**vue基础**

1 vm.$mount('#root')

el的两种写法

        const v = new Vue({

            //el:'#root', //第一种写法

        })

        v.$mount('#root') //第二种写法

2 MVVM模型

         1. M：模型(Model) ：data中的数据

         2. V：视图(View) ：模板代码

        3. VM：视图模型(ViewModel)：Vue实例

            观察发现：

         1.data中所有的属性，最后都出现在了vm身上。

         2.vm身上所有的属性 及 Vue原型上所有属性，在Vue模板中都可以直接使用。

3 Object.defineProperty

       let obj = {x:100}; let obj2 = {};

            Object.defineProperty(obj2,'x',{

                get(){

                    return obj.x

                },

                set(value){

                    obj.x = value

                }

            })

    通过Object.defineProperty()把data对象中所有属性添加到vm上。

    为每一个添加到vm上的属性，都指定一个getter/setter。

    在getter/setter内部去操作（读/写）data中对应的属性。

4 Vue中的事件修饰符：

    1.prevent：阻止默认事件（常用）；

    2.stop：阻止事件冒泡（常用）；

    3.once：事件只触发一次（常用）；

    4.capture：使用事件的捕获模式；

    5.self：只有event.target是当前操作的元素时才触发事件；

    6.passive：事件的默认行为立即执行，无需等待事件回调执行完毕；

5 Vue中常用的按键别名

            回车 => enter

            删除 => delete (捕获“删除”和“退格”键)

            退出 => esc

            空格 => space

            换行 => tab (特殊，必须配合keydown去使用)

            上 => up

            下 => down

            左 => left

            右 => right

系统修饰键（用法特殊）：ctrl、alt、shift、meta

            (1).配合keyup使用：按下修饰键的同时，再按下其他键，随后释放其他键

            (2).配合keydown使用：正常触发事件。

Vue.config.keyCodes.huiche = 13 //定义了一个别名按键

@keydown.huiche="showInfo"

6 computed:{

       fullName:{

          get(){},

          set(value){}

         }

        }

7 watch:{

    isHot:{

        deep:true,

        immediate:true,

        handler(newValue,oldValue){}

    }

}

8 绑定class样式

:class="mood" mood:'normal',

:class="classArr" classArr:['atguigu1','atguigu2','atguigu3'],

:class="classObj"

classObj:{

           atguigu1:false,

           atguigu2:false,

                }

9 绑定style样式

:style="styleObj"

styleObj:{

           fontSize: '40px',

           color:'red',

                }

:style="styleArr"

styleArr:[

          {

             fontSize: '40px',

             color:'blue',

              },

           {

                backgroundColor:'gray'

               }

             ]

10 条件渲染：v-if   v-show

            1.v-if

                        写法：

                                (1).v-if="表达式"

                                (2).v-else-if="表达式"

                                (3).v-else="表达式"

                        适用于：切换频率较低的场景。

                        特点：不展示的DOM元素直接被移除。

                        v-if v-else-if、v-else一起使用，结构不能被“打断”。

            2.v-show

                        写法：v-show="表达式"

                        适用于：切换频率较高的场景。

                        特点：不展示的DOM元素未被移除，仅仅是使用样式隐藏掉

            3.备注：使用v-if的时，元素可能无法获取到，而使用v-show可以获取到。

11 v-for指令:

        1.用于展示列表数据

        2.语法：v-for="(item, index) in xxx" :key="yyy" //key必须使用

        3.可遍历：数组、对象、字符串（用的很少）、指定次数（用的很少）

12 面试题：react、vue中的key有什么作用？（key的内部原理）

1. 虚拟DOM中key的作用：

        key是虚拟DOM对象的标识，Vue会根据【新数据】生成【新的虚拟DOM】,

        随后Vue进行【新虚拟DOM】与【旧虚拟DOM】的差异比较，比较规则如下：

2.对比规则：

        旧虚拟DOM中找到了与新虚拟DOM相同的key：

            ①.若虚拟DOM中内容没变, 直接使用之前的真实DOM！

            ②.若虚拟DOM中内容变了, 则生成新的真实DOM

        旧虚拟DOM中未找到与新虚拟DOM相同的key

            创建新的真实DOM，随后渲染到到页面。

3. 用index作为key可能会引发的问题：

            若对数据进行：逆序添加、逆序删除等破坏顺序操作:

