Vue  
**1,数据驱动---组件化**

1.安装node--- **npm install -g cnpm --registry=https://registry.npm.taobao.org**

2.cnpm I webpack -g

查看版本 npm info webpack

在全局下删除    npm  uninstall  webpack  -g

实现Object.defineProperty(obj,’username’,{set:function(){},get:function(){}});--

初始化一个单页面

3.Cnpm I vue-cli -g 查看版本vue -V（必须大写）===vue --version；

最新版本npm install -g @vue/cli

4. Vue init webpack demo

Vue Init webpack-simple demo

运行

5.Cnpm i

6.Cnpm run dev 设置自动启动 config/index.js-- autoOpenBrowser: true, 控制面板-设置默认程序

生产

Cnpm run build

普通项目

Vue init webpack demo

除了数据属性，Vue 实例还提供了一些有用的实例属性与方法。它们都有前缀 $，以便与用户定义的属性区分开来

vm.$data === data--true

vm.$el === document.getElementById('vue\_det')—true

**模板语法**

1文本**{{msg}}**；

2,htnl代码**v-html=”msg**“;<h1>文本</h1>;

3,属性**v-bind:id**=”index”;或者:id=’index’

4js表达式{{ok?true:false}};

5,**v-text=’**’;

6,指令是带有 v- 前缀的特殊属性**v-if=’’;**

7,过滤器，管道符连接，可以串联**{{msg1|msg2}}**或者**v-bind=’msg1|msg2’**

**{{ message | filterA('arg1', arg2) }}**在**filters:{}**中定义函数

这里，message 是第一个参数，字符串 'arg1' 将传给过滤器作为第二个参数， arg2 表达式的值将被求值然后传给过滤器作为第三个参数。

8.双向绑定<div id="app"> <p>{{ message }}</p> <input **v-model="message"**> </div>

v-model 指令用来在 input、select、textarea、checkbox、radio 等表单控件元素上创建双向数据绑定，根据表单上的值，自动更新绑定的元素的值。<input v-bind:value='msg' v-model='m' ref='inp' />

获取输入框的值；

**v-model实现原理<input placeholder="inputValue2" v-on:input="inputValue ($event.target.value)">**

{{test}}

**8.1用v-model绑定另一个span获取；**

**8.2<input v-bind:value='msg' v-model='m' ref='inp' /> this.$refs.inp.value—如果是span用innerHTML**

8.3复选框<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>

多个的话会被放进**v-modle**绑定的值中，是一个数组，放进去的是value

9.样式绑定：

v-bind:class=”**{active:true或者false}**

v-bind**:class=’[active]”**date里边为active名字，样式名称，也可以为数组三元

v-bind**:style=’{color:curcolor,fontSize:curfontsize+’px’}’**

10.条件渲染

**v-if=’’**

**v-else-if=’’**

**v-else=’’**

**v-show=’’**

**v-cloak=’’;//刷新太快某些html没有加载，隐藏html**

11.事件处理

**v-on:click**=’great’或**者@click**=’great’也可以**v-on:click=’count++**’,或者v-on:click=’count（item）’

v-on**:stop**=’’阻止冒泡—子元素

**v-on:click.stop.prevent**=’’阻止默认行为a标签，submit

v-**on:click.self**=’’

v-**on:click.once**=’’

v-on:keyup,**enter**=’’键盘输入，如：**tab,delete,esc,space,top,left,down,right**;

其他的如hover，自定义事件

<!-- 添加事件侦听器时使用事件捕获模式 -->

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

12.条件循环

v-for 指令需要以 **site in sites** 形式的特殊语法， sites 是源数据数组并且 site 是数组元素迭代的别名。Tips:模板循环时,需要绑定在template上；**也可以迭代对象site为属性值或者第二个参数为属性名,第三个为下标(value, key，index) in object**

<li v-for="site in sites"> {{ site.name }} </li>：

13．计算属性—computed 多对一

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。

computed 属性默认只有 getter ，不过在需要时你也可以提供一个 setter ：

computed:{

site:{

set:function(){

},

get:function(){

}

}

}

14.计算属性watch 一对多，监听某个属性的变化，回调函数处理业务

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

<button @click = "counter++" style = "font-size:25px;">点我</button>直接操作变量+1

Watch:{ kilometers :function(val){}}可以监听到文本框值的变化

Vm.$watch(‘监听的数据’function(新数据，老数据){})

