一、js 的数据类型有哪些？

Number String Undefined Null Boolean 和一种复杂的数据类型 Array，Function，Object

二、js 的事件循环机制

首先，Javasript 是单线程的语言，为了解决单线程的阻塞问题，就出现了事件的异步。

例如：

1、普通事件，如 click、resize 等

2、资源加载，如 load、error 等

3、定时器，包括 setInterval、setTimeout 等

4、promise，async await

以上这些都属于异步任务。

而异步任务的有俩个队列. 一个是 microtask 队列（微任务），一个 macrotask （宏任务）队列.

所以 js 的事件循环就是说

1. 主线程任务执行完了
2. 开始执行微任务队列的事件，每次处理直到队列为空
3. 开始执行宏任务队列的事件，每次只处理宏任务队列里第一个任务
4. 宏任务第一个任务执行完了，就又回到微任务队列里 （重复 2.3）

注意： new Promise(fn)中的 fn 是同步执行（属于主线程的一部分）；
microtask 微任务队列里边的优先级 process.nextTick()>Promise.then()
macrotask 宏任务队列里边的优先级 setTimeout>setImmediate。

https://note.youdao.com/web/#/file/WEBb017bdf106cf4f0880f03385fa288be7/note/WEB9c6b97e7a3385883e105d4363d0b3510/

三、什么是函数执行栈？

栈内存：存放基本类型。 堆内存：存放引用类型（**在栈内存中存一个基本类型值保存对象在堆内存中的地址，用于引用这个对象**。）

基本类型在当前执行环境结束时销毁，而引用类型不会随执行环境结束而销毁，只有当所有引用它
的变量不存在时这个对象才被垃圾回收机制回收。

当一个方法执行时，每个方法都会建立自己的内存栈，在这个方法内定义的变量将会逐个放入这块栈内存里，（如果方法里的有一个变量 var obj = {}；那么 obj 变量存放对象的访问地址，存放在栈内存中，真正的对象值放在堆内存中），随着方法的执行结束，这个方法的内存栈也将自然销毁了（但如果这个方法返回了一个方法，被返回的方法在外层又被调用的话，即闭包，那么这个

内存栈暂时不会被销毁的）。因此，所有在方法中定义的变量都是放在栈内存中的；而放在堆内存中的对象不会随方法的结束而销毁，即使方法结束后，这个对象还可能被另一个引用变量所引用（方法的参数传递时很常见），则这个对象依然不会被销毁，只有当一个对象没有任何引用变量引用它时，系统的垃圾回收机制才会在核实的时候回收它。
执行栈（函数调用栈）
理解完栈的存取方式，我们接着分析 JavaScript 中如何通过栈来管理多个执行上下文。
程序执行进入一个执行环境时，它的执行上下文就会被创建，并被推入执行栈中(入栈)；
程序执行完成时，它的执行上下文就会被销毁，并从栈顶被推出(出栈)，控制权交由下一个执行上下文。
因为 JS 执行中最先进入全局环境，所以处于"栈底的永远是全局环境的执行上下文"。而处于"栈顶的是当前正在执行函数的执行上下文"，当函数调用完成后，它就会从栈顶被推出（理想的情况下，闭包会阻止该操作，闭包后续文章深入详解）。
"全局环境只有一个，对应的全局执行上下文也只有一个，只有当页面被关闭之后它才会从执行栈中被推出，否则一直存在于栈底"

四、什么是事件委托？

在 JavaScript 中，事件委托 *Event delegation* 是一种事件的响应机制，当需要监听不存在的元素或是动态生成的元素时，可以考虑事件委托。
事件委托得益于事件冒泡，当监听子元素时，事件冒泡会通过目标元素向上传递到父级，直到 *document*，如果子元素不确定或者动态生成，可以通过监听父元素来取代监听子元素。
通过给父级而非子集添加事件，当事件被抛到更上层的父节点的时候，我们通过检查事件的目标对象 *target* 来判断并获取事件源 *li*，当子节点被点击的时候，*click* 事件会从子节点开始向上冒泡。父节点捕获到事件之后，通过判断 *e.target.nodeName* 来判断是否为我们需要处理的节点。并且通过 *e.target* 拿到了被点击的 *Li* 节点。从而可以获取到相应的信息，并作处理。

currentTarget 始终是监听事件者，而 **target** 是事件的真正发出者。

五、事件的三个阶段
target.addEventListener(type, listener[, useCapture]); 第三个参数是 false，全部事件默认是冒泡阶段触发。
DOM 事件流：将事件分为三个阶段：捕获阶段、目标阶段、冒泡阶段。先调用捕获阶段的处理函数，其次调用目标阶段的处理函数，最后调用冒泡阶段的处理函数。

