**dns查询 --2--**

**DNS解析过程（简易）**

域名解析的过程实际是将域名还原为IP地址的过程。

1. 解析这个域名，查看本地hosts文件是否有这个网址映射关系，如果有就调用这个IP地址映射
2. 浏览器发送请求到本地域名服务器查询本地缓存，如果有就直接返回，该过程以递归方式查询。
3. 最后迭代查询，按根域服务器 ->顶级域,.com->第二层域，baidu.com ->子域，www.baidu.com的顺序找到IP地址。
4. 本地域名服务器返回结果给客户端，并保存到缓存

所有网址真正的解析过程为: . -> .com -> google.com. -> www.google.com.。最后面并不是多打了一个.，这个.对应的就是根域名服务器，默认情况下所有的网址的最后一位都是.

**数组去重 --2--**

**1、双重循环：**创建一个临时数组，每个元素都向后遍历一遍查找是否有相同值，没有时加入临时数组

**2、使用splice去重：**还是使用双重循环，但这次不用创建临时数组，而是直接用splice方法，在遇到相同值时直接删除掉该元素，虽说减少了内存占用，但时间复杂度较大。

**3、使用indexOf或includes的API：**与双重循环的方法类似，就是把第二重循环改成用indexOf判断临时数组中是否有该元素存在，或者用includes的API效果与类似

**4、排序后遍历：**先排序得到有序数组后，遍历数组对比当前元素与下一索引的元素值是否相同，不相同时加入临时数组

**5、Set去重：**创建Set将数组整个传入，再用Array.from方法传入Set将其转化成数组，或者用扩展运算符 [...Set]

**6、利用对象的属性或哈希表；**遍历数组，遇到的每个值直接查看对象中是否有该属性，有的话直接跳过，没有时给对象添加以该值为键名的属性，并将该值加入临时数组。对象的属性可能会有点小问题，最好还是用哈希表

前三种时间复杂度n^2(第二种可能为n^3)，第四种nlogn，五、六种为n

**两种盒子模型，应用场景 --2--**

CSS的盒子模型包括内容（content），还有margin（外边距）、border（边框）、padding（内边框）四个部分

两个盒子模型唯一的区别在于元素宽高度的定义

标准盒模型的width是content的宽度

IE的怪异盒模型的width是content+padding+border这三部分的宽度

如果想要切换盒模型也很简单，这里需要借助css3的box-sizing属性

·box-sizing: content-box 是W3C盒子模型

·box-sizing: border-box 是IE盒子模型

**box-sizing的默认属性是content-box**

**判断数组的方法 --2--**

1、使用instanceof运算符

let arr = [];

console.log(arr instanceof Array)

2、使用constructor方法

console.log([].constructor == Array); //true

3、使用Object.prototype.toString.call(arr) === '[object Array]'方法

4、ES5定义了Array.isArray:  
 Array.isArray([]) //true

**深浅拷贝的区别，如何解决循环引用 --2--**

浅拷贝和深拷贝的区别主要在于引用类型的值改变时的情况不同。由于引用类型的值在传递时传递的是内存地址，所以存在的问题就是浅拷贝就只是存储了引用值传递过来的内存地址，拷贝对象和原对象的变量指向的是同一地址，无论哪一方修改这个变量，都会导致另一方中变量值的改变。

而深拷贝遇到引用类型时是会创建一个新的对象，将引用类型中的属性一一复制到新对象上，如果遇到引用类型的属性则继续递归处理。最终得到的值是存储在一个新的内存中的，所以拷贝对象和原对象之间就不存在浅拷贝的关联关系了

