**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。

## 一个场景

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

// src/core/components/keep-alive.js

export default {

name: 'keep-alive',

abstract: true, // 判断当前组件虚拟dom是否渲染成真实dom的关键

props: {

include: patternTypes, // 缓存白名单

exclude: patternTypes, // 缓存黑名单

max: [String, Number] // 缓存的组件

},

created() {

this.cache = Object.create(null) // 缓存虚拟dom

this.keys = [] // 缓存的虚拟dom的键集合

},

destroyed() {

for (const key in this.cache) {

// 删除所有的缓存

pruneCacheEntry(this.cache, key, this.keys)

}

},

mounted() {

// 实时监听黑白名单的变动

this.$watch('include', val => {

pruneCache(this, name => matched(val, name))

})

this.$watch('exclude', val => {

pruneCache(this, name => !matches(val, name))

})

},

render() {

// 先省略...

}

}

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。

// src/core/components/keep-alive.js

function pruneCacheEntry (

cache: VNodeCache,

key: string,

keys: Array<string>,

current?: VNode

) {

const cached = cache[key]

if (cached && (!current || cached.tag !== current.tag)) {

cached.componentInstance.$destroyed() // 执行组件的destroy钩子函数

}

cache[key] = null

remove(keys, key)

}

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

function pruneCache (keepAliveInstance: any, filter: Function) {

const { cache, keys, \_vnode } = keepAliveInstance

for (const key in cache) {

const cachedNode: ?VNode = cache[key]

if (cachedNode) {

const name: ?string = getComponentName(cachedNode.componentOptions)

if (name && !filter(name)) {

pruneCacheEntry(cache, key, keys, \_vnode)

}

}

}

}

* render

render () {

const slot = this.$slots.defalut

const vnode: VNode = getFirstComponentChild(slot) // 找到第一个子组件对象

const componentOptions : ?VNodeComponentOptions = vnode && vnode.componentOptions

if (componentOptions) { // 存在组件参数

// check pattern

const name: ?string = getComponentName(componentOptions) // 组件名

const { include, exclude } = this

if (// 条件匹配

// not included

（include && (!name || !matches(include, name))）||

// excluded

(exclude && name && matches(exclude, name))

) {

return vnode

}

const { cache, keys } = this

// 定义组件的缓存key

const key: ?string = vnode.key === null ? componentOptions.Ctor.cid + (componentOptions.tag ? `::${componentOptions.tag}` : '') : vnode.key

if (cache[key]) { // 已经缓存过该组件

vnode.componentInstance = cache[key].componentInstance

remove(keys, key)

keys.push(key) // 调整key排序

} else {

cache[key] = vnode //缓存组件对象

keys.push(key)

if (this.max && keys.length > parseInt(this.max)) {

//超过缓存数限制，将第一个删除

pruneCacheEntry(cahce, keys[0], keys, this.\_vnode)

}

}

vnode.data.keepAlive = true //渲染和执行被包裹组件的钩子函数需要用到

}

return vnode || (slot && slot[0])

