# v-model:

双向绑定，应用在<input><textarea>等标签中，将输入值和data值绑定起来，

|  |  |
| --- | --- |
| **修饰符** | |
| v-model**.lazy** | 控件失焦后更改data数据 |
| v-model**.number** | 将输入值转变为数字，若是字符则不生效 |
| v-model**.trim** | 清空输入数据的前后空格 |

# v-bind:

主要用在动态更新页面元素属性值、父传子、CSS渲染方面。

## CSS渲染

* 利用Boolean值控制class，class修饰来自CSS文件中：v-bind:class=”{style: vm.boolean,…}”。
* 通过string控制v-bind:style=”{CSS属性:vm.data,}”，{ }中的内容引用data、computed中的数据，此方式中Vue支持三元判定式bool?vm.strArry1:vm.strArry2
* 利用data中数据批量控制class：v-bind:class=“[vm.strArray]”,但此方式无法进行判断操作；仅能通过pop()、push()、shift()、unshift()操作数组元素的方式控制class；
* 利用Boolean值对标签进行使能控制，v-bind:disabled=”true”或者v-bind:disabled=”flag”【flag定义在data中】

数组：v-bind:style=[ ]，[ ]中的内容引用data、computed中的对象型数据，对象格式如下：

**border\_class:function(){**

**return {width: '80%',margin: '10%',padding: '10px',borderRadius:'13px',border:'4px solid '+this.color};**

**},**

**数组与对象复用模式** ：[ { },{ } ]

数组中插入data、computed中的字段组合成的对象，这一部分写在{ }中，data、computed中的对象型值写在{ }外[ ]里并用‘，’隔开。

# v-on/@

触发methods中的方法，若需传入data中数据，在标签中直接传定义的data中的变量名即可，不能写this。如send(msg)。

|  |  |  |  |
| --- | --- | --- | --- |
| **v-on:/@** | | | |
| .dbclick | 双击 |  |  |
| .click | 单击 |  |  |
| .keyup | 键盘按键抬起触发 |  |  |
| .keydown | 键盘按键按下触发 |  |  |
| .mousermove | 鼠标移动触发 |  |  |

# v-if

v-if、v-else-if、v-else这三个命令常一起联用，v-if=”…”可写判断式，也可引用方法【要加括号】、计算属性。v-if控制的是DOM的新增删除，其效果和v-show相差无几；使用v-if更安全而且对性能消耗较大。

★v2.1.0之后的vue版本对v-if做了优化，并不是将控制的dom中的内容全部移除，那些没有绑定vue的控件并不会删除，只是替换部分不同的属性（前提是两者属性不同）。

# v-show

v-show="…"只接收true/false的变量，这些变量可以直接写上去或来自于data，计算属性

# 侦听器watch

监听某个data的数据，数据发生变化则触发watch方法；具备实时性，依赖缓存，执行效率高的特点。

**定义：**

watchName:function(**New**,**Old**){……}或简写为watchName(**New**,**Old**){……}//New和Old分别代表新值和旧值。

# 过滤器filters

filter用于简单的计算，比如小写转大写，数字隔开等等，定义时必须传参。

**定义**

filterName:function(value){ return …}或简写为filterName(value){return …}，value值是引用过滤方法的data数据的原值，可以不写；

**使用**：{{ data | filterName }}这样用 | 隔开

# 计算属性computed

用于复杂计算，被用于v-model；它通常依赖data计算，data中数据改变时将触发它[依赖缓存]；具备不重新渲染，计算效率高的特点。但也可以在使用set/get格式时不依赖data。

**默认格式**：默认格式下不可传值；

computedName(){ return …… }

**set/get格式**：set传值（建议传值**value**，且只有一个值），set方法中可获取v-model绑定的计算属性的值——**value**，建议

computedName:{

get(){……},

set(value){ ……}

}

# 组件components

定义：略，使用：略

组件传值——父传子：

子组件通过props中的元素承接父元素传来的数据，props中的单数据元素值和父组件的元素值无关而对象、数组则有关；

**使用：v-bind:props中的元素=”父中传递的元素”**

**对象传递缩写**：如果传递给子组件的是一个对象，而这个对象的所有属性名与子组件props中的全部元素名均吻合，则可将**v-bind:props中的元素=”父中传递的元素”**缩写为**v-bind=“父中传递的元素”，**

**props注意事项**

建议使用kebab-case 方式命名props元素

props中的值虽可略写，但建议使用propsName:{type:…,default:…,required:…}格式

|  |  |  |
| --- | --- | --- |
| type | 数据种类 | Number、String、Array、Function、Boolean、Object |
| default | 默认值 |  |
| required | 是否必传此选项数据 | true、false |