**循环引用问题**

概念：对象A中包含指向对象B的指针，对象B中包含指向对象A的指针，会引发内存泄漏现象

父级引用：对象属性是对象本身，导致：子->父->子....循环，栈溢出  
 解决：while循环判断一个对象的字段是否引用了这个对象或这个对象的任意父级

同级引用：对象中的子对象引用了其他的子对象  
 解决：用 WeakMap() 记录下对象中的所有对象，并与新创建的对象一一对应，即记录引用关系

**var let const 的不同 - 概念的不同，变量提升，暂时性死区，顶层对象 --2--**

1. var在全局作用域声明的变量会挂载到window上，let和const不会

2、var声明变量存在变量提升，let和const不存在变量提升

3、let和const声明形成块作用域

4、同一作用域下let和const不能声明同名变量，而var可以

5、暂时性死区

**defineproperty和proxy有什么区别 --2--**

Proxy 的优势如下:

·Proxy 可以直接监听对象而非属性；

·Proxy 可以直接监听数组的变化；

·Proxy 有多达 13 种拦截方法,不限于 apply、ownKeys、deleteProperty、has 等等是 Object.defineProperty 不具备的；

·Proxy 返回的是一个新对象,我们可以只操作新的对象达到目的,而 Object.defineProperty 只能遍历对象属性直接修改；

·Proxy 作为新标准将受到浏览器厂商重点持续的性能优化，也就是传说中的新标准的性能红利；

Object.defineProperty 的优势如下:

·兼容性好，支持 IE9，而 Proxy 的存在浏览器兼容性问题,而且无法用 polyfill 磨平，因此 Vue 的作者才声明需要等到下个大版本( 3.0 )才能用 Proxy 重写。



**图片懒加载 --2--**

**懒加载也就是延迟加载。**  
 当访问一个页面的时候，先把img元素或是其他元素的背景图片路径替换成一张大小为1\*1px图片的路径（这样就只需请求一次，俗称占位图），只有当图片出现在浏览器的可视区域内时，才设置图片真正的路径，让图片显示出来。这就是图片懒加载。

**懒加载原理**

页面中的img元素，如果没有src属性，浏览器就不会发出请求去下载图片，只有通过javascript设置了图片路径，浏览器才会发送请求。  
 懒加载的原理就是先在页面中把所有的图片统一使用一张占位图进行占位，把真正的路径存在元素的“data-url”（这个名字起个自己认识好记的就行）属性里，要用的时候就取出来，再设置

**懒加载思路及实现**

1、首先，不要将图片地址放到src属性中，而是放到其它属性(data-original)中。  
 2、页面加载完成后，根据scrollTop判断图片是否在用户的视野内，如果在，则将data-original属性中的值取出存放到src属性中。  
 3、在滚动事件中重复判断图片是否进入视野，如果进入，则将data-original属性中的值取出存放到src属性中。

优点：页面加载速度快、可以减轻服务器的压力、节约了流量,用户体验好

**DOM0事件和DOM2事件的区别 --1--**

**0级DOM**

分为2个： 一是在标签内写onclick事件

二是在JS写onclick=function（）{}函数

**2级DOM**

只有一个：监听方法，有两个方法用来添加和移除事件处理程序：addEventListener()和removeEventListener()。

它们都有三个参数：第一个参数是事件名（如click）；

　　　　　　　　　第二个参数是事件处理程序函数；

　　　　　　　　   第三个参数如果是true则表示在捕获阶段调用，为false表示在冒泡阶 段调用。

1. **DOM0级和DOM2级的共同优点：**  
   能添加多个事件处理程序，按顺序执行，HTML事件处理程序无法做到
2. **关于dom0级和dom2级的区别**  
    DOM0级事件处理：

同时绑定几个不同的事件，例如在绑定onclick的基础上再绑定一个onmouseover为按钮2设置背景颜色（这里注意不能onclick、onmouseover事件都设为alert弹出哦，可能有冲突，dom0和dom2都不能成功）；但是不能同时绑定多个相同的事件，比如onclick；会覆盖，只会执行最后一个的函数；  
 DOM2级事件处理：

优点：同时绑定几个事件（相同或不同），然后顺序执行，不会覆盖。

缺点：不具有跨浏览器优势.  
  