            如果结构中还包含输入类的DOM：会产生错误DOM更新

13 模拟一个数据监测

let data = {

    name:'尚硅谷',

    address:'北京',

}

//创建一个监视的实例对象，用于监视data中属性的变化

const obs = new Observer(data)

//准备一个vm实例对象

let vm = {}

vm.\_data = data = obs

function Observer(obj){

    //汇总对象中所有的属性形成一个数组

    const keys = Object.keys(obj)

    keys.forEach((k)=>{

        Object.defineProperty(this,k,{

            get(){

                return obj[k]

            },

            set(val){

                console.log(`${k}被改了，我要去解析模板，生成虚拟DOM`)

                obj[k] = val

            }

        })

    })

}

14 Vue.set(this.student,'sex','男') Vue监视数据的原理

Vue监视数据的原理：

    1. vue会监视data中所有层次的数据。

    2. 如何监测对象中的数据？

        通过setter实现监视，且要在new Vue时就传入要监测的数据。

        对象中，后追加的属性，Vue默认不做响应式处理

        如需给后添加的属性做响应式，请使用如下API：

                Vue.set(this.student,'sex','男')

                this.$set(this.student,'sex','男')

    3.在Vue修改数组中的某个元素一定要用如下方法：

        1.push()、pop()、shift()、unshift()、splice()、sort()、reverse()

        2.Vue.set() 或 vm.$set()

15 v-model的三个修饰符：

         lazy：失去焦点再收集数据

         number：输入字符串转为有效的数字

         trim：输入首尾空格过滤

16 Vue.filter(name,callback) 或 new Vue{filters:{}}

备注：

        1.过滤器也可以接收额外参数、多个过滤器也可以串联

        2.并没有改变原本的数据, 是产生新的对应的数据

{{time | timeFormater('YYYY\_MM\_DD') | mySlice}}

Vue.filter('mySlice',function(value){

            return value.slice(0,4)

        })

filters:{

           timeFormater(value,str='YYYY年MM月DD日 HH:mm:ss'){

                return dayjs(value).format(str)

                }

            }

17 自定义指令 Vue.directive directives

      //定义全局指令

         Vue.directive('fbind',{

            //指令与元素虚拟dom成功绑定时

            bind(element,binding){

                element.value = binding.value

            },

            //指令所在元素被插入页面时

            inserted(element,binding){

                element.focus()

            },

            //指令所在的模板被重新解析时

            update(element,binding){

                element.value = binding.value

            }

        })

  //定义局部指令

directives:{

                //1.指令与元素虚拟dom成功绑定时（一上来）。

                //3.指令所在的模板被重新解析时。

                big(element,binding){

                    element.innerText = binding.value \* 10

                },

                fbind:{

                    //指令与元素虚拟dom成功绑定时（一上来）

                    bind(element,binding){

                        element.value = binding.value

                    },

                    //指令所在元素被插入页面时

                    inserted(element,binding){

                        element.focus()

                    },

                    //指令所在的模板被重新解析时

                    update(element,binding){

                        element.value = binding.value

                    }

                }

            }

18 分析生命周期 mounted() {}

            beforeCreate() {}      created() {}

            beforeMount() {}       mounted() {}

            beforeUpdate() {}      updated() {}

            beforeDestroy() {}     destroyed() {}

19 Vue.component中使用组件

  Vue中使用组件的三大步骤：

            一、定义组件(创建组件)

                Vue.extend(options)

产生了VueComponent的构造函数

            二、注册组件

                1.局部注册：靠new Vue的时候传入components选项

                2.全局注册：靠Vue.component('组件名',组件)

            三、使用组件(写组件标签)

                <School></School>

相当于new VueComponent(options)

组件名字写法

<School></School>

<MySchool></MySchool>

<MySchool/>

定义组件(创建组件)可以简写，

因为Vue.component('组件名',组件)注册组件,内部包含了定义组件

        const school = Vue.extend(options)

