【vue核心知识点】

1、对于Vue是一套渐进式框架的理解？

就是vue没有多做职责之外的事情，它是渐进的，可以在原有的大系统的上面，把一两个组件改用它实现，就是当成jQuery来使用；

可以整个用它全家桶开发，当Angular来使用；

它只是个轻量视图而已，只做了自己该做的事，没有做不该做的事

2、vue.js的两个核心是什么?

双向数据绑定；组件化

3、请问v-if和v-show有什么区别？

（1）共同点：

都是动态显示DOM元素

（2）区别：

1）手段：

v-if是动态的向DOM树内添加或者删除DOM元素；

v-if是动态的向DOM树内添加或者删除DOM元素；

2）编译过程：

v-if切换有一个局部编译/卸载的过程，切换过程中合适地销毁和重建内部的事件监听和子组件；

v-show只是简单的基于css切换；

3）性能消耗：

v-if有更高的切换消耗；

v-show有更高的初始渲染消耗；

4）使用场景：

v-if适合运营条件不大可能改变；

v-show适合频繁切换；

4、vue常用的修饰符

.prevent: 提交事件不再重载页面；

.stop: 阻止单击事件冒泡；

.self: 当事件发生在该元素本身而不是子元素的时候会触发；

.capture: 事件侦听，事件发生的时候会调用

5、v-on可以监听多个方法吗?

可以监听多个方法

例：<input type="text" :value="name" @input="onInput" @focus="onFocus" @blur="onBlur" />

6、vue中key 值的作用

用于 管理可复用的元素，当在组件中使用 v-for 时，key 是必须的

7、vue-cli程升级vue版本

npm install -g @vue/cli

8、vue事件中如何使用event对象?

1）使用不带圆括号的形式，event 对象将被自动当做实参传入；

2）使用带圆括号的形式，我们需要使用 $event 变量显式传入 event 对象。

9、$nextTick的使用

在Vue生命周期的created()钩子函数进行的DOM操作一定要放在Vue.nextTick()的回调函数中；为了在数据变化之后等待 Vue 完成更新 DOM ，可以在数据变化之后立即使用Vue.nextTick(callback) 。这样回调函数在 DOM 更新完成后就会调用。

10、vue组件中data为什么必须是函数

组件中的data写成一个函数，数据以函数返回值形式定义，这样每复用一次组件，就会返回一份新的data，类似于给每个组件实例创建一个私有的数据空间，让各个组件实例维护各自的数据。而单纯的写成对象形式，就使得所有组件实例共用了一份data，就会造成一个变了全都会变的结果。

11、v-for和v-if的优先级？

v-for比v-if具有更高的优先级。

12、vue中子组件调用父组件的方法

1）直接在子组件中通过this.$parent.event来调用父组件的方法；

2）在子组件里用$emit向父组件触发一个事件，父组件监听这个事件；

3）父组件把方法传入子组件中，在子组件里直接调用这个方法；

13、vue中keep-alive组件的作用

在做电商有关的项目中，当我们第一次进入列表页需要请求一下数据，当从列表页进入详情页，详情页不缓存也需要请求下数据，然后返回列表页，这时候使用keep-alive来缓存组件，防止二次渲染，这样会大大的节省性能。

14、vue中如何编写可复用的组件?

可复用组件一定要定义一个清晰的公开接口：

1）prop 允许外部环境传递数据给组件，在vue-cli工程中也可以使用vuex等传递数据。

2）事件允许组件触发外部环境的 action

3）slot 允许外部环境将内容插入到组件的视图结构内。

例子：<my-component

:foo="bar"

:bar="qux"

//子组件调用父组件方法

@event-a="doThis"

@event-b="doThat">

<!-- content -->

<img slot="icon" src="..." />

<p slot="main-text">Hello!</p>

</my-component>

15、什么是vue生命周期和生命周期钩子函数?

组件从创建，挂载，更新，销毁的过程，称为生命周期，而在这些过程中，会触发不同的钩子函数，从而给了开发者在不同的时机（阶段）添加自己代码的机会

分为：四个阶段，八个函数，这个八个钩子函数指向的是window，而不是vue实例！！

1.创建阶段 除了一开始外，在创建时也会触发如v-if

前 beforeCreate

后 created

2.挂载阶段

前 beforeMount vue还没开始工作（数据都没替换）

后 mounted vue开始工作（数据替换）

3.更新阶段 更新阶段的两个钩子，在初次渲染时，不会执行，他们

执行是伴随着视图的更新触发的，而视图的更新是数据的更新导致的，所以说这个两个钩子是伴随着数据的改变触发的。视图更新才触发!

