**VUE面试题**

1、v-show 和 v-if的区别，v-show 和 keep-alive 的区别

答案：v-show是 CSS display 控制显示和隐藏

v-if 是组件真正的渲染和销毁，而不是显示和隐藏

频繁切换显示状态用 v-show，否则用 v-if

keep-alive 是在vue 框架层级进行的JS 对象渲染

一般简单的可用 v-show，

复杂一点的一般用 keep-alive，keep-alive 通常用于 tab 的切换

## 2.1 keep-alive是什么

keep-alive是一个抽象组件：它自身不会渲染一个DOM元素，也不会出现在父组件链中；使用keep-alive包裹动态组件时，会缓存不活动的组件实例，而不是销毁它们。

用户在某个列表页面选择筛选条件过滤出一份数据列表，由列表页面进入数据详情页面，再返回该列表页面，我们希望：列表页面可以保留用户的筛选（或选中）状态。  
keep-alive就是用来解决这种场景。当然keep-alive不仅仅是能够保存页面/组件的状态这么简单，它还可以避免组件反复创建和渲染，有效提升系统性能。总的来说，**keep-alive用于保存组件的渲染状态。**

## keep-alive用法

* 在动态组件中的应用

<keep-alive :include="whiteList" :exclude="blackList" :max="amount">

<component :is="currentComponent"></component>

</keep-alive>

* 在vue-router中的应用

<keep-alive :include="whiteList" :exclude="blackList" :max="amount">

<router-view></router-view>

</keep-alive>

**include**定义缓存白名单，keep-alive会缓存命中的组件；**exclude**定义缓存黑名单，被命中的组件将不会被缓存；**max**定义缓存组件上限，超出上限使用LRU的策略置换缓存数据。

[内存管理](https://links.jianshu.com/go?to=https%3A%2F%2Fbaike.baidu.com%2Fitem%2F%25E5%2586%2585%25E5%25AD%2598%25E7%25AE%25A1%25E7%2590%2586%2F5633616)的一种页面置换算法，对于在内存中但又不用的[数据块](https://links.jianshu.com/go?to=https%3A%2F%2Fbaike.baidu.com%2Fitem%2F%25E6%2595%25B0%25E6%258D%25AE%25E5%259D%2597%2F107672)（内存块）叫做LRU，操作系统会根据哪些数据属于LRU而将其移出内存而腾出空间来加载另外的数据。

keep-alive.js内部还定义了一些工具函数，我们按住不动，先看它对外暴露的对象

可以看出，与我们定义组件的过程一样，先是设置组件名为keep-alive，其次定义了一个abstract属性，值为true。这个属性在vue的官方教程并未提及，却至关重要，后面的渲染过程会用到。props属性定义了keep-alive组件支持的全部参数。

keep-alive在它生命周期内定义了三个钩子函数：

* created  
  初始化两个对象分别缓存VNode(虚拟DOM)和VNode对应的键集合
* destroyed  
  删除this.cache中缓存的VNode实例。我们留意到，这不是简单地将this.cache置为null，而是遍历调用pruneCacheEntry函数删除。

删除缓存的VNode还要对应组件实例的destory钩子函数

* mounted  
  在mounted这个钩子中对include和exclude参数进行监听，然后实时地更新（删除）this.cache对象数据。pruneCache函数的核心也是去调用pruneCacheEntry
* 第一步：获取keep-alive包裹着的第一个子组件对象及其组件名；
* 第二步：根据设定的黑白名单（如果有）进行条件匹配，决定是否缓存。不匹配，直接返回组件实例（VNode），否则执行第三步；
* 第三步：根据组件ID和tag生成缓存Key，并在缓存对象中查找是否已缓存过该组件实例。如果存在，直接取出缓存值并更新该key在this.keys中的位置（更新key的位置是实现LRU置换策略的关键），否则执行第四步；
* 第四步：在this.cache对象中存储该组件实例并保存key值，之后检查缓存的实例数量是否超过max设置值，超过则根据LRU置换策略删除最近最久未使用的实例（即是下标为0的那个key）;
* 第五步：最后并且很重要，将该组件实例的keepAlive属性值设置为true。

## 4.2 keep-alive组件的渲染

我们用过keep-alive都知道，它不会生成真正的DOM节点，这是怎么做到的？

Vue在初始化生命周期的时候，为组件实例建立父子关系会根据abstract属性决定是否忽略某个组件。在keep-alive中，设置了abstract:true，那Vue就会跳过该组件实例。

**最后构建的组件树中就不会包含keep-alive组件，那么由组件树渲染成的DOM树自然也不会有keep-alive相关的节点了。**

**keep-alive包裹的组件是如何使用缓存的？**  
在patch阶段，会执行createComponent函数：

* 在首次加载被包裹组建时，由keep-alive.js中的render函数可知，vnode.componentInstance的值是undfined，keepAlive的值是true，因为keep-alive组件作为父组件，它的render函数会先于被包裹组件执行；那么只执行到i(vnode,false)，后面的逻辑不执行；
* 再次访问被包裹组件时，vnode.componentInstance的值就是已经缓存的组件实例，那么会执行insert(parentElm, vnode.elm, refElm)逻辑，这样就直接把上一次的DOM插入到父元素中。

2、为何 v-for 要用 key

答案：必须要用 key, 而且不能用 index 和 random，

key是vue中vnode的唯一标记，通过这个key，我们的diff操作可以更准确，更快速

在 diff 算法中用 tag 和 key来判断，是否是sameNode

可以减少渲染次数，提高渲染性能

3、描述 Vue 组件生命周期（有父子组件的情况）

答案：单组件生命周期，生命周期可分为

* 挂载阶段（
  + beforeCreate:此阶段为实例初始化之后,此时数据观察和事件机制还没有形成,不能获取到dom节点；
  + created：此阶段的vue实例已经创建,仍不能获取DOM 节点.把vue 的一个实例给初始化了，只是存在于 js 内存的一个变量而已，这个时候并没有开始渲染；
  + beforeMount:在这一阶段,我们虽然还不能获取到具体 DOM 元素,但 vue 挂载的根节点已经创建,下面 vue 对DOM 的操作将围绕这个根元素继续进行,beforeMount 这个阶段是过渡性的,一般一个项目只能用到一两次；
  + mounted：组件真正绘制完成了，页面已经渲染完了,数据和DOM 都已被渲染出来,一般我们的异步请求都写在这里)
* 更新阶段（
  + beforeUpdate: 这一阶段，vue遵循数据驱动DOM 的原则，beforeUpdate 函数在数据更新后没有立即更新数据，但是DOM 数据会改变，这是双向数据绑定的作用；
  + updated：这一阶段，DOM 会和更改过的内容同步)
* 销毁阶段（
  + beforeDestroy：在上一阶段vue已经成功通过数据驱动DOM 的修改，当我们不再需要 vue 操纵 DOM 时，就要销毁 vue，也就是清除vue 实例与 DOM 的关联，调用destroy方法可以销毁当前组件。在销毁前，会触发 beforeDestroy 钩子函数；
  + destroyed：在销毁后，会触发destroyed 钩子函数）

beforeDestroy要做的事：

* 自定义事件解除绑定：（eventBus 等）
* 销毁定时任务：（setTimeout,setInterval等）
* 绑定的window 或 document 事件要销毁

总之就是该销毁的要在这里销毁，不要让他们留在内存中

具体参考：<https://www.cnblogs.com/queenya/p/13416654.html>

4、Vue 组件如何通讯

答案：

* 父子组件通讯：使用属性和触发事件，props，$emit，this.$emit 调用父组件的事件，父组件向子组件传递一个信息，或者说子组件向父组件触发一个事件
* 组件之间没有关系或层级较深：使用自定义事件 ，event是vue实例，vue本身就具有自定义事件的能力。调用自定义事件： event.$emit('xxx', 变量名)；绑定自定义事件：event.$on('xxx', 函数名字)。在beforeDestroy 要做的一件事是及时解绑自定义事件，及时销毁，否则可能造成内存泄漏，写法：event.$off('xxx', 函数名).
* vuex 通讯： Vuex 实现了一个单向数据流，在全局拥有一个 State 存放数据，当组件要更改 State 中的数据时，必须通过 Mutation 进行，Mutation 同时提供了订阅者模式供外部插件调用获取 State 数据的更新。而当所有异步操作(常见于调用后端接口异步获取更新数据)或批量的同步操作需要走 Action，但 Action 也是无法直接修改 State 的，还是需要通过 Mutation 来修改 State 的数据。最后，根据 State 的变化，渲染到视图上。vuex 是 vue 的状态管理器，存储的数据是响应式的。但是并不会保存起来，刷新之后就回到了初始状态，**具体做法应该在 vuex 里数据改变的时候把数据拷贝一份保存到 localStorage 里面，刷新之后，如果 localStorage 里有保存的数据，取出来再替换 store 里的 state。**
* 多级组件嵌套需要传递数据时，通常使用的方法是通过 vuex。但如果仅仅是传递数据，而不做中间处理，使用 vuex 处理，未免有点大材小用。为此 Vue2.4 版本提供了另一种方法----$attrs/$listeners
* $attrs：包含了父作用域中不被 prop 所识别 (且获取) 的特性绑定 (class 和 style 除外)。当一个组件没有声明任何 prop 时，这里会包含所有父作用域的绑定 (class 和 style 除外)，并且可以通过 v-bind="$attrs" 传入内部组件。通常配合 interitAttrs 选项一起使用。
* $listeners：包含了父作用域中的 (不含 .native 修饰器的) v-on 事件监听器。它可以通过 v-on="$listeners" 传入内部组件

