|  |
| --- |
| 总结: this 8种: 判断this，~~一定不要看定义在哪儿！~~只看调用时！  1. obj.fun() this->obj  2. fun() 或 (function(){ ... })() 或 多数回调函数 或 定时器函数 this->window  3. new Fun() this->new正在创建的新对象  4. 类型名.prototype.共有方法=function(){ ... } this->将来谁调用指谁，同第一种情况  5. DOM或jq中事件处理函数中的this->当前正在触发事件的DOM元素对象  如果需要使用简化版函数，必须$(this)  6. 箭头函数中的this->箭头函数外部作用域中的this  7. jQuery.fn.自定义函数=function(){ ... } this->将来调用这个自定义函数的.前的jQuery子对象，~~不用再$(this)~~  8. new Vue()中methods中的函数中的this->当前new Vue()对象 |

总结:

1. MVVM: 界面View+模型Model+视图模型ViewModel

2. Vue绑定原理: 访问器属性+虚拟DOM树

变量被修改时: 访问器属性发出通知，虚拟DOM树扫描并仅更新受影响的元素

3. 虚拟DOM树优点:

(1). 小: 只包含可能变化的元素。

(2). 遍历查找快

(3). 修改效率高: 只修改受影响的元素。

(4). 避免重复编码: 已封装DOM增删改查代码

4. Vue功能3步:

(1). 先创建增强版的界面:

a. 整个界面必须包含在一个唯一的父元素下:

通常是<div id="app">  
 b. 可能变化的元素内容用{{自定义变量名}}标记

c. 触发事件的元素用@click="自定义处理函数名"标记

(2). 再创建new Vue()对象,其中el:指向new Vue()要监控的页面区域

(3). 在new Vue()对象内定义模型对象data和methods

a.界面所需的所有变量都放在data中

b.界面所需的所有事件处理函数都放在methods中

5. 总结: 绑定语法+13种指令

(1). 如果元素的内容需要随变量自动变化: {{}}

(2). 如果元素的属性值需要随变量自动变化: :

(3). 控制一个元素显示隐藏: v-show //使用display:none隐藏元素

(4). 控制两个元素二选一显示: v-if v-else //使用删除元素方式隐藏元素

(5). 多个元素多选一显示: v-if v-else-if v-else

(6). 只要反复生成多个相同结构的元素组成列表时: v-for :key="唯一标识"

强调: 为什么必须加:key="i"？给每个元素副本添加唯一标识。修改数组中某个元素值时，避免重建整个列表，只需要修改一个DOM元素副本即可！提高修改效率。

(7). 只要绑定事件: @ $event

(8). 防止用户短暂看到{{}}: v-cloak和v-text

(9). 只要绑定原始HTML代码片段内容: v-html

(10). 如果元素的内容只在首次加载时绑定一次，之后都不会改变: v-once

优化: 减少虚拟DOM树中元素个数。

(11). 保护内容中的{{}}不被编译: v-pre

(12). 今后只要想获得表单元素的值或状态: v-model

6. 绑定样式:

(1). 需要精确修改某一个css属性，就绑定style:

a. <元素 style="固定样式" :style="{css属性:变量名, ...}"

data:{  
 变量名:css属性值

... : ...

}

b. <元素 style="固定样式" :style="变量名"

data:{  
 变量名:{

css属性名: 属性值,

... : ...

}

}

(2). 只要批量修改一个元素的多个css属性就绑定class

a. <元素 class="固定class" :class="{class名:变量名, ...}"

data:{  
 变量名:true或false,

... : ...

}  
 b. <元素 class="固定class" :class="变量名"

data:{  
 变量名:{  
 class名:true或false,

... : ...

}

}

7. 只要希望在页面加载时自动对元素执行一些初始化操作时就用自定义指令:

(1). 添加自定义指令:

Vue.directive("自定义指令名",{

inserted(domElem){  
 对domElem执行DOM操作

}

})

(2). 使用自定义指令:

<元素 v-自定义指令名>

8. 今后只要根据其他变量的值动态计算出一个属性值就用计算属性:

<元素>{{计算属性}}</元素>

new Vue({

el:"#app",

data:{...},

methods:{...},

computed:{  
 计算属性名(){

计算过程

return 计算结果

}  
 }

})

9. 希望将变量的原始值先加工后再显示给用户看时就用过滤器:

Vue.filter("过滤器名",function(oldVal, 自定义形参,...){

return 加工后的新值

})

<元素>{{ 变量 | 过滤器(实参值, ...) | ... }}</元素>

10. 只要在vue中发送ajax请求，就用axios

axios.defaults.baseURL="服务器端接口的公共基础地址部分"

axios.get(  
 "服务器端接口地址的相对路径",

{

params:{ 参数名: 参数值, ... }

}  
).then(result=>{

... result.data...

