vue 面试题-4

1.vuex 的作用？

vuex 做全局数据状态管理，vuex 是单向数据流（箭头一个方向）actions->mutations->state

vuex 的几个核心概念：

一个 store 包括：state,getter，mutation,action 四个属性

state：存储状态 ，【初始化的】相当于变量存状态数据用的，只有 mutations 能改。不是谁都能改 state（单向数据流）

getter:getter 的作用与 filters 有一些相似，可以将 state 进行过滤后输出

mutation:门用来修改 state,不宜过大。专用的并且只能同步操作

action:一些对 state 的异步操作可以放在 action 中，actions:组合 mutations 完成复杂操作，【写功能的】

module:当 store 对象过于庞大时，可以根据具体的业务需求分为多个 module

我们可以在组件中触发 Action，Action 则会提交 Mutation，Mutaion 会对 State 进行修改，组件再根据 State 、Getter 渲染页面

getters 使用：

getters 里的东西以变量形式用,不是以函数形式但可以模拟函数使用，return 返回个函数就可以了，但不能返数字那样就 11()调数字了

```
getters:{
    getDataById(state) {
        return 111
    }
}
```

当变量用：var a = this.$store.getters.getDataById;console.log(a)//111

模拟返回函数巧用上边改成: return function(id){return ...} 用时 a(传的值)

getters 可以帮助重用

getters 是专门帮助返回数据的，正常 store 的所有的读都应该通过 getters，只是有时偷赖没用

getters 是用来读的不是写的，不要用它来写。写东西用 this.$store.dispatch('del',id)

修改时带数据，二者相同用 getters 提取更好可重用代码推荐，把判断放 getters 中

form:this.$store.state.find(item => item.id == this.$route.params.id) || {}

form: this.$store.getters.getDataById(this.$route.params.id) || {}

Vuex 用于存储:

答:vue 框架中状态管理。在 main. js 引入 new Vue({store,})，新建了一个目录 store。**引入:import Vuex from 'vuex'; Vue.use(Vuex);**

场景有：单页应用中，组件之间的状态。音乐播放、登录状态、加入购物车

```
import Vue from 'vue';
import App from './App.vue';
import axios from 'axios';
import vueAxios from 'vue-axios';
Vue.use(vueAxios, axios);
// 引入 vuex
import Vuex from 'vuex';
Vue.use(Vuex); //插件都得 use
let store = new Vuex.Store({
```

```
    state: {
        num: 0
    },
    mutations: {
        setNum(state, arg) {state.num = arg;}
    },
    actions: {
    //actions 名可以和 mutations 里名一样,
       aaa(context, arg) { //context 对象有很多操作
            context.commit('setNum', arg) //调 mutations 里 setNum,
        },
        // 等同于 aaa({ commit }, arg) {commit('setNum', arg)}
    }
})
new Vue({
    store, //和 router 一样与 vue 产生关联
    render: h => h(App),
}).$mount('#app')
```

本地定位信息 locationInfo

用户信息 userInfo

用户登录 Token userToken

分享 shareCofing

```
export const state={
    locationInfo:{
        lng:"", //经度
        lat:"", //纬度
        addressComponent:"", //定位的详细信息
        formattedAddress: "" //定位的位置名称
    },
    userInfo:{ //用户信息

    },
  userToken:{  //用户登录 Token
    accessToken:""
  },
  shareCofing:{  //分享
    url:"",
    img:"",
    title:"",
    content:""
  }
}
```

state->mutation ->action
data ->原子操作（指这个操作是不可分割的） -> methods
mutation 是不对外的，理论上来说可以直接调用 mutation 但没有意义，
vuex 是单向的 是私有的只有 mutation 能改 state,尽量保证 mutation 作用单一如就是修改用户 ID，就是修改用户某向数据，mutation 不要包含太多东西，同时改很多东西不便组合。
用 action 组合，
变量是双向的 alert(a) a=12 想读就读想写就写
state 和 getters 是用来读的：
getters 类似于 computed，它不是一个真实的数据但方便去用，
action 用来写的修改操作。纯粹的写
1、什么时候用 vuex，不用普通 data,不用父级 data
2、vuex 放什么数据
3、vuex 怎么规划
vuex 是来个保底的最不应该用的一个技术，相当于全局变量，什么都往里放不好，
父级管理：数据在很多不相关的地方用，如两个相关的组件分不开 90%要一起用说明关联性很强就是拿父级把他们包裹起来让父级来管理它们的数据交互。
用 vuex：像数据分的很散 十个八个一百个都会用到这个数据这种大量重用的数据放别的地很麻烦，为了‘方便’但为了眼前的方便也要平横以后的利益。
用户的信息很多组件都用，用户登录组件，推荐商品，大量组件用同一数据都是 vuex,
还有一种也许数据不是很多组件用但使用这个数据的组件关联很弱，作为一个组件这会和 A 配合，过会和 B 配合，又和 C 配合，这种数据也是往 VUEX 里放，因为这时没办法很明确的级别把他们管理起来
如不会 VUEX 需要 50 行代码，用了 VUEX3 行就行了 就会 VUEX
如淘宝和天猫是一个网站，在淘宝加了购物车项目在在天猫也要看到购物车数据，像这种它们两个组件差异很大就用 vuex,当然购物车数据一定要放服务器端的要不一刷新就没了
两个不相关的组件用同一个数据就 vuex

