都有哪些方式实现回到顶部功能?

1. 使用锚点，页面顶部放置一个锚点链接，然后在页面下方放置一个返回到该锚点的链接，用户点击该链接即可返回到该锚点所在的顶部位置。

```
//锚点方式
<body style="height:2000px;">
    <div id="topAnchor"></div>
    <a href="#topAnchor" style="position:fixed;right:0;bottom:0">回到顶部</a>
</body>
```

2. 监听浏览器的scollTop事件,当用户滚动到页面底部时，出现回到顶部按钮，点击之后重置浏览器滚动条的高度.

```
//兼容性获取并监听浏览器滚动条
var osTop=document.documentElement.scrollTop|| document.body.scrollTop;
document.documentElement.scrollTop=document.body.scrollTop = 0;
```

3.第三方插件，例如jquery中:

```
$('html,body').animate({srollTop:0},500);
```

javaScript是一种什么样的语言?

- javascript是一门弱类型的，直译型的，运行在浏览器上的脚本语言，用于给网页添加动态效果和交互。

如何判断用户的设备来源?

- 使用Navigator.userAgent方法来判断

js中forEach和map的区别?

1. 两者都可以用来遍历数组，forEach无返回值，map会映射并返回一个新的数组。
2. forEach会允许修改原数组的值，而map不能修改原数组的值。

为什么操作dom会消耗性能

- 因为JS解释引擎和浏览器的渲染引擎是各自独立的，JS引擎通过渲染引擎的接口来获取文档中的元素，每次操作DOM时都会先进行连接，而每次连接都会消耗性能。
- 因为频繁操作dom的话，会触发浏览器的重排和重绘，而重排和重绘会占用CPU和GPU从而导致性能的下降。

浏览器渲染引擎的工作原理和工作流程?

1. 解析HTML文档，构建DOM树（DOM节点树。
2. 解析CSS文档，生成CSS规则树。
3. 合并DOM树和CSS规则树，生成渲染树render树。
4. 布局render树，计算每个元素的大小，位置等信息（重排会走这一步）。
5. 绘制render树，绘制页面的像素信息，（根据render树上每个节点的几何属性计算每个节点的像素数，重绘会走这一步）

6. 浏览器会将各层节点的像素信息发给GPU，GPU将各层进行合成和渲染，最终展示到页面上。

什么是重排和重绘?

- 当DOM节点宽高等（几何属性）属性发生改变时，浏览器需要重新计算元素的几何属性，这时候浏览器就会重新构造渲染树，这个过程就叫做重排。
- 完成重排后，浏览器会重新绘制把受影响的部分到浏览器上，这个过程就成为重绘，重排必然会引发重绘。

从网站上输入url到用户展示界面，中间都发生了什么?

1. DNS解析
2. TCP连接(三次握手)
3. 客户端向服务端发送http请求
4. 服务器处理请求，返回http报文
5. 浏览器解析和渲染页面
6. 连接结束

你做的页面都在哪些浏览器测试过，这些浏览器的内核分别是什么?

- IE浏览器：Trident内核
- Chrome浏览器：webkit（过去）、blink（现在）
- FireFox：Gecko内核
- Edge：chormium内核
- Safari：webkit内核

HTML文件开头的Doctype的作用?

- 用来声明文档类型，告知浏览器应该以何种规范来解析
- 这叫W3C标准，如果不加浏览器会用自己的规范去解析文档，可能在不同浏览器会造成不同的效果

渐进增强和优雅降级如何理解?

- 渐进增强，写样式的时候从低版本的浏览器开始兼容，从下向上兼容。
- 优雅降级，构建页面的时候，先不考虑兼容问题，先构建出完整的页面，再由上向下进行兼容。

src和href的区别?

- src是source的缩写，表示将外部资源加载到文档内
- href用于在当前文档和引用资源之间确认联系（侧重点）
- href会在页面加载时同时加载，src会在页面加载完加载。

你在开发过程中都用过什么格式的图片?

- png-8，png-24，jpeg，gif，svg等

网页上图片非常多，加载会比较慢，如何优化?

