue:

<template>

<div class="counter-component">

<div class="counter-btn" @click="minus"> - </div> (减号加号输入框)

<div class="counter-show">

<input type="text" v-model="number" @keyup="fixNumber">

</div>

<div class="counter-btn" @click="add"> + </div>

</div>

</template>

<script>

export default {

props: {

max: {

type: Number,

default: 5

},

最大最小值通过父组件传递进来

min: {

type: Number,

default: 1

}

},

data () {

return {

number: this.min

}

},

不确定number会在哪里发生改变，使用watch监听

watch: {

number () {

this.$emit('on-change', this.number)

}

},

methods: {

将所有不是数字的字符删掉

fixNumber () {

let fix

if (typeof this.number === 'string') {

fix = Number(this.number.replace(/\D/g, ''))

}

else {

fix = this.number

}

if (fix > this.max || fix < this.min) {

fix = this.min

}

this.number = fix

},

minus () {

if (this.number <= this.min) {

return

}

this.number --

},

add () {

if (this.number >= this.max) {

return

}

this.number ++

}

}

}

</script>

<style scoped>

.counter-component {

position: relative;

display: inline-block;

overflow: hidden;

vertical-align: middle;

}

.counter-show {

float: left;

}

.counter-show input {

border: none;

border-top: 1px solid #e3e3e3;

border-bottom: 1px solid #e3e3e3;

height: 23px;

line-height: 23px;

width: 30px;

outline: none;

text-indent: 4px;

}

.counter-btn {

border: 1px solid #e3e3e3;

float: left;

height: 25px;

line-height: 25px;

width: 25px;

text-align: center;

cursor: pointer;

}

.counter-btn:hover {

border-color: #4fc08d;

background: #4fc08d;

color: #fff;

}

</style>

父组件：

<v-counter :max="300" :min="20"></v-counter>

进行max 和 min 的给子组件传值

# 十六、选择数量之后将价格返回到界面上

思路：

1、首先在后台需要记录一下传过来的值，因为是四项，所以设定四个变量



buyNum: 0, （买多少个）

buyType: {}, （类型，是个对象）

versions: [], （版本，因为可以多选，所以为数组【】）

period: {}, （有效时间，为对象）

2、首在组件发生改变的时候，将改变的值赋给对应的变量

(1) choose.vue:

methods: {

chosenSelection (index) {

this.nowIndex = index

this.$emit('on-change', this.selections[index]) (根据index取出selections的值)

}

}

(2) counter.vue:

watch: {

number () {

this.$emit('on-change', this.number)

}

},

(3) selections.vue:

methods: {

toggleDrop () {

this.isDrop = !this.isDrop

},

chooseSelection (index) {

this.nowIndex = index

this.isDrop = false

this.$emit('on-change', this.selections[this.nowIndex])

}

}

(4) multychooser.vue:

methods: {

toggleSelection (index) {

if (this.nowIndexes.indexOf(index) === -1) {

this.nowIndexes.push(index)

}

else {

this.nowIndexes = \_.remove(this.nowIndexes, (idx) => {

return idx !== index

})

}

因为是数组的集合，多选所以采用lodash。通过map方法给nowIndex做一个映射，对里面的每一项进行改变，把每一项的Index变成了一个从selections里面取的对象

let nowObjArray = \_.map(this.nowIndexes, (idx) => {

return this.selections[idx]

})

this.$emit('on-change', nowObjArray)

},

checkActive (index) {

return this.nowIndexes.indexOf(index) !== -1

}

}

3、想每个组件共享一个方法，onParamChange

<v-counter @on-change="onParamChange('buyNum', $event)"></v-counter>

<v-selection :selections="buyTypes" @on-change="onParamChange('buyType', $**event**)"></v-selection>

<v-chooser :selections="periodList" @on-change="onParamChange('period', $event)"></v-chooser>

<v-mul-chooser :selections="versionList" @on-change="onParamChange('versions', $event)"></v-mul-chooser>

methods: {

onParamChange (attr, val) {

this[attr] = val

this.getPrice()

},

（手动将组件传过来的信息自己手动打进去，用到 $event 事件【用来传递信息】）

4、需要一个方法，发送一个AJAX请求

getPrice () {

let buyVersionsArray = \_.map(this.versions, (item) => {

return item.value （将version通过lodash变成对象的value）

})

let reqParams = {

buyNumber: this.buyNum,

buyType: this.buyType.value, （拼接好请求的参数）

period: this.period.value,

version: buyVersionsArray.join(',')

}

this.$http.post('/api/getPrice', reqParams) （通过一个vue-resourse发送一个post请求，注意是post请求）

.then((res) => {

this.price = res.data.amount

let data= JSON.parse(res.data)

this.price = data.amount （如果后台传递的是字符串，需要进行JSON.parse变成JSON形式）

})

},

5、刚进入界面时候应该显示为多少钱，而不是0元

（1）在页面渲染完毕之后，进行一个初始化内容：

mounted () {

this.buyNum = 1

this.buyType = this.buyTypes[0] （注意是“=”号）

this.versions = [this.versionList[0]]

this.period = this.periodList[0]

this.getPrice()

}

# 十七、VScode快捷键

**主命令框**

F1 或 Ctrl+Shift+P: 打开命令面板。在打开的输入框内，可以输入任何命令，例如：

* 按一下 Backspace 会进入到 Ctrl+P 模式
* 在 Ctrl+P 下输入 > 可以进入 Ctrl+Shift+P 模式

在 Ctrl+P 窗口下还可以:

* 直接输入文件名，跳转到文件
* ? 列出当前可执行的动作
* ! 显示 Errors或 Warnings，也可以 Ctrl+Shift+M
* : 跳转到行数，也可以 Ctrl+G 直接进入
* @ 跳转到 symbol（搜索变量或者函数），也可以 Ctrl+Shift+O 直接进入
* @ 根据分类跳转 symbol，查找属性或函数，也可以 Ctrl+Shift+O 后输入:进入
* # 根据名字查找 symbol，也可以 Ctrl+T

**常用快捷键**

**编辑器与窗口管理**

1. 打开一个新窗口： Ctrl+Shift+N
2. 关闭窗口： Ctrl+Shift+W
3. 同时打开多个编辑器（查看多个文件）
4. 新建文件 Ctrl+N
5. 文件之间切换 Ctrl+Tab
6. 切出一个新的编辑器（最多 3 个） Ctrl+\，也可以按住 Ctrl 鼠标点击 Explorer 里的文件名
7. 左中右 3 个编辑器的快捷键 Ctrl+1 Ctrl+2 Ctrl+3
8. 3 个编辑器之间循环切换 Ctrl+
9. 编辑器换位置， Ctrl+k然后按 Left或 Right