}

* 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

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

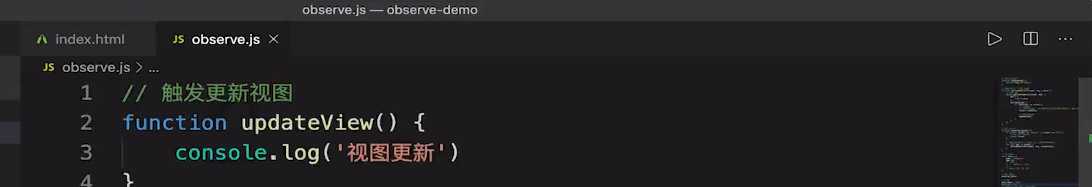
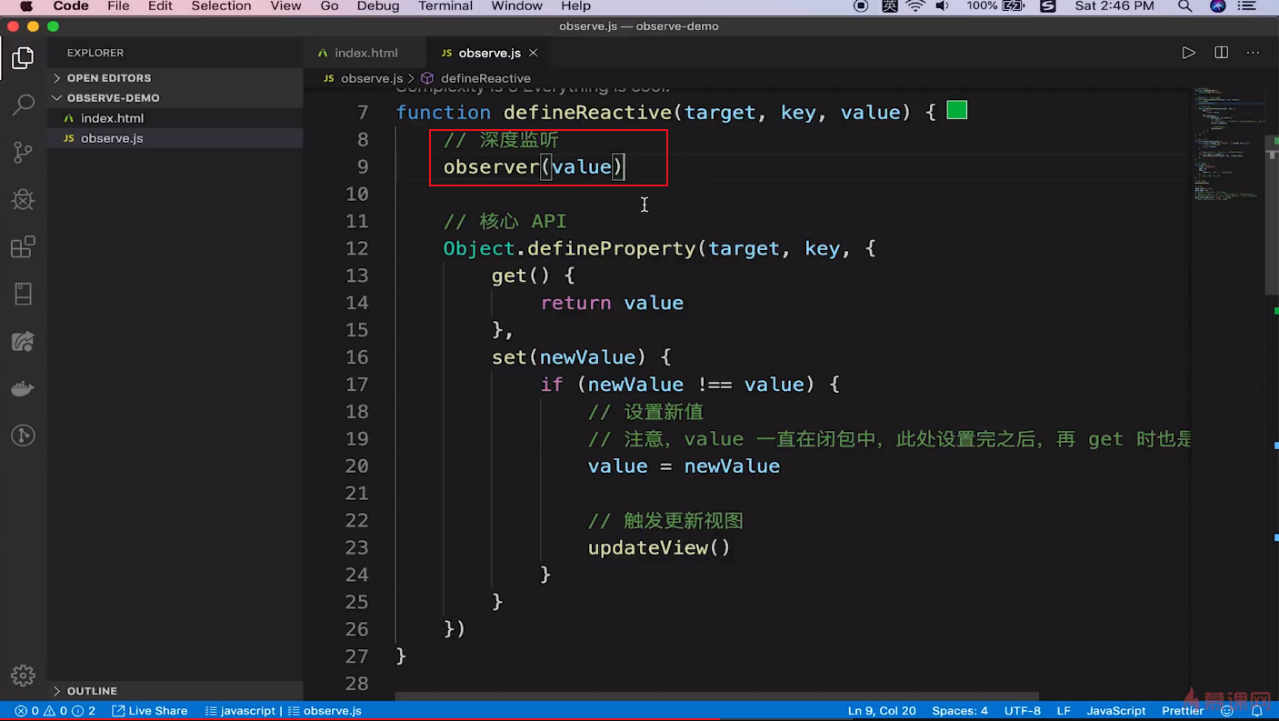
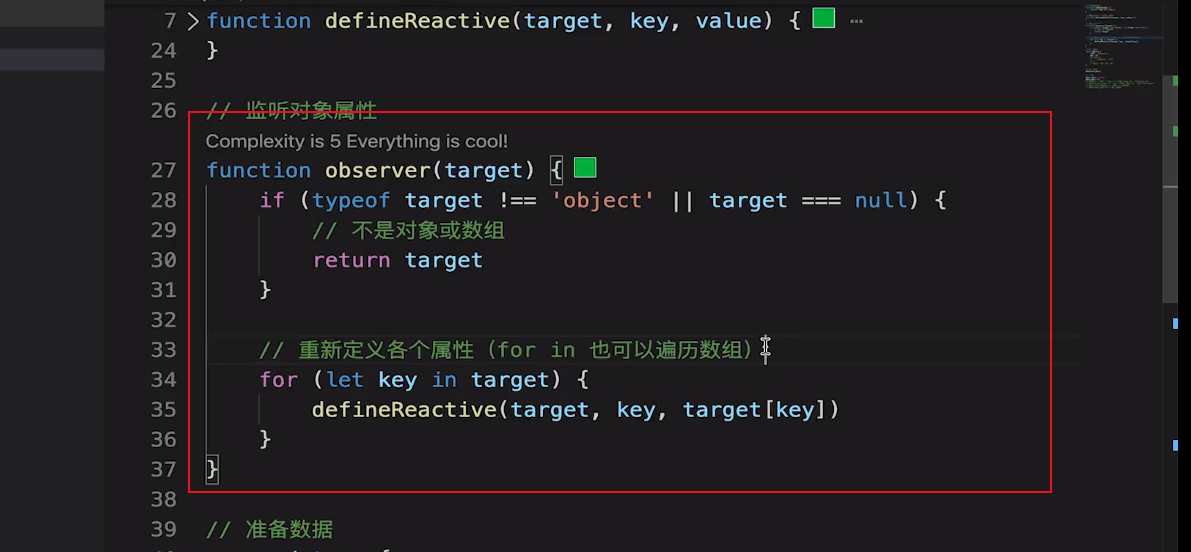
答案：