另外，IE9能兼容dom2

**怎么判断一个对象，判断空对象？怎么遍历这个对象 --1--**

//法一：**将json对象转化为json字符串**，再判断该字符串是否为"{}"

let a = {};

let b={cc: 1}

console.log(JSON.stringify(a)==='{}')//true

console.log(JSON.stringify(b)==='{}')//false

//法二：**for in 循环判断**

let mm = function(obj) {

for(let key in obj) {

if(obj.hasOwnProperty(key)) { //只遍历对象自身的属性

return false;/\*能遍历，不为空\*/

}

}

return true;

}

console.log(mm(a));//true

console.log(mm(b));//false

//法三：用Object对象的**getOwnPropertyNames方法，获取到对象中的属性名**，存到一个数组中，返回数组对象，我们可以通过判断数组的length来判断此对象是否为空

let aa = Object.getOwnPropertyNames(a);

let bb = Object.getOwnPropertyNames(b);

console.log(aa.length == 0); //true

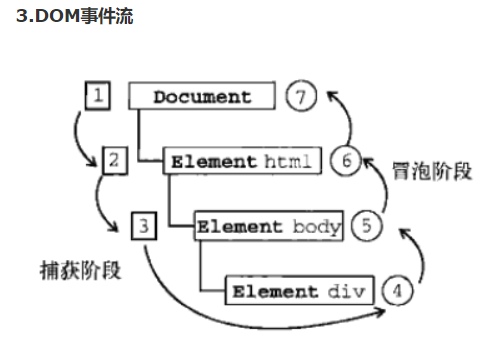
console.log(bb.length == 0); //false

//法四：**ES6的Object.keys()方法，返回值也是对象中属性名组成的数组**

let aaa = Object.keys(a);

console.log(aaa.length == 0); //true

**事件冒泡和事件捕获 --1--**



先从最外层元素（document）逐层进入事件触发的元素的过程是事件捕获

从事件触发的元素逐层向外直到最外层元素（document）的过程是事件冒泡

先进行事件捕获，再事件冒泡

**箭头函数和普通函数的区别 --1--**

·箭头函数是匿名函数，不能作为构造函数，不能使用new

·箭头函数不能绑定arguments，取而代之用rest参数...解决

·箭头函数没有原型属性

·箭头函数的this永远指向其上下文的this，没有办改变其指向，没有自己的this。  
 普通函数的this指向调用它的对象

·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

·箭头函数不能当做Generator函数,不能使用yield关键字

·箭头函数内的this指向上层对象，bind()、call()、apply()均无法改变指向。

**v-for中key的作用 --1--**

key值大多情况下使用在循环语句中，从本质来讲主要作用大概有以下两点：

1、主要用在 Vue 的虚拟 DOM 算法，在新旧 nodes 对比时辨识 VNodes，相当于唯一标识ID。

2、Vue 会尽可能高效地渲染元素，通常会复用已有元素而不是从头开始渲染， 因此使用key值可以提高渲染效率，同理，改变某一元素的key值会使该元素重新被渲染。

key的特殊attribute主要用在Vue的虚拟DOM算法，在新旧Nodes对比时辨识VNodes。如果不使用key，Vue会使用一种最大限度减少动态元素并且尽可能的尝试就地修改、复用相同类型元素的算法，而使用key时，它会基于key的变化重新排列元素顺序，并且会移除key不存在的元素。此外有相同父元素的子元素必须有独特的key，重复的key会造成渲染错误。

1. 两个相同的组件产生类似的DOM结构，不同的组件产生不同的DOM结构。

2. 同一层级的一组节点，他们可以通过唯一的id进行区分。

基于以上这两点假设，使得虚拟DOM的Diff算法的复杂度从O(n^3)降到了O(n)。

所以一句话，\*\*key的作用主要是为了高效的更新虚拟DOM\*\*。另外vue中在使用相同标签名元素的过渡切换时，也会使用到key属性，其目的也是为了让vue可以区分它们

**vue路由的实现原理 --1--**

vue-router通过hash与History interface两种方式实现前端路由，更新视图但不重新请求页面”是前端路由原理的核心之一，目前在浏览器环境中这一功能的实现主要有两种方式：

1.Hash --- 利用 URL 中的hash("#");

2.利用 History interface 在HTML5中新增的方法。

Vue 中，它是通过 mode 这一参数控制路由的实现模式：

const router=new VueRouter({

mode:'history',

routes:[...]