**代码编辑**

**格式调整**

1. 代码行缩进 Ctrl+[ 、 Ctrl+]
2. Ctrl+C 、 Ctrl+V 复制或剪切当前行/当前选中内容
3. 代码格式化： Shift+Alt+F，或 Ctrl+Shift+P 后输入 format code
4. 上下移动一行： Alt+Up 或 Alt+Down
5. 向上向下复制一行： Shift+Alt+Up 或 Shift+Alt+Down
6. 在当前行下边插入一行 Ctrl+Enter
7. 在当前行上方插入一行 Ctrl+Shift+Enter

**光标相关**

1. 移动到行首： Home
2. 移动到行尾： End
3. 移动到文件结尾： Ctrl+End
4. 移动到文件开头： Ctrl+Home
5. 移动到定义处： F12
6. 定义处缩略图：只看一眼而不跳转过去 Alt+F12
7. 移动到后半个括号： Ctrl+Shift+]
8. 选择从光标到行尾： Shift+End
9. 选择从行首到光标处： Shift+Home
10. 删除光标右侧的所有字： Ctrl+Delete
11. 扩展/缩小选取范围： Shift+Alt+Left 和 Shift+Alt+Right
12. 多行编辑(列编辑)：Alt+Shift+鼠标左键，Ctrl+Alt+Down/Up
13. 同时选中所有匹配： Ctrl+Shift+L
14. Ctrl+D 下一个匹配的也被选中 (在 sublime 中是删除当前行，后面自定义快键键中，设置与 Ctrl+Shift+K 互换了)
15. 回退上一个光标操作： Ctrl+U

**重构代码**

1. 找到所有的引用： Shift+F12
2. 同时修改本文件中所有匹配的： Ctrl+F12
3. 重命名：比如要修改一个方法名，可以选中后按 F2，输入新的名字，回车，会发现所有的文件都修改了
4. 跳转到下一个 Error 或 Warning：当有多个错误时可以按 F8 逐个跳转
5. 查看 diff： 在 explorer 里选择文件右键 Set file to compare，然后需要对比的文件上右键选择 Compare with file\_name\_you\_chose

**查找替换**

1. 查找 Ctrl+F
2. 查找替换 Ctrl+H
3. 整个文件夹中查找 Ctrl+Shift+F

**显示相关**

1. 全屏：F11
2. zoomIn/zoomOut：Ctrl +/-
3. 侧边栏显/隐：Ctrl+B
4. 显示资源管理器 Ctrl+Shift+E
5. 显示搜索 Ctrl+Shift+F
6. 显示 Git Ctrl+Shift+G
7. 显示 Debug Ctrl+Shift+D
8. 显示 Output Ctrl+Shift+U

**其他**

* 自动保存：File -> AutoSave ，或者 Ctrl+Shift+P，输入 auto

**修改默认快捷键**

打开默认键盘快捷方式设置：  
File -> Preferences -> Keyboard Shortcuts，或者：Alt+F -> p -> k

修改 keybindings.json：

// Place your key bindings in this file to overwrite the defaults

[

// ctrl+space 被切换输入法快捷键占用

{

"key": "ctrl+alt+space",

"command": "editor.action.triggerSuggest",

"when": "editorTextFocus"

},

// ctrl+d 删除一行

{

"key": "ctrl+d",

"command": "editor.action.deleteLines",

"when": "editorTextFocus"

},

// 与删除一行的快捷键互换

{

"key": "ctrl+shift+k",

"command": "editor.action.addSelectionToNextFindMatch",

"when": "editorFocus"

},

// ctrl+shift+/多行注释

{

"key":"ctrl+shift+/",

"command": "editor.action.blockComment",

"when": "editorTextFocus"

},

// 定制与 sublime 相同的大小写转换快捷键，需安装 TextTransform 插件

{

"key": "ctrl+k ctrl+u",

"command": "uppercase",

"when": "editorTextFocus"

},

{

"key": "ctrl+k ctrl+l",

"command": "lowercase",

"when": "editorTextFocus"

}

]

# 十八、点击购买显示购买购买弹窗包含一个购买信息表格

layout布局表单

slideshow幻灯片表单

analysis.vue:

import Dialog from '../../components/base/dialog'

export default {

components: {

MyDialog: Dialog (注意dialog是html内置标签，不能使用dialog)

},

<my-dialog :is-show="isShowPayDialog" @on-close="hidePayDialog">

isShowPayDialog: false,

showPayDialog () {

this.isShowPayDialog = true

}, （is-show默认是关闭的，关闭的时候设置为false）

hidePayDialog () {

this.isShowPayDialog = false

},

（2）analysis.vue的mydialog中写入表格

<table class="buy-dialog-table">

<tr>

<th>购买数量</th>

<th>产品类型</th>

<th>有效时间</th>

<th>产品版本</th>

<th>总价</th>

</tr>

<tr>

<td>{{ buyNum }}</td>

<td>{{ buyType.label }}</td>

<td>{{ period.label }}</td>

<td>

<span v-for="item in versions">{{ item.label }}</span>

</td>

<td>{{ price }}</td>

</tr>

</table>

buyNum: 0,

mounted () {

this.buyNum = 1

this.buyType = this.buyTypes[0]

this.versions = [this.versionList[0]]

this.period = this.periodList[0]

this.getPrice()

}

选择银行界面：

<h3 class="buy-dialog-title">请选择银行</h3>

<bank-chooser @on-change="onChangeBanks"></bank-chooser> （选择银行组件引入）

<div class="button buy-dialog-btn" @click="confirmBuy">

确认购买

</div>

bankChooser:

新建一个bankChooser.vue 级别同logForm.vue

<template>

<div class="chooser-component">

<ul class="chooser-list">

<li

v-for="(item, index) in banks"

@click="chooseSelection(index)"

:title="item.label"

:class="[item.name, {active: index === nowIndex}]"

></li>

</ul>

</div>

</template>

<script>

export default {

data () {

return {

nowIndex: 0, (当前选择的银行，默认是0)

banks: [

{

id: 201,

label: '招商银行',

name: 'zhaoshang'