15.指令注册

<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() } })**

// 创建根实例 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() } } } })

Tips

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>input获取</title>

<script src="https://cdn.staticfile.org/vue/2.2.2/vue.min.js"></script>

</head>

<body>

<div id="app">

<input v-bind:value='msg' v-model='m' ref='inp' />

<span>{{m}}</span>

<span>{{mCompute}}</span>

<button v-on:click="reverseMessage" ref='btn'>获取</button>

<p>单个复选框：</p>

<input type="checkbox" id="checkbox" v-model="checked" ref='checkbox1'>

<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>

<span>选择的值为: **{{ checkedNames }}**</span>

<!-- 度个的话value的值会被放入一个数组 即v-modole绑定的文本框 -->

<input @click='f' type='button' value='提交' />

<select v-model="selected" name="fruit" @change="sel">

<option value="">选择一个网站</option>

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

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

</select>

修饰符

**.lazy**

在默认情况下， v-model 在 input 事件中同步输入框的值与数据，但你可以添加一个修饰符 lazy ，从而转变为在 change 事件中同步：

<!-- 在 "change" 而不是 "input" 事件中更新 -->

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

**.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">

这通常很有用，因为在 type="number" 时 HTML 中输入的值也总是会返回字符串类型。

**.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">

</div>

<script>

new Vue({

el: '#app',

data: {

message: 'Runoob!',

msg: 'fasdf',

m: '',

checked: false,

checkedNames: [],

selected: ''

},

computed: {

mCompute: function() {

return this.m ? parseInt(this.m) + 10 : 10

}

},

methods: {

reverseMessage: function() {

console.log(`input:${this.$refs.inp.value}`)

console.log(`btn:${this.$refs.btn.innerHTML}`)

},

chekbook: function() {

console.log(123)

console.log(this.$refs.checkedNames.value)

},

f: function() {

console.log(this.$refs.checkbox1.checked)

console.log(this.checkedNames)

},

sel: function() {

console.log(this.selected)

}

}

})

</script>

</body>

</html>

**组件化-模块化**

全局组件 **Vue.component(‘组件名’，{template:’<h1>组件</h1>’}),**直接放在页面中用；

局部组件child={template: ’<h1>组件</h1>};使用的话需要在vue实例中components{‘组件名’：child}

父组件—》子组件props

**父组件声明引入 import组件名from子组件目录，components:{组件名}；父组件v-bind:属性名，子组件proprs:[属性名]==》单项传递**

**<templateName V-bind:子组件属性=’父组件属性’></templateName>**

**Props:[];**

**子组件—》父组件emits原则上不允许直接修改，比如说套了三层**

**<templateName V-on:子组件事件=’父组件事件’></templateName>**

**This.$emit(‘子组件事件’)；**

**Methods:{‘父组件事件’}**

**父组件v-bind:自定义事件名字=’子组件事件名称’，methods:{父组件事件名称}；子组件methods:{this.$emit(子组件事件名称)}**

**父子组件双向板顶 v-model**

**<v-model-child v-model="txt（父组件数据）"></v-model-child>**

**子组件接受**

**model:{**

**prop:'childTxt',**

**event:'change'**

**},**

**props:{**

**childTxt:String**

**}**

父子组件通信证明，只能从父组件流向子组件

可以使用vuex,全局声明

<router-link></router-link>跳转相当于a标签；this.$router.push({path:””});

<router-view></router-view>展示，渲染位置

Vue内部组件

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title></title>

</head>

<body>

<script src="https://unpkg.com/vue/dist/vue.js"></script>

<script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>

<div id="app">

<h1>Hello App!</h1>

<p>

<!-- 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>

</p>

<!-- 路由出口 -->

<!-- 路由匹配到的组件将渲染在这里 -->

<router-view></router-view>

</div>

<script type="text/javascript">

// 0. 如果使用模块化机制编程，导入 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！

</script>

</body>

</html>

**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。

**This.$router.push({path:””})—编程式路由**

<!-- 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>访问时必须严格按照path路径规定的格式比如path=’/goods/：goodsId’==>localhost:8080/#/goods/321321

其实也就是对history的封装，history.pushState(‘desc-页面描述’,’text-标题’,’/admin-页面路径’)

**动态路由—嵌套路由**

**Router/index.js //对history封装；history.pushState(‘描述’，‘标题’，‘地址’)**

import Vue from 'vue'

import Router from 'vue-router'

import GoodsList from './../views/GoodsList.vue'

import Image from '@/views/Image.vue'

import Title from '@/views/Title.vue'