5、描述组件渲染和更新的过程

答案：

* 初次渲染过程：

1. 解析模板为 render 函数（或在开发环境已完成， vue-loader）
2. 触发响应式，监听 data 属性 getter，setter
3. 执行 render 函数，生成 vnode, patch(elem, vnode)

* 更新过程：

1. 修改 data，触发 setter （此前在 getter 中 已被 监听）
2. 重新执行 render 函数，生成 newVnode
3. patch(vnode, newVnode)

* 异步渲染：

1. 回顾 $nextTick，（以下这是对 $nextTick的回顾： $nextTick: vue 是异步渲染；data改变之后，DOM 不会立刻渲染；$nextTick 会在 DOM 渲染之后被触发，以获取最新 DOM 节点。vue 为何是异步渲染，$nextTick何用？异步渲染（以及合并data修改），以提高渲染性能，$nextTick 在DOM 更新完之后，触发回调。另外，在 vue 中可以通过 ref 获取元素：给元素添加ref属性并设置名称，然后通过 this.$refs.ref 属性名称获取该DOM 元素）总结：1、异步渲染，$nextTick待 DOM 渲染完再回调；2、页面渲染时会将 data 的修改做整合，多次data修改只做一次渲染。
2. 汇总 data 的修改，一次性更新视图
3. 减少 DOM 操作次数，提高性能

6、双向事件绑定 v-model 的实现原理

答案：通过 input 元素的 value = this.name

绑定 input 事件 this.name = $event.target.value

data 更新触发 re-render

数据双向绑定的原理可参考：<https://www.cnblogs.com/queenya/p/13426695.html> 的第3 点

7、对 MVVM 的理解

　　M (Model): 数据来源，服务器上业务逻辑操作

　　V (View): 界面，页面

　　VM (ViewModel): view 和 model 的核心枢纽，如 vue.js

　　Model 和 ViewModel 的双向关系：

　　1、 Model 通过 Ajax (服务器) 通信，发送数据给 ViewModel

　　2、 ViewModel 通过 Ajax (服务器) 通信，回传数据给 Model

　　View  和 ViewModel 的双向关系

　　1、 ViewModel 的数据改变，会映射到 View(即可以即时改变 View 显示的内容)

　　2、View 的内容改变，也会同时改变 ViewModel 对应的数据

241、说下你所理解的mvc

View 传送指令到 Controller

Controller 完成业务逻辑后，要求 Model 改变状态

Model 将新的数据发送到 View，用户得到反馈

所有的通信都是单向的。

8、computed 有何特点，computed 和 watch，methods 的区别

答案：

* computed： 有缓存，data 不变不会重新计算；提高性能。
* computed 为什么需要缓存？简单说就是可以提高性能。假设我们有一个性能开销比较大的计算属性A,它需要遍历一个巨大的数组做大量的计算，然后我们可能有其他的属性依赖于A，如果没有缓存，将不可避免的多次执行A 的getter，如果不希望有缓存请用方法代替
* computed 和 methods的区别: computed 计算属性是基于它的响应式依赖进行缓存的，只在相关响应式依赖发生改变时它们才会重新求值，这意味着只要原属性还没发生改变，多次访问相关属性，计算属性会立即返回之前的计算结果，而不必再次执行函数；而 methods 每当触发重新渲染时，调用方法总会再次执行函数
* computed 和 watch的区别：computed 默认只要 getter，不过需要时也可以提供 setter；watch 侦听器，当需要在数据变化时执行异步或开销较大的操作时，watch是最有用的，使用 watch选项允许执行异步操作（访问一个API）,限制我们执行该操作的频率，并在得到最终结果前，设置中间状态，这些都是计算属性无法做到的

（

* computed 是属性

1. 当需要根据已有数据产生一些派生数据的时候，可使用计算属性
2. 注意：计算属性不支持异步操作，因为计算属性一般要绑定到模板中
3. 更重要的一点是：计算属性会缓存调用的结果，提高性能
4. 计算属性必须有返回值，没有返回值就没有意义

* watch 是一个功能：

1. watch不需要返回值，根据某个数据变化执行xx逻辑
2. watch可以执行异步操作

）

* computed 和 watch的使用场景：如果一个数据需要经过复杂计算就用 computed；如果一个数据需要被监听并且对数据做一些操作就用watch；watch擅长处理的场景：一个数据影响多个数据；computed擅长处理的场景：一个数据受多个数据影响

9、为何组件 data 必须是一个函数？

答案：防止组件重用的时候导致数据相互影响。根本上 .vue 文件编译出来是一个类，这个组件是一个class，我们在使用这个组件的时候相当于对class 实现实例化，在实例化的时候执行data，如果 data不是函数的话拿每个组件的实例结果都一样了，共享了，如果 data不是函数的话在一个地方改了，另一个地方也改了。如果data是函数在左边实例化一个右边实例化一个都会执行这个函数，这两个data都在闭包中，两个不会相互影响

10、Ajax 请求应该放在哪个生命周期

答案：应该放在 mounted 生命周期，JS 是单线程的，Ajax 异步获取数据，放在 mounted 之前没有用，只会让逻辑更加混乱

11、如何将组件所有 props 传递给子组件？

答案：$props　　<user v-bind="$props">

12、如何自己实现 v-model？

答案：

13、多个组件有相同逻辑，如何抽离？

答案： 用 mixin, mixin 的一些缺点

mixin 的用法：；定义一个 js文件将export default 中的共有内容写到里面，然后在组件中import，放到 mixin数组中

mixin 的一些缺点：

1. 变量来源不明，不利于阅读。我们希望编程红的变量和方法是可查找的，但是 mixin 引入的内容编辑是不可寻找
2. 多mixin 可能造成命名冲突
3. 迷信和组件可能出现多对多的关系（一个组件引用多个 mixin, 一个mixin被多个组件引用），复杂度较高。多对多是最复杂的关系，很容易剪不断理还乱

在vue3 提出的 Composition API 旨在解决这些问题

14、何时使用异步组件？

答案：

* 加载大组件
* 路由异步加载

15、何时需要使用 keep-alive？

答案：

* 缓存组件，不需要重复渲染
* 如多个静态 tab 页的切换
* 优化性能

16、何时需要使用 beforeDestroy？

答案：

* 解除自定义事件 event.$off
* 清除定时器
* 解绑自定义的 DOM 事件，如 window scroll等

 17、什么是作用域插槽？

答案：父组件通过 slot 获取子组件中的的值：子组件中通过自定义属性绑定数据，父组件通过 template的 v-slot 属性来接收数据



18、vuex 中 action 和 mutation有何区别？

答案：

* action 中处理异步，mutation 不可以
* mutation 做原子操作
* action 可以整合多个 mutation

19、vue-router 常用的路由模式

答案：

* hash 默认：有 #，也就是路由的hash，后面是路由
* H5 history(需要服务端支持)：没有 #，需要服务端支持，无特殊需求可选择 hash模式
* 随着history api的到来，前端路由开始进化了，前面的hashchange，你只能改变#后面的url片段，而history api则给了前端完全的自由
* history api可以分为两大部分：切换和修改
* （1）切换历史状态
* 包括back、forward、go三个方法，对应浏览器的前进，后退，跳转操作
* （2）修改历史状态
* 包括了pushState、replaceState两个方法，这两个方法接收三个参数：stateObj，title，url

20、如何配置 vue-router 异步加载？

答案：异步加载性能会优化很多，配置：component: () => import(......)