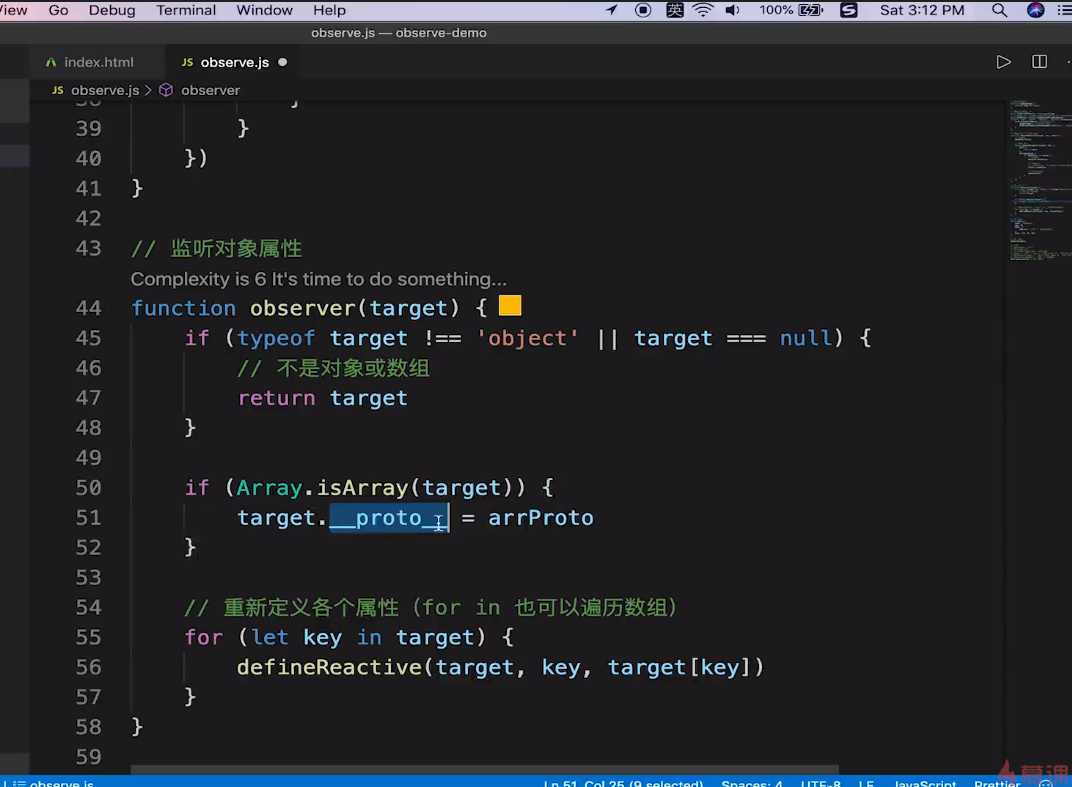
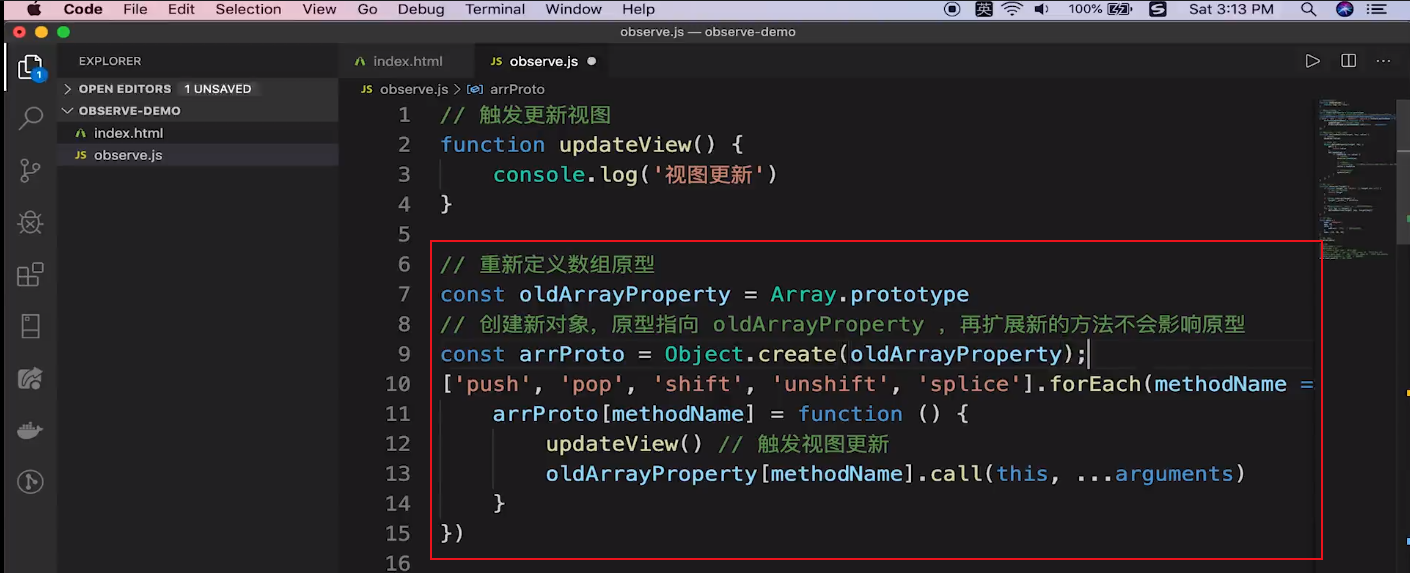
22、监听 data 变化的核心 API 是什么？