Vue.use(Router)

export default new Router({

mode:’hash/histroy’

routes: [

{

1. **// path: '/goods/:goodsId/admin/:username', //动态路由—为访问路径，自定义，但是必须按照此种格式通过$route.params.goodsId获得路径参数**

**Router/index.js //对history封装；history.pushState(‘描述’，‘标题’，‘地址’)**

<router-link :to="{name:'mainContent',**'params'**:{links:'recommend'}}">

<router-link :to="{name:'mainContent',**'query'**:{links:'recommend'}}">

**使用params传参时必须用name,用path会被忽略，url中不会出现参数的值(多个参数时会出现最后一个)，页面刷新后参数会消失(可以在路由中path:’index/:id/:age’)避免消失**

**Params—name; query----name/path**

**使用query传参时，url中会出现参数可以用path或者是name，页面刷新后参数不会消失,用query的path传递参数时,切记路由配置不能有：id**

**路径+？的话用query获得参数**

<div>{{msg+**$route.params.goodsId**+$route.params.username}}</div>

path:'/goods',

name: 'GoodsList',

component:GoodsList,，

**redirect:'/w/i'—重定向，设置为首页**

children:[

{

path:'title',**//2.不能加斜杠，否则会变成跟路径--一级路路由//<router-link to='/goods/image'>图片</router-link>**

name:'title',

component:Title

},{

path:'image',//**不能加斜杠，否则会变成根路径--一级路路由**

name:'image',

component:Image

},

]

}

]

})

goodsList组件  
router-link 中必须为绝对路径

**编程路由**

methods:{

jump(){

// this.$router.push('/cart')

this.$router.push({path:'/cart?goodsId=123'})

}

}

组件跳转接收参数$route.params.userId

页面跳转接受参数

<div>这是购物车页面{{$route.query.goodsId}}</div>

**命名路由**

<router-link v-bind:to="{path:'/cart',params:{userId:555}}">命名路由去购物车</router-link>

{

path: "/cart",

name:'cart',

component: Cart

}

命名视图.vue文件中；

<router-view name='title1'></router-view>

<router-view name='image1'></router-view>

.js中配置路由

components:{

default:GoodsList,

title1:Title,

image1:Image

}

**数据请求axios**

el: '#app',

data: {

msg: ''

},

mounted: function() {

axios.interceptors.request.use(function(res) {

console.log('init request')

return res

});

// axios.interceptors.response.use(function(res) {

// setTimeout(function() {

// // console.log('init response'+JSON.stringify(res))

// return res

// }, 1000)

// // return res

// })

},

methods: {

get: function() {

// https://www.osborne.work/spQuery/getActive -跨域

axios.get('../package.json', {

params: {

userId: 'get101'

},

headers: {

token: 'get-header'

}

}).then(res => {

console.log(res)

this.msg = res.data

}).catch(function(err) {

console.log('get:' + err) //真实环境才会报错,比如跨域

})

},

post: function() {

axios.post('http://127.0.0.1:8848/shop/shop/package.json', {

userId: 'post102'

}, {

headers: {

token: 'post-header'

}

}).then(res => {

this.msg = res.data

}).catch(function(err) {

console.log('get:' + err)

})

},

**http**: function() {

axios({

url: '../package.json',

method: 'post',

**data**: {

userId: 'http103post'

}, //此处为post携带的参数

**params**: {

userId: 'http103get'

},

headers: {

token: 'http-header'

},

}).then(res => {

this.msg = res

}).catch(function(err) {

console.log("http--:" + err)

})

},

getUserAccount: function() {

return axios.get('../package.json', {

params: {

userId: 'get101-1'

},

headers: {

token: 'get-header1'

}

});

},

getUserPermissions: function() {

return axios.get('../package.json', {

params: {

userId: 'get101-2'

},

headers: {

token: 'get-header2'

}

});

},

all: function() {

axios.all([this.getUserAccount(), this.getUserPermissions()])

.then(axios.**spread**(function(acct, perms) {

// 两个请求现在都执行完成

console.log(acct)

console.log(perms)

// console.log('acct'+acct)

// console.log('perms'+perms)

}));

}

}

Vue:

Build—打包配置所在文件夹

根目录postcssrc.js为css添加前缀的配置

Config/.index.js—webpack配置，包括端口，代理之类的、包括打包

Main.js—:

webpack.base.conf.js里边配置的，为入口文件。里边包含了entry--output-module{rules: [

{

test: /\.vue$/,

loader: 'vue-loader',

options: vueLoaderConfig

}}；

1. 实例化vue;引入vue

2．引入router(实例化的router)

3.声明模板conpments{}.