21、请用 vnode 描述一个 DOM 结构

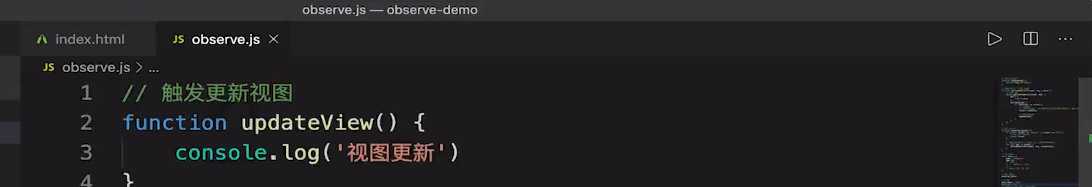
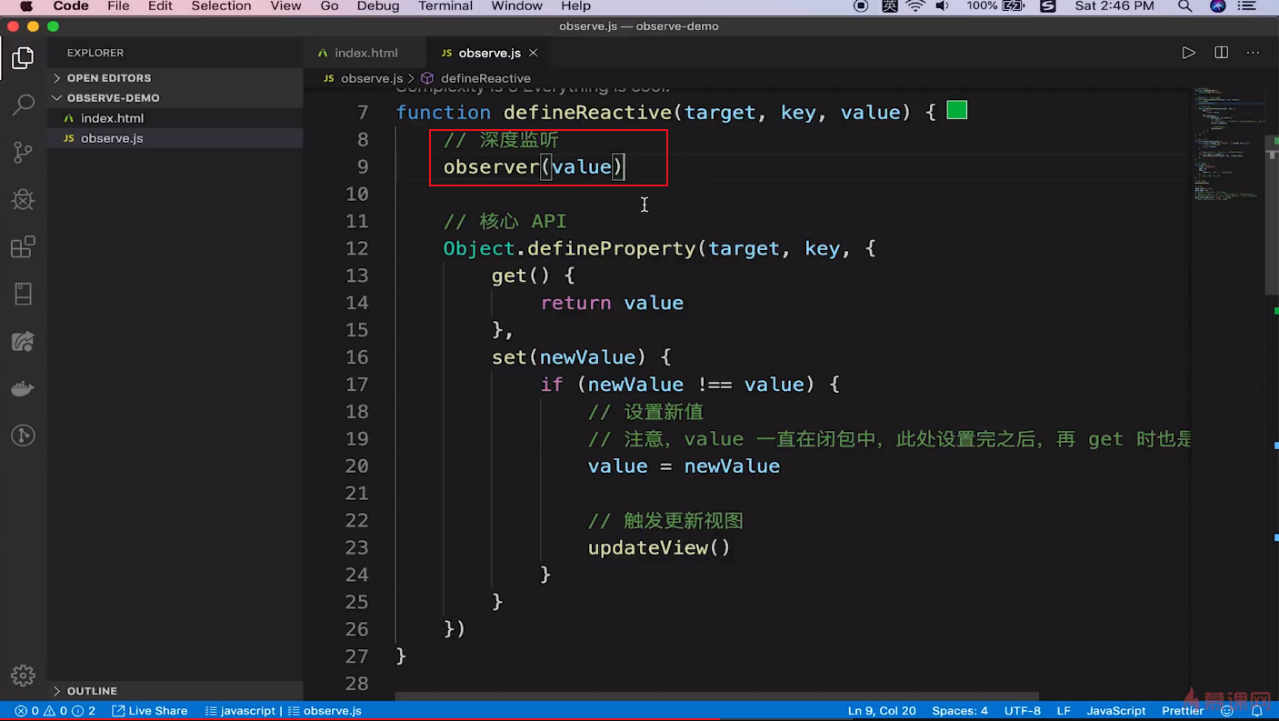
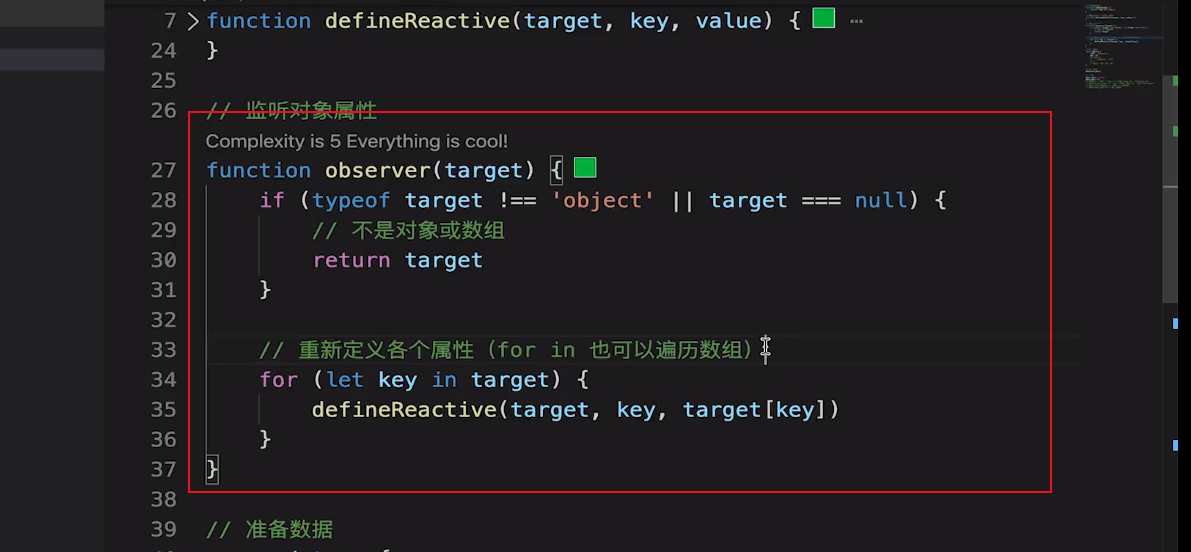
答案：



22、监听 data 变化的核心 API 是什么？

答案： Object.defineProperty,以及要想一下如何深度监听、监听数组，有何缺点

如何深度监听：



24、请描述响应式原理

答案：

* 监听data变化，监听data属性getter，setter（包括数组）

27、Vue 常见性能优化方式

答案：

* 合理使用v-show 和 v-if
* 合理使用computed
* v-for 时要加key，以及避免和 v-if 同时使用
* 自定义事件、DOM 事件及时销毁
* 合理使用异步组件
* 合理使用keep-alive
* data层级不要太深（因为深度监听一次性监听到底）
* 使用 vue-loader 在开发环境做模板编译（预编译）
* webpack层面的优化
* 前端通用的性能优化，如果图片懒加载
* 使用 SSR

28、vuex

vuex 是一个专门为 vue.js 应用程序开发的状态管理模式,

* state,驱动应用的数据源
* view,以声明方式将state映射到视图
* actions,响应在view上的用户输入导致的状态变化

几个基本概念(属性):

1. state:单一状态树,储存的单一状态,是储存的基本数据.vuex 的状态储存是响应式的
2. getters:可以认为是 store的计算属性,对 state加工,是派生出来的数据,返回值会根据它的依赖被缓存起来,且只有当它的依赖值发生了改变才会重新计算
3. mutation:更改 vuex 的store中的状态的唯一方法是提交 mutation(mutation提交修改状态).每个mutation 都有一个字符串的事件类型(type)和一个回调函数(handler),这个回调函数就是我们实际进行状态更改的地方,并且它会接受 state 作为第一个参数,使用 store.commit, (mutation是同步的)
4. action: 像一个装饰器,action提交的是mutation,而不是直接更改状态,action可以包含任意异步操作,通过store.dispatch 方法触发,也可以使用 mapAction
5. module: 是 store 分割的模块,每个模块拥有直接的 state,getter,mutation,action, 甚至是嵌套子模块 -- 从上至下进行同样方式的分割模块内部的 action,局部状态通过 context.state暴露出来,根节点则为 context.rootState

用于Vue 组件(API):

* dispatch
* commit
* mapState
* mapGetters
* mapMutations
* mapActions

vuex的store是如何挂载注入到组件中的呢?

1. 在vue 项目中先安装 vuex
2. 利用vue 的插件机制,使用 vue.use(vuex)时,会调用 vuex 的install方法,安装 vuex
3. applyMixin 方法使用 vue 混入机制,vue的生命周期 beforeCreate 钩子函数混入 vuexInit 方法

vuex是利用 vue 的 mixin 混入机制,在beforeCreate 钩子函数混入 vuexInit 方法,vuexInit 方法实现了 store 注入 vue 组件实例,并注册了 vuex store 的引用属性 $store

vuex 的state 和 getter 是如何映射到各个组件实例中响应式更新状态的?

vuex 的state 状态是响应式,是借助 vue的data是响应式,将 state存入vue实例组件的data中;vuex 的getters则是借助 vue的计算属性 computed 实现数据实时监听

2、module、chunk、bundle分别是什么意思，有何区别

答案：

* module -- 各个源码文件，webpack 中一切皆模块
* chunk -- 多模块合并成的，如 entry import() splitChunk
* bundle -- 最终的输出文件

3、loader 和 plugin 的区别

答案：

loader：模块转换器，如 less --> css, 如识别 js 结尾的，css 结尾的，图片格式结尾的，通过 loader 转换成相应的文件格式

plugin：扩展插件，如 HtmlWebpackPlugin

常见 loader 和 plugin：

<https://www.webpackjs.com/loaders/>，

本人在项目中常用到的loader有

1. babel-loader -- This package allows transpiling JavaScript files using [Babel](https://github.com/babel/babel) and [webpack](https://github.com/webpack/webpack).
2. css-loader -- css-loader 解释(interpret) @import 和 url() ，会 import/require() 后再解析(resolve)它们。引用资源的合适 loader 是 [file-loader](https://www.webpackjs.com/loaders/file-loader/)和 [url-loade](https://www.webpackjs.com/loaders/url-loader/)
3. expose-loader -- The expose loader adds modules to the global object. This is useful for debugging, or [supporting libraries that depend on libraries in globals](https://webpack.js.org/guides/shimming/).，模块必须在你的 bundle 中被 require() 过
4. file-loader -- Instructs webpack to emit the required object as file and to return its public URL
5. json-loader -- 注意：由于 webpack >= v2.0.0 默认支持导入 JSON 文件。如果你使用自定义文件扩展名，你可能仍然需要使用此 loader。See the [v1.0.0 -> v2.0.0 Migration Guide](https://webpack.js.org/guides/migrating/#json-loader-is-not-required-anymore) for more information
6. less-loader -- Compiles Less to CSS.
7. postcss-loader -- Loader for [webpack](http://webpack.js.org/) to process CSS with [PostCSS](http://postcss.org/)
8. style-loader --  Adds CSS to the DOM by injecting a <style> tag
9. url-loader -- Loads files as base64 encoded URL

