1. **说一说cookie sessionStorage localStorage 区别？**

Cookie、SessionStorage、 LocalStorage都是浏览器的本地存储。(数据存储位置) 它们的共同点：都是存储在浏览器本地的。它们的区别：cookie是由服务器端写入的，而SessionStorage、 LocalStorage都是由前端写入的，cookie的生命周期是由服务器端在写入的时候就设置好的，LocalStorage是写入就一直存在，除非手动清除，SessionStorage是页面关闭的时候就会自动清除。cookie的存储空间比较小大概4KB，SessionStorage、 LocalStorage存储空间比较大，大概5M。Cookie、SessionStorage、 LocalStorage数据共享都遵循同源原则，SessionStorage还限制必须是同一个页面。在前端给后端发送请求的时候会自动携带Cookie中的数据，但是SessionStorage、 LocalStorage不会 加分回答 由于它们的以上区别，所以它们的应用场景也不同，Cookie一般用于存储登录验证信息SessionID或者token，在有效的过期时间内一直保存。LocalStorage常用于存储不易变动的数据，减轻服务器的压力，关闭浏览器存储的信息不会丢失除非手动清除。SessionStorage可以用来检测用户是否是刷新进入页面，如音乐播放器恢复播放进度条的功能，关闭浏览器则信息自动删除。

1. **说一说你对闭包的理解？**闭包 一个函数和词法环境的引用捆绑在一起，这样的组合就是闭包（closure）。一般就是一个函数A，return其内部的函数B，被return出去的B函数能够在外部访问A函数内部的变量，这时候就形成了一个B函数的变量背包，A函数执行结束后这个变量背包也不会被销毁，并且这个变量背包在A函数外部只能通过B函数访问。 闭包形成的原理：作用域链，当前作用域可以访问上级作用域中的变量 闭包解决的问题：能够让函数作用域中的变量在函数执行结束之后不被销毁，同时也能在函数外部可以访问函数内部的局部变量。 闭包带来的问题：由于垃圾回收器不会将闭包中变量销毁，于是就造成了内存泄露，内存泄露积累多了就容易导致内存溢出。 加分回答 闭包的应用，能够模仿块级作用域，能够实现柯里化，在构造函数中定义特权方法、Vue中数据响应式Observer中使用闭包
2. **说一说promise是什么与使用方法？**Promise的作用：Promise是异步微任务，解决了异步多层嵌套回调的问题，让代码的可读性更高，更容易维护 Promise使用：Promise是ES6提供的一个构造函数，可以使用Promise构造函数new一个实例，Promise构造函数接收一个函数作为参数，这个函数有两个参数，分别是两个函数 `resolve`和`reject`，`resolve`将Promise的状态由等待变为成功，将异步操作的结果作为参数传递过去；`reject`则将状态由等待转变为失败，在异步操作失败时调用，将异步操作报出的错误作为参数传递过去。实例创建完成后，可以使用`then`方法分别指定成功或失败的回调函数，也可以使用catch捕获失败，then和catch最终返回的也是一个Promise，所以可以链式调用。 Promise的特点： 1. 对象的状态不受外界影响（Promise对象代表一个异步操作，有三种状态）。 - pending（执行中） - Resolved（成功，又称Fulfilled） - rejected（拒绝） 其中pending为初始状态，fulfilled和rejected为结束状态（结束状态表示promise的生命周期已结束）。 2. 一旦状态改变，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（状态凝固了，就不会再变了，会一直保持这个结果）： - 从Pending变为Resolved - 从Pending变为Rejected 3. resolve 方法的参数是then中回调函数的参数，reject 方法中的参数是catch中的参数 4. then 方法和 catch方法 只要不报错，返回的都是一个fullfilled状态的promise 加分回答 Promise的其他方法： Promise.resolve() :返回的Promise对象状态为fulfilled，并且将该value传递给对应的then方法。 Promise.reject()：返回一个状态为失败的Promise对象，并将给定的失败信息传递给对应的处理方法。 Promise.all()：返回一个新的promise对象，该promise对象在参数对象里所有的promise对象都成功的时候才会触发成功，一旦有任何一个iterable里面的promise对象失败则立即触发该promise对象的失败。 Promise.any()：接收一个Promise对象的集合，当其中的一个 promise 成功，就返回那个成功的promise的值。 Promise.race()：当参数里的任意一个子promise被成功或失败后，父promise马上也会用子promise的成功返回值或失败详情作为参数调用父promise绑定的相应句柄，并返回该promise对象。
3. **说一说跨域是什么？如何解决跨域问题？**跨域：当前页面中的某个接口请求的地址和当前页面的地址如果协议、域名、端口其中有一项不同，就说该接口跨域了。跨域限制的原因：浏览器为了保证网页的安全，出的同源协议策略。跨域就是要通过各种方法，避开浏览器的安全限制。跨域办法：  
   cors：目前最常用的一种解决办法，通过设置后端允许跨域实现。  
   res.setHeader('Access-Control-Allow-Origin', '\*');  
   res.setHeader("Access-Control-Allow-Methods", "GET, PUT, OPTIONS, POST");  
   node中间件、nginx反向代理：跨域限制的时候浏览器不能跨域访问服务器，node中间件和nginx反向代理，都是让请求发给代理服务器，静态页面面和代理服务器是同源的，然后代理服务器再向后端服务器发请求，服务器和服务器之间不存在同源限制。  
   JSONP：利用的原理是script标签可以跨域请求资源，将回调函数作为参数拼接在url中。后端收到请求，调用该回调函数，并将数据作为参数返回去，注意设置响应头返回文档类型，应该设置成javascript。  
   postmessage：H5新增API，通过发送和接收API实现跨域通信。  
   加分回答 跨域场景：前后端分离式开发、调用第三方接口
4. **说一说BFC**