},

{

id: 301,

label: '中国建设银行',

name: 'jianshe'

},

{

id: 601,

label: '浦发银行',

name: 'pufa'

},

{

id: 1101,

label: '交通银行',

name: 'jiaotong'

},

{

id: 101,

label: '中国工商银行',

name: 'gongshang'

},

{

id: 401,

label: '中国农业银行',

name: 'nongye'

},

{

id: 1201,

label: '中国银行',

name: 'zhongguo'

},

{

id: 501,

label: '中信银行',

name: 'zhongxin'

}

]

}

},

methods: {

chooseSelection (index) {

this.nowIndex = index

this.$emit('on-change', this.banks[index])

}

}

}

</script>

在analysis.vue中，引入

import BankChooser from '../../components/bankChooser'

export default {

components: {

BankChooser

},

》》》》》》

》》》》》》

confirmBuy () {

let buyVersionsArray = \_.map(this.versions, (item) => {

return item.value

})

let reqParams = {

buyNumber: this.buyNum,

buyType: this.buyType.value,

period: this.period.value,

version: buyVersionsArray.join(','),

bankId: this.bankId

}

this.$http.post('/api/createOrder', reqParams)

.then((res) => {

this.orderId = res.data.orderId

this.isShowCheckOrder = true

this.isShowPayDialog = false

}, (err) => {

this.isShowBuyDialog = false

this.isShowErrDialog = true

})

}

# 十九、vue-date-picker时间函数引入

https://githup.com/fishenal 中找到awesome vue

1、安装

npm install vue-date-picker --save

2、

import VDatePicker from 'vue-date-picker'

3、components:{

VDatePicker

}

4、界面引入<v-data-picker></ v-data-picker>

5、node-modules中找到 vue-date-picker，==》src ==》 datePicker.vue复制到需要应用的组件文件夹下面

6、

改变dataPicker.vue为以下代码：详情见同文件夹下的datePicker.vue

7、import VDatePicker from 'vue-date-picker' 改变引入路径为最后复制的vue-date-picker.vue的路径

# 二十、在json中读取数据存放到表格当中

1、有这样一个表格

<table>

<tr>

<th v-for="head in tableHeads">{{ head.label }}</th> （表头部分）

</tr>

<tr>

<td>

</td>

</tr>

</table>

2、data数据存在 tableHeads

tableHeads: [

{

label: '订单号',

key: 'orderId'

},

{

label: '购买产品',

key: 'product'

},

{

label: '版本类型',

key: 'version'

},

{

label: '有效时间',

key: 'period'

},

{

label: '购买日期',

key: 'date'

},

{

label: '数量',

key: 'buyNum'

},

{

label: '总价', （lable与表头一一对应）

key: 'amount'

}

],

3、此时显示表头信息



4、db.json 存在数据： （tableHeads中的key应该与list键值对中的key一一对应）

"getOrderList": {

"list": [

{

"orderId": "ddj123",

"product": "数据统计",

"version": "高级版",

"period": "1年",

"buyNum": 2,

"date": "2016-10-10",

"amount": "500元"

},

{

"orderId": "yuj583",

"product": "流量分析",

"version": "户外版",

"period": "3个月",

"buyNum": 1,

"date": "2016-5-2",

"amount": "2200元"

},

{

"orderId": "pmd201",

"product": "广告发布",

"version": "商铺版",

"period": "3年",

"buyNum": 12,

"date": "2016-8-3",

"amount": "7890元"

}

]

}

5、vue中写方法：

getTableData(){

let reqParam={

query:this.query,

productId:this.productId,

startDate:this.startDate, （此处是你想传入json的值）

endDate:this.endDate

}

this.$http.post('api/getOrderList',reqParam) （用post请求页面跳转到getOrderList'，db.json中的数据名称）

.then((res)=>{

this.tableData =res.data.list （将获得的数据复制给tableData）

},(err)=>{

})

}

6、界面刚进入是触发：

mounted() {

this.getTableData() （触动这个方法，不要忘记括号）

}

<tr v-for="item in tableData" :key="item.period">

<td v-for="head in tableHeads">{{ item[head.key] }}</td>

</tr>

7、编辑表格

<tr v-for="item in tableData" :key="item.period">

<td v-for="head in tableHeads">{{ item[head.key] }}</td>

</tr>

8、点击表头自动排序

（1）在表头上有这样的一个方法

<table>

<tr>

<th v-for="head in tableHeads" @click="changeOrderType(head)" :class="{active:head.active}">{{ head.label }}</th>

</tr>

<tr v-for="item in tableData" :key="item.period">

<td v-for="head in tableHeads">{{ item[head.key] }}</td>

</tr>

</table>

methods{

changeOrderType (headItem) {

this.tableHeads.map((item) => {

item.active = false

return item

})

headItem.active = true

if (this.currentOrder === 'asc') {

this.currentOrder = 'desc'

}

else if (this.currentOrder === 'desc') {

this.currentOrder = 'asc'

} （lodash的orderby方法返回的是排序好的数组，直接将它赋值给this.tableData即可）

this.tableData = \_.orderBy(this.tableData, headItem.key, this.currentOrder)

} (第一个参数你要排序的数组,第二个参数根据哪一个项目排序,第三个参数排序的方式【正序和倒叙】)

}

初始化排序方式： currentOrder: 'asc' 正序排序

# 二十一、this.$route.path点击对应文字切换图片this.$route.push点击关闭跳转页面

detail.vue:

用法一：

<img :src="productIcon">

data数据：

imgMap: {

'/detail/count': require("../assets/images/1.png"),

'/detail/forecast': require("../assets/images/2.png"),

'/detail/analysis': require("../assets/images/3.png"),

'/detail/publish': require("../assets/images/4.png")

}

计算属性：

computed:{

productIcon(){

return this.imgMap[this.$route.path]

}

}

在这个里面，this.$router.path与'/detail/count '相同，为url

用法二：

<this-dialog :is-show="isShowFailDialog" @on-close="toOrderList">

购买失败！

</this-dialog>

toOrderList () {

this.$router.push({path: '/orderList'})

}

# 二十二、eventBus和stopPropagation阻止时间冒泡

## 一、一个selection组件的情况

1、新建eventBus.js和main.js同级,写入

// The Vue build version to load with the `import` command

// (runtime-only or standalone) has been set in webpack.base.conf with an alias.