不用 Vuex 会带来什么问题？

答：
① 可维护性下降，修改数据，需要维护好个地方
② 可读性下降，组件内的数据来源不明确
③增加耦合，Vue 用 Component 的初衷在于降低耦合性，如果大量的上传分发，反而会增加耦合度。

vuex 有哪几种属性？

答：State、 Getters、Mutation 、Action、 Module。
state => 基本数据
getters => 从基本数据派生的数据
mutations => 提交更改数据的方法，同步!
actions => 像一个装饰器，包裹 mutations，使之可以异步。
modules => 模块化 Vuex

vuex 的 State 特性是？

答：Vuex 就好比是一个仓库，仓库里面放了很多对象。其中 state 就是存放数据源的地方，对应 Vue 对象里面的 data。state 里面存放的数据是响应式的，Vue 组件从 store 中读取数据，若是 store 中的数据发生改变，依赖这个数据的组件也会发生更新。它通过 mapState 把全局的 state 和 getters 映射到当前组件的 computed 计算属性中

vuex 的 Getter 特性是？

答：getters 可以对 State 进行计算操作，它就是 Store 的计算属性。是 getters 可以在多组件之间复用。如果一个状态只在一个组件内使用，可以不用 getters

vuex 的 Mutation 特性是？

答：Action 类似于 mutation，不同在于：Action 提交的是 mutation，而不是直接变更状态。Action 可以包含任意异步操作

Vue.js 中 ajax 请求代码应该写在组件的 methods 中还是 vuex 的 actions 中？

答：

① 如果请求来的数据不需要被其他组件复用，只在请求的组件内使用，就不需要放入 vuex 的 state 里。

② 如果被其他地方复用，就可以把请求放入 action 里，包装成 promise 返回，在调用的地方用 async await 处理返回的数据。

2.vue 中的路由拦截器的作用？

路由拦截，权限设置

例如：当用户没有登录权限的时候就会跳转到登录页面，用到的字段 requireAuth:true

3.axios 的作用与拦截登录？

vue 中的 ajax，用于向后台发起请求

特点：从浏览器中创建 XMLHttpRequests、从 node.js 创建 http 请求、支持 Promise API、拦截请求和响应、转换请求数据和响应数据、取消请求、自动转换 json 数据、

客户端支持防御 XSRF

promise：

一个对象用来传递异步操作的信息、

promise 的出现主要是解决地狱回调的问题，无需多次嵌套

本质：分离异步数据获取和业务

axios 的拦截器分为请求拦截器跟响应拦截器,每个拦截器都可以设置两个拦截函数，一个为成功拦截，一个为失败拦截。

响应拦截器很好理解，就是服务器返回给我们的数据，我们在拿到之前可以对他进行一些处理

比如，有些请求是需要用户登录之后才能访问的，或者 post 请求的时候，我们需要序列化我们提交的数据。这时候，我们可以在请求被发送之前进行一个拦截，从而进行我们想要的操作。

```
拦截器分为请求拦截器和响应拦截器
#请求拦截器
axios.interceptors.request.use(function(config){
    return config;
},function(error){
    return Promise.reject(error);
});
#响应拦截器
axios.interceptors.response.use(function(response){
    return response;
```

```
},function(error){
    return Promise.reject(error);
});
```

4.列举 vue 的常见指令。

1.文本插值：{{}} Mustache

```
<div id='app'>
    {{message}}
</div>
```

2.DOM 属性绑定：v-bind

```
<div id='app-2'>
    <span v-bind:title='message'>
        鼠标悬停几秒钟查看此处动态绑定的提示信息
    </span>
</div>
```

3.指令绑定一个事件监听器：v-on

```
<div id='app-5'>
    <p>{{message}}</p>
    <button v-on:click='reverseMessage'>逆转消息</button>
</div>
```

4.实现表单输入和应用状态之间的双向绑定：v-model

```
<div id='app-6'>
    <p>{{message}}</p>
    <input v-model='message'>
</div>
```

5.控制切换一个元素的显示：v-if 和 v-else

```
<div id='app-3'>
    <p v-if='seen'>现在你看到我了</p>
</div>
```

6.列表渲染:v-for

```
<div id='app-4'>
    <ol>
        <li v-for='todo in todos'>
            {{todo.text}}
        </li>
    </ol>
```

5.列举 Http 请求中常见的请求方式？

Get　向特定的路径资源发出请求，数据暴露在 url 中

post　向指定路径资源提交数据进行处理请求，数据包含在请求体中

options 返回服务器针对特定资源所支持的 http 请求方法，允许客户端查看，测试服务器性能

head　向服务器与 get 请求相一致的响应，响应体不会返回，可以不必传输整个响应内容

put 从客户端向服务器端传送的数据取代指定的文档的内容

delete 请求服务器删除指定的页面

trace 回显服务器收到的请求，主要用于测试或者诊断
connecthttp/1.1 协议中预留给能够将连接改为管道方式的代理服务

6.对于 MVVM 的理解

MVVM 是 Model-View-ViewModel 的缩写。
Model 代表数据模型，也可以在 Model 中定义数据修改和操作的业务逻辑。
View 代表 UI 组件，它负责将数据模型转化成 UI 展现出来。
ViewModel 监听模型数据的改变和控制视图行为、处理用户交互，简单理解就是一个同步 View 和 Model 的对象，连接 Model 和 View。viewmodel 和 model 实现双向数据绑定

7.Vue 的生命周期

1.什么是 vue 生命周期？
答： Vue 实例从创建到销毁的过程，就是生命周期。从开始创建、初始化数据、编译模板、挂载 Dom→渲染、更新→渲染、销毁等一系列过程，称之为 Vue 的生命周期。
2.vue 生命周期的作用是什么？
答：它的生命周期中有多个事件钩子，让我们在控制整个 Vue 实例的过程时更容易形成好的逻辑。
3.vue 生命周期总共有几个阶段？
答：它可以总共分为 8 个阶段：创建前/后，载入前/后,更新前/后,销毁前/销毁后。
4.第一次页面加载会触发哪几个钩子？
答：会触发 下面这几个 beforeCreate, created, beforeMount, mounted 。
5.DOM 渲染在 哪个周期中就已经完成？
答：DOM 渲染在 mounted 中就已经完成了。