BFC(Block Formatting Context)块级格式化上下文，是Web页面一块独立的渲染区域，内部元素的渲染不会影响边界以外的元素。 BFC布局规则 内部盒子会在垂直方向，一个接一个地放置。 -Box垂直方向的距离由margin决定。属于同一个BFC的两个相邻Box的margin会发生重叠。 -每个盒子（块盒与行盒）的margin box的左边，与包含块border box的左边相接触(对于从左往右的格式化，否则相反)。即使存在浮动也是如此。 -BFC的区域不会与float box重叠。 -BFC就是页面上的一个隔离的独立容器，容器里面的子元素不会影响到外面的元素。反之也如此。 -计算BFC的高度时，浮动元素也参与计算。 BFC形成的条件 -`float `设置成 `left `或 `right` ,不为none ;绝对元素定位`position `是`absolute`或者`fixed` -`overflow `不是`visible`，为 `auto`、`scroll`、`hidden` -行内块和弹性盒`display`是`flex`或者`inline-block` 等 。

BFC能解决的问题：清除浮动,BFC的方式都能清除浮动，但是常使用的清除浮动的BFC方式只有`overflow:hidden`,原因是使用float或者position方式清除浮动，虽然父级盒子内部浮动被清除了，但是父级本身又脱离文档流了，会对父级后面的兄弟盒子的布局造成影响。如果设置父级为`display:flex`，内部的浮动就会失效。所以通常只是用`overflow: hidden`清除浮动。

IFC（Inline formatting contexts）：内联格式上下文。IFC的高度由其包含行内元素中最高的实际高度计算而来（不受到竖直方向的padding/margin影响)，IFC中的line box一般左右都贴紧整个IFC，但是会因为float元素而扰乱。 GFC（GrideLayout formatting contexts）：网格布局格式化上下文。当为一个元素设置display值为grid的时候，此元素将会获得一个独立的渲染区域。 FFC（Flex formatting contexts）：自适应格式上下文。display值为flex或者inline-flex的元素将会生成自适应容器。