import Vue from 'vue'

const eventBus = new Vue()

export { eventBus }

2、在layout.vue中引入eventBus.js

import { eventBus } from '../eventBus.js'

<template>

<div id="app" @click="resetComponent"> (为了实现点击div控件，所有的selection组件隐藏)

methods:

resetComponent(){

eventBus.$emit('reset-component')

}

3、在selection.vue中引入eventBus.js

import { eventBus } from '../../eventBus.js'

mounted() {

eventBus.$on('reset-component',() => {

this.isDrop = false （点击的时候selection组件关闭）

})

},

methods: {

toggleDrop (event) {

event.stopPropagation(); （阻止时间冒泡）

this.isDrop = !this.isDrop

},

## 二、两个selection组件的情况

只需要再调用一次$emit事件就可以了

methods: {

toggleDrop (event) {

event.stopPropagation();

eventBus.$emit('reset-component')

this.isDrop = !this.isDrop

},

# 二十三、Vue基本组件构成

1、el、template、data含义

var newVue=new Vue({

el:'页面装载的位置'，

template:'装载的东西，html片段<div>hello</div>,模板',

data:{

name:'数据'

}

})

2、Vue实例生命周期：

beforeCreate =(初始化完成)=>created=>beforeMount=>mounted(渲染)

=> beforeUpdate => updated =>beforeDestroy => destroyed

3、箭头函数

render：function（h）{

return h(App)

}

相当于：

render：h=>h(App) return 相当于箭头

4、模板渲染

（1）{{}}进行渲染 不可以使用到属性里面，比如title

{{hello}}

data(){

retuen{

hello:'world'

}

}

（2）v-html/v-text:区别是v-html存在html标记性语言可以直接转化

<p v-html="hello"><p>

<p v-text="hello"><p>

（3）进行表达式书写

{{ num +1 }}

{{ status ? 'true': 'false' }}

data(){

retuen{

hello:'world，

num:1,

status:false

}

}

5、列表渲染v-for

数组渲染：

<ul>

<li v-for="item in list">{{ item.value }} </li>

或者

<li v-for="item in list v-text="item.name + ‘-' + item.value></li>

/ul>

(item ,index) in list 序号index这种方式被渲染

data(){

retuen{

hello:'world，

num:1,

status:false,

list:[

{

name:'a1',

value:'11'

},

{

name:'a2',

value:'22'

},

{

name:'a3',

value:'33'

}

]

}

}

对象渲染v-for：

v-for="value in obj"

data(){

return {

obj:{

name:'apple',

value:'111'

}

}

}

# 二十四、Vue中引入jQuery

1：因为已经安装了vue脚手架，所以需要在webpack中全局引入jquery

      打开package.json文件，在里面加入这行代码，jquery后面的是版本，根据你自己需求更改。

        dependencies:{

  "jquery":"^2.2.3"

       }

      然后在命令行中cnpm install

      大多人应该都是使用的淘宝镜像，所以使用cnpm，如果你不是 ，可以使用npm安装。

2：在webpack.base.conf.js中加入一行代码

 var webpack=require("webpack")

3：在webpack.base.conf.js中module.exports的最后加入这行代码，

  plugins: [  
new webpack.optimize.CommonsChunkPlugin('common.js'),  
new webpack.ProvidePlugin({  
    jQuery: "jquery",  
    $: "jquery"  
})  
]

4：在main.js中引入,加入下面这行代码

      import $ from 'jquery'

5：最后一步，重新跑一边就好，cnpm run dev

# 二十五、Vue之css样式

一：border-radius

border-radius：用来设置边框圆角。

有两种语法：

border-radius:2px 3px;

border-radius:2px 4px/6px;

* 一个值

border-radius: radius 表示边框四角的圆角半径。只在单值语法中使用。

* 两个值

border-radius: top-left-and-bottom-right top-right-and-bottom-left

第一个值表示边框**左上角和右下角**的圆角半径。   
第一个值表示边框**右上角和左下角**的圆角半径

* 三个值

border-radius: top-left top-right-and-bottom-left bottom-right

第一个值为左上角；   
第二个值为右上角和左下角；   
第三个值为右下角。

* 四个值

border-radius：top-left top-right bottom-right bottom-left

从左上角开始顺时针声明四个角的值。

**椭圆半径**

半径的第二个语法，“/” 前后分别可取1~4个值，对应的角的顺序与第一种语法相同。   
“/”前后相对应的第一个值与第二个值确定了一个角的**椭圆半径**。

border-radius: 1em/5em;

圆形：.div1{

border-radius:50px; /\* 或者50% \*/

width:100px;

height:100px;

background-color:lightgreen;

}

二：padding: 0; margin: 0;

就是全部样式的外边距和内边距都为0， 也就是写CSS之前把所有浏览器的默认样式统一。

主要意义是为了兼容不同的浏览器，消除浏览器的默认值。

三、！important

.xzjxsc .el-radio + .el-radio {

margin-left: 15px !important;

}

!important的作用是提高指定样式规则的应用优先权。写在定义的最后面，例如：box{color:red !important;} 最重要的一点是：IE 6.0一直都不支持这个语法，而其他的浏览器都支持。因此我们就可以利用这一点来分别 比如：  
.bor{ border:20px solid red !important; border:20px solid green;padding: 30px;width : 300px;}  
因为ie6不认识!important，后面的绿色边框样式就会覆盖前面那句红色边框样式，而如ie7、火狐等浏览器认识!important，知道这一条样式的优先级要比后面那句高，就只会执行这句红色边框的样式。

四、height：100% ：想让根节点满屏

五、display：flex

父级元素定义display:flex,子元素宽度用flex来定义,flex:1 是均分父级元素。占的比例相同

main:{

padding :20px;

display: flex

}

.left{

flex:1

}

.right{

flex:1

}

.clear{

clear:both

}

这样是一比一分布

如果是flex:1 flex:2那么就是一比二

如果是flex:1 flex:2 flex:1 那面就是左：中：右为1：2：1

flex-direction: 属性规定灵活项目的方向。

六：颜色渐变  
 background: linear-gradient(to right, #f5adc9, #f38eb6, #f977ab);

七：禁止鼠标点击：  
 pointer-events: none;