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

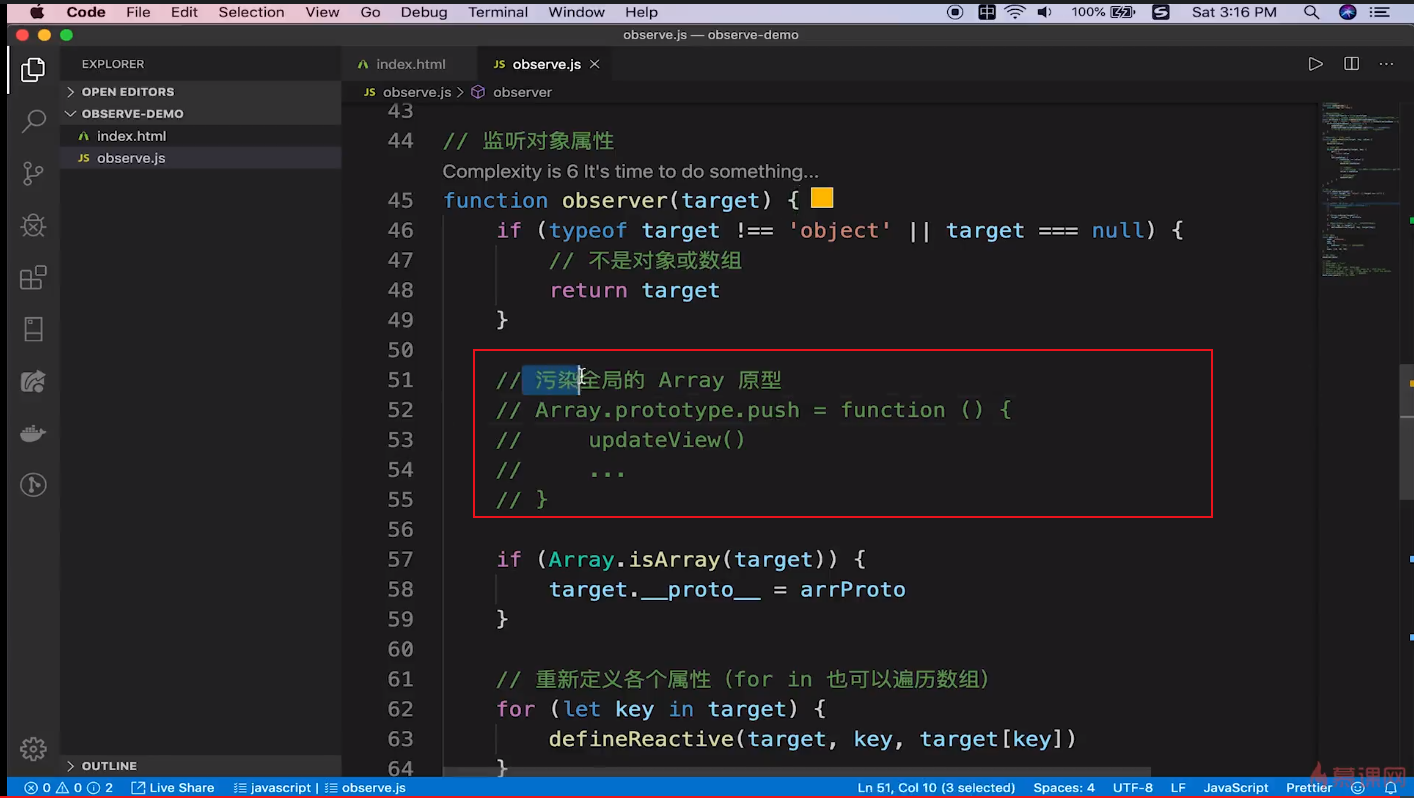
如何深度监听：



如何监听数组：



不可像以下这样做会污染全局的Array 原型：



缺点：

* + 深度监听，需要递归到底，一次性计算量大
  + 无法监听新增/删除属性（所以需要 vue.set vue.delete 实现新增/删除属性）
  + 无法监听原生数组，需要特殊处理

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）参考：<https://www.cnblogs.com/queenya/p/13426695.html> 第3点
* 组件渲染和更新的流程：

25、diff 算法的时间复杂度

答案：

O(n)

在O(n^3)基础上做了一些调整

26、简述diff算法过程：

答案：

* patch(elem, vnode) 和 patch(vnode, newVnode)
* patchVnode 和 addVnode 和 removeVnode
* updateChildren(key 的重要性)

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、手写jsonp的实现

2、不过我们又发现，Web页面上调用js文件时则不受是否跨域的影响（不仅如此，我们还发现凡是拥有"src"这个属性的标签都拥有跨域的能力，比如script、img、iframe）。

3、于是可以判断，当前阶段如果想通过纯web端（ActiveX控件、服务端代理、属于未来的HTML5之Websocket等方式不算）跨域访问数据就只有一种可能，那就是在远程服务器上设法把数据装进js格式的文件里，供客户端调用和进一步处理。

4、恰巧我们已经知道有一种叫做JSON的纯字符数据格式可以简洁的描述复杂数据，更妙的是JSON还被js原生支持，所以在客户端几乎可以随心所欲的处理这种格式的数据。

5、这样子解决方案就呼之欲出了，web客户端通过与调用脚本一模一样的方式，来调用跨域服务器上动态生成的js格式文件（一般以JSON为后缀），显而易见，服务器之所以要动态生成JSON文件，目的就在于把客户端需要的数据装入进去。

6、客户端在对JSON文件调用成功之后，也就获得了自己所需的数据，剩下的就是按照自己需求进行处理和展现了，这种获取远程数据的方式看起来非常像AJAX，但其实并不一样。