1. **说一说Vuex是什么，每个属性是干嘛的，如何使用？**Vuex是集中管理项目公共数据的。Vuex 有state、mutations 、getters、actions、module属性。 state 属性用来存储公共管理的数据。 mutations 属性定义改变state中数据的方法， 注意：不要在mutation中的方法中写异步方法ajax，那样数据就不可跟踪了 。 getters 属性可以认为是定义 store 的计算属性。就像计算属性一样，getter 的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被重新计算。 action属性类似于 mutation，不同在于：Action 提交的是 mutation，而不是直接变更状态。Action 可以包含任意异步操作。 moudle属性是将store分割成模块。每个模块拥有自己的 state、mutation、action、getter、甚至是嵌套子模块，从上至下进行同样方式的分割 使用方法： state ：直接以对象方式添加属性 mutations ：通过`store.commit`调用 action：通过 `store.dispatch` 方法触发 getters：直接通过store.getters.调用 加分回答 可以使用mapState、mapMutations、mapAction、mapGetters一次性获取每个属性下对应的多个方法。 VueX在大型项目中比较常用，非关系组件传递数据比较方便。
2. **JavaScript有几种方法判断变量的类型？**  
    JavaScript有4种方法判断变量的类型，分别是typeof、instanceof、Object.prototype.toString.call()（对象原型链判断方法）、 constructor (用于引用数据类型) typeof：常用于判断基本数据类型，对于引用数据类型除了function返回’function‘，其余全部返回’object'。 instanceof：主要用于区分引用数据类型，检测方法是检测的类型在当前实例的原型链上，用其检测出来的结果都是true，不太适合用于简单数据类型的检测，检测过程繁琐且对于简单数据类型中的undefined, null, symbol检测不出来。 constructor：用于检测引用数据类型，检测方法是获取实例的构造函数判断和某个类是否相同，如果相同就说明该数据是符合那个数据类型的，这种方法不会把原型链上的其他类也加入进来，避免了原型链的干扰。 Object.prototype.toString.call()：适用于所有类型的判断检测，检测方法是Object.prototype.toString.call(数据) 返回的是该数据类型的字符串。 这四种判断数据类型的方法中，各种数据类型都能检测且检测精准的就是Object.prototype.toString.call()这种方法。 加分回答 instanceof的实现原理：验证当前类的原型prototype是否会出现在实例的原型链\_\_proto\_\_上，只要在它的原型链上，则结果都为true。因此，`instanceof` 在查找的过程中会遍历左边变量的原型链，直到找到右边变量的 `prototype`，找到返回true，未找到返回false。 Object.prototype.toString.call()原理：Object.prototype.toString 表示一个返回对象类型的字符串，call()方法可以改变this的指向，那么把Object.prototype.toString()方法指向不同的数据类型上面，返回不同的结果。  
   ⭐⭐new⭐⭐
3. **说一说样式优先级的规则是什么**  
    CSS样式的优先级应该分成四大类 -第一类`!important`，无论引入方式是什么，选择器是什么，它的优先级都是最高的。 -第二类引入方式，行内样式的优先级要高于嵌入和外链，嵌入和外链如果使用的选择器相同就看他们在页面中插入的顺序，在后面插入的会覆盖前面的。 -第三类选择器，选择器优先级：id选择器>（类选择器 | 伪类选择器 | 属性选择器 ）> （后代选择器 | 伪元素选择器 ）> （子选择器 | 相邻选择器） > 通配符选择器 。 -第四类继承样式，是所有样式中优先级比较低的。 -第五类浏览器默认样式优先级最低。
4. **说一说js实现异步的方法**  
   所有异步任务都是在同步任务执行结束之后，从任务队列中依次取出执行。 回调函数是异步操作最基本的方法，比如AJAX回调，回调函数的优点是简单、容易理解和实现，缺点是不利于代码的阅读和维护，各个部分之间高度耦合，使得程序结构混乱、流程难以追踪（尤其是多个回调函数嵌套的情况），而且每个任务只能指定一个回调函数。此外它不能使用 try catch 捕获错误，不能直接 return Promise包装了一个异步调用并生成一个Promise实例，当异步调用返回的时候根据调用的结果分别调用实例化时传入的resolve 和 reject方法，then接收到对应的数据，做出相应的处理。Promise不仅能够捕获错误，而且也很好地解决了回调地狱的问题，缺点是无法取消 Promise，错误需要通过回调函数捕获。 Generator 函数是 ES6 提供的一种异步编程解决方案，Generator 函数是一个状态机，封装了多个内部状态，可暂停函数, yield可暂停，next方法可启动，每次返回的是yield后的表达式结果。优点是异步语义清晰，缺点是手动迭代`Generator` 函数很麻烦，实现逻辑有点绕 async/awt是基于Promise实现的，async/awt使得异步代码看起来像同步代码，所以优点是，使用方法清晰明了，缺点是awt 将异步代码改造成了同步代码，如果多个异步代码没有依赖性却使用了 awt 会导致性能上的降低，代码没有依赖性的话，完全可以使用 Promise.all 的方式。 加分回答 JS 异步编程进化史：callback -> promise -> generator/yield -> async/awt。 async/awt函数对 Generator 函数的改进，体现在以下三点： - 内置执行器。 Generator 函数的执行必须靠执行器，而 async 函数自带执行器。也就是说，async 函数的执行，与普通函数一模一样，只要一行。 - 更广的适用性。 yield 命令后面只能是 Thunk 函数或 Promise 对象，而 async 函数的 awt 命令后面，可以跟 Promise 对象和原始类型的值（数值、字符串和布尔值，但这时等同于同步操作）。 - 更好的语义。 async 和 awt，比起星号和 yield，语义更清楚了。async 表示函数里有异步操作，awt 表示紧跟在后面的表达式需要等待结果。 目前使用很广泛的就是promise和async/awt
5. **说一说数组去重都有哪些方法？**  
   第一种方法：利用对象属性key排除重复项：遍历数组，每次判断对象中是否存在该属性，不存在就存储在新数组中，并且把数组元素作为key，设置一个值，存储在对象中，最后返回新数组。这个方法的优点是效率较高，缺点是占用了较多空间，使用的额外空间有一个查询对象和一个新的数组 第二种方法：利用Set类型数据无重复项：new 一个 Set，参数为需要去重的数组，Set 会自动删除重复的元素，再将 Set 转为数组返回。这个方法的优点是效率更高，代码简单，思路清晰，缺点是可能会有兼容性问题 第三种方法：filter+indexof 去重：这个方法和第一种方法类似，利用 Array 自带的 filter 方法，返回 arr.indexOf(num) 等于 index 的num。原理就是 indexOf 会返回最先找到的数字的索引，假设数组是 [1, 1]，在对第二个1使用 indexOf 方法时，返回的是第一个1的索引0。这个方法的优点是可以在去重的时候插入对元素的操作，可拓展性强。 第四种方法：这个方法比较巧妙，从头遍历数组，如果元素在前面出现过，则将当前元素挪到最后面，继续遍历，直到遍历完所有元素，之后将那些被挪到后面的元素抛弃。这个方法因为是直接操作数组，占用内存较少。 第五种方法：reduce +includes去重：这个方法就是利用reduce遍历和传入一个空数组作为去重后的新数组，然后内部判断新数组中是否存在当前遍历的元素，不存在就插入到新数组中。这种方法时间消耗多，内存空间也有额外占用。 方法还有很多，常用的、了解的这些就可以 加分回答 以上五个方法中，在数据低于10000条的时候没有明显的差别，高于10000条，第一种和第二种的时间消耗最少，后面三种时间消耗依次增加，由于第一种内存空间消耗比较多，且现在很多项目不再考虑低版本浏览器的兼容性问题，所以建议使用第二种去重方法，简洁方便。
6. **说一说null 和 undefined 的区别，如何让一个属性变为null**  
   undefind 是全局对象的一个属性，当一个变量没有被赋值或者一个函数没有返回值或者某个对象不存在某个属性却去访问或者函数定义了形参但没有传递实参，这时候都是undefined。undefined通过typeof判断类型是'undefined'。undefined == undefined undefined === undefined 。 null代表对象的值未设置，相当于一个对象没有设置指针地址就是null。null通过typeof判断类型是'object'。null === null null == null null == undefined null !== undefined undefined 表示一个变量初始状态值，而 null 则表示一个变量被人为的设置为空对象，而不是原始状态。在实际使用过程中，不需要对一个变量显式的赋值 undefined，当需要释放一个对象时，直接赋值为 null 即可。 让一个变量为null，直接给该变量赋值为null即可。 加分回答 null 其实属于自己的类型 Null，而不属于Object类型，typeof 之所以会判定为 Object 类型，是因为JavaScript 数据类型在底层都是以二进制的形式表示的，二进制的前三位为 0 会被 typeof 判断为对象类型，而 null 的二进制位恰好都是 0 ，因此，null 被误判断为 Object 类型。 对象被赋值了null 以后，对象对应的堆内存中的值就是游离状态了，GC 会择机回收该值并释放内存。因此，需要释放某个对象，就将变量设置为 null，即表示该对象已经被清空，目前无效状态。