<https://www.webpackjs.com/plugins/>

4、babel 和 webpack 的区别

答案：

* Babel --> JS 新语法编译工具，不关心模块化
* webpack --> 打包构建工具，是多个 loader plugin的集合

5、webpack 如何实现懒加载

答案：

import()

结合Vue React 异步组件

结合vue-router React-router 异步加载路由

6、为何 Proxy 不能被 Polyfill

答案：

如class 可以用 function 模拟

如 Promise 可以用 callback 模拟

但 Proxy 的功能用 Object.defineProperty 无法模拟（没有任何一个语法可以模拟 Proxy）

9、babel-runtime 和 babel-polyfill 的区别

答案：

* babel-polyfill 会污染全局 支持es6新语法
* babel-runtime 不会污染全局
* 产出第三方 lib 要用 babel-runtime

1. 跨域问题解决方式
2. **（1）通过jsonp跨域**  
   原理：动态添加一个<script>标签，而script标签的src属性是没有跨域的限制的。

<script type="text/javascript">

　　function jsonpCallback(res) {

　　　　alert(res.msg);

　　}

</script>

<script type="text/javascript"src="http://xxx.com/xx?jsonp=jsonpCallback"></script>

**（2）jquery ajax:jq实现**

$.ajax({

　　url: 'http://xxxx.com',

　　type: 'get',

　　dataType: 'jsonp', //请求方式为jsonp

　　jsonpCallback: "callBack",//自定义回调函数名

　　data: {}

});

**JSONP的优点是：**它不像XMLHttpRequest对象实现的Ajax请求那样受到同源策略的限制；它的兼容性更好，在更加古老的浏览器中都可以运行，不需要XMLHttpRequest或ActiveX的支持；并且在请求完毕后可以通过调用callback的方式回传结果。  
**JSONP的缺点则是：**它只支持GET请求而不支持POST等其它类型的HTTP请求；它只支持跨域HTTP请求这种情况，不能解决不同域的两个页面之间如何进行JavaScript调用的问题。

4、原型链的解释

在js里，继承机制是原型继承。继承的起点是 对象的原型（Object prototype）。

一切皆为对象，只要是对象，就会有 proto 属性，该属性存储了指向其构造的指针。

Object prototype也是对象，其 proto 指向null。

 对象分为两种：函数对象和普通对象，只有函数对象拥有『原型』对象（prototype）。

* prototype的本质是普通对象。
* Function prototype比较特殊，是没有prototype的函数对象。
* new操作得到的对象是普通对象。

 当调取一个对象的属性时，会先在本身查找，若无，就根据 proto 找到构造原型，若无，继续往上找。最后会到达顶层Object prototype，它的 proto 指向null，均无结果则返回undefined，结束。

 由 proto 串起的路径就是『原型链』。

5、对闭包的理解，实现一个暴露内部变量，而且外部可以访问修改的函数

闭包是就是函数中的函数，里面的函数可以访问外面函数的变量，外面的变量的是这个内部函数的一部分。

闭包是指有权访问另一函数作用域中的变量的函数。创建闭包的方式是在一个函数内部创建另一个函数。

闭包的主要作用 **可以读取函数内部的变量**，另一个就是**让这些变量的值始终保持在内存中**，不会在调用后被自动清除。

5个简单数据类型（基本数据类型）+ 1个复杂数据类型

undefiend, number string null boolean + object

ES6 新增Symbol

17、你对es6的了解

(1).新增模板字符串（为JavaScript提供了简单的字符串插值功能）

(2).箭头函数

(3).for-of（用来遍历数据—例如数组中的值。）

(4).arguments对象可被不定参数和默认参数完美代替。

(5).ES6将promise对象纳入规范，提供了原生的Promise对象。

Promise是异步编程的一种解决方案，比传统的解决方案--回调函数和事件监听--更合理和强大。

特点：

(1)对象的状态不受外界的影响。promise对象代表一个异步操作，有3种状态：pending(进行中)、Fulfiled(已成功)、Rejected(已失败)。

(2)一旦状态发生改变就不会再变，任何时候都可以得到这个结果。Promise对象的状态改变只有两种可能：从Pending变为Fulfiled和从Pending到Rejected。

缺点：  
 (1)无法取消Promise,一旦新建它就会立即执行，无法中途取消。  
 (2)如果不设置回调函数，Promise内部抛出的错误不会反映到外界。  
 (3)当处于pending状态时，无法得知目前进行到哪个阶段。

(6).增加了let和const命令，用来声明变量。

(7).增加了块级作用域。

(8).let命令实际上就增加了块级作用域。

(9).还有就是引入module模块的概念

22、用promise手写ajax

function getJson(url){

return new Promise((resolve, reject) =>{

var xhr = new XMLHttpRequest();

xhr.open('open', url, true);

xhr.onreadystatechange = function(){

if(this.readyState == 4){

if(this.status = 200){

resolve(this.responseText, this)

}else{

var resJson = { code: this.status, response: this.response }

reject(resJson, this)

}

}

}

xhr.send()

})

}

function postJSON(url, data) {

return new Promise( (resolve, reject) => {

var xhr = new XMLHttpRequest()

xhr.open("POST", url, true)

xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

xhr.onreadystatechange = function () {

if (this.readyState === 4) {

if (this.status === 200) {

resolve(JSON.parse(this.responseText), this)

} else {

var resJson = { code: this.status, response: this.response }

reject(resJson, this)

}

}

}

xhr.send(JSON.stringify(data))

})

}

getJSON('/api/v1/xxx') // => 这里面是就try

.catch( error => {

// dosomething // => 这里就是catch到了error，如果处理error以及返还合适的值

})

.then( value => {

// dosomething // 这里就是final

})

23、手写一个类的继承，并解释一下

继承的形式有很多中，js高程里面归纳了其中，我简单说一下前三种。

**1.原型继承**

function Father () {

this.name = "smd";

this.arr = [1, 2, 3]

}

function Son () { }

Son.prototype = new Father()

var s1 = new Son(), s2 = new Son();

s1.arr.push(4);

console.log(s1.arr) //--------> [1,2,3,4]

console.log(s2.arr) //--------->[1,2,3,4]

// Son的实例化对象s1,s2继承了Father的属性arr,但是s1,s2是同时指向这一属性的

**2.构造函数继承**

1. function Father () {
2. this.name = "smd";
3. this.age = 26
4. };
6. function Son () {
7. Father.call(this)
8. // Father.apply(this)
9. }
10. // 弊端：Son只能继承Father自身的属性，而无法继承Father原型中的方法。

24、解释一下call函数和apply函数的作用，以及用法

改变this的指向。

1.默认绑定 （指 直接调用 foo(), this指向window）

2.隐式绑定（obj.foo(), this指向obj 这里会出现很多坑，下面的问题应该会有解答）

3.显示绑定（利用call、apply、bind改变this）

4.new（var cat = new Animal() , this指向cat对象）

38、手写一个js的深克隆

function deepCopy(obj){

//判断是否是简单数据类型，

if(typeof obj == "object"){

//复杂数据类型

var result = obj.constructor == Array ? [] : {};

for(let i in obj){

result[i] = typeof obj[i] == "object" ? deepCopy(obj[i]) : obj[i];

}

}else {

//简单数据类型 直接 == 赋值

var result = obj;

}

return result;

}

39、for函数里面setTimeout异步问题

for (var i = 1; i <= 5; i++) {

setTimeout(function timer() {

console.log(i);

}, i\*1000)

}

40、手写归并排序

<1>.长度为n(n>1),把该输入序列分成两个长度为n/2的子序列；

<2>.对这两个子序列分别采用归并排序，直到长度n小于2；

<3>.将两个排序好的子序列合并成一个最终的排序序列。

function mergeSort(arr) {

var len = arr.length;

if(len < 2) {

return arr;

} else {

middle = Math.floor(len / 2);

var left = arr.slice(0, middle);

var right = arr.splice(middle);

return merge(mergeSort(left), mergeSort(right));

}

}

function merge(left, right) {

var result = [];

while(left.length && right.length) {

left[0] > right[0] ? result.push(right.shift()): result.push(left.shift());

}

if(left.length) {

result = result.concat(left);

}

if(right.length) {

result = result.concat(right);

}

return result;