7、为了便于客户端使用数据，逐渐形成了一种非正式传输协议，人们把它称作JSONP，该协议的一个要点就是允许用户传递一个callback参数给服务端，然后服务端返回数据时会将这个callback参数作为函数名来包裹住JSON数据，这样客户端就可以随意定制自己的函数来自动处理返回数据了。

JSONP的客户端具体实现：

1、我们知道，哪怕跨域js文件中的代码（当然指符合web脚本安全策略的），web页面也是可以无条件执行的。

远程服务器remoteserver.com根目录下有个remote.js文件代码如下：

jQuery如何实现jsonp调用？

这里针对ajax与jsonp的异同再做一些补充说明：

1、ajax和jsonp这两种技术在调用方式上"看起来"很像，目的也一样，都是请求一个url，然后把服务器返回的数据进行处理，因此jquery和ext等框架都把jsonp作为ajax的一种形式进行了封装。

2、但ajax和jsonp其实本质上是不同的东西。ajax的核心是通过XmlHttpRequest获取非本页内容，而jsonp的核心则是动态添加script标签来调用服务器提供的js脚本。

3、所以说，其实ajax与jsonp的区别不在于是否跨域，ajax通过服务端代理一样可以实现跨域，jsonp本身也不排斥同域的数据的获取。

4、还有就是，jsonp是一种方式或者说非强制性协议，如同ajax一样，它也不一定非要用json格式来传递数据，如果你愿意，字符串都行，只不过这样不利于用jsonp提供公开服务。

总而言之，jsonp不是ajax的一个特例，哪怕jquery等巨头把jsonp封装进了ajax，也不能改变这一点！

2、手写单链表查找倒数第k个节点

1、为了找出倒数第k个元素，最容易想到的办法是首先遍历一遍单链表，求出整个单链表的长度n，然后将倒数第k个，转换为正数第n-k个，接下来遍历一次就可以得到结果。但是该方法存在一个问题，即需要对链表进行两次遍历，第一次遍历用于求解单链表的长度，第二次遍历用于查找正数第n-k个元素。

这种思路的时间复杂度是O(n)，但需要遍历链表两次。

2、如果我们在遍历时维持两个指针，第一个指针从链表的头指针开始遍历，在第k-1步之前，第二个指针保持不动；在第k-1步开始，第二个指针也开始从链表的头指针开始遍历。由于两个指针的距离保持在k-1，当第一个（走在前面的）指针到达链表的尾结点时，第二个指针（走在后面的）指针正好是倒数第k个结点。这种思路只需要遍历链表一次。对于很长的链表，只需要把每个结点从硬盘导入到内存一次。因此这一方法的时间效率前面的方法要高。

class Node{

Node next=null;

int data;

public Node(int data){

this.data=data;

}

}

public class MyLinkedList {

Node head=null;//链表头的引用

public Node findElem(Node head,int k){

if(k<1||k>this.length()){

return null;

}

Node p1=head;

Node p2=head;

for(int i=0;i<k;i++)

p1=p1.next;

while(p1!=null){

p1=p1.next;

p2=p2.next;

}

return p2;

}

public static void main(String[] args) {

MyLinkedList list=new MyLinkedList();

list.addNode(1);

list.addNode(2);

list.addNode(3);

list.addNode(4);

list.addNode(5);

MyLinkedList p=new MyLinkedList();

p.head=list.findElem(list.head, 3);

p.printList();

}

}

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。

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

闭包的作用：

**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。

6、基本的数据类型

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

9、OSI模型，HTTP,TCP,UDP分别在哪些层

这个可以参考我另一个博客：

OSI：物理层-数据链路层-网络层-传输层-会话层-表现层-应用层

13、在jquery方法和原型上面添加方法的区别和实现，以及jquery对象的实现

参考上一个问题答案~

使用jquery的第一件事就是要使用jquery对象，jquery对象和javascript中的DOM对象是不同的。

什么是jquery对象？jquery将一个DOM对象转化为jquery对象后就可以使用jquery类库提供的各种函数。可以将jquery对象理解为一个类，并且封装了很多的方法，而且可以动态的通过加载插件扩展这个类。除了jQuery工具函数，jQuery的操作都是从jQuery对象开始。比如：

attr(key,value)

<img id="myphoto" alt="my photo" src=""/>

$("#myphoto").attr("src","/pic/1.jpg");

jQuery对象是一个特殊的集合对象。即使只有一个元素，jQuery对象仍然是一个集合。说其特殊是因为实际上jQuery对象是包含一个集合对象和各种函数的类。

14、手写一个递归函数

function fact(num) {

if (num <= 1) {

return 1;

} else {

return num \* fact(num - 1);

}

}

以下代码可导致出错：

var anotherFact = fact;

fact = null;

alert(antherFact(4)); //出错

1

2

3

由于fact已经不是函数了，所以出错。

用arguments.callee可解决问题，这是一个指向正在执行的函数的指针,arguments.callee返回正在被执行的对现象。

新的函数为：

function fact(num) {

if (num <= 1) {

return 1;

} else {

return num \* arguments.callee(num - 1); //此处更改了。

}

}

var anotherFact = fact;

fact = null;

alert(antherFact(4)); //结果为24.