})

**mode 参数：**

1.默认 hash

2. history。如果浏览器不支持 history 新特性，则采用 hash

3. 如果不在浏览器环境下，就采用 abstract（Node环境下）

**Hash模式**

hash即浏览器url中#后面的内容，包含#。hash是URL中的锚点，代表的是网页中的一个位置，单单改变#后的部分，浏览器只会加载相应位置的内容，不会重新加载页面。

也就是说

·即#是用来指导浏览器动作的，对服务器端完全无用，HTTP请求中，不包含#。

·每一次改变#后的部分，都会在浏览器的访问历史中增加一个记录，使用”后退”按钮，就可以回到上一个位置。

所以说Hash模式通过锚点值的改变，根据不同的值，渲染指定DOM位置的不同数据。

**HTML5History**

push()：将新路由添加到浏览器访问历史的栈顶

replace()：与push()方法不同它并不是将新路由添加到浏览器访问历史的栈顶，而是替换掉当前的路由

history.back()：与浏览器点击后退按钮相同

history.forward()：与浏览器点击前进按钮相同

history.go(n)：接受一个整数作为参数，移动到该整数指定的页面，比如go(1)相当于forward()，go(-1)相当于back()，go(0)相当于刷新当前页面。如果移动的位置超出了访问历史的边界，以上三个方法并不报错，而是静默失败

**两种模式比较：**

·pushState设置的新URL可以是与当前URL同源的任意URL；而hash只可修改#后面的部分，故只可设置与当前同文档的URL

·pushState通过stateObject可以添加任意类型的数据到记录中；而hash只可添加短字符串

·pushState可额外设置title属性供后续使用

·history模式则会将URL修改得就和正常请求后端的URL一样,如后端没有配置对应/user/id的路由处理，则会返回404错误

**hash模式：**

      a: 点击或浏览器历史跳转时，触发onhashchange事件,然后根据路由规则匹配显示相应页面(遍历路由表，装载相应组件到router-link)；

      b: 手动刷新时,不会像服务器发送请求（不会触发onhashchange），触发onload事件，然后根据路由规则匹配显示相应页面；

**history模式：**

     a:跳转时会调用history.pushState方法,根据to属性改变地址，并切换相应组件到router-link;

     b:浏览器历史操作（前进，后退）,只会改变地址栏（页面内容不会变）,不会切换组件，需要使用popstate方法来切换组件；

     c: 手动刷新,需要后端配合重定向，不然404

**v-if和v-show的区别 --1--**

**1、手段：**

v-if 是动态的向DOM树内添加或者删除DOM元素；  
 v-show 是通过设置DOM元素的display样式属性控制显隐；

**2、编译过程：**

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

**3、编译条件：**

v-if 是惰性的，如果初始条件为假，则什么也不做；只有在条件第一次变为真时才开始局部编译（编译被缓存？编译被缓存后，然后再切换的时候进行局部卸载);  
 v-show 是在任何条件下（首次条件是否为真）都被编译，然后被缓存，而且DOM元素保留；

**4、性能消耗：**

v-if 有更高的切换消耗；  
 v-show 有更高的初始渲染消耗；

**5、使用场景：**

v-if 适合运营条件不大可能改变；  
 v-show 适合频繁切换。

**vue中$nextTick的作用 --1--**

·首先Vue中有一个重要的概念：异步更新队列

·Vue异步更新DOM的原理：Vue在观察到数据变化时，并不是直接更新DOM，而是开启一个队列，并且缓存同一轮事件循环中的所有数据改变。在缓冲时会除去重复的操作,等到下一轮事件循环时，才开始更新。