        可简写为：const school = options

1.重要的关系：VueComponent.prototype.\_\_proto\_\_ === Vue.prototype

2.为什么要有这个关系：让组件实例对象（vc）可以访问到 Vue原型上的属性、方法。

**vue-cli**

1 render: h => h(App)

    关于不同版本的Vue：

  1.vue.js与vue.runtime.xxx.js的区别：

       (1).vue.js是完整版的Vue，包含：核心功能+模板解析器。

       (2).vue.runtime.xxx.js是运行版的Vue，只包含：核心功能；没有模板解析器。

2.因为vue.runtime.xxx.js没有模板解析器，不能使用template配置项，需要使用

render函数接收到的createElement函数去指定具体内容。

new Vue({

    el:'#app',

    //render函数完成了这个功能：将App组件放入容器中

  render: h => h(App),

    // template:`<h1>你好啊</h1>`,

    // components:{App},

})

2 this.$refs.btn

<button ref="btn">DOM元素</button>

<School ref="sch"/>

 this.$refs.sch//真实DOM元素

 this.$refs.btn

3 props配置

<Student name="李四" sex="女" :age="18"/>

Student.vue

//简单声明接收

    props:['name','age','sex']

//接收的同时对数据进行类型限制

    props:{

    name:String,

    age:Number,

    sex:String

}

//接收的同时对数据：进行类型限制+默认值的指定+必要性的限制

props:{

    name:{

        type:String, //name的类型是字符串

        required:true, //name是必要的

    },

    age:{

        type:Number,

        default:99 //默认值

    },

    sex:{

        type:String,

        required:true

    }

}

4 \_src\_mixin混入(合)

mixin.js

export const hunhe = {

    methods: {

        showName(){

            alert(this.name)

        }

    },

    mounted() {

        console.log('你好啊！')

    },

}

import {hunhe} from '../mixin'

    export default {

        name:'Student',

        data() {

            return {

                name:'张三',

                sex:'男'

            }

        },

         mixins:[hunhe]

    }

5 \_src\_插件

import plugins from './plugins'

Vue.use(plugins,1,2,3)

export default {

    install(Vue,x,y,z){

        console.log(x,y,z)

        //全局过滤器

        Vue.filter('mySlice',function(value){

            return value.slice(0,4)

        })

        //定义全局指令

        Vue.directive('fbind',{

            //指令与元素成功绑定时（一上来）

            bind(element,binding){

                element.value = binding.value

            },

            //指令所在元素被插入页面时

            inserted(element,binding){

                element.focus()

            },

            //指令所在的模板被重新解析时

            update(element,binding){

                element.value = binding.value

            }

        })

        //定义混入

        Vue.mixin({

            data() {

                return {

                    x:100,

                    y:200

                }

            },

        })

        //给Vue原型上添加一个方法（vm和vc就都能用了）

        Vue.prototype.hello = ()=>{alert('你好啊')}

    }

}

6 <style scoped>

7 nanoid随机id

import {nanoid} from 'nanoid'

const todoObj = {id:nanoid(),title:this.title,done:false}

8 localStorage sessionStorage

localStorage.setItem('msg','hello!!!')

localStorage.getItem('msg')

localStorage.removeItem('msg')

localStorage.clear()

const result = localStorage.getItem('msg')

JSON.parse(result)

sessionStorage.setItem('msg','hello!!!')

sessionStorage.getItem('msg')

sessionStorage.removeItem('msg')

sessionStorage.clear()

const result = sessionStorage.getItem('msg')

9子传父，绑定属性传入方法 :checkTodo="checkTodo"

MyList.vue

//绑定属性传入方法

<MyItem :checkTodo="checkTodo"/>

import MyItem from './MyItem'

methods: {

    checkTodo(id){

        this.todos.forEach((todo)=>{

            if(todo.id === id) todo.done = !todo.done

        })

}

}

MyItem.vue

this.checkTodo(id)