8、设置透明，并兼并浏览器

{

filter: alpha(opacity=50); /\* internet explorer \*/

-khtml-opacity: 0.5; /\* khtml, old safari \*/

-moz-opacity: 0.5; /\* mozilla, netscape \*/

opacity: 0.5; /\* fx, safari, opera \*/

}

# 二十六、前台直接跳转后台

前台代码：

openWs() {

var docUrl =

window.location.protocol +

"//" +

window.location.host +

"/zhzj/writ/" +

this.caseId +

"/load/" + this.selectValue;

this.openFromURL(docUrl);

},

后台接收： 一一对应

@Controller

@RequestMapping("writ")

public class DocumentController {

static final File BASEPATH = new File(new File(ArteryWebUtil.getWebPath()), "artery" + "/" + "downloads"

+ "/" + "files");

@RequestMapping(value = "{caseId}/{load}/{selectValue}", method = RequestMethod.GET)

public void loadWrit(@PathVariable("caseId") String caseId, @PathVariable("selectValue") String selectValue,

@PathVariable("load") String load, HttpServletResponse res) throws StoreException {

InputStream inputStream = null;

OutputStream outputStream = null;

String autoDocType=ApiKeyToType.selectTypeByApiKey(selectValue);

ZhzjAutoDoc zhzjAutoDoc=new ZhzjAutoDoc();

zhzjAutoDoc.setAutoDocType(autoDocType);

zhzjAutoDoc.setCaseId(caseId);

List<ZhzjAutoDoc> fileList = autoDocService.getZhzjAutoDocByCaseIdAndType(zhzjAutoDoc);

try {

if (CollectionUtils.isNotEmpty(fileList)) {

String fileName = caseId + "\_"+autoDocType+".docx";

String ftpPath = "/zhzj/" + caseId + "/" + fileName;

inputStream = FtpUtil.downloadFileAsStream(fileName, ftpPath);

}

outputStream = res.getOutputStream();

IOUtils.copy(inputStream, outputStream);

} catch (IOException e) {

logger.error("数据流转换出错", e);

} finally {

IOUtils.closeQuietly(inputStream);

IOUtils.closeQuietly(outputStream);

}

}

# 二十七、1变一 11变十一

num="1"

number = this.Arabia\_To\_SimplifiedChinese(num);

Arabia\_To\_SimplifiedChinese(Num) {

//字符处理完毕后开始转换,采用前后两部分分别转换

let part = String(Num).split(".");

let newchar = "";

//小数点前进行转化

for (i = part[0].length - 1; i >= 0; i--) {

if (part[0].length > 10) {

return "";

} //若数量超过拾亿单位,提示

let tmpnewchar = "";

let perchar = part[0].charAt(i);

switch (perchar) {

case "0":

tmpnewchar = "零" + tmpnewchar;

break;

case "1":

tmpnewchar = "一" + tmpnewchar;

break;

case "2":

tmpnewchar = "二" + tmpnewchar;

break;

case "3":

tmpnewchar = "三" + tmpnewchar;

break;

case "4":

tmpnewchar = "四" + tmpnewchar;

break;

case "5":

tmpnewchar = "五" + tmpnewchar;

break;

case "6":

tmpnewchar = "六" + tmpnewchar;

break;

case "7":

tmpnewchar = "七" + tmpnewchar;

break;

case "8":

tmpnewchar = "八" + tmpnewchar;

break;

case "9":

tmpnewchar = "九" + tmpnewchar;

break;

}

switch (part[0].length - i - 1) {

case 0:

tmpnewchar = tmpnewchar;

break;

case 1:

if (perchar != 0) tmpnewchar = tmpnewchar + "十";

break;

case 2:

if (perchar != 0) tmpnewchar = tmpnewchar + "百";

break;

case 3:

if (perchar != 0) tmpnewchar = tmpnewchar + "千";

break;

case 4:

tmpnewchar = tmpnewchar + "万";

break;

case 5:

if (perchar != 0) tmpnewchar = tmpnewchar + "十";

break;

case 6:

if (perchar != 0) tmpnewchar = tmpnewchar + "百";

break;

case 7:

if (perchar != 0) tmpnewchar = tmpnewchar + "千";

break;

case 8:

tmpnewchar = tmpnewchar + "亿";

break;

case 9:

tmpnewchar = tmpnewchar + "十";

break;

}

newchar = tmpnewchar + newchar;

}

//替换以“一十”开头的,为“十”

if (newchar.indexOf("一十") == 0) {

newchar = newchar.substr(1);

}

//替换以“零”结尾的,为“”

if (newchar.lastIndexOf("零") == newchar.length - 1) {

newchar = newchar.substr(0, newchar.length - 1);

}

return newchar;

},

# 二十八、文字超长显示问题

超出文本长度的内容省略号显示，鼠标放上去显示全部内容

<span class="ahNameClass" :title="item.name">{{item.name}}</span>

.ahNameClass{

display: inline-block;

width: 251px;

overflow: hidden;

text-overflow: ellipsis;

white-space: nowrap;

position: absolute;

}

# 二十九、返回主页面的时候保存页面上的纠违或者其他未保存的信息

1、home.vue里面点击事件：

handleClick() {

this.$store.dispatch('saveSclcOk',true)

setTimeout(()=>{

this.goAjlb() //500毫秒之后再执行这个操作

},500)

2、 btnset.vue里面

computed: {还原证据名称重新识别瑕疵点/获取最后修改时间格式问题/修改自动同步文书名称初始化的错误赋值

getSaveSclc(){

return this.$store.getters['saveSclcOk']

},

watch: {

getSaveSclc(newValue, oldValue){

if(newValue){

this.saveSclc()

}

},

//保存方法里面

saveSclc() {

if (this.disableEsitors) {

return;

}

this.$store.dispatch('saveSclcOk',false)

3、 保存方法里面

handleSave() {

this.$store.dispatch('saveSclcOk',true)