15、对前端路由的理解？前后端路由的区别？

前端的路由和后端的路由在实现技术上不一样，但是原理都是一样的。在 HTML5 的 history API 出现之前，前端的路由都是通过 hash 来实现的，hash 能兼容低版本的浏览器。

http://10.0.0.1/

http://10.0.0.1/#/about

http://10.0.0.1/#/concat

1

2

3

服务端路由：每跳转到不同的URL，都是重新访问服务端，然后服务端返回页面，页面也可以是服务端获取数据，然后和模板组合，返回HTML，也可以是直接返回模板HTML，然后由前端JS再去请求数据，使用前端模板和数据进行组合，生成想要的HTML。

前端路由：每跳转到不同的URL都是使用前端的锚点路由，实际上只是JS根据URL来操作DOM元素，根据每个页面需要的去服务端请求数据，返回数据后和模板进行组合，当然模板有可能是请求服务端返回的，这就是 SPA 单页程序。

在js可以通过window.location.hash读取到路径加以解析之后就可以响应不同路径的逻辑处理。

history 是 HTML5 才有的新 API，可以用来操作浏览器的 session history (会话历史)。基于 history 来实现的路由可以和最初的例子中提到的路径规则一样。

H5还新增了一个hashchange事件，也是很有用途的一个新事件：

当页面hash(#)变化时，即会触发hashchange。锚点Hash起到引导浏览器将这次记录推入历史记录栈顶的作用，window.location对象处理“#”的改变并不会重新加载页面，而是将之当成新页面，放入历史栈里。并且，当前进或者后退或者触发hashchange事件时，我们可以在对应的事件处理函数中注册ajax等操作！

但是hashchange这个事件不是每个浏览器都有，低级浏览器需要用轮询检测URL是否在变化，来检测锚点的变化。当锚点内容(location.hash)被操作时，如果锚点内容发生改变浏览器才会将其放入历史栈中，如果锚点内容没发生变化，历史栈并不会增加，并且也不会触发hashchange事件。

16、介绍一下webpack和gulp，以及项目中具体的使用

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

20、前后端分离的意义以及对前端工程化的理解

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）

26、对前端前景的展望，以后前端会怎么发展

27、手写第一次面试没有写出来的链表问题，要求用es6写

33、手写一个简单遍历算法

34、解释一下react和vue，以及区别

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;

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

39、for函数里面setTimeout异步问题

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;

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

41、介绍自己的项目

略

42、实现两个排序数组的合并

参考42题中的merge函数。

43、手写一个原生ajax

ajax：一种请求数据的方式，不需要刷新整个页面；

ajax的技术核心是 XMLHttpRequest 对象；

ajax 请求过程：创建 XMLHttpRequest 对象、连接服务器、发送请求、接收响应数据；

一个在stackoverflow的高分回答结合上面的代码，给出get和post的两种不同请求方法：

var ajax = {};

ajax.x = function () {

if (typeof XMLHttpRequest !== 'undefined') {

return new XMLHttpRequest();

}

var versions = [

"MSXML2.XmlHttp.6.0",

"MSXML2.XmlHttp.5.0",

"MSXML2.XmlHttp.4.0",

"MSXML2.XmlHttp.3.0",

"MSXML2.XmlHttp.2.0",

"Microsoft.XmlHttp"

];

var xhr;

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

try {

xhr = new ActiveXObject(versions[i]);

break;

} catch (e) {

}

}

return xhr;

};

ajax.send = function (url, method, data, success,fail,async) {

if (async === undefined) {

async = true;

}

var x = ajax.x();

x.open(method, url, async);

x.onreadystatechange = function () {

if (x.readyState == 4) {

var status = x.status;

if (status >= 200 && status < 300) {

success && success(x.responseText,x.responseXML)

} else {

fail && fail(status);

}

}

};

if (method == 'POST') {

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

}

x.send(data)

};

ajax.get = function (url, data, callback, fail, async) {

var query = [];

for (var key in data) {

query.push(encodeURIComponent(key) + '=' + encodeURIComponent(data[key]));

}

ajax.send(url + (query.length ? '?' + query.join('&') : ''), 'GET', null, success, fail, async)

};

ajax.post = function (url, data, callback, fail, async) {

var query = [];

for (var key in data) {

query.push(encodeURIComponent(key) + '=' + encodeURIComponent(data[key]));

}

ajax.send(url,'POST', query.join('&'), success, fail, async)

};

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

使用方法：GET

ajax.get('/test.php', {foo: 'bar'}, function(response,xml) {

//success

},

function(status){

//fail

});

1

2

3

4

5

6

POST

ajax.post('/test.php', {foo: 'bar'}, function(response,xml) {

//succcess

},function(status){

//fail

});