8.Vue 实现数据双向绑定的原理：Object.defineProperty（）

采用**数据劫持结合发布者-订阅者模式**的方式，通过 **Object.defineProperty（）**来劫持各个属性的 setter，getter，在数据变动时发布消息给订阅者，触发相应监听回调。

9.Vue 组件间的参数传递

1.父组件与子组件传值
父组件传给子组件：子组件通过 props 方法接受数据;
子组件传给父组件：$emit 方法传递参数
2.非父子组件间的数据传递，兄弟组件传值
eventBus，就是创建一个事件中心，相当于中转站，可以用它来传递事件和接收事件。项目比较小时，用这个比较合适。（虽然也有不少人推荐直接用 VUEX，具体来说看需求咯。技术只是手段，目的达到才是王道。）

10.Vue 的路由实现：hash 模式 和 history 模式

hash 模式：在浏览器中符号“#”，#以及#后面的字符称之为 hash，用 window.location.hash 读取；
特点：hash 虽然在 URL 中，但不被包括在 HTTP 请求中；用来指导浏览器动作，对服务端安全无用，hash 不会重加载页面。
hash 模式下，仅 hash 符号之前的内容会被包含在请求中，如 http://www.xxx.com，因此对于后端来说，即使没有做到对路由的全覆盖，也不会返回 404 错误。
history 模式：history 采用 HTML5 的新特性；且提供了两个新方法：pushState（），replaceState（）可以对浏览器历史记录栈进行修改，以及 popState 事件的监听到状态变更。

history 模式下，前端的 URL 必须和实际向后端发起请求的 URL 一致，
如 http://www.xxx.com/items/id。后端如果缺少对 /items/id 的路由处理，将返回 404 错误。**Vue-Router 官网里如此描述：**“不过这种模式要玩好，还需要后台配置支持……所以呢，你要在服务端增加一个覆盖所有情况的候选资源：如果 URL 匹配不到任何静态资源，则应该返回同一个 index.html 页面，这个页面就是你 app 依赖的页面。”

11.Vue 组件化开发

组件化，就是把页面拆分成多个组件，每个组件依赖的 CSS、JS、模板、图片等资源放在一起开发和维护。因为组件是资源独立的，所以组件在系统内部可复用，组件和组件之间可以嵌套，如果项目比较复杂，可以极大简化代码量，并且对后期的需求变更和维护也更加友好

全局 `Vue.component('my-component', { })`

局部

```
import HelloWorld from './components/HelloWorld'
components: {     HelloWorld   }
```

12.vue-cli 如何新增自定义指令？

1.创建局部指令

```
var app = new Vue({
    el: '#app',
    data: {},
    // 创建指令(可以多个)
    directives: {
    // 指令名称
        dir1: {
            inserted(el) {
                // 指令中第一个参数是当前使用指令的 DOM
                console.log(el);
                console.log(arguments);
                // 对 DOM 进行操作
                el.style.width = '200px';
                el.style.height = '200px';
                el.style.background = '#000';
            }
        }
    }
});
```

2.全局指令

```
Vue.directive('dir2', {
    inserted(el) {
     console.log(el);
    }
})
```

3．指令的使用

```
<div id="app">
    <div v-dir1></div>
    <div v-dir2></div>
</div>
```

13.vue 如何自定义一个过滤器？

html 代码：

```
<div id="app"><input type="text" v-model="msg" />{{msg| capitalize }}</div>
```

JS 代码：

```
var vm=new Vue({
    el:"#app",
    data:{
        msg:''
    },
    filters: {
      capitalize: function (value) {
        if (!value) return ''
        value = value.toString()
        return value.charAt(0).toUpperCase() + value.slice(1)
      }
    }
})
```

全局定义过滤器

```
Vue.filter('capitalize', function (value) {
  if (!value) return ''
  value = value.toString()
  return value.charAt(0).toUpperCase() + value.slice(1)
})
```

过滤器接收表达式的值 (msg) 作为第一个参数。capitalize 过滤器将会收到 msg 的值作为第一个参数。

14.keep-alive

keep-alive 是 Vue 内置的一个组件，可以使被包含的组件保留状态，或避免重新渲染。

在 vue 2.1.0 版本之后，keep-alive 新加入了两个属性：include(包含的组件缓存) 与 exclude(排除的组件不缓存，优先级大于 include) 。使用方法：

```
<keep-alive include="include_components" exclude="exclude_components">
    <component>
    <!-- 该组件是否缓存取决于 include 和 exclude 属性 -->
    </component>
</keep-alive>
```

参数解释

include - 字符串或正则表达式，只有名称匹配的组件会被缓存

exclude - 字符串或正则表达式，任何名称匹配的组件都不会被缓存

include 和 exclude 的属性允许组件有条件地缓存。二者都可以用"，"分隔字符串、正则表达式、数组。当使用正则或者是数组时，要记得使用 v-bind 。

使用示例

```
<!-- 逗号分隔字符串，只有组件 a 与 b 被缓存。 -->
<keep-alive include="a,b"><component></component></keep-alive>
<!-- 正则表达式 (需要使用 v-bind，符合匹配规则的都会被缓存) -->
<keep-alive :include="/a|b/"><component></component></keep-alive>
<!-- Array (需要使用 v-bind，被包含的都会被缓存) -->
<keep-alive :include="['a', 'b']"><component></component></keep-alive>
```

15.v-if 与 v-show 区别(就这个)