组件传值——子传父：

使用**$emit(“object”,component.data.value)**，在子元素的method，watch，computed，filter的function中注册object，将值赋予object，然后object再将值传给父元素，父元素需要使用methods中的方法接收值：@object=“father.method.function”。

原生事件绑定到组件：

引用子组件时标签里写**v-on:原生事件名.native=“父组件中的方法”**，将原生事件绑定到组件，组件触发原生事件后会牵连触发父组件中定义的方法。

父链：子组件可使用this.$parent.父中元素访问父组件中的数据，并且支持同步修改。【此方法看似方便，但不建议使用，因为容易导致安全问题】

子链：给子组件加一个ref="…"标签，此标签是命名子组件的，父可通过this.$ref.….子组件元素访问到子组件中内容。

监听子组件

定义在子组件中一个事件名，在父中使用时监听

使用：<blog-post **v-on:enlarge-text**="postFontSize += 0.1" ></blog-post>

定义：<button v-on:click="**$emit('enlarge-text')**">

Enlarge text

</button>

就像父传子一样，不同的是并未传值，只是注册了一个父传子事件，子触发，父监听。

v-model在子组件中使用

使用<base-model v-model="text"></base-model>

定义<input type="text" v-bind:value=" value" v-on:input="$emit('input',$event.target.value)"/>// props中必须定义value

.sync同步父传子的数据

父传子，子中修改props中的值（虽然可行，但Vue会警告，但出于维护的角度考虑这种方法并不推荐使用），父中值并不受影响，当业务需要同步子修改的值到父，请使用.sync关键字。格式如下，父传子text，子传给父title事件名，父接收时并不引用函数而是引用data中的text。

<base-sync v-bind:title**.sync**="text" :title="text"></base-sync>//组件使用

this.$emit('update:title',……);//组件中定义

子组件中需要双向绑定props中的值，请在data中定义一个数据接收props中的数据，并结合以上方法。

## 内联模板

应用组件后，给组件标签加上inline-template，组件就会把写在其标签中的内容展示出来。

<test **inline-template** >

<div class="border-style">

这里就使用了内联模板，并且会将这个div中的内容展现出来

</div>

</test>

# 动态组件

**<component :is="componentId"></component>**

动态切换控件，通过操作父中元素操控子组件切换。其中componentId是子组件名，父元素可以不断改变componentId这个值，前提是父中在computed中定义一个componentId的计算方法，如下：

computed: {

componentId(){ return ……; }

},

return的值即子组件名【string类型】。

在<ul>/<ol>/<table>/<select>标签里面使用组件，请使用<… is=”组件名”>这样的格式。

为保持被切换的组件的现有状态，使用<keep-alive></keep-alive>标签包裹组件。

# 异步组件

通过工厂函数的方式加载组件，而不是一次性全部引入，这样可以节省资源开销

# Render渲染函数

通过JS构造组件，可以有效避免重复冗余的代码但随着业务的增加也带来了阅读不便的烦恼，所以**慎用**。在CLI编程的环境下，需删除vue默认模板的<template></template>（包括自身）中的全部代码：保留<script>和<style>这两个。

代码格式如下：

render(createElement){

return createElement("div",{……},);

}

createElement(1,2,3);这三个参数的意思

1：HTML标签或组件名，若是组件名则需要引入并注册（引入注册和平常一样）。

2：标签修饰

|  |  |  |
| --- | --- | --- |
| class | 样式引入，用Boolean修饰，如：'border-style':true |  |
| style | 样式修饰，‘-’用大写替代，如：fontSize:'80px' |  |
| attrs | 普通HTML特性，如id、placeholder、type、name、href等 |  |
| domProps | DOM属性：如innerHTML:`BAZ` |  |
| on | 事件监听 |  |
| nativeOn | 原生事件监听 |  |
| directives:[] | 自定义指令 |  |

3：标签中的内容

或写具体值【this.$slots.default;】，或标签嵌套格式如下，有分支就另写createElement：

[

createElement(1,2,3);

createElement(1,2,3);

]