六、MVVM 框架双向绑定

Model 指的是数据部分，对应到前端就是 Javascript 对象，（react 中的 state，反正就是 object 对象携带的数据）View 指的是视图部分，对应到前端就是 DOM,ViewModel 就是连接视图和数据的中间件，在 MVM 架构下视图和数据是不能直接通讯的,通常会通过 ViewModel 来做通讯，ViewModel 通常要实现一个 Observer 观察者,当数据发生变化，ViewModel 能够观察到这种数据的变化，然后通知到对应的视图进行更新，而当用户操作视图 ViewModel 也能监听到视图的变化，然后通知数据做改动，这实际上就实现了数据的双向绑定

七、this 指向

普通函数：this 指向的是最终调用并且执行该函数的对象。**（切记 this 指向的是最终执行该函数的对象）**

箭头函数不绑定 this，会捕获其所在的上下文的 this 值，作为自己的 this 值

八、fun.bind 、fun.call、 fun.apply （三者的调用者都是函数）

bind 返回对应函数，便于稍后调用；apply，call 则是立即调用

bind 、call、 apply 这三个函数的第一个参数都是 this 的指向对象，第二个参数差别就来了：

bind 除了返回是函数以外，它 的参数和 call 一样。

call 的参数是直接放进去的，第二第三第 n 个参数全都用逗号分隔，直接放到后面 obj.myFun.call(db,'成都', ... ,'string');

apply 的所有参数都必须放在一个数组里面传进去 obj.myFun.apply(db,['成都', ..., 'string']);

当然，三者的参数不限定是 string 类型，允许是各种类型，包括函数 、 object 等等！

九、vue 的 method 和 compute 的区别

作用机制上

1.watch 和 computed 都是以 Vue 的依赖追踪机制为基础的，它们都试图处理这样一件事情：当某一个数据（称它为依赖数据）发生变化的时候，所有依赖这个

数据的"相关"数据"自动"发生变化，也就是自动调用相关的函数去实现数据的变动。

2.对 methods:methods 里面是用来定义函数的，很显然，它需要手动调用才能执行。而不像 watch 和 computed 那样，"自动执行"预先定义的函数

【总结】:methods 里面定义的函数，是需要主动调用的，而和 watch 和 computed 相关的函数，会自动调用,完成我们希望完成的作用

从性质上看

1.methods 里面定义的是函数，你显然需要像"fuc()"这样去调用它（假设函数为 fuc）

2.computed 是计算属性，事实上和和 data 对象里的数据属性是同一类的（使用上），

3.watch:类似于监听机制+事件机制：

watch 和 computed 的对比

1.watch 擅长处理的场景：一个数据影响多个数据

2.computed 擅长处理的场景：一个数据受多个数据影响

利用 computed 处理 methods 存在的重复计算情况

只有符合：1.存在依赖型数据 2.依赖型数据发生改变这两个条件,computed 才会重新计算。

而 methods 下的数据，是每次都会进行计算的

而 methods 下的数据，是每次都会进行计算的

https://blog.csdn.net/m0_37257670/article/details/80311478

十、什么是深拷贝和浅拷贝？

浅拷贝是按位拷贝对象，**它会创建一个新对象**，这个对象有着原始对象属性值的一份精确拷贝。如果属性是基本类型，拷贝的就是基本类型的值；如果属性是内存地址（引用类型），拷贝的就是内存地址 ，因此如果其中一个对象改变了这个地址，就会影响到另一个对象。即默认拷贝构造函数只是对对象进行浅拷贝复制(逐个成员依次拷贝)，

浅拷贝只复制对象的第一层属性。

深拷贝，**它也会创建一个新对象**，对原始对象的属性进行递归复制。修改新对象的引用类型属性的值不会影响到旧对象得引用类型属性，因为它们不共享内存。**深拷贝**会另外创造一个一模一样的对象，新对象跟原对象不共享内存，修改新对象不会改到原对象。