4、配置文件：evidence.js

const state = {

evidenceId: '',

caseId: '',

evidenceType: [],

currEvidenceType: [],

addEvidenceVisible: false, // 是否点击了证据补录

changeRecordVisible: false, // 是否点击了变更历史

changeReasonVisible: false, // 是否弹出了变更原因窗口

exitVisible: false, // 退出弹窗是否显示，是否点击了退出按钮

currCaseCompleted: -1, // 当前案件上传的文件是否解析完成:0解析中，1解析完成

savePageBeforeGoAjlb: 0, // 返回案件列表前先保存,0不保存 1保存 2保存完成

savePageBeforeExit: 0, // 退出前先保存,0不保存 1保存 2保存完成

eviManagePdfVisible: true, // 证据管理，是否点击了删除节点，隐藏pdf

pageFlag: '', // 页面类型：ajlb案件列表页 zjcxsc证据程序性审查

currClickEviId: '', // 点击全局的证据审查点，保存其证据id，切换目录

uploadSuccess: 0, // 证据补录是否上传成功,0初始状态没有上传 1上传成功 2上传中 3上传失败

uploadClearFiles: false, // 清空上传的文件列表

nodeOldInfoList: [], // 记录页面原来需要展开的节点信息，目的：新增节点或子节点，保留原树的收起展开状态

clickAppendChildNode: false, // 记录是否点击了新增子节点

clickIgnoreBtn: false, //是否点击了忽略按钮

evidenceList: [] ,

changechartOpation:false, //是否改变图表的数据

clickSave:true, //是否点击保存

saveOk:false, //是否保存成功

clickTreeNode:'9999',//点击树结构时的id值

repDis:false,//是否点击的重新识别

zjglClick:false,//是否点击的证据管理

closePdf:false,//点击查看文书隐藏pdf

exportWord:false,//导出文书

userIdentity:'',//登录人身份

disableEsitors:'',//根据不同身份进入案件详情，是否为不可编辑状态

btnsShow:false,//按钮组的显示状态

saveSclcOk:false,//审查流程是否需要保存

}

// getters

const getters = {

evidenceId: state => state.evidenceId,

closePdf: state => state.closePdf,

caseId: state => state.caseId,

evidenceType: state => state.evidenceType,

currEvidenceType: state => state.currEvidenceType,

addEvidenceVisible: state => state.addEvidenceVisible,

changeRecordVisible: state => state.changeRecordVisible,

changeReasonVisible: state => state.changeReasonVisible,

exitVisible: state => state.exitVisible,

currCaseCompleted: state => state.currCaseCompleted,

savePageBeforeGoAjlb: state => state.savePageBeforeGoAjlb,

savePageBeforeExit: state => state.savePageBeforeExit,

eviManagePdfVisible: state => state.eviManagePdfVisible,

pageFlag: state => state.pageFlag,

currClickEviId: state => state.currClickEviId,

uploadSuccess: state => state.uploadSuccess,

uploadClearFiles: state => state.uploadClearFiles,

nodeOldInfoList: state => state.nodeOldInfoList,

clickAppendChildNode: state => state.clickAppendChildNode,

clickIgnoreBtn: state => state.clickIgnoreBtn,

evidenceList: state => state.evidenceList,

changechartOpation: state => state.changechartOpation,

saveOk: state => state.saveOk,

clickTreeNode: state => state.clickTreeNode,

repDis: state => state.repDis,

clickSave: state => state.clickSave,

zjglClick: state => state.zjglClick,

exportWord: state => state.exportWord,

userIdentity: state => state.userIdentity,

disableEsitors: state => state.disableEsitors,

btnsShow: state => state.btnsShow,

saveSclcOk: state => state.saveSclcOk,

}

// actions

const actions = {

getCurrEvidence ({ commit, state }, obj) {

commit('getCurrEvidence', obj)

},

closePdf ({ commit, state }, obj) {

commit('closePdf', obj)

},

zjglClick ({ commit, state }, obj) {

commit('zjglClick', obj)

},

exportWord ({ commit, state }, obj) {

commit('exportWord', obj)

},

getCurrCaseId ({ commit, state }, obj) {

commit('getCurrCaseId', obj)

},

getEvidenceType ({ commit, state }, obj) {

commit('getEvidenceType', obj)

},

getCurrEvidenceType ({ commit, state }, obj) {

commit('getCurrEvidenceType', obj)

},

getAddEvidenceVisible ({ commit, state }, obj) {

commit('getAddEvidenceVisible', obj)

},

getChangeRecordVisible ({ commit, state }, obj) {

commit('getChangeRecordVisible', obj)

},

getChangeReasonVisible ({ commit, state }, obj) {

commit('getChangeReasonVisible', obj)

},

getExitVisible ({ commit, state }, obj) {

commit('getExitVisible', obj)

},

getCurrCaseCompleted ({ commit, state }, obj) {

commit('getCurrCaseCompleted', obj)

},

getSavePageBeforeGoAjlb ({ commit, state }, obj) {

commit('getSavePageBeforeGoAjlb', obj)

},

getSavePageBeforeExit ({ commit, state }, obj) {

commit('getSavePageBeforeExit', obj)

},

getEviManagePdfVisible ({ commit, state }, obj) {

commit('getEviManagePdfVisible', obj)

},

getPageFlag ({ commit, state }, obj) {

commit('getPageFlag', obj)

},

getCurrClickEviId ({ commit, state }, obj) {

commit('getCurrClickEviId', obj)

},

getUploadSuccess ({ commit, state }, obj) {

commit('getUploadSuccess', obj)

},

getUploadClearFiles ({ commit, state }, obj) {

commit('getUploadClearFiles', obj)

},

getNodeOldInfoList ({ commit, state }, obj) {

commit('getNodeOldInfoList', obj)

},

getClickAppendChildNode ({ commit, state }, obj) {

commit('getClickAppendChildNode', obj)

},

getClickIgnoreBtn ({ commit, state }, obj) {

commit('getClickIgnoreBtn', obj)

},

evidenceList ({ commit, state }, obj) {

commit('evidenceList', obj)

},

changechartOpation ({ commit, state }, obj) {

commit('changechartOpation', obj)

},

saveOk ({ commit, state }, obj) {

commit('saveOk', obj)

},

clickTreeNode ({ commit, state }, obj) {

commit('clickTreeNode', obj)

},

repDis ({ commit, state }, obj) {

commit('repDis', obj)

},

clickSave ({ commit, state }, obj) {

commit('clickSave', obj)

},

userIdentity ({ commit, state }, obj) {

commit('userIdentity', obj)

},

disableEsitors ({ commit, state }, obj) {

commit('disableEsitors', obj)

},

btnsShow ({ commit, state }, obj) {

commit('btnsShow', obj)

},

saveSclcOk ({ commit, state }, obj) {

commit('saveSclcOk', obj)