·异步更新队列实现的选择 ： Vue会根据当前浏览器环境优先使用原生的Promise.then和MutationObserver（现已经换为MessageChannel ,是宏任务），如果都不支持，就会采用setTimeout代替。

·最后：$nextTick就是用来告知DOM什么时候更新完，当DOM更新完毕后，nextTick方法里面的回调就会执行。

·在Vue生命周期的created()钩子函数进行的DOM操作一定要放在Vue.nextTick()的回调函数中

·在created()钩子函数执行的时候DOM 其实并未进行任何渲染，而此时进行DOM操作无异于徒劳，所以此处一定要将DOM操作的js代码放进Vue.nextTick()的回调函数中。与之对应的就是mounted()钩子函数，因为该钩子函数执行时所有的DOM挂载和渲染都已完成，此时在该钩子函数中进行任何DOM操作都不会有问题 。

·在数据变化后要执行的某个操作，而这个操作需要使用随数据改变而改变的DOM结构的时候，这个操作都应该放进Vue.nextTick()的回调函数中。

**应用场景：**mounted 获取ajax数据，后根据页面渲染数据的样式（文字占宽）来修改布局。几乎所有更新数据后操作dom的操作，都需要用到异步更新队列

**commonJS和es6模块化的区别，es6的模块化和common相互引入的优先级等 --1--**

**CommonJS**

1、对于基本数据类型，属于复制。即会被模块缓存。同时，在另一个模块可以对该模块输出的变量重新赋值。

2、对于复杂数据类型，属于浅拷贝。由于两个模块引用的对象指向同一个内存空间，因此对该模块的值做修改时会影响另一个模块。

3、当使用require命令加载某个模块时，就会运行整个模块的代码。

4、当使用require命令加载同一个模块时，不会再执行该模块，而是取到缓存之中的值。也就是说，CommonJS模块无论加载多少次，都只会在第一次加载时运行一次，以后再加载，就返回第一次运行的结果，除非手动清除系统缓存。

5、 循环加载时，属于加载时执行。即脚本代码在require的时候，就会全部执行。一旦出现某个模块被"循环加载"，就只输出已经执行的部分，还未执行的部分不会输出。

**ES6模块**

1、ES6模块中的值属于【动态只读引用】。

2、对于只读来说，即不允许修改引入变量的值，import的变量是只读的，不论是基本数据类型还是复杂数据类型。当模块遇到import命令时，就会生成一个只读引用。等到脚本真正执行时，再根据这个只读引用，到被加载的那个模块里面去取值。

3、对于动态来说，原始值发生变化，import加载的值也会发生变化。不论是基本数据类型还是复杂数据类型。

4、循环加载时，ES6模块是动态引用。只要两个模块之间存在某个引用，代码就能够执行。



**unicode和utf-8编码 --1--**

简单来说：

Unicode 是「字符集」

UTF-8 是「编码规则」

其中：

字符集：为每一个「字符」分配一个唯一的 ID（学名为码位 / 码点 / Code Point）

编码规则：将「码位」转换为字节序列的规则（编码/解码 可以理解为 加密/解密 的过程）

Unicode 是全球文字统一编码。它把世界上的各种文字的每一个字符指定唯一编码，实现跨语种、跨平台的应用。

Unicode 只是一个符号集，它只规定了每个符号的二进制数，却没有规定这个二进制数应该如何存储。比如，汉字‘严’的 Unicode 是十六进制数4E25，转换成二进制数足足有15位（100111000100101），也就是说，这个符号的表示至少需要2个字节。表示其他更大的符号，可能需要3个字节或者4个字节，甚至更多。

可以想象，如果有一种编码，将世界上所有的符号都纳入其中。每一个符号都给予一个独一无二的编码，那么乱码问题就会消失。这就是 Unicode，就像它的名字都表示的，这是一种所有符号的编码。