答：v-if 按照条件是否渲染，v-show 是 display 的 block 或 none；

v-if 元素真的被删掉 ，只剩下一行注释 ，<!-------->占位符

v-show 元素只是隐藏了，display

v-show 用于频繁显示隐藏，【隐藏在显示比直接删除显示快】v-if 用的更多。大量的隐藏-也会影响性能。

v-show 某些元素隐藏了也会起作用-比如：表单，【v-if 不会有这个问题】

16.vue 几种常用的指令

答：v-for 、 v-if 、v-bind、v-on、v-show、v-else 、v-html、v-text、v-model

17.怎么定义 vue-router 的动态路由？怎么获取传过来的值

答：在 router 目录下的 index.js 文件中，对 path 属性加上 /:id，使用 router 对象的 params.id 获取。

跳路由：

```
$router.push({ name: 'add' }) this.$router.push({ path: 'add' });
```

参数：

```
$route.params.id
```

返回：

```
@click="$router.back()"
```

18.vue 常用的修饰符？

答：.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

19.vue.js 的两个核心是什么？

答：数据驱动、组件系统

20.vue 中 key 值的作用？

答：当 Vue.js 用 v-for 正在更新已渲染过的元素列表时，它默认用"就地复用"策略。如果数据项的顺序被改变，Vue 将不会移动 DOM 元素来匹配数据项的顺序， 而是简单复用此处每个元素，并且确保它在特定索引下显示已被渲染过的每个元素。key 的作用主要是为了高效的更新虚拟 DOM。

key 作用高效更新虚拟 dom,元素切换时使用 key 属性可以做区分，提高性能

21. **什么是 vue 的计算属性？**

答：在模板中放入太多的逻辑会让模板过重且难以维护，在需要对数据进行复杂处理，且可能多次使用的情况下，尽量采取计算属性的方式。好处：①使得数据处理结构清晰；②依赖于数据，数据更新，处理结果自动更新；③计算属性内部 this 指向 vm 实例；④在 template 调用时，直接写计算属性名即可；⑤常用的是 getter 方法，获取数据，也可以使用 set 方法改变数据；⑥相较于 methods，不管依赖的数据变不变，methods 都会重新计算，但是依赖数据不变的时候 computed 从缓存中获取，不会重新计算。

22.vue 等单页面应用及其优缺点

答：优点：Vue 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件，核心是一个响应的数据绑定系统。MVVM、数据驱动、组件化、轻量、简洁、高效、快速、模块友好。

缺点：不支持低版本的浏览器，最低只支持到 IE9；不利于 SEO 的优化（如果要支持 SEO，建议通过服务端来进行渲染组件）；第一次加载首页耗时相对长一些；不可以使用浏览器的导航按钮需要自行实现前进、后退。

23.怎么定义 vue-router 的动态路由？怎么获取传过来的值

答：在 router 目录下的 index.js 文件中，对 path 属性加上 /:id，使用 router 对象的 params.id 获取

24、v-on 可以监听多个方法

可以

```
<button @click="a(),b()">点我 ab</button>
methods: {
    a() {console.log('a');},
    b() {console.log('b');}
}
```

25、为什么避免 v-if 和 v-for 用在一起

当 Vue 处理指令时，v-for 比 v-if 具有更高的优先级，这意味着 v-if 将分别重复运行于每个 v-for 循环中。通过 v-if 移动到容器元素，不会再重复遍历列表中的每个值。取而代之的是，我们只检查它一次，且不会在 v-if 为否的时候运算 v-for。

26、vue-loader 是什么？使用它的用途有哪些？

解析.vue 文件的一个加载器。（深入理解见 https://www.jb51.net/article/115480.htm）

用途：js 可以写 es6、style 样式可以 scss 或 less、template 可以加 jade 等

根据官网的定义，vue-loader 是 webpack 的一个 loader，用于处理 .vue 文件。

其次，使用 vue-cli 脚手架，作者已经配置好了基本的配置，开箱及用，你需要做的就是 npm install 安装下依赖，然后就可以开发业务代码了。当然，如果你想进阶，最好熟悉下 vue-loader 的具体配置，而不要依赖脚手架

27、active-class 是哪个组件的属性？

vue-router 模块的 router-link 组件。

28、请列举出 3 个 Vue 中常用的生命周期钩子函数 （见博客）

created: 实例已经创建完成之后调用,在这一步,实例已经完成数据观测, 属性和方法的运算, watch/event 事件回调. 然而, 挂载阶段还没有开始, $el 属性目前还不可见

mounted: el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子。如果 root 实例挂载了一个文档内元素，当 mounted 被调用时 vm.$el 也在文档内。

activated: keep-alive 组件激活时调用

29、在 Vue 中使用插件的步骤

采用 ES6 的 import ... from ...语法或 CommonJSd 的 require()方法引入插件

使用全局方法 Vue.use(plugin)使用插件,可以传入一个选项对象 Vue.use(MyPlugin, { someOption: true })

如使用懒加载插件：

```
Vue.use(VueLazyload, {
  loading: require('common/image/default.png')
})
```

30、指令 v-el 的作用是什么?

提供一个在页面上已存在的 DOM 元素作为 Vue 实例的挂载目标.可以是 CSS 选择器，也可以是一个 HTMLElement 实例

31、Vue 中引入组件的步骤?