、

52、低版本浏览器不支持HTML5标签怎么解决？

IE支持：

IE8或更早的版本不支持HTML5的<nav>标签和媒介查询（MediaQuery）。

我们需要引用css3-mediaqueries.js（或者respond.js）和html5shim.js来提供支持。

如果你不打算使用html5shim.js，可以把<nav>标签用<div>代替。

兼容处理，如下：

<!--[if lt IE 9]>

<script src="http://css3-mediaqueries-js.googlecode.com/files/css3-mediaqueries.js"></script>

<script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>

<![endif]-->

57、Array对象自带的方法，一一列举

**不改变原数组：**  
1、concat()  连接两个或多个数组 不改变原数组 返回被连接数组的一个副本

2、join()  把数组中所有元素放入一个字符串 不改变原数组 返回字符串

3、slice()  从已有的数组中返回选定的元素 不改变原数组 返回一个新数组

4、toString()  把数组转为字符串 不改变原数组 返回数组的字符串形式

**改变原数组：**  
5、pop()  删除数组最后一个元素，如果数组为空，则不改变数组，返回undefined 改变原数组 返回被删除的元素

6、push()  向数组末尾添加一个或多个元素 改变原数组 返回新数组的长度

7、reverse()  颠倒数组中元素的顺序 改变原数组 返回该数组

8、shift()  把数组的第一个元素删除，若空数组，不进行任何操作，返回undefined 改变原数组 返回第一个元素的值

9、sort()  对数组元素进行排序(ascii) 改变原数组 返回该数组

58、若干个数字，怎么选出最大的五个

**var** arr =  [1, 3, 10, 6, 1001,500,6988];  
  
 **function** sortNumber(a,b) {  
 **return** b - a  
 }  
arr= arr.sort(sortNumber)  
 arr= arr.slice(0,5)  
 console.log(arr)

60、常用的排序算法有哪些，介绍一下选择排序

##### 1冒泡排序:

##### 冒泡排序思想：比较相邻两个数的大小（两两比较，除了首尾两个数字，其他数字都要比较两次），将较大（或小）的数换到右边（或左边），这样就能将最大（或最小）的数一个个比较出来，放到一端。循环这个过程，就能将所有数字的顺序排列出来。

1. function bubbleSort(array){
2. for(let i = array.length - 1;i > 0;i--){ *// 冒泡次数*
3. for(let j = 0;j < i;j++){ *// 冒泡循环*
4. if(array[j] > array[j + 1]){
5. let temp = array[j]
6. array[j] = array[j + 1]
7. array[j + 1] = temp
8. }
9. }
10. }
11. return array
12. }

#### 2选择排序

选择排序思想：在序列中将最大（最小）的数找出来，放到一边。之后重复这个过程，直到所有数字都被找出来。

function selectSort(array){

for(let i = 0;i < array.length - 1;i++){

for(let j = i + 1;j < array.length;j++){

if(array[i] > array[j]){

let temp = array[i]

array[i] = array[j]

array[j] = temp

}

}

}

return array

}

#### 3. 计数排序

使用一个额外的哈希，将需要排序的数组的数字变作哈希的键，之后遍历哈希的键，就得到排序后的数组。

#### 插入排序

通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入

#### 堆排序算法

把最大堆堆顶的数字取出，将剩余的数字继续调整为最大堆，再取出，重复过程，知道所有数字全部被取出。

#### 快速排序

设置一个基准数，将基准数交换到数组的末尾，遍历数组，将小于基准数的换到左边，大于基准数的换到右边，之后将基准书换到他们的中间，这时候基准数就完成排序。这称为分区。快速排序就是不断分区的过程。完成第一个分区之后，再对左右两边分成的数组进行分区，重复以上过程，直到数组排序完毕。

61、了解navigator对象吗？

Navigator 对象包含有关浏览器的信息

63、link和@import引入CSS的区别？

首页link和import语法结构不同，前者<link>是[html标签](http://www.divcss5.com/html/h323.shtml)，只能放入[html源代码](http://www.divcss5.com/html/h26.shtml)中使用，后者可看作为[css样式](http://www.divcss5.com/rumen/r29.shtml)，作用是引入css样式功能。import在html使用时候需要<style type="text/css">标签，同时可以直接“@import url(CSS文件路径地址);”放如css文件或[css代码](http://www.divcss5.com/rumen/r95.shtml)里引入其它css文件。

本质上两者使用选择区别不大，但为了软件中编辑布局网页html代码，一般使用link较多，也推荐使用link

66、jquery绑定click的方法有几种

jQuery中提供了四种事件监听方式，分别是bind、live、delegate、on，

对应的解除监听的函数分别是unbind、die、undelegate、off。在开始看他们之前

68、移动端适配问题

<meta name="viewport" content="width=device-width, initial-scale=1.0">

@media 媒体查询

70、做过css动画吗

73、你做过最困难的事情是啥？

74、css3 html5新特性

1新增标签

header：定义文档的页眉 头部

    - nav：定义导航链接的部分

    - footer：定义文档或节的页脚 底部

    - article：定义文章。

    - section：定义文档中的节（section、区段）

    - aside：定义其所处内容之外的内容 侧边

- datalist   标签定义选项列表。请与 input 元素配合使用该元素

2常用新属性

    placeholder\*\*\*\*\*\*<input type="text" placeholder="请输入用户名">占位符  当用户输入的时候 里面的文字消失

    autofocus\*\*\*\*\*\*<input type="text" autofocus>规定当页面加载时 input 元素应该自动获得焦点

    autocomplete\*\*\*\*\*\*<input type="text" autocomplete="off">规定表单是否应该启用自动完成功能  有2个值，一个是on 一个是off      on 代表记录已经输入的值  1.autocomplete 首先需要提交按钮 <br/>2.这个表单您必须给他名字

    required\*\*\*\*\*\*<input type="text" required>必填项  内容不能为空

    accesskey\*\*\*\*\*\*<input type="text" accesskey="s">规定激活（使元素获得焦点）元素的快捷键   采用 alt + s的形式

# 3.CSS3 新增选择器

    - :first-child :选取属于其父元素的首个子元素的指定选择器  
    - :last-child :选取属于其父元素的最后一个子元素的指定选择器

4伪元素选择器（CSS3)

    1. E::first-letter文本的第一个单词或字（如中文、日文、韩文等）

    2. E::first-line 文本第一行；

    3. E::selection 可改变选中文本的样式；

    CSS3的规范里“:”用来表示伪类，“::”用来表示伪元素

5，背景缩放(CSS3)

    通过background-size设置背景图片的尺寸，就像我们设置img的尺寸一样，在移动Web开发中做屏幕适配应用非常广泛。

6 凹凸文字

    text-shadow: 水平位置  垂直位置  模糊距离 阴影颜色;

    text-shadow: 1px 1px 1px #000, -1px -1px 1px #fff;

7 CSS3盒模型

    CSS3中可以通过box-sizing 来指定盒模型，即可指定为content-box、border-box，这样我们计算盒子大小的方式就发生了改变。

    可以分成两种情况：

    1、box-sizing: content-box  盒子大小为 width + padding + border   content-box:此值为其默认值，其让元素维持W3C的标准Box Mode

    2、box-sizing: border-box  盒子大小为 width    就是说  padding 和 border 是包含到width里面的

    注：上面的标注的width指的是CSS属性里设置的width: length，content的值是会自动调整的。

8 过渡(CSS3)

    transition: 要过渡的属性  花费时间  运动曲线  何时开始;

    如果有多组属性变化，还是用逗号隔开

9 2D变形(CSS3) transform

animation:动画名称 动画时间 运动曲线  何时开始  播放次数  是否反方向;

linear： 线性过渡。等同于贝塞尔曲线(0.0, 0.0, 1.0, 1.0)

ease： 平滑过渡。等同于贝塞尔曲线(0.25, 0.1, 0.25, 1.0)

ease-in： 由慢到快。等同于贝塞尔曲线(0.42, 0, 1.0, 1.0)

ease-out： 由快到慢。等同于贝塞尔曲线(0, 0, 0.58, 1.0)

ease-in-out： 由慢到快再到慢。等同于贝塞尔曲线(0.42, 0, 0.58, 1.0)

step-start： 等同于 steps(1, start)

step-end： 等同于 steps(1, end)

infinite： 无限循环

normal： 正常方向

reverse： 反方向运行

alternate： 动画先正常运行再反方向运行，并持续交替运行

alternate-reverse： 动画先反运行再正方向运行，并持续交替运行

    10 触摸事件 touchstart

盒模型

可以分成两种情况：

    1、box-sizing: content-box  盒子大小为 width + padding + border   content-box:此值为其默认值，其让元素维持W3C的标准Box Mode

    2、box-sizing: border-box  盒子大小为 width    就是说  padding 和 border 是包含到width里面的

77、兼容性

、

79、Array的unshift() method的作用是什么？如何连接两个Array？如何在Array里移除一个元素？

unshift() 方法可向数组的开头添加一个或更多元素，并返回新的长度。

Concat()

首先可以给JS的数组对象定义一个函数，用于查找指定的元素在数组中的位置，即索引，代码为：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Array.prototype.indexOf = function(val) {  for (var i = 0; i < this.length; i++) {  if (this[i] == val) return i;  }  return -1;  }; |