约束：

1. VNode必须唯一：组件只能存在一个根节点。
2. 如果需要渲染多个相同格式但不同内容，请使用工厂函数实现

# 插槽slot

<v-slot>……</v-slot>写在组件中，……可以被替换成任意内容，如果使用组件时，没有使用插槽，Vue会默认使用……中的内容。

作用域：插槽只能访问到父中内容（包括data，methods等），无法访问子中data，methods

具名插槽

定义格式：<slot **name=**"header">……</slot>

使用格式：<template **v-slot:**header>……</template> //具名slot必须使用template

作用域插槽

意义：父级可以用子组件中的内容

定义格式：<slot name="slot1" v-bind:user="user">{{ user.lastName }}</slot>

使用格式：<template v-slot:slot1 **v-slot:default="slotProps"**>

{{ **slotProps.**user.firstName }}

<!—slotProps是自定义的变量名，代表子组件中可供父组件使用的变量，注意：作用域插槽不能和具名插槽共用；**v-slot:default** 可以缩写成**v-slot=””**，建议写完整的语法-->

</template>

解构插槽：调用{…}并用子组件中对象替换即可，用于作用域插 槽。

动态插槽名：v-slot:[动态名]//用于具名插槽

# 混入mixin

定义一个vue模板，将这个vue模板中的data，computed，methods的元素混入引用它的模板中，若vue模板中存在引用vue中已定义的data数据，或是方法，对象数据会合并，基础数据会直接忽略vue模板中的内容，方法也是直接忽略模板中内容。

注意：定义混入vue模板一定是js

let mixin = {

data(){

return {

}

},methods:{

}

}

引用：let vm2 = new Vue({ mixins:[mixin],/\*引入mixin\*/

注意：mixin函数引入后就成了vm中的数据了，该表模板中的数据，不会影响到其他引入vm中

# 自定义vue指令

vue提供的供开发者根据需求设定vue指令的功能；搭配钩子函数bind、inserted、update、unbind、componentUpdated使用。

定义：

**directives:{**

**//**格式：'指令名':{钩子函数:(el)=>{el.focus();}},

'focus':{**inserted**:(el)=>{el.focus();}},

**},**

钩子函数

|  |  |
| --- | --- |
| bind | 指令第一次绑定到元素时调用 |
| **inserted** | 被绑定元素插入父节点时调用【比较常用】 |
| update | 被绑定元素所在的模板更新时调用 |
| componentUpdated | 被绑定元素所在模板完成一次更新周期时调用 |
| unbind | 仅调用一次，指令与元素解绑时调用 |

# Vue-Router

1. 安装vue-router插件，使用命令**npm install --save vue-router**
2. 建立router.js文件：【建议建一个router文件夹，其中放vue-router的文件】，js内容如下：

import Vue from 'vue'

import Router from 'vue-router'

import Home from '../views/Home.vue'

Vue.use(Router)

export default new Router({

**mode:"history"**,//history模式，取消#号路径

routes: [//路由配置

{ **path**: '……',**name**: '……',**meta**:{……},**component**: ……,**children**:[……]},

]

})

|  |  |  |
| --- | --- | --- |
| path | 地址栏路径 | /代表根路径，\*代表通配符【404页面】，传参用冒号+参名 |
| name | 名称 | 使用“命名路由”时使用 |
| alias | 别名 | alias="…"，用户访问别名时也访问的path，但地址栏显示为别名 |
| props | 传参 | ：型传参导致耦合过高，可使用props解耦  Boolean形式//子组件prop中需有name接收。  {path: '/p/:name', component: Promotion, props: true }  对象形式：//子组件prop中需有name和age，缺点就是改不了值  {path: '/p',component:Promotion,props:{name:"…",age:"…"}} |
| component | 组件路径 | 直接引入   * () => import('../views/About.vue') * Home   间接引入   * (resolve)=> require([`./views/home.vue`],resolve) |
| redirect | 路由重定向 | { path: '/a', redirect: '/b' }//访问a时重定向到b  { path: '/a', redirect: { name: 'foo' }}  { path: '/a', redirect: to => {return 重定向的字符串路径/路径对象}}  路由重定向不可与component同用 |
| children:[] | 子路由 | [……]中是子路由的信息，唯一的不同是path不用写/了 |
| 编程式导航  【写在methods的方法中】 | | router.push('home')  router.push({ path: 'home'，params:{} })  router.push({ name: 'user', params: { userId: '123' }})//带参  router.push({ path: 'register', query: { plan: 'private'}}) |