1）采用 ES6 的 import ... from ...语法或 CommonJS 的 require()方法引入组件

2）对组件进行注册,代码如下

```
// 注册
Vue.component('my-component', {
  template: '<div>A custom component!</div>'
})
```

3）使用组件<my-component></my-component>

33、vue 中是如何检测数组变化的

答：vue 将数组原型上的方法进行了重新编写，更改了一些数组的方法，比如 push、shift、pop、splice、unshift、sort、reverse，这些方法都有一个特点，就是可以改变数组原来的值。当我们用了这些方法来操作数组时，就会把原来的方法进行劫持，可以在函数内部添加自己的功能。如果想跟新数组的索引，需要使用 vue.$set 方法来实现。

34、 vue.set 方法

答：vue 不允许在已经创建的实例上动态添加新的根级响应式属性，$set 可以触发更新，当对象新增不存在的属性时，会触发对象依赖的 watcher 去更新，当更改数组索引时，我们调用数组的 splice 方法去更新数组。

操作数组示列：

```
this.$set(arr,  index,  val)
```

操作对象示例：

```
this.$set( obj, key, val)
```

36、递归组件的用法

答：在 export default 中，有一个属性是 name。这属性对递归组件来说非常重要。递归组件只能通过 name 选项来做事。递归组件一定要有一个结束的条件，否则就会使组件循环引用，最终出现“max stack size exceeded”的错误，也就是栈溢出。那么，我们可以使用 v-if="false"作为递归组件的结束条件。当遇到 v-if 为 false 时，组件将不会再进行渲染。

37、$route 和 router 的区别

router 是 VueRouter 的一个对象，通过 Vue.use(VueRouter)和 VueRouter 构造函数得到一个 router 的实例对象,这个对象中是一个全局的对象,他包含了所有的路由包含了许多关键的对象和属性。$router == new Router({})

$router.replace({path:'home'});//替换路由，没史记录

接收参数的方式是

this.$router.currentRoute.params.userId == this.$route.params.userId //内是路由相关参数
跳转链接用 this.$router.push({path:`/user/${userId}`})，和 router-link 跳转一样

38、vue-router 响应路由参数的变化

答：

① 用 watch 监听

② 使用组件内 beforeRouteUpdate(to,from,next)导航钩子函数，to 表示将要跳转的路由对象，from 表示从哪个路由跳转过来，next 是进行下一个钩子函数

vue-router 传参

答：① 使用 Params:只能使用 name 属性，不能使用 path，参数不会显示在路径上，浏览器强制刷新参数会被清空

② 使用 Query：参数会显示在路径上，刷新不会被清空，name 可以使用 path 路径

38、 vue 的模板编译原理

答：vue 的生命周期钩子实际上就是一个回调函数。当我们传入一个钩子函数时，vue 内部会帮我们调用，并将生命周期钩子转换成数组，调用的时候，就又会把数组遍历一遍，类似一个发布订阅的模式。

过 render 函数返回虚拟 DOM，再把虚拟的 DOM 变成真正的 DOM。

40、说说 vue 的动态组件。

答：多个组件通过同一个挂载点进行组件的切换，is 的值是哪个组件的名称，那么页面就会显示哪个组件。

39、yarn

1、创建项目：

旧版本：vue init webpack 项目名

新版本：vue create 项目名 --no-git --bare //用这个嫌脏

建议加上--no-git：不加--no-git 会自动初始化 git 项目，建议加上

建议加上--bare: -bare 裸仓库/空白文件 ，不加会给一堆乱乱的东西

2、安装新版本 vue-cli3

yarn add global @vue/cli@3.12.1

3、配置 vue.config.js

4、安装 axios vue-router vue-axios

命令：yarn add axios vue-axios

命令：yarn add vue-router

5、运行 yarn run serve

41、为什么 vue 中配置项要放 data 中

放 data 的配置项才能实现数所的双向绑定。否则 vue 的配置项就不是响应式。响应式原理就是 defineproperty 会遍历 data 对象里的属性。转变为 getter 和 setter。从而在 getter 中依赖收集，setter 中依赖更新。

42、单页面应用和多页面应用区别

组成：(单)一个外壳页面和多个页面片段组成， (多)多个完整页面构成

资源共用（css,js）：(单)共用，只需在外壳部分加载 (多)不共用，每个页面都需加载

刷新方式：（单）页面局部刷新或更改，（多）整页刷新

Url 模式 (单)a.com/#/pageone (多)a.com/pageone.html

用户体验 （单）页面片段的切换快，用户体验良好，（多）页面切换加载缓慢，流畅度不够，体验差

数据传输：（单）容易 （多）依赖 URL 传参，或者 cookie，localStorage 等

搜索引擎优化（SEO）（单）需要单独方案，实现较为困难，不利于 SEO 检索，可利用服务器端渲染（SSR）优化（多）实现方法简易

试用范围 （单）高要求体验度，追求界面流畅的应用 （多）适用于追求高度支持搜索引擎的应用

开发成本（单）较高，常需借助专业框架（多）较低，但页面重复代码多

维护成本 （单）本对容易 （多）相对复杂

43、自定义指令

Vue2.0 中，代码复用和抽象的主要形式是组件，有的情况下，你仍然需要对普通 DOM 元素进行底层操作，这时候就会用到自定义指令。举个聚焦输入框的例子，如下：

5 种钩子函数：

一个指令定义对象可以提供如下几个钩子函数（均为可选）：

bind:指令绑定时触发。（el,val）传元素与值

el：<h2 style="color: red;">color</h2>

val：{name: "color", rawName: "v-color", value: "red", expression: "color", modifiers: {…}, …}