**12、说一下浮动**浮动的特点：

（1）浮动在父元素范围内，不会从父元素中移出。

（2）元素设置浮动后，会完全从文档流中脱离，不再占用文档流的位置。之后的元素会自动向上移动占用文档流的位置。

（3）浮动元素不会超过前面的浮动元素，利用此特点可以设置水平排列布局。

（4）浮动元素不会盖住文字，文字会自动环绕在浮动元素周围，利用此特点来设置文字环绕图片效果。  
设置浮动的图片，可以实现文字环绕图片，设置了浮动的块级元素可以排列在同一行，设置了浮动的行内元素可以设置宽高，同时可以按照浮动设置的方向对齐排列盒子。 设置浮动元素的特点： -设置了浮动，该元素脱标。元素不占位置 -浮动可以进行模式转换（行内块元素） 浮动造成的影响，使盒子脱离文档流，如果父级盒子没有设置高度，需要被子盒子撑开，那么这时候父级盒子的高度就塌陷了(“子元素设置浮动或者设置绝对定位后,子元素会完全脱离文档流,此时将会导致子元素无法撑起父元素的高度,导致父元素高度塌陷。”)，同时也会造成父级盒子后面的兄弟盒子布局受到影响。如果浮动元素后面还有其他兄弟元素，其他兄弟元素的布局也会受到影响。 清除浮动的方法： -伪元素清除浮动：给浮动元素父级增加 .clearfix::after { content: ''; display: table; clear: both; } /\*兼容IE低版本 \*/ .clearfix { \*zoom: 1; } overflow：hidden`：给浮动元素父级增加`overflow：hidden`属性 额外标签法：给浮动元素父级增加标签

加分回答 三种清除浮动的特点和影响 -伪元素清除浮动：不会新增标签，不会有其他影响，是当下清除浮动最流行的方法 -`overflow：hidden`：不会新增标签，但是如果父级元素有定位元素超出父级，超出部分会隐藏，在不涉及父级元素有超出内容的情况，overflow：hidden比较常用，毕竟写法方便简洁 -标签插入法：清除浮动的语法加在新增标签上，由于新增标签会造成不必要的渲染，所以这种方法目前不建议使用。