互联网的普及，强烈要求出现一种统一的编码方式。UTF-8 就是在互联网上使用最广的一种 Unicode 的实现方式。

UTF-8 最大的一个特点，就是它是一种变长的编码方式。它可以使用1~4个字节表示一个符号，根据不同的符号而变化字节长度。

**什么是伪类？什么是伪元素？ --1--**

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/伪对象**：不存在在DOM文档中，是虚拟的元素，是创建新元素。代表某个元素的子元素，这个子元素虽然在逻辑上存在，但却并不实际存在于文档树中。

**伪类**：用于向某些选择器添加特殊的效果  
 **伪元素**：用于将特殊的效果添加到某些选择器  
 其实根本意思就是就是对那些不能通过class、id等选择元素的补充

在使用伪类和伪元素的时候，有一点要特别注意，伪类就像真正的类一样，可以叠加使用，没有数量上限，只要不是互斥的

伪元素就要严格的多，****伪元素在一个选择器中只能出现一次，并且只能出现在末尾****



·伪类的效果可以通过添加实际的类来实现  
 ·伪元素的效果可以通过添加实际的元素来实现  
 所以它们的本质区别就是是否抽象创造了新元素

**注意：**  
 伪类只能使用“：”，而伪元素既可以使用“:”，也可以使用“::”  
 因为伪类是类似于添加类所以可以是多个，而伪元素在一个选择器中只能出现一次，并且只能出现在末尾

**注意：**

·伪对象要配合content属性一起使用  
 ·伪对象不会出现在DOM中，所以不能通过js来操作，仅仅是在 CSS 渲染层加入  
 ·伪对象的特效通常要使用:hover伪类样式来激活

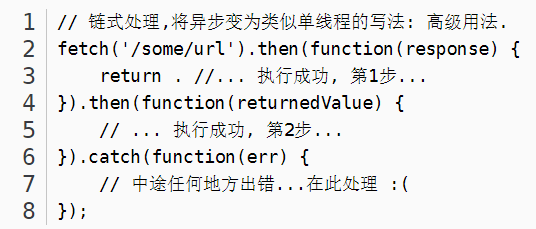
总之，伪元素的操作对象是新生成的dom元素，而不是原来dom结构里就存在的；而伪类恰好相反，伪类的操作对象是原来的dom结构里就存在的元素。

伪元素与伪类的根本区别在于：操作的对象元素是否存在于原来的dom结构里。

**ajax和fetch区别 --1--**

Ajax的本质是使用XMLHttpRequest对象来请求数据

fetch 是全局量 window 的一个方法，它的主要特点有：  
 1、第一个参数是URL:  
 2、第二个是可选参数，可以控制不同配置的 init 对象  
 3、使用了 JavaScript Promises 来处理结果/回调:



fetch号称是AJAX的替代品，是在ES6出现的，使用了ES6中的promise对象。Fetch是基于promise设计的。Fetch的代码结构比起ajax简单多了，参数有点像jQuery ajax。但是，一定记住fetch不是ajax的进一步封装，而是原生js，没有使用XMLHttpRequest对象。

**fetch规范与jQuery.ajax()主要有两种方式的不同，牢记：**

1、从 fetch()返回的 Promise 将不会拒绝HTTP错误状态, 即使响应是一个 HTTP 404 或 500。相反，它会正常解决 (其中ok状态设置为false), 并且仅在网络故障时或任何阻止请求完成时，它才会拒绝。

2、默认情况下, fetch在服务端不会发送或接收任何 cookies, 如果站点依赖于维护一个用户会话，则导致未经认证的请求(要发送 cookies，必须发送凭据头).