10 this.$emit自定义事件，子传父

App.vue

(1)<Student @atguigu="getStudentName" />

(2)<Student ref="student"/>

this.$refs.student.$on('atguigu',this.getStudentName)

getStudentName(name,...params){

      console.log('App收到了学生名：',name,params)

          this.studentName = name

        }

//接收子组件的数据

Student.vue

 this.$emit('atguigu',this.name,666,888,900)

//触发事件，向父组件传入参数

11 全局事件总线 this.$bus.$on

main.js

new Vue({

    el:'#app',

    render: h => h(App),

    beforeCreate() {

        Vue.prototype.$bus = this //安装全局事件总线

    },

})

//接收数据的地方

this.$bus.$on('hello',(data)=>{

     console.log('我是School组件，收到了数据',data)

})

//传入数据的地方

this.$bus.$emit('hello',this.name)

12 pubsub.subscribe，消息订阅与发布

import pubsub from 'pubsub-js'

pubsub.subscribe('hello',(msgName,data)=>{}) //接收数据

pubsub.publish('hello',666) //传入数据

13 this.$nextTick 数据插入真实dom后运行，可以获得真实dom

this.$nextTick(function(){

         this.$refs.inputTitle.focus()

       })

14 transition 过度与动画

   <transition name="hello" appear>

            <h1 v-show="isShow">你好啊！</h1>

   </transition>

    .hello-enter-active{

        animation: atguigu 0.5s linear;

    }

    .hello-leave-active{

        animation: atguigu 0.5s linear reverse;

    }

    @keyframes atguigu {

        from{

            transform: translateX(-100%);

        }

        to{

            transform: translateX(0px);

        }

}

        <transition-group name="hello" appear>

            <h1 v-show="!isShow" key="1">你好啊！</h1>

            <h1 v-show="isShow" key="2">尚硅谷！</h1>

        </transition-group>

    .hello-enter,.hello-leave-to{

        transform: translateX(-100%);

    }

    .hello-enter-active,.hello-leave-active{

        transition: 0.5s linear;

    }

    /\* 进入的终点、离开的起点 \*/

    .hello-enter-to,.hello-leave{

        transform: translateX(0);

    }

<transition-group

            appear

            name="animate\_\_animated animate\_\_bounce"

            enter-active-class="animate\_\_swing"

            leave-active-class="animate\_\_backOutUp"

        >

            <h1 v-show="!isShow" key="1">你好啊！</h1>

            <h1 v-show="isShow" key="2">尚硅谷！</h1>

        </transition-group>

15 devServer配置代理服务器，避免跨域

axios.get('http://localhost:8080/demo/cars').then(

        response => {

           console.log('请求成功了',response.data)

          }

         )

vue.config.js

module.exports = {

devServer: {

    proxy: {

      '/demo': {

        target: 'http://localhost:5001',

        pathRewrite:{'^/demo':''},

      }

    }

  }

}

16 this.$http.get

import vueResource from 'vue-resource'

Vue.use(vueResource)

this.$http.get(`https://api.github.com/users?q=${this.keyWord}`).then(

    response => {

        console.log(response.data)

    },

    error => {}

)

17 slot

App.vue

<Category title="美食" >

   <img slot="center" src="srJlq0.jpg" alt="">

    <a slot="footer" href="http://www.atguigu.com">更多美食</a>

</Category>

Category.vue

<template>

    <div class="category">

        <slot name="center">当使用者没有传递具体结构时，我会出现1</slot>

        <slot name="footer">当使用者没有传递具体结构时，我会出现2</slot>

    </div>

</template>

18 vuex的模块化写法

calculation.js

const state={

    result:0

}

const actions={

    calculate(context,calculateValue){

        context.commit("CALCULATE",result)

    }

}

const mutations={

    CALCULATE(state,result){

       state.result=result

    }

}

const getters={

    bigResult(state){

        return state.result\*10

    }

}

const calculation={

    namespaced:true,

    state,

    mutations,

    actions,

    getters

}

export default calculation;

//store的index.js

import Vue from 'vue'

import Vuex from "vuex"

Vue.use(Vuex)

import calculation from './calculation'

 export default new Vuex.Store({

    modules:{

        calculation

    }

})

//使用vuex

//高级写法

import {mapGetters,mapActions,mapMutations,mapState} from 'vuex'