})

或

axios.post(  
 "服务器端接口地址的相对路径",

"参数名1=参数值1&参数名2=参数值2&..."  
).then(result=>{

... result.data...

})

强调: 在vue内使用axios，then中必须用箭头函数，保持then内this与外部this一致，都指向当前new Vue()对象

11. vue生命周期4个阶段 8个钩子函数

beforeCreate(){ ... }

(1). 创建(create)

created(){ ... }

beforeMount(){ ... }

(2). 挂载(mount)

mounted(){ ... 经常在这里发送ajax请求 ... }

beforeUpdate(){ ... }

(3). 更新(update)

updated(){ ... }

beforeDestroy(){ ... }

(4). 销毁(destroy)

destroyed(){ ... }

12. 只要希望重用一块独立的功能区域就用组件:

(1). 定义组件

Vue.component(`组件标签名`,{

template:`HTML内容片段`,

data(){ return { 变量 } },

//其余和new Vue()完全相同

})

(2). 在HTML中使用自定义组件

<组件标签名/>或双标记也行

(3). 原理: new Vue()扫描到自定义组件标签，

a.组件的template中的HTML内容代替页面中<组件标签>位置。

b. 并为这个小区域专门创建一个缩微版的vue类型对象。

1). 调用组件的data()函数为当前组件副本创建一个专属数据对象副本。

2). 引入组件对象中的methods等其他内容到当前组件对象副本中

13. 组件化开发:  
(1). 步骤:  
 a. 拿到页面先划分功能区域

1). 从上到下，按功能不同划分区域

2). 按是否重用划分

b. 为每个组件创建独立的.js文件，其中包含一个组件对象及其内容

c. 将所有组件引入唯一完整的html页面中，并在<div id=”app”></div>中添加父组件标签。  
(2). 运行时:

a. new Vue()扫描<div id=”app”>，发现父组件标签，创建并替换父组件

b. 父组件扫描自己内部的template内容，创建并替换子组件

(3). 三种组件:

a. 根组件: new Vue()

b. 全局组件: Vue.component(...)

c. 子组件: 3步

1). var 子组件对象名={

内容必须符合组件的要求

}

子组件对象名必须是驼峰命名

2). 父组件对象中:{

... ...

components{ 子组件对象名, ... ,... }

}

子组件对象名必须是驼峰命名

3). 父组件template中用<子组件标签名/>引入子组件内容

components会将子组件对象名的驼峰命名自动翻译为-分隔

所以, 使用子组件标签时，要用-分隔多个单词

(2). 组件间传参: 父给子

a. 父组件给:

<子组件 :自定义属性名="父组件变量">

b. 子组件取:

props:["自定义属性名"]

结果: 在子组件内，props中的"自定义属性名"与子组件自己data中的变量用法完全相同！

14. SPA

(1). 3步:

a. 先创建唯一完整的HTML页面

1). 包含vue基本的页面结构

<div id="app"> new Vue({el:"#app"})

2). 引入所有必要的文件和组件

vue-router.js, 其它页面或组件文件, router.js路由器对象所在文件

3). <div id="app">中用<router-view/>为今后页面组件预留空位

b. 再为每个页面组件创建独立的文件。每个页面组件其实都是一个子组件

c. 创建router.js文件，创建路由器对象

1). 创建路由字典对象:

var routes=[

{path:"/", component:首页组件对象名},

{path:"/相对路径" , component: 其它页面组件对象名},

{path:"\*", component: 404页面组件对象 }

]

2). 创建路由器对象，并将路由字典对象转入路由器对象中  
 var router=new VueRouter({ routes })

3). 将router对象加入到new Vue()中

回到唯一完整的HTML页面中: new Vue({ el:"#app", router })

(2). 页头等全局组件：

a. 创建独立的文件保存页头组件的内容

b. 使用Vue.component("my-header",{ ... })将页头创建为全局组件

c. 在唯一完整的HTML页面中引入页头组件文件

d. 使用页头组件标签<my-header/>: 2种:

1). 如果所有页面都有统一的页头:

就放在唯一完整的html页面中<router-view>外部上方

2). 如果有的页面有页头，有的页面没有页头:

就只放在需要页头的组件中的template中

(3). 路由跳转: 2种:

a. html中: <router-link to="/相对路径">文本<router-link>

b. js中: this.$router.push("/相对路径")

(4). 路由传参:

a. 修改路由字典:

{path:"/相对路径/:自定义参数名", component:页面组件对象, props:true}

b. 跳转时:

<router-link to="/相对路径/参数值"

或

this.$router.push("/相对路径/参数值")

c. 下一个页面接:

1). props:[ '自定义参数名' ]