**渲染页面的过程 --1--**

1、解析HTML，生成DOM树，解析CSS，生成CSSOM树

2、将DOM树和CSSOM树结合，生成渲染树(Render Tree)

3、Layout(回流):根据生成的渲染树，进行回流(Layout)，得到节点的几何信息（位置，大小）

4、Painting(重绘):根据渲染树以及回流得到的几何信息，得到节点的绝对像素

5、Display:将像素发送给GPU，展示在页面上。（这一步其实还有很多内容，比如会在GPU将多个合成层合并为同一个层，并展示在页面中。而css3硬件加速的原理则是新建合成层）

**computed、watch区别**

**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。

.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

**计算属性computed :**

1. 支持缓存，只有依赖数据发生改变，才会重新进行计算

2. 不支持异步，当computed内有异步操作时无效，无法监听数据的变化

3.computed 属性值会默认走缓存，计算属性是基于它们的响应式依赖进行缓存的，也就是基于data中声明过或者父组件传递的props中的数据通过计算得到的值

4. 如果一个属性是由其他属性计算而来的，这个属性依赖其他属性，是一个多对一或者一对一，一般用computed

5.如果computed属性属性值是函数，那么默认会走get方法；函数的返回值就是属性的属性值；在computed中的，属性都有一个get和一个set方法，当数据变化时，调用set方法。

**侦听属性watch：**

1. 不支持缓存，数据变，直接会触发相应的操作；

2.watch支持异步；

3.监听的函数接收两个参数，第一个参数是最新的值；第二个参数是输入之前的值；

4. 当一个属性发生变化时，需要执行对应的操作；一对多；

5. 监听数据必须是data中声明过或者父组件传递过来的props中的数据，当数据变化时，触发其他操作，函数有两个参数，

　　immediate：组件加载立即触发回调函数执行，

deep: 深度监听，为了发现对象内部值的变化，复杂类型的数据时使用，例如数组中的 对象内容的改变，注意监听数组的变动不需要这么做。注意：deep无法监听到数组的变动和对象的新增，参考vue数组变异,只有以响应式的方式触发才会被监听到。

**浏览器的存储方式有哪些 --1--**

**轮播图的实现 --1--**

**BOM和DOM的联系和区别是啥 --1--**

**Vue和React的区别 --1--**

**vue的缺陷和解决方法**

**null和undefined的不同，为什么 == 结果是true，而 === 结果是false**

**BOM：window location navigator - 解析url会用到location对象**

**写一个通用组件需要考虑哪些问题**

**H5 CSS3特性**

**手撕每隔一秒输出数组中的一项，多种方式 setInterval，setTimeout递归，遍历setTimeout 时间为index\*1000,async/await和Promise实现   await返回的是什么？Promise**

**禁止js读取cookie**

**全等，相等判断。几个小题目**

**CSS布局方式**

**script是否会阻塞其他的运行，解决方法**

JavaScript 在浏览器中的性能，可以认为是开发者所面临的最严重的可用性问题。这个问题因为 JavaScript 的阻塞特性而变得复杂，也就是说浏览器在执行 JavaScript 时不能同时做任何事情。多数浏览器使用单一进程来处理用户界面（UI）刷新和 JavaScript ，所以只能同一时刻做一件事。  
 简单的说，就是 <scritp>标签每次出现，都会让页面等待脚本的解析和执行，无论当前的 JavaScript 代码是内嵌的还是包含在外链文件中，页面的下载和渲染都必须停下来等待脚本执行完成。  
 同样的情况也发生在使用 src 属性加载 JavaScript 的过程中，浏览器必须先话时间下载外链文件中的代码，然后解析并执行，在这个过程中，页面渲染和用户交互式完全被阻塞的。

所以，优化 JavaScript 的首要规则：将脚本放在底部！

还可以使用defer和async属性异步加载脚本

或者用动态创建script标签并在需要时引入的方法也能在一定程度上解决这个 问题

**对语义化标签的理解 - 让人更可读，让机器更可读（利于SEO）**

语义化的标签，旨在让标签有自己的含义。

**HTML语义化的原因（优点）：**

1、即使在没有CSS样式的条件下，也能很好地呈现出内容结构、代码结构；（HTML结构清晰）

2、语义化HTML会使HTML结构变的清晰，有利于维护代码和添加样式；（代码可读性较好）

3、方便其他设备解析（如屏幕阅读器、盲人阅读器、移动设备）以意义的方式来渲染网页；（无障碍阅读 ）

4、提升搜索引擎优化(SEO)的效果。和搜索引擎建立良好沟通，有助于爬虫抓取更多的有效信息：爬虫依赖于标签来确定上下文和各个关键字的权重；  
 SEO，中文称搜索引擎优化，一种利用搜索引擎的搜索规则来提高目前网站在有关搜索引擎内的自然排名的方式。