inserted:元素插入页面时触发，不怎么用

updated：节点更新时触发

componentUpdated：指令所在组件的 VNode 及其子 VNode 更新时调用

unbind：指令与元素解绑时调用。

钩子函数的参数：

主要有四个参数:el,binding,vnode,oldvnode。

el:指令所绑定的元素，可以用来直接操作 DOM

bingding:一个对象，包含很多属性

vnode:Vue 生成的虚拟节点

oldvnode:上一个虚拟节点。仅在 update 和 componentUpdated 钩子中可用

除了 el 之外，其它参数都应该是只读的，切勿进行修改。如果需要在钩子之间共享数据，建议通过元素的 dataset 来进行。全局和局部：

```
<div id="app">
    <h2 v-color="color">color</h2>
    <h2 v-fontsize="'80px'">fontsize</h2>
</div>
<script src="vue.js"></script>
<script>
// 全局指令
Vue.directive('color', {
    // bind:指令绑定时触发;el 元素,val 传的值
    bind(el, val) {
        el.style.color = val.value; // 写要做的事情
    },
    // 元素插入页面时触发
    inserted() {},
    // 节点更新时触发
    update() {}
})
var app = new Vue({
    el: '#app',
```

```
    data: {
        color: 'red'
    },
    // 局部指令
    directives: {
        'fontsize': {
            bind(el, val) {
                el.style.fontSize = val.value
            }
        }
    }
})
</script>
```

color

fontsize

44、vue 优化

1、代码模块化，咱们可以把很多常用的地方封装成单独的组件，在需要用到的地方引用，而不是写过多重复的代码，每一个组件都要明确含义，复用性越高越好，可配置型越强越好，包括咱们的 css 也可以通过 less 和 sass 的自定义 css 变量来减少重复代码。

2、for 循环设置 key 值，在用 v-for 进行数据遍历渲染的时候，为每一项都设置唯一的 key 值，为了让 Vue 内部核心代码能更快地找到该条数据，当旧值和新值去对比的时候，可以更快的定位到 diff。

3、Vue 路由设置成懒加载，当首屏渲染的时候，能够加快渲染速度。

4、更加理解 Vue 的生命周期，不要造成内部泄漏，使用过后的全局变量在组件销毁后重新置为 null。

5、可以使用 keep-alive，keep-alive 是 Vue 提供的一个比较抽象的组件，用来对组件进行缓存，从而节省性能。

45、git 命令 如何提交新建的文件

Git add.

Git commit -m 备注

Git push

46、vuex 页面 F5 刷新后维持刷新前的状态不变

（1）

首先，一样需要在 store 的 index.js 中，state，增加页面要保存的变量，并且将它们的值和 localStorage 里面的绑定

```
// 初始化时用sessionStore.getItem('token'),这样子刷新页面就无需重新登录
const state = {
  user: window.sessionStorage.getItem('user'),
  token: window.sessionStorage.getItem('token'),
  UserImage: window.sessionStorage.getItem('UserImage'),
};
```

然后再新增 mutations，这是将变量写入 sessionStorage 的方法

```
const mutations = {
  // 将token保存到sessionStorage里，token表示登陆状态
  SET_TOKEN: (state, data) => {
    state.token = data;
    window.sessionStorage.setItem('token', data);
  },
  // 获取用户名
  GET_USER: (state, data) => {
  // 把用户名存起来
    state.user = data;
    window.sessionStorage.setItem('user', data);
  },
  SET_UserImage: (state, value) => {
    state.UserImage = value;
    window.sessionStorage.setItem('UserImage', value);
  },
```

接着就可以在登录函数中这样调用，登录成功后将需要的变量写进 sessionStorage

```
    $this.AlertMessage = res.data.errorMsg;
  } else {
    $this.$store.commit("SET_TOKEN", res.data.return.session_id);
```

在路由的钩子函数里面可以这样将变量取回来，那样页面刷新后，也能将刷新前的状态保持

```
router.beforeEach(function (to, from, next) {
  // debugger;
  const token = store.state.token ? store.state.token : window.sessionStorage.getItem(
```

（2）

解决 Vuex-在 F5 刷新页面后数据不见

问题场景

页面刷新后，想恢复刷新之前的某些数据状态，我们总是习惯于将数据放在浏览器的 sessionStorage 和 localStorage 中。但是用了 vue 后，vuex 便可以被应用了。

vuex 优势：相比 sessionStorage，存储数据更安全，sessionStorage 可以在控制台被看到。

vuex 劣势：在 F5 刷新页面后，vuex 会重新更新 state，所以，存储的数据会丢失。

为了克服这个问题， vuex-persistedstate 出现了~~

原理：将 vuex 的 state 存在 localStorage 或 sessionStorage 或 cookie 中一份

刷新页面的一瞬间，vuex 数据消失，vuex 会去 sessionStorage 中拿回数据，变相的实现了数据刷新不丢失~

使用方法：

安装：

```
npm install vuex-persistedstate  --save
```

在 store 下的 index.js 中，引入并配置

```
import createPersistedState from "vuex-persistedstate"
const store = new Vuex.Store({
  // ...
```

```
  plugins: [createPersistedState()]
})
```

此时可以选择数据存储的位置，可以是 localStorage/sessionStorage/cookie，此处以存储到 sessionStorage 为例，配置如下：

```
import createPersistedState from "vuex-persistedstate"
const store = new Vuex.Store({
  // ...
  plugins: [createPersistedState({
    storage: window.sessionStorage
  })]
})
```

存储指定 state：

vuex-persistedstate 默认持久化所有 state，指定需要持久化的 state,配置如下：

```
import createPersistedState from "vuex-persistedstate"
const store = new Vuex.Store({
  // ...
  plugins: [createPersistedState({
    storage: window.sessionStorage,
    reducer(val) {
      return {
        // 只储存 state 中的 user
        user: val.user
      }
    }
  })]
```

此刻的 val 对应 store/modules 文件夹下几个 js 文件存储的内容，也就是 stor/index 中 import 的几个模块，可以自己打印看看。希望哪一部分的数据持久存储，将数据的名字在此配置就可以，目前我只想持久化存储 user 模块的数据。