然后使用通过得到这个元素的索引，使用js数组自己固有的函数去删除这个元素：

代码为：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Array.prototype.remove = function(val) {  var index = this.indexOf(val);  if (index > -1) {  this.splice(index, 1);  }  }; |

84、如何做一个AJAX Request？

function ajax(a, b, f, e, d, c){ // URL, callback, errorCallback, method, formdata, just a placeholder

c = new XMLHttpRequest;

c.open(e||'GET', a);

c.onload = b;

c.onerror = f;

c.send(d||null);

}

可以利用如下方式发送get和post请求，并且定制error错误处理回调。

function error(e){

console.log('--Error--', this.type);

console.log('this: ', this);

console.log('Event: ', e)

}

ajax(url, callback, error);

ajax(url, callback, error, 'post', {'key': 'val'});

86、前端安全方面有没有了解？XSS和CSRF如何攻防？

一 XSS攻击--跨站脚本攻击

* 1. 通过注入的标签事件触发  
     <body onload="alert('xss')"></body>
  2. 通过注入的带有src属性的标签触发

<iframe src="http://www.xss.com/xss.html"></iframe>

防御：利用正则检查输入内容进行过滤

二 CSRF攻击--跨站请求伪造攻击

1. 利用img可跨域伪造一个get请求：B页面嵌入标签，因为用户在A网站已登录，所以可伪造用户调用一个在A网站登录后才能进行的一个任意操作  
<img style="width:0;" src="http://www.cnblogs.com/mvc/Follow/FollowBlogger.aspx?blogUserGuid=4e8c33d0-77fe-df11-ac81-842b2b196315" />

2. 利用form可跨域伪造一个post请求：不嵌iframe页面会重定向，这样就降低了攻击的隐蔽性

<iframe style="display:none;" src="test2.html"></iframe>

防御：使用token

87、HTTP Response的Header里面都有些啥？

1、HTTP请求方式

说明：主要使用到“GET”和“POST”。

实例：POST /test/tupian/cm HTTP/1.1

分成三部分：

（1）POST：HTTP请求方式

（2）/test/tupian/cm：请求Web服务器的目录地址（或者指令）

（3）HTTP/1.1: URI（Uniform Resource Identifier，统一资源标识符）及其版本

备注：在Ajax中，对应method属性设置。

2、Host

说明：请求的web服务器域名地址

实例：例如web请求URL：http://zjm-forum-test10.zjm.baidu.com:8088/test/tupian/cm

Host就为zjm-forum-test10.zjm.baidu.com:8088

User-Agent

说明：HTTP客户端运行的浏览器类型的详细信息。通过该头部信息，web服务器可以判断到当前HTTP请求的客户端浏览器类别。

实例：User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.8.1.11) Gecko/20071127 Firefox/2.0.0.11

4、Accept说明：指定客户端能够接收的内容类型，内容类型中的先后次序表示客户端接收的先后次序。

实例：Accept:text/xml,application/xml,application/xhtml+xml,text/html;q=0.8,image/png,\*/\*;q=0.5

备注：在Prototyp（1.5）的Ajax代码封装中，将Accept默认设置为“text/javascript, text/html, application/xml, text/xml, \*/\*”。这是因为Ajax默认获取服务器返回的Json数据模式。

在Ajax代码中，可以使用XMLHttpRequest 对象中setRequestHeader函数方法来动态设置这些Header信息。

5、Accept-Language

说明：指定HTTP客户端浏览器用来展示返回信息所优先选择的语言。

实例：Accept-Language: zh-cn,zh;q=0.5 这里默认为中文。

6、Accept-Encoding

说明：指定客户端浏览器可以支持的web服务器返回内容压缩编码类型。表示允许服务器在将输出内容发送到客户端以前进行压缩，以节约带宽。而这里设置的就是客户端浏览器所能够支持的返回压缩格式。

实例：Accept-Encoding: gzip,deflate

备注：其实在百度很多产品线中，apache在给客户端返回页面数据之前，将数据以gzip格式进行压缩。

7、Accept-Charset

说明：浏览器可以接受的字符编码集。

实例：Accept-Charset: gb2312,utf-8;q=0.7,\*;q=0.7

8、Content-Type说明：显示此HTTP请求提交的内容类型。一般只有post提交时才需要设置该属性。

实例：Content-type: application/x-www-form-urlencoded;charset:UTF-8

9、Connection

说明：

表示是否需要持久连接。如果web服务器端看到这里的值为“Keep-Alive”，或者看到请求使用的是HTTP 1.1（HTTP 1.1默认进行持久连接），它就可以利用持久连接的优点，当页面包含多个元素时（例如Applet，图片），显著地减少下载所需要的时间。要实现这一点， web服务器需要在返回给客户端HTTP头信息中发送一个Content-Length（返回信息正文的长度）头，最简单的实现方法是：先把内容写入ByteArrayOutputStream，然 后在正式写出内容之前计算它的大小。

实例：

Connection: keep-alive

10、Keep-Alive说明：显示此HTTP连接的Keep-Alive时间。使客户端到服务器端的连接持续有效，当出现对服务器的后继请求时，Keep-Alive功能避免了建立或者重新建立连接。以前HTTP请求是一站式连接，从HTTP/1.1协议之后，就有了长连接，即在规定的Keep-Alive时间内，连接是不会断开的。

实例：Keep-Alive: 300

11、cookie说明：HTTP请求发送时，会把保存在该请求域名下的所有cookie值一起发送给web服务器。

12、Referer说明：包含一个URL，用户从该URL代表的页面出发访问当前请求的页面

·服务器端返回HTTP头部信息

1、Content-Length说明：表示web服务器返回消息正文的长度

2、Content-Type:说明：返回数据的类型（例如text/html文本类型）和字符编码格式。

实例：Content-Type: text/html;charset=utf-8

3、Date说明：显示当前的时间

　　Cache-Control头域

89、输入URL后发生了什么？

1.DNS域名解析；  
2.建立TCP连接；  
3.发送HTTP请求；  
4.服务器处理请求；  
5.返回响应结果；  
6.关闭TCP连接；  
7.浏览器解析HTML；  
8.浏览器布局渲染；

91、面向对象的特征有哪些？

基于类的面向对象编程有3大特征：封装、继承、多态。

封装：把一个具有相同的特征和行为的客观事物封装成抽象的类。类可以把自己的数据及方法让可信的类或则对象操作，类中的数据和方法可以私有化，让对象中的数据和方法提供了不同程度的保护，但是值得注意的是，在ES6中的类，没有提供类的私有化属性，[参考上文](https://zhuanlan.zhihu.com/p/148185550)，因此严格意义上说，ES中的面向对象思想与传统的基于类的面向对象有一定的差异。

继承：让某一个类型的对象获得另一个类型对象的属性和方法。继承能够使用现有类的所有功能，并能无需重新编写原有类的情况下，对现在功能进行扩展。继承可分为实现继承（直接使用父类的属性及方法）和接口继承（使用属性和方法的名称，在子类中提供实现的能力）。不过在js中继承被简单化为：获取存在对象属性和方法的一种方式。在js中有很多种方式来实现继承下文中我们慢慢来实现关于js的继承。

多态：一个类实例中的相同方法和属性在不同场景有不同的表现形式。多态能够使具有不同内部结构的对象共享相同的外部接口。在java中的面向对象中的多态是建立在继承和重载的基础之上的，它能够创建具有多种形式的变量，函数或对象。它主要用途在于不同类型的对象响应相同名称的方法，字段或属性调用的能力，每个对象根据适当的类型特定行为进行响应。但是前端中ES语言本身就具有一种多态的特性，具体下文介绍。

92、做一个两栏布局，左边fixed width，右边responsive，用纸笔手写

102、冒泡和捕获，事件流哪三个阶段？

捕获阶段 (从根节点开始顺着目标节点构建一条事件路径，即事件由页面元素接收，逐级向下，到具体的元素)

目标阶段 (到达目标节点，即元素本身)

冒泡阶段 (从目标节点顺着捕获阶段构建的路径回去， 即跟捕获相反具体元素本身，逐级向上，到页面元素)

取消冒泡

function stopBubble(e) {

//如果提供了事件对象，则这是一个非IE浏览器

if ( e && e.stopPropagation )

//因此它支持W3C的stopPropagation()方法

e.stopPropagation();

else

//否则，我们需要使用IE的方式来取消事件冒泡

window.event.cancelBubble = true;

}

取消默认事件

function stopDefault( e ) {

//阻止默认浏览器动作(W3C)

if ( e && e.preventDefault )

e.preventDefault();

//IE中阻止函数器默认动作的方式

else

window.event.returnValue = false;

return false;

}

103、实现事件代理

由于事件在冒泡阶段向上传播到父节点，因此可以把子节点监听的事件在父节点的监听，由父节点的监听函数统一处理多个子元素的的事件，这种方法就叫做事件的代理，处理如下：

var ul = document.querySelector('ul');

ul.addEventListener('click', function (event) {

if (event.target.tagName.toLowerCase() === 'li') {

// some code

}

});