    ...mapMutations('countAbout',{increment:'JIA',decrement:'JIAN'}),

    ...mapState("calculation",["result"]),

    ...mapGetters("calculation",["bigResult"]) ,

    ...mapActions("calculation",["calculate"]),

//普通写法

    this.$store.state.personAbout.personList

    this.$store.getters['personAbout/firstPersonName']

    this.$store.commit('personAbout/ADD\_PERSON',personObj)

    this.$store.dispatch('personAbout/addPersonWang',personObj)

19 router-link query params active-class replace name

<router-link

replace

    class="list-group-item"

    active-class="active"

    :to="{ name: 'message', query: { message } }"

>message</router-link>

    //接受传递的参数query

     this.$route.query.message

//以params的形式进入路由间的传递参数

<router-link

replace

    class="list-group-item"

    active-class="active"

    :to="{

    name: 'news',

    params: {

        newsMsg,

    },

    }"

>News</router-link>

    name: "news",

    //传递变量的方式是params时必须用:newsMsg占位

    path: '/news/:newsMsg',

    component: News,

//接受传递的参数params

$route.params.newsMsg

20 props三种配置方法

routes:[

{

    path:'message',

    component:Message,

    children:[

        {

            name:'xiangqing',

            path:'detail',

            component:Detail,

            //props的第一种写法，

            props:{a:1,b:'hello'}

            //收到的所有params参数，以props的形式传给Detail组件。

            props:true

            //props的第三种写法，值为函数

            props($route){

                return {

                    id:$route.query.id,

                    title:$route.query.title,

                    a:1,

                    b:'hello'

                }

            }

        }

     ]

}

]

21 缓存路由组件 keep-alive include="News"

  <!-- 缓存多个路由组件 -->

            <keep-alive :include="['News','Message']">

            <!-- 缓存一个路由组件 -->

            <keep-alive include="News">

                <router-view></router-view>

            </keep-alive>

22 activated() deactivated()两个组件生命周期钩子

   activated() {

        console.log('News组件被激活了')

        this.timer = setInterval(() => {

        this.opacity -= 0.01

         if(this.opacity <= 0) this.opacity = 1

            },16)

        },

   deactivated() {

       console.log('News组件失活了')

       clearInterval(this.timer)

        },

23 beforeEach afterEach meta全局路由守卫

routes:[

{

    name:'xinwen',

    path:'news',

    component:News,

    meta:{isAuth:true,title:'新闻'}

}

]

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

    console.log('前置路由守卫',to,from)

    if(to.meta.isAuth){ //判断是否需要鉴权

  next()

    }else{

        next()

    }

})

//全局后置路由守卫————初始化的时候被调用、每次路由切换之后被调用

router.afterEach((to,from)=>{

    console.log('后置路由守卫',to,from)

    document.title = to.meta.title || '硅谷系统'

})

24 beforeEnter独享路由守卫

routes:[

{

    name:'xinwen',

    path:'news',

    component:News,

    meta:{isAuth:true,title:'新闻'},

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

        console.log('独享路由守卫',to,from)

        if(to.meta.isAuth){ //判断是否需要鉴权

            next()

        }else{

            next()

        }

    }

}

]

25 beforeRouteEnter beforeRouteLeave组件内路由守卫

About.vue

beforeRouteEnter (to, from, next) {

    if(to.meta.isAuth){

        next()

    }else{

        next()

    }

}

beforeRouteLeave (to, from, next) {

    next()

}

43\_src\_history模式与hash模式

在router里面配置

const router =  new VueRouter({

    mode:'history',

    routes:[



1. 对于一个url来说，什么是hash值？—— #及其后面的内容就是hash值。

2. hash值不会包含在 HTTP 请求中，即：hash值不会带给服务器。

3. hash模式：

   1. 地址中永远带着#号，不美观 。

   2. 若以后将地址通过第三方手机app分享，若app校验严格，则地址会被标记为不合法。

   3. 兼容性较好。

4. history模式：

   1. 地址干净，美观 。

   2. 兼容性和hash模式相比略差。

   3. 应用部署上线时需要后端人员支持，解决刷新页面服务端404的问题。