1

2

3

4

5

6

7

44、手写一个promise版的ajax

45、手写实现一个promise

46、手写实现requireJS模块实现

47、手写实现jquery里面的insertAfter

48、react和vue的介绍以及异同

49、AMD和CMD，commonJS的区别

50、介绍一下backbone

51、了解过SEO吗？

52、低版本浏览器不支持HTML5标签怎么解决？

53、用js使低版本浏览器支持HTML5标签 底层是怎么实现的？

54、实现一个布局：左边固定宽度为200，右边自适应，而且滚动条要自动选择只出现最高的那个

55、画出盒子模型，要使谷歌浏览器的盒子模型显示得跟IE浏览器一致（让谷歌跟ie一致，不是ie跟谷歌一致），该怎么做？

56、手写JS实现类继承，讲原型链原理，并解释new一个对象的过程都发生了什么

57、Array对象自带的方法，一一列举

58、若干个数字，怎么选出最大的五个

59、Array对象自带的排序函数底层是怎么实现的？

60、常用的排序算法有哪些，介绍一下选择排序

61、了解navigator对象吗？

62、手写一个正则表达式，验证邮箱

63、link和@import引入CSS的区别？

64、刚才说有些浏览器不兼容@import，具体指哪些浏览器？

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

66、jquery绑定click的方法有几种

67、你的优点/竞争力

68、移动端适配问题

69、react的难点在哪里

70、做过css动画吗

71、如何优化网站

72、以后的规划

73、你做过最困难的事情是啥？

74、css3 html5新特性

75、闭包，ES6，跨域

76、问做过啥项目，用到什么技术，遇到什么困难

77、兼容性

78、盒子模型

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？

80、用纸笔写一个Closure，任意形式和内容

81、知不知道Array-like Object？

82、如何用Native JavaScript来读写Cookie？

83、知不知道CSS Box-model？

84、如何做一个AJAX Request？

85、Cross-domain access有没有了解？

86、前端安全方面有没有了解？XSS和CSRF如何攻防？

87、HTTP Response的Header里面都有些啥？

88、知不知道HTTP2？

89、输入URL后发生了什么？

90、new operator实际上做了什么？

91、面向对象的属性有哪些？

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

93、讲一下AJAX Request

94、讲一下Cross-domain access

95、介绍一下做过的项目

96、问到了多个服务器怎么弄，架构之类的

97、angular的渲染流程

98、脏检查

99、nodejs的架构、优缺点、回调

100、css 盒模型

101、css 布局，左边定宽右边自适应

102、冒泡和捕获，事件流哪三个阶段？

103、实现事件代理

104、原型链

105、继承的两种方法

106、ajax，原生ajax的四个过程

107、闭包，简单说一个闭包的应用，然后闭包的主要作用是什么

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

110、cookie和session有什么区别

111、网络分层结构

112、你的不足是什么？

113、做了那么多项目，有没有自己的归纳总结

114、工程怎么进行文件管理

115、less和sass掌握程度

116、Cookie 是否会被覆盖，localStorage是否会被覆盖

117、事件代理js实现

118、Css实现动画效果

119、Animation还有哪些其他属性

120、Css实现三列布局

121、Css实现保持长宽比1:1

122、Css实现两个自适应等宽元素中间空10个像素

123、requireJS的原理是什么

124、如何保持登录状态

125、浮动的原理以及如何清除浮动

126、Html的语义化

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？

128、Jsonp的原理。怎么去读取一个script里面的数据？

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。

130、304是什么意思？有没有方法不请求不经过服务器直接使用缓存

131、http请求头有哪些字段

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。

133、js异步的方法（promise，generator，async）

134、Cookie跨域请求能不能带上

135、最近看什么开源项目？

136、commonJS和AMD

137、平时是怎么学习的？

138、为什么要用translate3d？

139、对象中key-value的value怎么再放一个对象？

140、Get和post的区别？

145、Post一个file的时候file放在哪的？

146、说说你对组件的理解

147、组件的html怎么进行管理

148、js的异步加载，promise的三种状态，ES7中的async用过么

149、静态属性怎么继承

150、js原型链的继承

151、jquery和zepto有什么区别

152、angular的双向绑定原理

153、angular和react的认识

154、MVVM是什么

155、移动端是指手机浏览器，还是native，还是hybrid

156、你用了移动端的什么库类和框架？

157、移动端要注意哪些？

158、适配有去考虑么，retina屏幕啊？

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？

160、怎么测试的？会自动化测试么？

161、你觉得你什么技术最擅长？

162、你平时有没有什么技术的沉淀？

163、单向链表怎么查找有没有环？

164、怎么得到一个页面的a标签？

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

166、正则表达式判断url

167、怎么去除字符串前后的空格

168、实现页面的局部刷新

169、绝对定位与相对定位的区别

170、js轮播实现思路