106、ajax，原生ajax的四个过程

1.new一个实例对象，new XMLHttpRequest（）;

2打开请求，开（参数1，参数2，参数3）参数一可以为：获取或交参数2为我们请求的地址，参数

3是同步异步默认异步3.发送请求发送（空）

4使用。 onreadystatechange去监听新的实例，在监听当中，还有状态码判断readyState == 4 && status == 200请求过来的不是我们想要的格式我们可以使用，JSON.parse（xhr.responseText）转换我们所需要的格式

function loadXMLDoc()

{

var xmlhttp;

if (window.XMLHttpRequest)

{

// IE7+, Firefox, Chrome, Opera, Safari 浏览器执行代码

xmlhttp=new XMLHttpRequest();

}

else

{

// IE6, IE5 浏览器执行代码

xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");

}

xmlhttp.onreadystatechange=function()

{

if (xmlhttp.readyState==4 && xmlhttp.status==200)

{

document.getElementById("myDiv").innerHTML=xmlhttp.responseText;

}

}

xmlhttp.open("GET","/try/ajax/ajax\_info.txt",true);

xmlhttp.send();

}

108、css:两个块状元素上下的margin-top和margin-bottom会重叠。啥原因？怎么解决？

109、js：写一个递归。就是每隔5秒调用一个自身，一共100次

111、网络分层结构

124、如何保持登录状态

125、浮动的原理以及如何清除浮动

127、原生js添加class怎么添加，如果本身已经有class了，会不会覆盖，怎么保留？

129、如果页面初始载入的时候把ajax请求返回的数据存在localStorage里面，然后每次调用的时候去localStorage里面取数，是否可行。

132、数组去除一个函数。用arr.splice。又问splice返回了什么？应该返回的是去除的元素。

140、Get和post的区别？

145、Post一个file的时候file放在哪的？

149、静态属性怎么继承

151、jquery和zepto有什么区别

157、移动端要注意哪些？

159、rem是什么？em是什么？如果上一层就是根root了，em和rem等价么？

1. em的值并不是固定的；

2. em会继承父级元素的字体大小。

   rem是[CSS3](http://www.html5cn.org/portal.php?mod=list&catid=16)新增的一个相对单位（root em，根em），这个单位引起了广泛关注。这个单位与em有什么区别呢？区别在于使用rem为元素设定字体大小时，仍然是相对大小，但相对的只是HTML根元素。这个单位可谓集相对大小和绝对大小的优点于一身，通过它既可以做到只修改根元素就成比例地调整所有字体大小，又可以避免字体大小逐层复合的连锁反应。目前，除了IE8及更早版本外，所有浏览器均已支持rem。

165、怎么在页面里放置一个很简单的图标，不能用img和background-img？

canvas，或者一些库有icon库，data-icon

166、正则表达式判断url

167、怎么去除字符串前后的空格

正则匹配^\s和\s$并且替代，Jquery的$.trim，string.trim()

168、实现页面的局部刷新

局部自动刷新：没有请求局部页面也会自动刷新

1、通过定时器函数如setTimeout(),让Ajax每隔一段时间从服务器获取数据；

2、通过DOM方法或jQuery方法将数据载入页面，实现页面局部自动刷新。

178、ES6里头的箭头函数的this对象与其他的有啥区别

普通函数中的this:

this总是代表它的直接调用者, 例如 obj.func , 那么func中的this就是obj

在默认情况，没找到直接调用者, 则this指的是 window

在严格模式下,没有直接调用者的函数中的this是 undefined

使用call,apply, bind(ES5新增)绑定的, this指的是 绑定的对象

箭头函数中的this：

默认指向在定义时作用域, 它所处的对象,而不是执行时的对象, 定义它的时候,可能环境是window（即继承父级的this）

191、说下你知道的响应状态码

1XX Informational（信息性状态码） 接收的请求正在处理

2XX Success（成功状态码） 请求正常处理完毕

3XX Redirection（重定向状态码） 需要进行附加操作以完成请求

4XX Client Error（客户端错误状态码） 服务器无法处理请求

5XX Server Error（服务端错误状态码） 服务器处理请求出错

194、cookie与session的区别

(1)、cookie数据存放在客户的浏览器上，session数据放在服务器上 ；

(2)、cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗,如果主要考虑到安全应当使用session ；

(3)、session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能，如果主要考虑到减轻服务器性能方面，应当使用COOKIE ；

(4)、单个cookie在客户端的限制是3K，就是说一个站点在客户端存放的COOKIE不能3K；

(5)、所以将登陆信息等重要信息存放为SESSION;其他信息如果需要保留，可以放在COOKIE中。

3、cookie和session的共同之处在于：cookie和session都是用来跟踪浏览器用户身份的会话方式。

4、cookie 是一种发送到客户浏览器的文本串句柄，并保存在客户机硬盘上，可以用来在某个WEB站点会话间持久的保持数据。

65、介绍一下cookie,localstorage,sessionstorage,session

**存储大小：**  
　　cookie在4K以内。  
　　sessionStorage和localStorage在5M以内。

**有效时间：**  
　　cookie：如果未设置过期时间，关闭浏览器时清空；如果设置了有效时间则在到期后清空。  
　　sessionStorage：关闭浏览器时清空。  
　　localStorage：关闭浏览器不清空，手动清除时清空。

**数据与服务器之间的交互方式：**  
　　cookie：自动携带数据发送到服务器。  
　　sessionStorage和localStorage不自动发送数据给服务器，仅在本地保存。

**数据作用域：**  
　　cookie和localStorage在同源窗口可以共享。  
　　sessionStorage即使在同源窗口也不共享。

**storage事件：**  
　　cookie和sessionStorage不支持。  
　　localStorage支持。

**path路径：**  
　　cookie有路径的概念，可以限制cookie只属于某个路径下。  
　　sessionStorage和localStorage没有path的概念。

**易用性：**  
　　原生的cookie接口不友好，需要自己封装。  
　　sessionStorage和localStorage的接口调用更方便。

**cookie和session有什么联系和区别：**

cookie：浏览器用来保存用户信息的文件，可以保存用户是谁，购物车中有哪些商品。  
session：指我们访问网站的一个周期。用户打开一个站点，然后点击多个超链接查看各个网页，然后关闭浏览器，这个过程称为一个会话。

因为http协议是无状态的协议，也就是网页一旦关闭，浏览器和服务端的连接就会断开，下次打开网页又要重新连接，这样可以缓解服务器压力，但是随着网页内容越来越丰富，  
单词请求成本增加，这样的效率就降低了。使用cookie和session可以缓解这种问题。

联系和区别：  
session需要借助cookie才能正常工作。  
（1）存储位置：cookie存放在客户端，session存放在服务端。  
（2）安全：cookie的安全性不是很高，任何人都可以在控制台输入document.cookie获取到cookie，考虑安全性应该要使用session。  
（3）效率问题：session会在一定时间保存在服务器上，访问增多服务器压力增大，考虑到服务器性能应当使用cookie。  
（4）大小：cookie的大小不能超过4K，session没有大小限制。  
（5）登录信息等重要的信息存放在session中，其他信息可以放在cookie中。

116、Cookie 是否会被覆盖，localStorage是否会被覆盖

Cookie是可以覆盖的

134、Cookie跨域请求能不能带上

前端设置 withCredentials: true

服务器端设置：

header("Access-Control-Allow-Credentials: true");

header("Access-Control-Allow-Origin: http://www.xxx.com");

3、http请求头，请求体，cookie在哪个里面？url在哪里面？

**客户端请求消息**

每个HTTP请求和响应都会带有相应的头部信息。默认情况下，在发送XHR请求的同时，还会发送下列头部信息：

* Accept:浏览器能够处理的内容类型
* Accept-Charset:浏览器能够显示的字符集
* Accept-Encoding：浏览器能够处理的压缩编码
* Accept-Language：浏览器当前设置的语言
* Connection：浏览器与服务器之间连接的类型
* Cookie：当前页面设置的任何Cookie
* Host：发出请求的页面所在的域
* Referer：发出请求的页面的URL
* User-Agent：浏览器的用户代理字符串

HTTP响应头部信息：

Date：表示消息发送的时间，时间的描述格式由rfc822定义  
server:服务器名字。  
Connection：浏览器与服务器之间连接的类型  
content-type:表示后面的文档属于什么MIME类型  
Cache-Control：控制HTTP缓存

HTTP响应也由四个部分组成，分别是：状态行、消息报头、空行和响应正文。

198、前端优化你知道哪些

1. 减少 HTTP请求数

(3). 资源合并与压缩  
　　如果可以的话，尽可能的将外部的脚本、样式进行合并，多个合为一个。另外， CSS、 Javascript、Image 都可以用相应的工具进行压缩，压缩后往往能省下不少空间。  
　　(4). CSS Sprites  
　　合并 CSS图片，减少请求数的又一个好办法。  
　　　(6). Lazy Load Images（自己对这一块的内容还是不了解）