十、一个页面从输入 URL 到页面加载的的过程发生了什么？

域名解析 --> 发起 TCP 的 3 次握手 --> 建立 TCP 连接后发起 http 请求 --> 服务器响应 http 请求，浏览器得到 html 代码 --> 浏览器解析 html 代码，并请求

html 代码中的资源（如 js、css、图片等） --> 浏览器对页面进行渲染呈现给用户

十一、性能优化？

一、减少 HTTP 请求

合理设置 HTTP 缓存

资源合并与压缩

合并 CSS 图片，减少请求数的又一个好办法。

减少 DNS 查找

使用外部的 JavaScript 和 CSS

Lazy Load Images（自己对这一块的内容还是不了解）

减少不必要的 HTTP 跳转

避免重复的资源请求

减少作用域链查找（这方面设计到一些内容的相关问题）

减少闭包

节流防抖

https://www.cnblogs.com/yzhihao/p/9385467.html

十一、**scroll 系列**

https://note.youdao.com/web/#/file/recent/note/WEB63ce04936ce0bd2fd74f177deb5f9dd9/?search=scrollTop

十二、vue 的生命周期

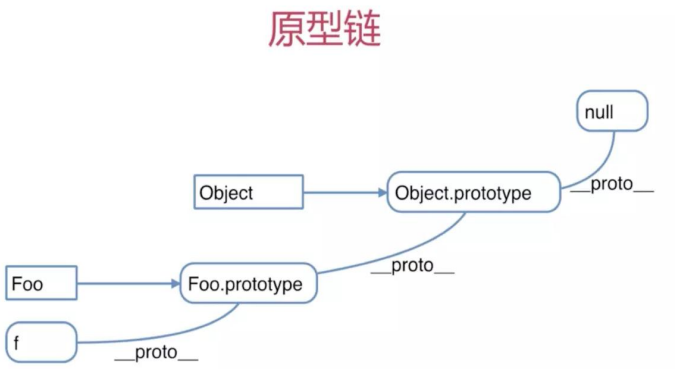
生命周期钩子	详细
beforeCreate	在实例初始化之后，数据观测(data observer) 和 event/watcher 事件配置之前被调用。
created	实例已经创建完成之后被调用。在这一步，实例已完成以下的配置：数据观测(data observer)，属性和方法的运算， watch/event 事件回调。然而，挂载阶段还没开始，$el 属性目前不可见。
beforeMount	在挂载开始之前被调用：相关的 render 函数首次被调用。
mounted	el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子。如果 root 实例挂载了一个文档内元素，当 mounted 被调用时 vm.$el 也在文档内。
beforeUpdate	数据更新时调用，发生在虚拟 DOM 重新渲染和打补丁之前。你可以在这个钩子中进一步地更改状态，这不会触发附加的重渲染过程。
updated	由于数据更改导致的虚拟 DOM 重新渲染和打补丁，在这之后会调用该钩子。当这个钩子被调用时，组件 DOM 已经更新，所以你现在可以执行依赖于 DOM 的操作。
activated	keep-alive 组件激活时调用。
deactivated	keep-alive 组件停用时调用。
beforeDestroy	实例销毁之前调用。在这一步，实例仍然完全可用。
destroyed	Vue 实例销毁后调用。调用后，Vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。

（除了beforeCreate和created钩子之外，其他钩子均在服务器端渲染期间不被调用。）

十三、什么是原型链？

1. 原型的五条规则

1、所有的引用类型都可以自定义添加属性
2、所有的引用类型都有自己的隐式原型（proto）
3、构造函数都有自己的显式原型（prototype）
4、所有的引用类型的隐式原型都指向对应构造函数的显示原型
5、使用引用类型的某个自定义属性时，如果没有这个属性，会去该引用类型的proto（也就是对应构造函数的 prototype）中去找



十四、类数组怎么转化为数组？
一、Array.prototype.slice.call(arrayLike)的结果是将 arrayLike 对象转换成一个 Array 对象。

slice 方法可以用来将一个类数组（Array-like）对象/集合转换成一个新数组。你只需将该方法绑定到这个对象上。 一个函数中的 arguments 就是一个类数组对象的例子。除了使用 ，你也可以简单的使用 来代替。
Array.prototype.slice.call(arguments) [].slice.call(arguments)
二、spread 扩展运算符
[...arrayLikeObject]
【注意】扩展运算符背后调用的是遍历器接口（ Symbol.iterator ），如果一个对象没有部署这个接口，就无法转换。
三、 es6 的 Array.from(arrayLike);

十五、数组快速去重复？

```
1、let arr = [1,2,3,3];
let resultarr = [...new Set(arr)];
console.log(resultarr); //[1,2,3]
2、
function dedupe(array){
 return Array.from(new Set(array));
}
dedupe([1,1,2,3]) //[1,2,3]
```