},

}

// mutations

const mutations = {

// data 载荷

getCurrEvidence (state, data) {

state.evidenceId = data

},

exportWord (state, data) {

state.exportWord = data

},

closePdf (state, data) {

state.closePdf = data

},

zjglClick (state, data) {

state.zjglClick = data

},

getCurrCaseId (state, data) {

state.caseId = data

},

getEvidenceType (state, data) {

state.evidenceType = data

},

getCurrEvidenceType (state, data) {

state.currEvidenceType = data

},

getAddEvidenceVisible (state, data) {

state.addEvidenceVisible = data

},

getChangeRecordVisible (state, data) {

state.changeRecordVisible = data

},

getChangeReasonVisible (state, data) {

state.changeReasonVisible = data

},

getExitVisible (state, data) {

state.changeReasonVisible = data

},

getCurrCaseCompleted (state, data) {

state.exitVisible = data

},

getSavePageBeforeGoAjlb (state, data) {

state.savePageBeforeGoAjlb = data

},

getSavePageBeforeExit (state, data) {

state.savePageBeforeExit = data

},

getEviManagePdfVisible (state, data) {

state.eviManagePdfVisible = data

},

getPageFlag (state, data) {

state.pageFlag = data

},

getCurrClickEviId (state, data) {

state.currClickEviId = data

},

getUploadSuccess (state, data) {

state.uploadSuccess = data

},

getUploadClearFiles (state, data) {

state.uploadClearFiles = data

},

getNodeOldInfoList (state, data) {

state.nodeOldInfoList = data

},

getClickIgnoreBtn (state, data) {

state.clickIgnoreBtn = data

},

evidenceList (state, data) {

state.evidenceList = data

},

changechartOpation (state, data) {

state.changechartOpation = data

},

saveOk (state, data) {

state.saveOk = data

},

clickTreeNode (state, data) {

state.clickTreeNode = data

},

repDis (state, data) {

state.repDis = data

},

userIdentity (state, data) {

state.userIdentity = data

},

clickSave (state, data) {

state.clickSave = data

},

disableEsitors (state, data) {

state.disableEsitors = data

},

btnsShow (state, data) {

state.btnsShow = data

},

saveSclcOk (state, data) {

state.saveSclcOk = data

},

// ,

// getClickAppendChildNode (state, data) {

// state.clickAppendChildNode = data

// }

}

export default {

state,

getters,

actions,

mutations

}

# 三十、防止重复增加数据

<el-button type="primary" class="blue" size="small" @click="checkSave()">保存</el-button>

successflag: true,

methods:

checkSave(){

if(this.successflag){

this.save()

this.successflag = false;

}

setTimeout(() => {

this.successflag = true;

}, 5000);

},

再调用保存方法，就不会重复添加了

save() {

this.sort(this.treeData);

let treeDataStr = JSON.stringify(this.treeData);

let deleteDocListStr = JSON.stringify(this.deleteDocList);

let deleteDocFileListStr = JSON.stringify(this.deleteDocFileList);

let params = {

appKey: this.global.config.appKey,

sign: this.$getUser(),

apiKey: "0304",

v: "1",

timestamp: new Date().getTime(),

caseId: this.caseId,

treeDataStr: treeDataStr,

deleteDocListStr: deleteDocListStr,

deleteDocFileListStr: deleteDocFileListStr

};

api

.post(this.global.config.baseApiUrl, params)

.then(response => {

if (response.data.code === "0000") {

console.log('保存成功！！！')

if (this.successflag) {

this.$notice("1","保存成功");

}

this.$emit("hiddenDialog", true);

} else {

if (this.failedflag) {

this.$notice("2","保存失败");

this.failedflag = false;

setTimeout(() => {

this.failedflag = true;

}, 3000);

}

}

})

.catch(error => {

console.log(error);

});

},

# 三十一、手动增加按钮鼠标划过变小手

mounted() {

this.caseId = this.$getCaseId();

this.getData("FactDimension");

let elTabs = document.getElementsByClassName("el-tabs\_\_header")[0];

let button = document.createElement("button"); <!--增加按钮-->

elTabs.appendChild(button); <!--增加按钮-->

let Btn = elTabs.lastChild;

Btn.innerHTML = "返回"; <!--按钮名称-->

Btn.classList.add("backRouter"); <!--按钮样式-->

let \_this = this;

document.getElementsByClassName("backRouter")[0].onclick = function() {

\_this.$router.go(-1); <!--返回上一页，点击按钮-->

};

},

样式：

.backRouter {

width: 80px;

height: 30px;

background: #409eff;

position: absolute;

border: 0;

color: #fff;

font-size: 14px;

padding: 0;

line-height: 30px;

border-radius: 3px;

display: block;

position: absolute;

left: 0;

bottom: 60px;

right: 0;

margin: 0 auto;

cursor:pointer <!--点击的时候出现小手，不是鼠标-->

}

.backRouter:active { <!--点击的时候，背景颜色变为下面这个颜色-->

background: #1579e0;

}

# 三十二、点击空白不可关闭组件

<el-dialog class="my-small" size="small" title="变更原因" :visible.sync="dialogVisibleReason" width="40%">

<el-dialog class="my-small" size="small" title="变更原因" :visible.sync="dialogVisibleReason" width="40%" :close-on-click-modal="false">

# 三十三、只在某个指定的页面显示某控件

<el-popover v-if="newsShow" class="popover"

placement="bottom"

:width="popoverWidth"

v-model="visible">

<div class="newsTitle">继续侦查消息提醒</div>

<div v-if="newsContent.length>0">

<ul class="newsList">

<li v-for="(data,index) in newsContent" :key="index">

<span :title='data.caseName +"一案，" +"继续侦查意见取证期限截止到" +data.date +"结束，请注意审核"'><span @click="getAjxxList(data.caseID,index)" style="color:rgb(26, 147, 250)">{{data.caseName}}</span>一案，继续侦查意见取证期限截止到{{data.date}}结束，请注意审核</span>

<span @click="removeNews(index)">已审核</span>

</li>

</ul>

<div @click="removeNews('removeall')"><a href="javascript:;" style="color:#1a93fa">已全部审核</a></div>

</div>

<div v-else>

<img src="./ajlb/img/newsNone.png" alt="" style="display:block; margin:0 auto;">

</div>

<el-button class="btn-bell" slot="reference" v-show="!disabled">

<div class="newsCount" v-show="newsContent.length>0">{{newsContent.length}}</div>