2. 将外部脚本置底（将脚本内容在页面信息内容加载后再加载）

　5. 将 CSS放在 HEAD中

7. 减少不必要的 HTTP跳转

204、遇到过哪些浏览器兼容性问题

1，浏览器兼容问题一：不同浏览器的标签默认的margin和padding不同

问题症状：随便写几个标签，不加样式控制的情况下，各自的margin 和padding差异较大。

碰到频率：100%

解决方案：

CSS里加一行

1

\*{margin:0；padding:0；}

备注：这个是最常见的也是最易解决的一个浏览器兼容性问题，几乎所有的CSS文件开头都会用通配符\*来设置各个标签的内外补丁是0。

2，浏览器兼容问题二：块属性标签float后，又有横行的margin情况下，在IE6显示margin比设置的大

问题症状：常见症状是IE6中后面的一块被顶到下一行

碰到频率：90%（稍微复杂点的页面都会碰到，float布局最常见的浏览器兼容问题）

解决方案：在float的标签样式控制中加入 display:inline；将其转化为行内属性

备注：最常用的就是div+CSS布局了，而div就是一个典型的块属性标签，横向布局的时候通常都是用div float实现的，横向的间距设置如果用margin实现，这就是一个必然会碰到的兼容性问题。

3，浏览器兼容问题三：设置较小高度标签（一般小于10px），在IE6，IE7，遨游中高度超出自己设置高度

问题症状：IE6、7和遨游里这个标签的高度不受控制，超出自己设置的高度

碰到频率：60%

解决方案：给超出高度的标签设置overflow:hidden；或者设置行高line-height 小于你设置的高度。

备注：这种情况一般出现在设置小圆角背景的标签里。出现这个问题的原因是IE8之前的浏览器都会给标签一个最小默认的行高的高度。即使你的标签是空的，这个标签的高度还是会达到默认的行高。

4，浏览器兼容问题四：行内属性标签，设置display:block后采用float布局，又有横行的margin的情况，IE6间距bug

问题症状：IE6里的间距比超过设置的间距

碰到几率：20%

解决方案：在display:block；后面加入display:inline；display:table；

备注：行内属性标签，为了设置宽高，需要设置display:block；(除了input标签比较特殊)。在用float布局并有横向的margin后，在IE6下，它就具有了块属性float后的横向margin的bug。不过因为它本身就是行内属性标签，所以再加上display：inline的话，它的高宽就不可设了。

5，浏览器兼容问题五：图片默认有间距

问题症状：几个img标签放在一起的时候，有些浏览器会有默认的间距，加了问题一中提到的通配符也不起作用。

碰到几率：20%

解决方案：使用float属性为img布局

备注：因为img标签是行内属性标签，所以只要不超出容器宽度，img标签都会排在一行里，但是部分浏览器的img标签之间会有个间距。去掉这个间距使用float是正道。（我的一个学生使用负margin，虽然能解决，但负margin本身就是容易引起浏览器兼容问题的用法，所以我禁止他们使用）

6，浏览器兼容问题六：标签最低高度设置min-height不兼容

问题症状：因为min-height本身就是一个不兼容的CSS属性，所以设置min-height时不能很好的被各个浏览器兼容

碰到几率：5%

解决方案：如果要设置一个标签的最小高度200px，需要进行的设置为：{min-height:200px； height:auto ！important； height:200px； overflow:visible；}

备注：在B/S系统前端开时，有很多情况下有这种需求。当内容小于一个值（如300px）时。容器的高度为300px；当内容高度大于这个值时，容器高度被撑高，而不是出现滚动条。这时候就会面临这个兼容性问题。

225、js中this的作用

1.当函数作为对象的方法调用时，this指向该对象。

2.当函数作为淡出函数调用时，this指向全局对象（严格模式时，为undefined）

3.构造函数中的this指向新创建的对象

4.嵌套函数中的this不会继承上层函数的this，如果需要，可以用一个变量保存上层函数的this。

再总结的简单点，如果在函数中使用了this，只有在该函数直接被某对象调用时，该this才指向该对象。

253、window.onload和$(document).ready()的区别，浏览器加载转圈结束时哪个时间点？

|  |  |  |
| --- | --- | --- |
|  | window.onload | $(document).ready() |
| 执行时机 | 必须等待网页全部加载完毕（包括  图片等），然后再执行JS代码 | 只需要等待网页中的DOM结构  加载完毕，就能执行JS代码 |
| 执行次数 | 只能执行一次，如果第二次，那么  第一次的执行会被覆盖 | 可以执行多次，第N次都不会被上  一次覆盖 |
| 简写方案 | 无 | $(function () {}) |

，window.onload很少直接去使用，因为他需要等待图片之类的大型元素加载完毕后才能执行JS代码。遇到网速慢或者存在大的图片要下载时，页面已经全面展开，图片还在缓慢加载，这时页面上任何的JS交互功能全部处在假死状态。并且只能执行单次在多次开发和团队开发中会带来困难。

254、form表单当前页面无刷新提交 target iframe

1. <form action="提交的action" method="post" target="theID">
2. .....................
3. </form>
4. <iframe name="theID" style="display: none;"></iframe>

提交到action后，action返回一串javascript语句

Java代码   收藏代码

1. String script = "<script>alert('ok!');</script>";
2. response.getOutputStream().write(script.getBytes("utf-8"));

alert('ok')将在当前页执行。原理是form提交后的结果在target指定的iframe里执行  
而iframe将其隐藏。这样提交后的效果就和无刷新的效果一样。

256、如何避免多重回调—promise，promise简单描述一下，如何在外部进行resolve()

258、伪类和伪元素区别

伪类和伪元素的根本区别在于：**它们是否创造了新的元素。**

伪元素 div:before

伪类：a:hover

a:first-child

260、数组去重

## 一、利用ES6 Set去重（ES6中最常用）

function unique (arr) {

return Array.from(new Set(arr))

}

## 二、利用for嵌套for，然后splice去重（ES5中最常用）

**function** **unique**(arr){

**for**(var i=0; i<arr.length; i++){

**for**(var j=i+1; j<arr.length; j++){

**if**(arr[i]==arr[j]){ //第一个等同于第二个，splice方法删除第二个

arr.splice(j,1);

j*--;*

}

}

}

**return** arr;

}

var arr = [1,1,'true','true',**true**,**true**,15,15,**false**,**false**, undefined,undefined, **null**,**null**, **NaN**, **NaN**,'NaN', 0, 0, 'a', 'a',{},{}];

console.log(**unique**(arr))

263、实现bind函数

1. var a = 10;
2. var test = function() *{*
3. *console.log(this.a);*
4. *}*
5. *// 如果直接执行test，最终打印的是10.*
6. var bindTest = test.bind(*{a: "111"}*)
7. 从上面这个例子可以看出来，bind函数改变了test函数中this的指向。 除此之外，bind函数还有两个特殊的用法，一个是柯里化，一个是绑定构造函数无效。

267、http状态码。。。401和403区别？

总的说401相应应该用来表示缺失或错误的认证，403表示用户认证后，但权限不足，无法对该资源进行操作

284、vue的特点？双向数据绑定是如何实现的

#### 1) 轻量级的框架

Vue.js 能够自动追踪依赖的模板表达式和计算属性，提供 MVVM 数据绑定和一个可组合的组件系统，具有简单、灵活的 API，使读者更加容易理解，能够更快上手。

#### 2) 双向数据绑定

声明式渲染是数据双向绑定的主要体现，同样也是 Vue.js 的核心，它允许采用简洁的模板语法将数据声明式渲染整合进 DOM。

#### 3) 指令

Vue.js 与页面进行交互，主要就是通过内置指令来完成的，指令的作用是当其表达式的值改变时相应地将某些行为应用到 DOM 上。

#### 4) 组件化

组件（Component）是 Vue.js 最强大的功能之一。组件可以扩展 HTML 元素，封装可重用的代码。  
  
在 Vue 中，父子组件通过 props 传递通信，从父向子单向传递。子组件与父组件通信，通过触发事件通知父组件改变数据。这样就形成了一个基本的父子通信模式。  
  
在开发中组件和 HTML、JavaScript 等有非常紧密的关系时，可以根据实际的需要自定义组件，使开发变得更加便利，可大量减少代码编写量。  
  
组件还支持热重载（hotreload）。当我们做了修改时，不会刷新页面，只是对组件本身进行立刻重载，不会影响整个应用当前的状态。CSS 也支持热重载。

#### 5) 客户端路由

Vue-router 是 Vue.js 官方的路由插件，与 Vue.js 深度集成，用于构建单页面应用。Vue 单页面应用是基于路由和组件的，路由用于设定访问路径，并将路径和组件映射起来，传统的页面是通过超链接实现页面的切换和跳转的。

#### 6) 状态管理

状态管理实际就是一个单向的数据流，State 驱动 View 的渲染，而用户对 View 进行操作产生 Action，使 State 产生变化，从而使 View 重新渲染，形成一个单独的组件。

285、Object.defineProperty

Object.defineProperty()的作用就是直接在一个对象上定义一个新属性，或者修改一个已经存在的属性

Object.defineProperty(obj, prop, desc)