1. **箭头函数**  
   箭头函数相当于匿名函数，简化了函数定义。箭头函数有两种写法，当函数体是单条语句的时候可以省略{}和return。另一种是包含多条语句，不可以省略{}和return。 箭头函数最大的特点就是没有this，所以this是从外部获取，就是继承外部的执行上下文中的this，由于没有this关键字所以箭头函数也不能作为构造函数， 同时通过 `call()` 或 `apply()` 方法调用一个函数时，只能传递参数（不能绑定this），第一个参数会被忽略。箭头函数也没有原型和super。不能使用yield关键字，因此箭头函数不能用作 Generator 函数。不能返回直接对象字面量。 加分回答 箭头函数的不适用场景： -定义对象上的方法 当调用` dog.jumps` 时，`lives` 并没有递减。因为 `this` 没有绑定值，而继承父级作用域。 var dog = { lives: 20, jumps: () => { this.lives--; } } -不适合做事件处理程序 此时触发点击事件，this不是button，无法进行class切换 var button = document.querySelector('button'); button.addEventListener('click', () => { this.classList.toggle('on'); }); 箭头函数函数适用场景： -简单的函数表达式，内部没有this引用，没有递归、事件绑定、解绑定，适用于map、filter等方法中，写法简洁 var arr = [1,2,3]; var newArr = arr.map((num)=>num\*num) -内层函数表达式，需要调用this，且this应与外层函数一致时 let group = { title: "Our Group", students: ["John", "Pete", "Alice"], showList() { this.students.forEach( student => alert(this.title + ': ' + student) ); } }; group.showList();
2. **说一说call apply bind的作用和区别？**  
   call、apply、bind的作用都是改变函数运行时的this指向。 bind和call、apply在使用上有所不同，bind在改变this指向的时候，返回一个改变执行上下文的函数，不会立即执行函数，而是需要调用该函数的时候再调用即可，但是call和apply在改变this指向的同时执行了该函数。 bind只接收一个参数，就是this指向的执行上文。 call、apply接收多个参数，第一个参数都是this指向的执行上文，后面的参数都是作为改变this指向的函数的参数。但是call和apply参数的格式不同，call是一个参数对应一个原函数的参数，但是apply第二个参数是数组，数组中每个元素代表函数接收的参数，数组有几个元素函数就接收几个元素。 加分回答 call的应用场景： 对象的继承，在子构造函数这种调用父构造函数，但是改变this指向，就可以继承父的属性 function superClass () { this.a = 1; this.print = function () { console.log(this.a); } } function subClass () { superClass.call(this); // 执行superClass，并将superClass方法中的this指向subClass this.print(); } subClass(); 借用Array原型链上的slice方法，把伪数组转换成真数组 let domNodes = Array.prototype.slice.call(document.getElementsByTagName("div")); apply的应用场景： Math.max，获取数组中最大、最小的一项 let max = Math.max.apply(null, array); let min = Math.min.apply(null, array); 实现两个数组合并 let arr1 = [1, 2, 3]; let arr2 = [4, 5, 6]; Array.prototype.push.apply(arr1, arr2); console.log(arr1); // [1, 2, 3, 4, 5, 6] bind的应用场景 在vue或者react框架中，使用bind将定义的方法中的this指向当前类
3. **说一说this指向（普通函数、箭头函数）？**  
    this关键字由来：在对象内部的方法中使用对象内部的属性是一个非常普遍的需求。但是 JavaScript 的作用域机制并不支持这一点，基于这个需求，JavaScript 又搞出来另外一套 this 机制。 this存在的场景有三种全局执行上下文和函数执行上下文和eval执行上下文，eval这种不讨论。在全局执行环境中无论是否在严格模式下，（在任何函数体外部）`this` 都指向全局对象。在函数执行上下文中访问this，函数的调用方式决定了 `this` 的值。在全局环境中调用一个函数，函数内部的 this 指向的是全局变量 window，通过一个对象来调用其内部的一个方法，该方法的执行上下文中的 this 指向对象本身。 普通函数this指向：当函数被正常调用时，在严格模式下，this 值是 undefined，非严格模式下 this 指向的是全局对象 window；通过一个对象来调用其内部的一个方法，该方法的执行上下文中的 this 指向对象本身。new 关键字构建好了一个新对象，并且构造函数中的 this 其实就是新对象本身。嵌套函数中的 this 不会继承外层函数的 this 值。 箭头函数this指向：箭头函数并不会创建其自身的执行上下文，所以箭头函数中的 this 取决于它的外部函数。 加分回答 箭头函数因为没有this，所以也不能作为构造函数，但是需要继承函数外部this的时候，使用箭头函数比较方便 var myObj = { name : "闷倒驴", showThis:function(){ console.log(this); // myObj var bar = ()=>{ this.name = "王美丽"; console.log(this) // myObj } bar(); } }; myObj.showThis(); console.log(myObj.name); // "王美丽" console.log(window.name); // ''
4. **说说未知宽高元素水平居中的办法**  
   设置元素相对父级定位`position:absolute;left:50%;right:50%`，让自身平移自身高度50% `transform: translate(-50%,-50%);`，这种方式兼容性好，被广泛使用的一种方式  
   设置元素的父级为弹性盒子`display:flex`，设置父级和盒子内部子元素水平垂直都居中`justify-content:center; align-items:center` ，这种方式代码简洁，但是兼容性ie 11以上支持，由于目前ie版本都已经很高，很多网站现在也使用这种方式实现水平垂直居中  
   设置元素的父级为网格元素`display: grid`，设置父级和盒子内部子元素水平垂直都居中`justify-content:center; align-items:center` ，这种方式代码简介，但是兼容性ie 10以上支持  
   设置元素的父级为表格元素`display: table-cell`，其内部元素水平垂直都居中`text-align: center;vertical-align: middle;` ，设置子元素为行内块`display: inline-block; `，这种方式兼容性较好
5. **创建ajax** 1. 创建XHR对象：new XMLHttpRequest() 2. 设置请求参数：request.open(Method, 服务器接口地址); 3. 发送请求: request.send()，如果是get请求不需要参数，post请求需要参数request.send(data) 4. 监听请求成功后的状态变化：根据状态码进行相应的处理。 XHR.onreadystatechange = function () { if (XHR.readyState == 4 && XHR.status == 200) { console.log(XHR.responseText); // 主动释放,JS本身也会回收的 XHR = null; } }; 加分回答 POST请求需要设置请求头 readyState值说明 0：初始化,XHR对象已经创建,还未执行open 1：载入,已经调用open方法,但是还没发送请求 2：载入完成,请求已经发送完成 3：交互,可以接收到部分数据 4：数据全部返回 status值说明 200：成功 404：没有发现文件、查询或URl 500：服务器产生内部错误