注意：如果此刻想配置多个选项，将 plugins 写成一个一维数组，不然会报错。

复制代码

```
import createPersistedState from "vuex-persistedstate"
import createLogger from 'vuex/dist/logger'
// 判断环境 vuex 提示生产环境中不使用
const debug = process.env.NODE_ENV !== 'production'
const createPersisted = createPersistedState({
  storage: window.sessionStorage
})
export default new Vuex.Store({
  // ...
  plugins: debug ? [createLogger(), createPersisted] : [createPersisted]
})
```

47、ajax 里 post 设置请求头的编码格式

常见请求头格式：

json：xhr.setRequestHeader("**Content-type**","application/json; charset=utf-8");//内容类型

form：xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded; charset=utf-8");

文本：xhr.setRequestHeader("Content-type", "text/plain; charset=utf-8");

48、vue 同级兄弟间数据传递

$emit 触发事件，$on 监听事件、vuex

bus 方式的组件间传值其实就是建立一个公共的 js 文件，专门用来传递消息

1、定义一个公共文件 public.js，index.js 同级（代码内容是创建一个空的 vue 实例）：

```
import Vue from 'vue'
export default new Vue()
```

2、创建好以后，同级（兄弟）组件分别引入 public.js 这个文件：A 与 B

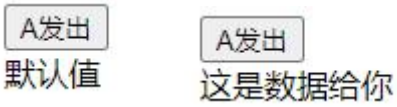
```
import Public from '../public.js'
```

3、例如现在有 A.vue 和 B.vue 两个同级（兄弟）组件，假设 A.vue 组件发送数据，通过$emit 将事件和参数'传递'给 B.vue（实际上是传递数据到 public.js 内）：

```
<input type="button" value='A 发出' @click="givedata"/>
import Public  from '../public'
import B from '@/components/B'
  export default{
    data(){
      return{
        str:'这是数据给你'
      }
    },
   components:{B},
   methods:{
     givedata(){
         Public.$emit('givedata1',this.str);
     }
   }
  }
```

4、在 B.vue 组件里接收数据，通过$on 将事件和数据从 A.vue 接收过来：

```
{{name}}
import Public from '../public'
export default{
     data() {return {name: '默认值'};},
     mounted(){
      Public.$on('givedata', (info) => { this.name = info ;});
    }
  }
```



49、vue 使用之路由跳转

1、需求：点击按钮，跳转到任意页面（不传参），router-view 内容内 login 组件

http://localhost:3000/login

```
export default new Router({
    mode: 'history',
        {
            path: '/details/:id/:name',
            name: 'details',
            meta: {
                requireAuth: true // 添加字段随意，表示进入这个路由是需要登录的，为 true
时不可切换路由进不去下页，false 时可正常进入
            },
            component: details
        },
    ]
});
```

声明式：<router-link :to="{ name: 'login'}">点击进入登录</router-link>

编程式：router.push(...)

方法一: this.$router.push({path:'路径'}); //如路由有参数 path 路径要对上：path: 'login/11/yjui'

方法二：this.$router.push({name:'组件名'});

2.需求：点击按钮，跳转到任意页面（传参）

路由设置和上一样，路径 http://localhost:3000/login?id=1&name=yjui

声明式：<router-link :to="{name:'index',query:{id:'xxx',name:'xxx'}}">

编程式：<button @click="fn">登录</button>触发用 router.push(...)

方法一 path-query：？,如路由有参数要对上：path: 'login/11/yjui',http://localhost:3000/login/11/刘?id=1&name=yjui

this.$router.push({path:'login/11/刘',query:{id: 1, name: 'yjui'}});

方法二 path-params：http://localhost:3000/login/1/yjui

this.$router.push({ path: '/login/' + 1 + '/yjui'})

this.$route.params.id，this.$route.params.name

方法三 name：http://localhost:3000/login/1/yjui

this.$router.push({name: 'login',params: { id: 1, name: 'yjui' }})

方法四 name：query://id=1&name=yjui，http://localhost:3000/login/11/刘?id=1&name=yjui

this.$router.push({name: 'login',query: { id: 1, name: 'yjui' },params: { id: 11, name: '刘' }})

注意：希望显示/login?id=1&name=yjui，但只加 query 会显示

http://localhost:3000/?id=1&name=yjui 所以加上 params

注：以上两种方法的 query 跳转路径也可以写成 name:'组件名'的形式

在 query 中放入需要传递的参数即可，多个参数之间用逗号隔开；

取值：this.$route.query.xx　(可在跳转的页面取得所传递的值)；

eg1:

跳转并传值：this.$router.push({path:'index',query:{id:'123'});　// 带查询参数，变成/index?id=123

取值：this.$route.query.id ;

eg2:

跳转并传值：this.$router.push({path:'xxx',params:{id:'123'});

取值：this.$route.params.id ;

非 vuex 中取值用 this.$store.state，vuex 中取值用 store.state

50、全局变量和 vuex 有什么区别，既然普通全局变量也要

Vuex 角色类似全局变量但比全局变量好，是个对象，改进-mutation 改 state，

模块化：（可以专门存用户相关的东西：是否登录，登录信息，token）；也可以在规划一个放购物数据如购物车，对比等等

普通全局变量不好没法监控，冲突很难控制

51、Vue+axios 的拦截器介绍：

拦截器 作用：在请求或响应被 then 或 catch 处理前拦截它们

请求拦截器：该类拦截器的作用是在请求发送前统一执行某些操作，比如在请求头中添加 token 字段。

响应拦截器：该类拦截器的作用是在接收到服务器响应后统一执行某些操作，比如发现响应状态码为 401 时，自动跳转到登录页。

执行顺序：这个 request 方法就是 axios 发送请求的核心方法，在源码中将发送请求分为了几个部分：请求拦截器，发送请求，响应拦截器，响应回调）

axios.interceptors.request->axios.interceptors.response->axios 请求

拦截器的使用方式如下：

在 axios 对象上有一个 interceptors 对象属性，该属性又有 request 和 response 2 个属性， axios.interceptors.request 和 axios.interceptors.response 对象提供了 use 方法，就可以分别设置请求拦截器和响应拦截器：（use 方法支持 2 个参数，第一个参数类似 Promise 的 resolve 函数，第二个参数类似 Promise 的 reject 函数。我们可以在 resolve 函数和 reject 函数中执行同步代码或者是异步代码逻辑。