前 beforeUpdate

后 updated

4.销毁阶段

前 beforeDestroy

后 destroyed

组件销毁时触发，有三种销毁方式，1.内部指令this.$destroy（）2.v-if销毁3.路由

组件销毁后，就不会在继续工作了，但本身还在只是不工作

8个钩子函数的特点：

创建：

beforeCreate： 1.无法操作data中的数据， 2.无法访问dom元素

created： 1.可以操作data的数据 2.无法访问dom元素

在beforeCreate后面的都可以操作data数据

挂载：

beforeMount： 1.可以访问dom，但是此时vue没有发挥作用

mounted： 1.可以访问dom，vue已经发挥作用了

更新：

beforeUpdate： 1.data的值是更新前的

updated： 1.data的值是更新后的

销毁：（适合做一些优化操作，比如关闭定时器，移除事件监听器等）

beforeDestroy 适合做一些优化操作，比如关闭定时器，移除事件监听器等

destroyed 适合做一些优化操作，比如关闭定时器，移除事件监听器等

8个钩子函数中最常用的是created和mounted这两个钩子，

created：发送数据请求（ajax）

mounted：获取dom元素（与dom交互）

16、vue的生命周期钩子函数有哪些？

创建：beforeCreate；created

挂载：beforeMount；mounted

更新：beforeUpdate；updated

销毁：beforeDestroy；destroyed

17、vue如何监听键盘事件中的按键?

在Vue中，已经为常用的按键设置了别名，这样我们就无需再去匹配keyCode，直接使用别名就能监听按键的事件。

例：<input @keyup.enter="function">

.delete delete（删除）/BackSpace（退格）

.tab Tab

.enter Enter（回车）

.esc Esc（退出）

.space Space（空格键）

.left Left（左箭头）

.up Up（上箭头）

.right Right（右箭头）

.down Down（下箭头）

.ctrl Ctrl

.alt Alt

.shift Shift

.meta (window系统下是window键，mac下是command键)

18、vue更新数组时触发视图更新的方法

push()

pop()

shift()

unshift()

splice()

sort()

reverse()

19、vue中对象更改检测的注意事项

/\* Vue.set(target,key,value)

参数：

{Object | Array} target

{string | number} key

{any} value

返回值：设置的值

用法：设置对象的属性。如果对象是响应式的，确保属性被创建后也是响应式的，同时触发视图更新。这个方法主要用于避开Vue不能检测属性被添加的限制。\*/

注意事项：由于JavaScript的限制，Vue不能检测对象属性的添加或删除。

对于已经创建的实例，Vue不能动态添加根级别的响应式属性。但是可以用Vue.set(object,key,value)方法向嵌套对象添加响应式属性。

20、解决非工程化项目初始化页面闪动问题

vue页面在加载的时候闪烁花括号{}}，v-cloak指令和css规则如[v-cloak]{display:none}一起用时，这个指令可以隐藏未编译的Mustache标签直到实例准备完毕。

21、v-for产生的列表，实现active的切换

<ul>

<li v-for="(info,index) in list" :key="info.id" @click="select(index)" v-bind:class="{'active':info.active}">{{info.name}}</li>

</ul>

data数据：

list:[

{

name:'a',

id:1,

active:false

},

{

name:'b',

id:2,

active:false

},

{

name:'c',

id:3,

active:false

},

{

name:'d',

id:4,

active:false

},

]

点击事件：

select(d){

this.list.map(s=>s.active=false);

this.list[d].active=true;

}

css样式：

<style scoped>

li.active{

background-color: red;

}

</script>

22、v-model语法糖的组件中的使用

/\* 语法糖：

<input type="text" v-model="mes"> 此时mes值就与input的值进行双向绑定

上面的代码是下面的代码的语法糖

<input v-bind:value="mes" v-on:input="mes= $event.target.value"/> \*/

1）父组件

<InputBox v-model="mes"></InputBox>

根据上面讲的v-model语法糖，所以 InputBox那行代码也可以写成

<InputBox v-bind:value="value" v-on:input="mes= $event.target.value"></InputBox>

2）子组件

1：接受一个value prop 2：在有新的值时触发input事件并将新值作为参数 。

23、十个常用的自定义过滤器

https://www.cnblogs.com/cangqinglang/p/11697718.html

24、vue等单页面应用及其优缺点

优点：

1、用户体验好，快，内容的改变不需要重新加载整个页面，对服务器压力较小。

2、前后端分离，比如vue项目

3、完全的前端组件化，前端开发不再以页面为单位，更多地采用组件化的思想，代码结构和组织方式更加规范化，便于修改和调整；

缺点：

1、首次加载页面的时候需要加载大量的静态资源，这个加载时间相对比较长。

2、不利于 SEO优化，单页页面，数据在前端渲染，就意味着没有 SEO。

3、页面导航不可用，如果一定要导航需要自行实现前进、后退。（由于是单页面不能用浏览器的前进后退功能，所以需要自己建立堆栈管理）

25、什么是vue的计算属性?