171、使用js画一个抛物线，抛物线上有个小球随着抛物线运动，有两个按钮能使小球继续运动停止运动

172、java五子棋，说下实现思路

173、如何让各种情况下的div居中(绝对定位的div,垂直居中,水平居中)？

174、display有哪些值？说明他们的作用

175、css定义的权重

176、requirejs实现原理

177、requirejs怎么防止重复加载

178、ES6里头的箭头函数的this对象与其他的有啥区别

179、tcp/udp区别

180、tcp三次握手过程

181、xss与csrf的原理与怎么防范

182、mysql与 MongoDB的区别

183、w3c事件与IE事件的区别

184、有没有上传过些什么npm模块

185、IE与W3C怎么阻止事件的冒泡

186、gulp底层实现原理

187、webpack底层实现原理

188、gulp与webpack区别

189、vuejs与angularjs的区别

190、vuex是用来做什么的

191、说下你知道的响应状态码

192、ajax的过程以及 readyState几个状态的含义

193、你除了前端之外还会些什么？

194、cookie与session的区别

195、一些关于php与java的问题

196、你觉得你哪个项目是你做的最好的

197、说说你在项目中遇到了哪些困难,是怎么解决的

198、前端优化你知道哪些

199、webpack是用来干嘛的

200、webpack与gulp的区别

201、es6与es7了解多少

202、说下你知道的响应状态码

203、看过哪些框架的源码

204、遇到过哪些浏览器兼容性问题

205、清除浮动有哪几种方式,分别说说

206、你知道有哪些跨域方式,分别说说

207、JavaScript有哪几种类型的值

208、使用 new操作符时具体是干了些什么

209、学习前端的方法以及途径

210、怎么实现两个大整数的相乘，说下思路

211、你学过数据结构没,说说你都了解些什么

212、你学过计算机操作系统没,说说你都了解些什么

213、你学过计算机组成原理没,说说你都了解些什么

214、你学过算法没,说说你都了解些什么

215、说下选择排序,冒泡排序的实现思路

216、用过哪些框架

217、让你设计一个前端css框架你怎么做

218、了解哪些设计模式说说看

219、说下你所了解的设计模式的优点

220、vue源码结构

221、状态码

222、浏览器缓存的区别

223、304与200读取缓存的区别

224、http请求头有哪些,说说看你了解哪些

225、js中this的作用

226、js中上下文是什么

227、js有哪些函数能改变上下文

228、你所了解的跨域的方法都说说看你了解的？

229、要是让你自己写一个js框架你会用到哪些设计模式

230、平常在项目中用到过哪些设计模式,说说看

231、一来给了张纸要求写js自定义事件

232、前端跨域的方法

233、call与apply的区别

234、h5有个api能定位你知道是哪个吗？

235、vue与angularjs中双向数据绑定是怎样实现的？

236、webpack怎样配置？

237、nodejs中的文件怎么读写？

238、link和@import有什么区别？

239、cookies，sessionStorage 和 localStorage 的区别

240、看过哪些前端的书？平时是怎么学习的

241、说下你所理解的mvc与mvvc

242、position有哪些值,说下各自的作用

243、写个从几个li中取下标的闭包代码

244、你的职业规划是怎么样的？

245、移动端性能优化

246、lazyload如何实现

247、点透问题

248、前端安全

249、原生js模板引擎

250、repaint和reflow区别

251、requirejs如何避免循环依赖？

252、实现布局：左边一张图片，右边一段文字（不是环绕）

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

254、form表单当前页面无刷新提交 target iframe

255、setTimeout和setInterval区别，如何互相实现？

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()

257、margin坍塌？水平方向会不会坍塌？

258、伪类和伪元素区别

259、vue如何实现父子组件通信，以及非父子组件通信

260、数组去重

261、使用flex布局实现三等分，左右两个元素分别贴到左边和右边，垂直居中

262、平时如何学前端的，看了哪些书，关注了哪些公众号

263、实现bind函数

264、数组和链表区别，分别适合什么数据结构

265、对mvc的理解

266、描述一个印象最深的项目，在其中担任的角色，解决什么问题

267、http状态码。。。401和403区别？

268、描述下二分查找

269、为什么选择前端，如何学习的，看了哪些书，《js高级程序设计》和《你不知道的js》有什么区别，看书，看博客，看公众号三者的时间是如何分配的？

270、如何评价BAT？

271、描述下在实习中做过的一个项目，解决了什么问题，在其中担任了什么角色？这个过程存在什么问题，有什么值得改进的地方？

272、如何看待加班，如果有个项目需要连续一个月加班，你怎么看？

273、遇到的压力最大的一件事是什么？如何解决的？

274、平时有什么爱好

275、自身有待改进的地方

276、n长的数组放入n+1个数，不能重复，找出那个缺失的数

277、手里有什么offer

278、你对于第一份工作最看重的三个方面是什么？

279、如何评价现在的前端？

280、用原生js实现复选框选择以及全选非全选功能

281、用4个颜色给一个六面体上色有多少种情况

282、amd和cmd区别

283、为什么选择前端，移动端性能优化

284、vue的特点？双向数据绑定是如何实现的

285、Object.defineProperty

286、算法题：数组去重，去除重复两次以上的元素，代码题：嵌套的ul-li结构，根据input中输入的内容，去除相应的li节点，且如果某个嵌套的ul下面的li都被移除，则该ul的父li节点也要被移除

287、页面加载过程

288、浏览器如何实现图片缓存

————————————————

版权声明：本文为CSDN博主「傻傻的鱼」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/qq\_22944825/article/details/78169321