4.引入模板template:{}

Main.js:

import Vue from 'vue'

import App from './App'

import router from './router'

import VueLazyload from 'vue-lazyload'

Vue.config.productionTip = false

Vue.use(VueLazyload, {

loading: '../static/loading-svg/loading-bars.svg'

})

/\* eslint-disable no-new \*/

new Vue({

1. el: '#app',// 实例化vue;引入vue

router, 引入router(实例化的router)

components: { App },声明模板conpments{}.

template: '<App/>'引入模板template:{}

})

Index.js:

import Router from 'vue-router'

import GoodsList from '../view/GoodsList'

Vue.use(Router)

export default new Router({

routes: [

{

path: '/',

name: 'GoodsList',

component: GoodsList

}

]

})

Es6

Rest参数

实例

1. 组件引入css

Script标签中Improt ‘’；或者style标签中 @import ‘路径’

1. 生成过程

Main.js（入口）🡪绑定el🡪引入路由router中index.js(控制组件) –>引入app.vue（最终生成的app.js）🡪声明模板components

插槽

2.60之前版本：slot 组件中<slot name=’s’></slot>;使用组件时<component-name><span slot=’s’>goods</span></component-name>

最新：

1. 具名插槽：  
   <slot name="firstnames">11</slot>  
   <template v-slot:firstnames>1-1t--{{info.firstname}}</template>  
   只能用template，默认名字default，如果有名字可省略为# firstnames
2. 作用域插槽  
   <template v-slot:lastnames ="slotProp">  
   {{ slotProp.ss }}  
   <slot name="lastnames" v-bind:ss="childData">  
   把内部信息传递出去
3. 解构插槽  
   <current-user v-slot="{ user = { firstName: 'Guest' } }">

{{ user.firstName }}

</current-user>

有值的话使用，没有值的话使用默认；

4.动态插槽名：<template v-slot:[dynamicSlotName]>

光标放到api处，按下**F1**，可直接在右侧打开对应的帮助文档，

vuex

1. 从**stroe**中读取的状态，stroe状态变，组件也更新；
2. 不能直接更改stroe状态，要显示的通过 **mutations**（mutations:{fn(state,payload(对象或者字符串)){}}）;/ commit(stroe.commit(‘fn’)-组件中

子组件获得this.$store.state.count

1. **Getters**通过this.$store.getters.count相当于computed(计算属性)

单选全选：计算属性computed：原理数组长度，是否和选中长度相等

全局过滤器

import {currency } from './util/currency.js'全局引入main.js

Vue.filter('currency' ,currency)

局部过滤器

import {currency } from './util/currency.js'局部引入

Filters:{currency:function(val){}}

跳转地址，需要两层判断，1.样式，2.js商品大于0；

地址列表： 使用slice 截取数组，limit提取出去，使用计算属性；判断limit改变

渐进式框架：

所有的一切适当的取舍；

声明式渲染 组件系统 路由系统 状态管理

**操作dom:**

**Dom(webcore)和js(v8)在浏览器(chrom)**中分在不同的模块,浏览器**重绘重排,性能差**

1. **解析html生成dom树，解析css生成css规则树，生成渲染树，包含背景色、边框之类的**
2. 重绘边框之类的不便，背景变
3. 重排，边框变

vue为何快？

1. **虚拟dom树：vue模板被vue编译器编译为一个渲染函数，渲染函数被调用时会渲染和返回一个一个虚拟dom,同时vue内部也有响应系统检测数据变化，当变化时，重新渲染虚拟Dom，新树久树比较较，再追加到真是dom上**

**Vue data,组件渲染时，每一个属性会被添加getter和setter属性，同时添加属性到（用到的getter）watcher,改变时触发setter=》需要重新渲染**

1. 数据驱动
2. 路由：作用就是让组件树和url对应起来

Vuex

1. state
2. getters
3. mutations
4. actions 通过context.commit()提交mutations,可以是异步的
5. modules: { a: moduleA, b: moduleB }

mouduleA={state..getters..mutations…actions}挂载在vuex上;store.state.a // -> moduleA 的状态

传代码。SIH