一、**js部分**

1、DOM树中总共分为如下几种节点格式：Element类型（元素节点）、Text类型（文本节点）、Comment类型（注释节点）、Document类型（document节点）。所有的 HTML elements 都是 element nodes，在 HTML element 內的 text 內容也是 text node

DHTML实现了网页从Web服务器下载后无需再经过服务的处理，而在浏览器中直接动态地更新网页的内容、排版样式和动画的功能。例如，当鼠标指针移到文章段落中时，段落能够变成蓝色，或者当鼠标指针移到一个超级链接上时，会自动生成一个下拉式子链接目录等。

包括：①动态内容(Dynamic Content)：动态地更新网页内容，可“动态”地插入、修改或删除网页的元件，如文字、图像、标记等。 ②动态排版样式(Dynamic Style Sheets)：W3C的CSS样式表提供了设定HTML标记的字体大小、字形、样式、粗细、文字颜色、行高度、加底线或加中间横线、缩排、与边缘距离、靠左右或置中、背景图片或颜色等排版功能，而“动态排版样式”即可以“动态”地改变排版样式。

2、DOM中的事件对象：（符合W3C标准）

   preventDefault()        取消事件默认行为

   stopImmediatePropagation() 取消事件冒泡同时阻止当前节点上的事件处理程序被调用。

   stopPropagation()      取消事件冒泡对当前节点无影响。

3、IE中的事件对象：

   cancelBubble()          取消事件冒泡

   returnValue()             取消事件默认行为

oncontextmenu 事件在元素中用户右击鼠标时触发并打开上下文菜单。

注意：所有浏览器都支持 oncontextmenu 事件， [contextmenu](https://www.runoob.com/tags/att-global-contextmenu.html) 元素只有 Firefox 浏览器支持

4、spellcheck 属性规定是否对元素进行拼写和语法检查。（不是很明白怎么进行检查的）可以对以下内容进行拼写检查：

5、简单数据类型(基本类型)：字符串（String）、数字(Number)、布尔(Boolean)、对空（Null）、未定义（Undefined）、Symbol(ES6新增), 复杂数据类型(引用数据类型)：对象(Object)、数组(Array)、函数(Function)

6、函数声明，函数声明有提升现象；函数表达式，函数表达式不会提升，实际代码执行顺序如下：

|  |  |
| --- | --- |
| 1  2  3  4  5 | var check；  check('second');  check= function(ars){      console.log(ars);  }//在未给check赋值的情况下调用，所以报错 |