<i class="el-icon-bell"></i>

</el-button>

</el-popover>

data:{

newsShow:true

}

$route: {

handler: function(val, oldVal) {

if (val.path == "/ajlb") { （只有在案件列表页面将newsShow设置为true）

localStorage.setItem("newsShow", true);

if (localStorage.getItem("newsShow") == "true") {

this.newsShow = true;

this.$nextTick(() => {

if (this.newsContent.length > 0 && !this.disabled) {

this.visible = true;

this.popoverWidth = "600";

} else {

this.visible = false;

this.popoverWidth = "350";

}

});

}

} else {

localStorage.setItem("newsShow", false);

if (localStorage.getItem("newsShow") == "false") {

this.newsShow = false;

this.visible = false;

}

}

},

// 深度观察监听

deep: true

}

# 三十四：样式子、孙css

.fxpgjya .el-tree-node\_\_children[role="group"]>div:last-child{

margin-left:3px!important;

}

解释：class = fxpghya的div下面的 class = el-tree-node\_\_children的div中 role="group"的div的孩子不包括孙子中的最后一个，将它设置成为margin-left:3px!important;

# 三十五:字母和数字也自动换行

word-break: break-all;

word-wrap: break-word;

# 三十六：在一个vue页面的方法里面调用另外一个vue页面的方法

a.vue里面有这样一个方法hanleUpdate()：，想要调用b.vue里面 的getLeftData()方法

（一、前端a.vue页面）

handleUpdate(){

let params = {

appKey: this.global.config.appKey,

sign: this.$getUser(),

apiKey: "0109",

v: "1",

timestamp: new Date().getTime(),

caseId: this.caseId

};

api

.post(this.global.config.baseApiUrl, params)

.then(response => {

if (response.data.data == true) {

console.log(response.data.data)

// this.$refs.gerLeftDataFather.getLeftData();

this.$notice("1", "更新成功");

**this.$store.dispatch('handleUpdata',true)**

} else {

this.$notice("2", "更新失败");

}

})

.catch(error => {

console.log(error);

});

// this.$refs.gerLeftDataFather.getLeftData();

},

（二、前端evidence.js页面）

使用vuex:

const state = {

handleUpdata:false, 定义这个变量的初始值是false

}

const getters = {

handleUpdata: state => state.handleUpdata, // 配置获得的状态

}

const actions = {

handleUpdata ({ commit, state }, obj) {

commit('handleUpdata', obj) //配置动作

}

}

const mutations = {

// data 载荷

handleUpdata (state, data) {

state.handleUpdata = data

}

}

export default {

state,

getters,

actions,

mutations

}

（三、前端b.vue页面）

computed: {

gethandleUpdata() {

return this.$store.getters["handleUpdata"]; ---handleUpdate要写对

},

}

watch: {

gethandleUpdata(newVal,oldVal){

if(newVal){

**this.getLeftData() ---- 此处就是调用b.vue的**

this.$store.dispatch('handleUpdata',false)

}

}

# 37、点击对应地方滚动条发生改变

clickTreeNode(newValue, oldValue){

let newTitleList = document.getElementsByClassName('title')

for(let i = 0; i<newTitleList.length;i++){

newTitleList[i].classList.remove('activeTitle')

}

let NewDom = document.querySelector('.XCname #title'+newValue+'.title')

if(NewDom){

NewDom.classList.add('activeTitle')

let scrollTop = document.getElementsByClassName("activeTitle")[0].offsetTop;

// alert(scrollTop)

let content = document.getElementsByClassName("content")[4]; 一定是对应的div

console.log(document.getElementsByClassName("content").length)

// content.style.background='#f0f'

content.scrollTop = scrollTop-458;

alert(content.scrollTop)

}else{

return

}

},

双向绑定：

麻烦的方法：

<input type="checkbox" v-bind:checked="checked" ref = "c2" @change = "change">v-bind</input><br/>

<label for="checkbox">{{ checked }}</label>

data(){

return {checked : true}

},

change(){

this.checked = this.$refs.c2.checked

}

# 38、js中的join(),reverse()与 split()函数用法解析

\* 1：arrayObject.reverse()

\* 注意： 该方法会改变原来的数组，而不会创建新的数组。

\* 2：arrayObject.join()

\* 注意：join() 方法用于把数组中的所有元素放入一个字符串。

\* 元素是通过指定的分隔符进行分隔的。

\* 指定分隔符方法join("#");其中#可以是任意

\* 3：stringObject.split(a,b)这是它的语法

\* 方法：用于把一个字符串分割成字符串数组.

\* a是必须的决定个从a这分割

\* b不是必须的，可选。该参数可指定返回的数组的最大长度 。如果设置了该参数，返回的子串不会多于这个参数指定的数组。如果没有设置该参数，整个字符串都会被分割，不考虑它的长度。

\* 注意：返回的数组中不包括a本身；

\* 提示和注释：

\* 注释：如果把空字符串 ("") 用作 a，那么 stringObject 中的每个字符之间都会被分割。

\* 注释：String.split() 执行的操作与 Array.join 执行的操作是相反的。

举例：

var str = "a,b,c,d,e,f,g";//声明一个字符串

str = str.split(',').reverse();//用split函数拆分成数组对象，再用reverse函数将数组倒序排列

console.log((str.length+":"+typeof(str)));

// 7:object

console.log(typeof(str.join)+":"+str.join('#')+":"+str.join('#').length);

// function:g#f#e#d#c#b#a:13

var s = str.reverse().join("#");

console.log(typeof(s)+":"+s);

// string:a#b#c#d#e#f#g

数组的操作：

https://blog.csdn.net/hehaoyang666/article/details/39137553

# 39、用key管理可复用元素

<span v-if="loginType === 'username'">

<label>Username</label>

<input placeholder="Enter your username" >

</span>

<span v-else>

<label>Email</label>

<input placeholder="Enter your email address" >

</span>

<button v-on:click="test">切换</button>

</br></br>

<span v-if="loginType === 'username'">

<label>Username</label>

<input placeholder="Enter your username" key="username-input">

</span>

<span v-else>

<label>Email</label>

<input placeholder="Enter your email address" key="email-input">

</span>

<button v-on:click="test">切换</button>

加上key之后再切换时，如果文本框中有输入的内容，那么会自动清除

# 40、v-show  & v-if

　　v-show也是根据条件展示元素的指令。带有 v-show 的元素始终会被渲染并保留在 DOM 中。v-show 是简单地切换元素的 CSS 属性 display 。