// 添加请求拦截器

```
axios.interceptors.request.use(
    function(config) {
      // 在发送请求之前可以做一些事情
      config.headers.token = 'added by interceptor';//在 headers 中携带 token
        return config;
      },function(error) {
           return Promise.reject(error);// 处理请求错误

     }
);
```

// 添加响应拦截器

```
axios.interceptors.response.use(
    function(response) {
       // 对响应数据做点什么
       return response;
   },function(error) {
       // 对响应错误做点什么
      return Promise.reject(error);
   });
```

vue+axios 前端实现登录拦截的两种方式（路由拦截、http 拦截）

第一步：路由拦截

首先在定义路由的时候就需要多添加一个自定义字段 requireAuth（名字随便起），用于判断该路由的访问是否需要登录。如果用户已经登录，则顺利进入路由，

否则就进入登录页面。

定义完路由后，我们主要是利用 vue-router 提供的钩子函数 beforeEach()（全局的前置路由守卫，跳转之前进行验证）对路由进行判断。

```
const routes = [
    {
          path: '/',
          name: '/',
          component: Index
    },
    {
         //假设进入购物车页面需要用户登录
          path: '/repository',
          name: 'repository',
          meta: {
              //是路由的原数据
              requireAuth: true // 添加该字段，表示进入这个路由是需要登录的
          },
         component: Repository
      },
      {
         path: '/login',
         name: 'login',
         component: Login
      }
];
```

第二步：拦截器

要想统一处理所有 http 请求和响应，就得用上 axios 的拦截器。通过配置 http response inteceptor，当后端接口返回 401 Unauthorized（未授权），让用户重新登录。

```
//to 表示目标路由 from 从什么地方来的路由、对象
//next 进行下一个路由 需要放行
router.beforeEach((to, from, next) => {
     if (to.meta.requireAuth) {
          // 判断该路由是否需要登录权限
          if (store.state.token) {
              // 通过 vuex state 获取当前的 token 是否存在 注意 token 有时效性
              next(); //存在就说明登录了，放行
          } else {
              next({
                 //不存在，回退到一个页面，例如登录页面
                 path: '/login',
```

```
                // 将跳转的路由 path 作为参数，登录成功后跳转到该路由
                query: { redirect: to.fullPath }
              });
            }
      } else {
            next();
      }
});
```

在 router.beforeEach 一般不去更改目标页面的东西一般是校验

---

elementUI 是别人封装好的一个 ui 库什么都有 很方便
设计最巧妙的是表单和弹出框，弹出框不难但很合理
重要的就是 form 表单校验这些东西

---

this.$delete 与 splice

educations.splice(位置,删除几个，添加内容)

this.$delete(educations,i)

this.$delete 方法是用来删除对象属性的。this.$delete(obj,key);

---

vue 中外边数据文件引入

单独引某个数据,引文件 src/libs/matas.js

```
export const genders = {
    male: '男',
    falmale: '女',
    other: '其他'
};
```

App.vue 使用

```
import { genders } from '@/libs/mates'//genders 加入 vue 的 data 中
```

引入全部

```
import * as metas from '@/libs/mates' //使用 metas.geners
```

v-for 可以循环数字， 循环数字从 1 开始索引

1、如访问的 http://localhost:3000/找不到自动跳到 redirect 里 list 相当于 404
只要访问的不是上边 path 里出现的就跳 redirect 里

```
export default new Router({
  routes:[
    {path:'/list',name:'list',component:list},
    {path:'*',redirect:'list'}
  ]
})
```

2、跳路由：

```
$router.push({ name: 'add' })
```

```
this.$router.push({ path: 'add' });
```

---

findIndex 返回匹配下标

find 返回某个匹配东西

12、typescript
14、登陆权限控制流程
16、后端状态码处理的封装
17、路由守卫（在哪里使用）
19、vue 打包后是单页面对百度优化怎么处理解决（百度不支持异步抓取）
24、父组件给更深层子组件传值（$attrs）..
31、移动端微信浏览器下载文件存到哪里了
34、mapstate 、mapgetters、map....
35、直播....
36、canvas 写动画原理
37、登录注册用什么她们区别
38、页面打开次数统计
39、查询嵌套对象里的某个字符串信息并打印引用地址
40、写一个对象的方法弹出 alert
41、写一个对象的方法得到数组最大值，数组可能有嵌套
42、过滤身份证的年月日为*
43、数组的方法那些可以改变原数组那些不可以
44、webpack 配置（跨域配置也是 webpack 配置的）
50、下拉父级代理事件子级如何确定是哪个点击了（e.target
54、一个字符串直接可以调用的方法
58、form 表单没有跨域
59、css3 变量
65、websocket
67、地图单点多点定位
80、怎么判断一个用户是否登陆了再跳转路由
82、如何让原生访问你定义的方法（app）
92、sass 与 less 区别
95、vue 是多项目还是单项目（使用公共内容）
116、原生 ajax 请求如何做
105、浏览器如何解析 js，css（服务器做什么处理
127、vue。loading 动画放哪里响应拦截