7、Java delete运算符只能删除自由属性，不能删除继承属性。

8、一般情况下，给参数进行复制修改，其arguments的值也会修改。在给参数进行赋值时，将其变为严格模式，对参数的修改，其arguments的值不被修改。

9、Math.random()方法返回指0到指定数之间的随机数,默认值为1，Math.random()\*4返回0到4之间的随机数;Math.floor()方法返回一个整数，向下取值，例如：Math.floor(5.2232)，取值为5;Math.ceil()方法返回一个整数，向上取值，例如：Math.ceil(5.2232)，取值为6;

eval：返回字符串表达式中的值unEscape：返回字符串ASCI码escape：返回字符的编码

parseFloat：返回实数

10、js中可以修改数组长度，java不能修改，其他语言应该不能修改的居多

11、Shift()返回的是删除的元素

12、检测是否为数组用instanceof，typeof用来检测基本类型

13、根据官方的JVM规范：如果try语句里有return，返回的是try语句块中变量值。详细执行过程如下：如果有返回值，就把返回值保存到局部变量中；执行jsr指令跳到finally语句里执行；执行完finally语句后，返回之前保存在局部变量表里的值。如果try，finally语句里均有return，略try的return，而使用finally的return.

14、null是关键字，NULL不是关键字，java区分大小写。 java关键字都是小写。

15、bootstrap框架，页面是采用的IE盒模型的方式呈现的。content-box指定盒子模型为W3C（标准盒模型），border-box为IE盒子模型（怪异盒模型）

16、let const var 相关

ES6之前创建变量用的是var,之后创建变量用的是let/const

三者区别：

（1）var定义的变量，没有块的概念，可以跨块访问, 不能跨函数访问。let定义的变量，只能在块作用域里访问，不能跨块访问，也不能跨函数访问。const用来定义常量，使用时必须初始化(即必须赋值)，只能在块作用域里访问，且不能修改。

（2）var可以先使用，后声明，因为存在变量提升；let必须先声明后使用。

（3）var是允许在相同作用域内重复声明同一个变量的，而let与const不允许这一现象。

在全局上下文中，基于let声明的全局变量和全局对象GO（window）没有任何关系 ;

（4）var声明的变量会和GO有映射关系；

（5）会产生暂时性死区：

暂时性死区是浏览器的bug：检测一个未被声明的变量类型时，不会报错，会返回undefined

如：console.log(typeof a) //undefined

而：console.log(typeof a)//未声明之前不能使用

let a

（6）let /const/function会把当前所在的大括号(除函数之外)作为一个全新的块级上下文，应用这个机制，在开发项目的时候，遇到循环事件绑定等类似的需求，无需再自己构建闭包来存储，只要基于let的块作用特征即可解决

**二、操作系统**

1、最佳适应:空闲区按容量大小递增 最差适应:空闲区按容量大小递减 最先适应:空闲区按地址大小递增

**三、mysql**

1、MYSQL中处理插入过程主键或唯一重复值的解决办法：

IGNORE:有则忽略，无则插入

REPLACE：有则删除再插入，无则插入

ON DUPLIACATE KEY UPDATE:有则更新，无则插入只更新新增部分，其余未涉及的字段不变；无则添加；

2、rank() over(pattition by id,order by score desc)the\_rank from SC 1 2 2 4

Dense\_rank() over(pattition by id,order by score desc)the\_rank from SC 1 2 2 3

Row\_number()over() 1 2 3 4

3、case

When then

When then

Else

End

4、DateDiff(datepart,startdate,enddate)

5、ifnull(a,b) 第一个参数为空则返回b

6、inner join 内连接或者等值连接 left join 获取左表所有记录

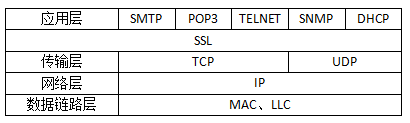
7、limit a offset b limit限制select的查询数量，offset表示要跳过的数量

8、min(日期)

9、select 没有则返回null

**四、计算机网路**

1、响应分为五类：信息响应(100–199)，成功响应(200–299)，重定向(300–399)，客户端错误(400–499)和服务器错误 (500–599)：



**五、网页设计html、css**

1、块状元素：div h dt dd li p；行内元素：a b span i input img label u var；行内块元素：canvas

块级元素总是从新的一行开始，即各个块级元素独占一行，默认垂直向下排列；高度、宽度、margin及padding都是可控的，设置有效，有边距效果；宽度没有设置时，默认为100%；块级元素中可以包含块级元素和行内元素。行内元素和其他元素都在一行，即行内元素和其他行内元素都会在一条水平线上排列；高度、宽度是不可控的，设置无效，由内容决定。根据标签语义化的理念，行内元素最好只包含行内元素，不包含块级元素。转换：当然块级元素与行内元素之间的特性是可以相互转换的。HTML可以将元素分为行内元素、块状元素和行内块状元素三种。使用display属性能够将三者任意转换：