2). 可将"自定义参数名"用作绑定或程序中都行

15. 安装vue脚手架:

(1). 设置npm默认使用国内淘宝镜像仓库:

npm config set registry http://registry.npm.taobao.org

备选: npm i -g cnpm --registry=http://registry.npm.taobao.org

(2). 安装可反复生成项目脚手架代码的工具:

npm i -g @vue/cli

备选: cnpm i -g @vue/cli

(3). npm run serve

启动vue脚手架项目  
(4). 打开浏览器，地址栏输入: http://localhost:8080

16. 脚手架文件夹结构:

(1). 唯一完整的HTML页面: 一分为三:

a. public文件夹

1). 图片img文件夹放在public文件夹下

2). 第三方css的压缩版和第三方js的压缩版都放在public文件夹下

3). 唯一完整的HTML文件index.html中，head中引入第三方的css和js

b. src/App.vue

1). <template>下只包含公共的页头组件和<router-view>

2). <style>下包含所有网页都要用到的公共css样式，比如css重置代码

c. src/main.js

1). import引入App.vue，router，axios，以及其他全局组件

2). 将全局组件对象转为真正的全局组件: Vue.component( "组件标签名", 全局组件对象 )

3). 配置axios并放入原型对象中:

axios.defaults.baseURL="服务器端基础路径"

Vue.prototype.axios=axios;

(2). 为每个页面创建.vue组件文件，都放在src/views文件夹下。每个.vue文件中:

a. <template>标签内，包含这个页面的HTML内容

b. <script>export default{ ... }</script>中包含组件对象的js内容。

c. <style>标签内包含仅这个页面组件内才使用的css

d. <template>中的HTML内容以及<script>export default{...}</script>中的js内容，和前四天将的是完全一样的写法，绑定，指令，函数，生命周期，axios请求等都一样。前四天怎么用，这里就怎么用。

(3). 路由字典和路由器对象，在src/router/index.js文件中

a. 仅import首页组件对象，不要过早引入其它页面组件

b. 路由字典中首页组件: { path:"/", component:首页组件对象}

c. 其余页面组件都做成懒加载:

{

path: '/相对路径',

component: () => import(/\* webpackChunkName: "组件名" \*/ '../views/其它页面组件.vue')

}

(4). 全局组件都放在src/components文件夹下，每个全局组件.vue文件中。但是，全局组件必须在main.js引入，并用Vue.component()转化为真正的全局组件，才能在其它组件的HTML中使用。

(5). 运行时: 路由器router对象监视浏览器地址栏路径变化，并查找对应的页面组件内容，先代替App.vue中的<router-view>，然后main.js再将包含要加载的页面内容的App.vue组件内容，替换到唯一完整的index.html中空<div id="app">位置。

17. 懒加载:

(1).异步延迟下载:

src/router/index.js

//不要import Details from "../views/Details"

const routes=[  
 ...

{

path:"/details/:lid",

component: () => import(

/\* webpackChunkName: "details" \*/

'../views/Details.vue'

),

props:true  
]

(2).彻底懒加载:

项目根目录创建vue.config.js

module.exports={

chainWebpack:config=>{

config.plugins.delete("prefetch")

//删除index.html开头的带有prefetch属性的link，不要异步下载暂时不需要的页面组件文件

},

}

18. http-proxy方式跨域:

在服务器端没有配置CORS或JSONP跨域的情况下:   
(1). vue.config.js中

module.exports={  
 ... ,

devServer: {

proxy: {

// detail: https://cli.vuejs.org/config/#devserver-proxy

'/api': {

target: `http://服务器端接口地址统一前缀部分`,

changeOrigin: true,

pathRewrite: {

'^/api' : '' //将程序中的/api，替换为空字符串，再和target中的基础路径拼接起来作为发送到服务器的最终请求地址。

}

}

}

}  
}  
(2). main.js中:   
...

axios.defaults.baseURL="/api"

19. 避免组件间样式冲突:

(1)<style scoped>  
(2)<template>  
 <组件父元素 class="组件名">  
 </template>  
 <style>  
 .组件名>子元素选择器{ ... }  
 </style>

20. watch+事件修饰符+防抖:

(1). 希望变量一变化就自动执行一项操作时:

data(){  
 return { 变量: 值 }  
},

watch:{  
 变量(){  
 要执行的操作

}  
}  
(2). 事件修饰符:

@事件.13 按回车时才执行

@事件.stop 阻止冒泡

@事件.prevent 阻止默认行为

(3). 防抖:

data(){

return {

... ,

timer:null

}

},

methods:{

查找方法(){

if(this.timer!=null){

clearTimeout(this.timer)

}

this.timer=setTimeout(()=>{

正式的查找操作

},等待时间)

}

}