1. main.js引入并注册router，直接引入router.js文件并注册即可

import **router** from './router/router'

new Vue({

**router**,store,render: h => h(App)

}).$mount('#app')

1. App.vue使用路由

<router-link to="/">home</router-link><!--路由链接-->

<router-link to="/about">about</router-link><!--路由链接-->

<router-view/><!--路由页面展示处-->

|  |  |
| --- | --- |
| tag="…" | 指定渲染成什么标签，如 tag="li"，渲染成li标签 |
| replace | 导航后无法返回上一个页面；直接添加即可 |
| active-class | 高亮显示当前页面对应的导航菜单项，exact |
| name="…" | 命名视图，一次链接对多个视图，路由就要写成  { path: '/',  components: {  default: Foo,  a: Bar,  b: Baz  }  } |
| $router | $router.push(…):转到……路径下的，括号中写path字符串  $router.go(…)：回退/前进n步，括号中写数字  $router.replace(…):替换当前的历史，括号中写path字符串 |

## 钩子函数

此处易发生内存溢出的错误，需谨慎编程

* to:到达的页面
* from:从哪个页面导航来的
* next:下一步执行

routers.beforeEach((to,from,next)=>{

if(to.path == '/page1'){

next('/page2');

}

else{

next();

}

})

# Vuex

Vuex用于管理共享数据，即多个组件都需要用到的数据；，

1. 首先CMD在项目目录下使用命令：**npm install vuex –save**安装模块vuex模块
2. 创建store.js文件，文件放置store目录下，js内容如下

import Vue from 'vue'//申明使用vue

import vuex from "vuex"//申明使用vuex

import A from "../store/a"//引入子模块

Vue.use(vuex);//使用vue

export default new vuex.Store({

**namespaced**: true,//可选项，命名空间，多用在modules的情况下。

**state**:{/\*数据\*/},

**getters**:{/\*返回state中数据值，相当于computed/filter，get…(state,getters){ return state.… }\*/},

**mutations**:{/\*设置state中数据的值，set…(state,value){ state.… = value; },\*/ }

**actions:**{/\*commit提交给mutation中的方法，支持异步：funName({commit},value){commit(mutations名字,value)}\*/},

modules:{

a:A//引入模块

}

});

export default store;

1. 使用vuex，不建议直接写成this.$store.state.a.name——耦合太高导致不易维护。建议使用下例，引入时，action中带有参数的可以不写参数，因为vuex会自动将参数构造好，但使用时必须传参。

import { mapActions } from 'vuex'

methods:{

...mapActions({

add:'increament',

dec:'decrease',

setName:'a/setName',//使用命名空间下的action

setSex:'a/setSex',//使用命名空间下的action

}),

}

======================================================================================

import { mapState, mapGetters } from 'vuex'

computed:{

...mapState({

count:state=> state.count,

aname:state=> state.a.name,//使用命名空间下的state数据

asex:state=>state.a.sex

}),

...mapGetters({

getCount:'getCount',

getName:'a/getName'//使用命名空间下的getter

}),

}

总结：使用模块化最好使用命名空间，那么使用时就可避免方法名重复的情况，定义：**命名空间名/方法**

# Ajax

## vue-resource

1. 需安装插件 npm install vue-resource
2. main.js引用
3. 直接使用
   1. GET模式

this.$http.get('/someUrl').then(response => {成功回调}, response => {失败回调});

* 1. POST模式

this.$http.post('/someUrl',{ 传递给后台的json格式数据}).then(response => {成功回调}, response => {失败回调});

## vue-axios（推荐使用）

1. 需安装插件：npm install --save axios vue-axios
2. main.js引用axios:

import axios from 'axios'

import VueAxios from 'vue-axios'

Vue.use(VueAxios, axios)

1. 直接使用
   1. GET模式

this.axios.get(`url`).then((response) => { 成功执行函数 }).catch((err)=>{ 失败执行函数 })

* 1. POST模式

this.axios.post(`url`,{传递给后台的json格式数据}).then((response) => { 成功 }).catch((err)=>{ 失败})

# Vue + ES6开发技巧

获取标签对象$event.target.……，后面跟标签属性，属性值可以更改。如获取属性值，获取控件的输入值。

时间格式化：