(1)display:inline;转换为行内元素；

(2)display:block;转换为块状元素；

(3)display:inline-block;转换为行内块状元素。

⭐⭐new⭐⭐2、伪类和伪元素

区别：是否创建列新元素

伪类：用于已有元素处于某种状态时为其添加对应的样式，这个状态是根据用户行为而动态变化的。例如：当用户悬停在指定元素时，可以通过:hover来描述这个元素的状态，虽然它和一般css相似，可以为 已有元素添加样式，但是它只有处于DOM树无法描述的状态下才能为元素添加样式，所以称为伪类。

伪元素：用于创建一些不在DOM树中的元素，并为其添加样式。例如，我们可以通过:before来在一个元素之前添加一些文本，并为这些文本添加样式，虽然用户可以看见 这些文本，但是它实际上并不在DOM文档中。

1. html语义化标签

HTML5中加入了一些语义化标签，来更清晰的表达文档结构。如 title、header、nav、article、section、aside、footer、 strong等。语义化更具有可读性，代码更好维护。

4、h5新增的特性

（一）媒体播放的 video 和 audio

<video src =# preload =auto/meta/none loop controls autoplay muted ></video>

cobtrols是设置是否有播放的组件

muted默认静音

Poster=”....”设置视频封面

<audio duration paused volume starttime currentTime.....></audio>

5、选择器类型

!important

内联样式（1000）

ID选择器（0100）

类选择器/属性选择器/伪类选择器（0010）(相等优先级)

元素选择器/伪元素选择器（0001）

关系选择器/通配符选择器（0000）

带!important 标记的样式属性优先级最高； 样式表的来源相同时：!important > 行内样式>ID选择器 > 类选择器 > 标签 > 通配符 > 继承 > 浏览器默认属性

6、position属性  
固定定位 fixed： 元素的位置相对于浏览器窗口是固定位置，即使窗口是滚动的它也不会移动。Fixed 定位使元素的位置与文档流无关，因此不占据空间。 Fixed 定位的元素和其他元素重叠。

*相对*定位 relative： 如果对一个元素进行相对定位，它将出现在它所在的位置上。然后，可以通过设置垂直或水平位置，让这个元素“相对于”它*自己*的起点进行移动（↓→为正）在使用相对定位时，无论是否进行移动，元素仍然占据原来的空间，不会脱离文档流和改变元素性质。【灵魂出窍】

绝对定位 absolute： 绝对定位的元素的位置相对于最近的已定位父元素，如果元素没有已定位的父元素，那么它的位置相对于absolute 定位使元素的位置与文档流无关，因此不占据空间。 absolute 定位的元素和其他元素重叠，定位元素提升一个层级覆盖其他元素。【层级特点：父子水涨船高；同级后者居上。】

粘性定位 sticky： 元素先按照普通文档流定位，然后相对于该元素在流中的 flow root（BFC）和 containing block（最近的块级祖先元素）定位。而后，元素定位表现为在跨越特定阈值前为相对定位，之后为固定定位。

默认定位 Static： 默认值。没有定位，元素出现在正常的流中（忽略 top, bottom, left, right 或者 z-index 声 明）。 inherit: 规定应该从父元素继承 position 属性的值。

7、让一个元素水平垂直居中  
水平居中  
（一）、对于行内元素 : text-align: center;

（二）、对于确定宽度的块级元素：

（1）width和margin实现。margin: 0 auto;

（2）绝对定位和margin-left: -width/2, 前提是父元素position: relative

（三）、对于宽度未知的块级元素

（1）table标签配合margin左右auto实现水平居中。使用table标签（或直接将块级元素设值为 display:table），再通过给该标签添加左右margin为auto。

（2）inline-block实现水平居中方法。display：inline-block和text-align:center实现水平居中。

（3）绝对定位+transform，translateX可以移动本身元素的50%。

（4）flex布局使用justify-content:center

垂直居中

（一）、利用 line-height 实现居中，这种方法适合纯文字类。

（二）、通过设置父容器 相对定位 ，子级设置 绝对定位，标签通过margin实现自适应居中。

（三）、弹性布局 flex :父级设置display: flex; 子级设置margin为auto实现自适应居中。

（四）、父级设置相对定位，子级设置绝对定位，并且通过位移 transform 实现。

（五）、table 布局，父级通过转换成表格形式，然后子级设置 vertical-align 实现。（需要注意的是：vertical-align: middle使用的前提条件是内联元素以及display值为table-cell的元素）

8、链接伪类的使用顺序是很重要的。对于超链接的伪类，推荐的使用顺序是：:link - :visited - :hover - :active。

9、脱离文档流：块元素变成行内块元素，宽高由内容撑开，也可自己设置。