模板内的表达式非常便利，但是设计它们的初衷是用于简单运算的。在模板中放入太多的逻辑会让模板过重且难以维护。遇到复杂逻辑时应该使用Vue特带的计算属性computed来进行处理。

26、vue-cli提是共的几种脚手架模板

1)webpack-simple模板

2)webpack模板

27、vue父组件如何向子组件中传递数据?

子组件在props中创建一个属性，用以接收父组件传过来的值

父组件中注册子组件

在子组件标签中添加子组件props中创建的属性

把需要传给子组件的值赋给该属性

28、vue-cli开发环境使用全局常量

开发环境的全局常量定义在.env里

29、vue-cli产―环境使用全局常量

生产环境的全局常量定义在.env.production里

30、vue如何禁止弹窗后面的滚动条滚动？

methods : {

//禁止滚动

stop(){

var mo=function(e){e.preventDefault();};

document.body.style.overflow=‘hidden’;

document.addEventListener(“touchmove”,mo,false);//禁止页面滑动

}

31、计算属性的缓存和方法调用的有什么区别？

计算属性必须返回结果

计算属性是基于它的依赖缓存的。一个计算属性所依赖的数据发生变化时，它才会重新取值。

使用计算属性还是methods取决于是否需要缓存，当遍历大数组和做大量计算时，应当使用计算属性，除非你不希望得到缓存。

计算属性是根据依赖自动执行的，methods需要事件调用

32、vue-cli中自定义指令的使用

【vue-router】

1、vue-router如何响应 路由参数 的变化？

什么是路由参数的变化：

当使用路由参数时，例如从 /user/foo 导航到 /user/bar，原来的组件实例会被复用。因为两个路由都渲染同个组件，比起销毁再创建，复用则显得更加高效。不过，这也意味着组件的生命周期钩子不会再被调用。

watch: { // watch的第一种写法

$route (to, from) {

console.log(to)

console.log(from)

}

},

watch: { // watch的第二种写法

$route: {

handler (to, from){

console.log(to)

console.log(from)

},

// 深度观察监听

deep: true

}

},

watch: { // watch的第三种写法

'$route':'getPath'

},

methods: {

getPath(to, from){

console.log(this.$route.path);

}

}

2、完整的vue-router导航解析流程

1）导航被触发。

2）在失活的组件里调用离开守卫。

3）调用全局的 beforeEach 守卫。

4）在重用的组件里调用 beforeRouteUpdate 守卫 (2.2+)。

5）在路由配置里调用 beforeEnter。

6）解析异步路由组件。

7）在被激活的组件里调用 beforeRouteEnter。

8）调用全局的 beforeResole 守卫 (2.5+)。

9）导航被确认。

10）调用全局的 afterEach 钩子。

11）触发 DOM 更新。

12）用创建好的实例调用 beforeRouteEnter 守卫中传给 next 的回调函数。

3、vue-router有哪几种导航钩子 (导航守卫) ?

1）全局守卫： router.beforeEach

2）全局解析守卫： router.beforeResolve

3）全局后置钩子： router.afterEach

4）路由独享的守卫： beforeEnter

5）组件内的守卫： beforeRouteEnter、beforeRouteUpdate (2.2 新增)、beforeRouteLeave

4、vue-router的几种实例方法以及参数传递

实例方法：

三个全局导航守卫

router.beforeEach

router.beforeResolve

router.afterEach

编程式路由导航

router.push

router.replace

router.go

router.back

router.forward

传参方式：

命名路由

this.$router.push({ name: 'news', params: { userId: 123 }}) //发

this.$route.params.userId //收

查询参数

this.$router.push({ path: '/news', query: { userId: 123 }}); //发

this.$route.query.userId //收

5、vue-router的动态路由匹配以及使用

配置：

routes:{

path:'/aaa/:bbb',动态路由

}

使用

传：this.$router.push('/aaa/123');

收：$route.params.id

6、vue-router如何定义 嵌套路由?

嵌套也就是路由中的路由的意思，组件中可以有自己的路由导航和路由容器（router-link、router-view），通过配置children可实现多层嵌套，在vue组件中使用<router-view>就可以了。

例：

{

path: '/standard',

name: 'standard',

component: resolve => require(['@/pages/template/standard'], resolve),

children: [

{

path: 'a',

component: resolve => require(['@/pages/practice/a'], resolve),

},

{

path: 'b',

component: resolve => require(['@/pages/practice/b'], resolve),

},

{

path: 'c',

component: resolve => require(['@/pages/practice/c'], resolve),

},

]

},

7、<router-link></router- link>组件及其属性

<router-link> 组件支持用户在具有路由功能的应用中（点击）导航。

属性；

to 表示目标路由的链接。

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

append 设置 append 属性后，则在当前（相对）路径前添加基路径。

tag 有时候想要 <router-link> 渲染成某种标签

active-class设置 链接激活时使用的 CSS 类名。

exact "是否激活" 默认类名