5、便于团队开发和维护，语义化更具可读性，是下一步网页的重要动向，遵循W3C标准的团队都遵循这个标准，可以减少差异化。

6、通常语义化HTML会使代码变的更少，使页面加载更快。

**在写HTML代码时应该注意：**

·尽可能少的使用无语义的标签div和span；

·在语义不明显时，既可以使用div或者p时，尽量用p, 因为p在默认情况下有上下间距，对兼容特殊终端有利；

·不要使用纯样式标签，如：b、font、u等，改用css设置。

·需要强调的文本，可以包含在strong或者em标签中（浏览器预设样式，能用CSS指定就不用他们），strong默认样式是加粗（不要用b），em是斜体（不用i）；

·使用表格时，标题要用caption，表头用thead，主体部分用tbody包围，尾部用tfoot包围。表头和一般单元格要区分开，表头用th，单元格用td；

·表单域要用fieldset标签包起来，并用legend标签说明表单的用途；

·每个input标签对应的说明文本都需要使用label标签，并且通过为input设置id属性，在lable标签中设置for=someld来让说明文本和相对应的input关联起来。

**Vue数据响应式怎么实现的？ 发布订阅模式**

**V-model实现原理**

v-model 本质上不过是语法糖，可以用 v-model 指令在表单 <input>、<textarea> 及 <select> 元素上创建双向数据绑定。它会根据控件类型自动选取正确的方法来更新元素。它负责监听用户的输入事件以更新数据，并对一些极端场景进行一些特殊处理。v-model 会忽略所有表单元素的 value、checked、

**v-model 在内部为不同的输入元素使用不同的属性并抛出不同的事件：**

·text 和 textarea 元素使用 value 属性和 input 事件；

·checkbox 和 radio 使用 checked 属性和 change 事件；

·select 字段将 value 作为 prop 并将 change 作为事件。

**实现原理：**

·v-model只不过是一个语法糖而已,真正的实现靠的还是

1、v-bind:绑定响应式数据

2、触发oninput 事件并传递数据

**vue的响应式原理**

**什么是响应式布局**

响应式布局就是一个网站能够兼容多个终端——而不是为每个终端做一个特定的版本。这个概念是为解决移动互联网浏览而诞生的。  
 响应式布局可以为不同终端的用户提供更加舒适的界面和更好的用户体验。

响应式布局的优点和缺点有哪些呢？

优点 ：1. 面对不同分辨率设备灵活性强  
       2. 能够快捷解决多设备显示适应问题  
缺点 ：1.不能完全兼容所有浏览器，代码累赘，会出现隐藏无用的元素,加载时间加长  
       2.一定程度上改变了网站原有的布局结构，会出现用户混淆的情况。

实现原理？

原理：简单点说响应式布局它是通过CSS中Media Query（媒介查询）@media功能，来判断我们的终端设备宽度在多少像素内，然后就执行与之对应的CSS样式。

**响应式布局一定要注意以下两点：**

    一 是图片，在移动设备上，要做一些特定适合的小图片来匹配，如果单纯使用压缩的图片会失真，影响用户体验；  
    二 是在头部加入如下代码，目的是声明在移动设备上设置原始大小显示和是否缩放



**怎么实现边框宽度是0.5px**

**[] === []说输出**

**src和href区别**

**JS中严格模式是什么，有什么作用；**

**MVVM和MVC的区别**

**css的动画怎么做**

**有一个有一百万个url的数组，如何从这一百万个url里获得资源**

**对元素的margin和padding设置百分比的时候，他的百分比是相对于父元素的宽度width计算的，margin上下左右都是如此**

**HTTP报文组成部分**

**常用的 git 命令**

**webpack，常用loader、plugin，如何配置**