十六、typeof 会返回一个变量的基本类型，

只有以下几种：number,boolean,string,object,undefined,function；
typeof 对 4 种基本类型（number，boolean，undefined，string），function，object，很方便，但是其他类型就没办法了。
1.javascript 的 typeof 返回哪些数据类型
object number function boolean underfind.string

十六、跨域
ajax 通过服务端代理一样可以实现跨域，跨域资源共享
服务器只需要在响应头部中设置 Access-Control-Allow-Origin 即可让客户端访问。
十七、节流和防抖
节流：规定在一个单位时间内，只能触发一次函数，如果这个单位时间内触发多次函数，只有一次生效
原理
原理是通过判断是否到达规定时间来触发函数。
运用场景
鼠标不断点击触发，mousedown 事件的执行(单位时间内只触发一次)
监听滚动事件，比如是否滑到底部自动加载更多，用节流来判断

比如在页面的无限加载场景下，我们需要用户在滚动页面时，

每隔一段时间

一次 Ajax 请求，而不是在用户停下滚动页面操作时才去请求数据。这样的场景，

就适合用节流技术来实现。
防抖
当持续触发事件时，一定时间段内没有再次触发事件，事件处理函数才会执行一次，如果在设定的时间到来之前，又一次触发了事件，就重新开始延时。
想象你将一个弹簧按下，继续加压，继续按下，只会在你最后放手的时候反弹，即我们希望函数只会调用一次，即使在这之前反复调用它，最终也只会调用一次而已。
原理
其原理是维护一个计时器，规定在 delay 时间后触发函数，但是在 delay 时间内再次触发的话，就会取消之前的计时器而重新设置；这样一来，只有最后一次操作能被触发。
运用场景
input 输入框实现模糊匹配功能，用户在不断输入值时，用防抖来节约请求资源
window 触发 resize 的时候，不断的调整浏览器窗口大小会不断的触发这个事件，用防抖来让其只触发一次

防抖动和节流本质是不一样的。防抖动是将多次执行变为最后一次执行，节流是

将多次执行变成每隔一段时间执行。

二十：webpack
webpack 就是识别你的 入口文件。识别你的模块依赖，来打包你的代码。
webpack 做的就是分析代码。转换代码，编译代码，输出代码。
webpack 是一个模块打包机，将根据文件间的依赖关系对其进行静态分析，然后将这些模块按指定规则生成静态资源
当 webpack 处理程序时，它会递归地构建一个依赖关系图(dependency graph)，其中包含应用程序需要的每个模块，然后将所有这些模块打包成一个或多个 bundle
css-loader，加载.css
style-loader 通过 style 标签 注入到 dom 中
babel-loader
告诉 webpack 我想要对我的 js 代码进行兼容性编译

二十一、什么是虚拟 dom？
什么是虚拟 DOM
0.0962018.12.03 19:33:41 字数 714 阅读 1139
虚拟 dom (virtual DOM)
这篇文档用于书写我对虚拟 dom 的一些自己的见解。
什么是虚拟 dom
虚拟 dom 是相对于浏览器所渲染出来的真实 dom 的，在 react，vue 等技术出现之前，我们要改变页面展示的内容只能通过遍历查询 dom 树的方式找到需要修改的 dom 然后修改样式行为或者结构，来达到更新 ui 的目的。
这种方式相当消耗计算资源，因为每次查询 dom 几乎都需要遍历整颗 dom 树，如果建立一个与 dom 树对应的虚拟 dom 对象（ js 对象），以对象嵌套的方式来表示 dom 树，那么每次 dom 的更改就变成了 js 对象的属性的更改，这样一来就能查找 js 对象的属性变化要比查询 dom 树的性能开销小。

vue

vue 采用的是虚拟 dom 通过重写 setter ， getter 实现观察者监听 data 属性的变化生成新的虚拟 dom 通过 h 函数创建真实 dom 替换掉 dom 树上对应的旧 dom。

react

react 也是通过虚拟 dom 和 setState 更改 data 生成新的虚拟 dom 以及 diff 算法来计算和生成需要替换的 dom 做到局部更新的。

公司的主营业务方向

公司主营业务方向为液晶显示主控板卡、工业电源、交互智能平板、移动智能终端和医疗等产品的设计、研发和销售

产业布局：**人工智能 健康医疗 智能硬件 智慧校园**

原型链.png