- 图片懒加载、图片压缩
- 使用CDN加速
- 与公司UI沟通，让他把压缩过的图片发过来

什么是异步加载？异步加载和延迟加载有什么区别?

- 异步加载时相对于同步加载而言的，我们平常书写的代码就是同步加载，代码自上而下执行，是阻塞式的，而异步加载是非阻塞式的，在执行同步代码时，并不会阻塞我后续代码的执行，（例如定时器，发送ajax请求），而延迟加载则是一开始并不加载，在我需要的时候再进行加载（例如图片的懒加载）

如何水平垂直居中一个盒子?

- 给父元素相对定位
- 给子元素绝对定位，并且：
- left: 50%;top: 50%;
- margin-left: 负的宽度一半。
- margin-top: 负的高度一半；
- 代码实现:

```
        #father{
            width: 500px;
            height: 500px;
            position: relative;
        }
        #son{
            width: 100px;
            height: 100px;
            position: absolute;
            top: 50%;
            left: 50%;
            margin-left: -50px;
            margin-top: -50px;
        }
```

- 使用display:flex；
- justify-content：center；
- align-items：center；
- 代码实现：

```
    .box {
        width: 500px;
        height: 500px;
        background-color: aquamarine;
        display: flex;
        align-items: center;
        justify-content: center;
      }
      .son {
        width: 200px;
        height: 200px;
        background-color: red;
      }
```

- 使用定位的方式
- 代码实现：

```
    .box {
        width: 500px;
        height: 500px;
```

```
        position: relative;
        }
     .son {
        width: 200px;
        height: 200px;
        position: absolute;
        top: 0;
        bottom: 0;
        left: 0;
        right: 0;
        margin: auto;
      }
```

- 定位+translate
- 代码实现：

```
    .box {
        width: 400px;
        height: 400px;
        background-color: cyan;
        position: relative;
     }
     .son {
        width: 200px;
        height: 200px;
        background-color: purple;
        position: absolute;
        left: 50%;
        top: 50%;
        transform: translate(-50%, -50%);
     }
```

flex是如何使用的?

- flex是一种css盒子的布局方式，通过改变父元素的属性来控制子元素的排列方式，其常用属性有：
- flex-direction 主轴的方向
- flex-wrap 子元素空间不足时候是否换行
- justify-content 主轴对齐方式
- align-items 交叉轴的对齐方式
- align-content 多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用

flex：1代表着什么?

- flex：1 是 flex-grow、flex-shrink、flex-basis三个属性的缩写，一般默认为flex：1 1 auto（存在剩余空间就放大，空间不足就缩小）；
- flex-grow：元素的放大比例，默认为1，即使存在剩余空间，也不会放大；
- flex-shrink：元素的缩小比例，默认为1，即如果空间不足，该元素将缩小;
- flex-basis:项目占据的主轴空间,默认为auto，即项目原本的大小。

px、rem、em、vw的区别?

- px是相对于显示器屏幕分辨率而言的，固定的一种单位。
- em是相对于父元素的字体大小而言的，譬如父元素字体大小为16px，那么1em=16px，2em=32px。

- rem是相对于根元素（html）字体大小而言的，浏览器默认的字体大小都是16px，那么1rem=16px，2rem=32px，可以根据设备的宽度，结合媒体查询（@media）来进行自适应的布局。
- vw是窗口的实际宽度，1vw=1% 窗口的实际宽度。

px 与 rem 的选择?

- 对于只需要适配少部分手机设备，且分辨率对页面影响不大的，使用px即可 。
- 对于需要适配各种移动设备，使用rem，例如只需要适配iPhone和iPad等分辨率差别比较挺大的设备。

CSS中的定位方式?

- 绝对定位，参照离自己的最近的，有定位属性的父元素定位。
- 相对定位，参照自己本身的位置进行定位。
- 固定定位，基于浏览器窗口定位。
- 默认定位，，由左到右，从上往下流式布局。

bootstarp实现响应式的原理?

- 媒体查询

布局效果?

- 圣杯布局，使用flex

数据去重?

- new Set
- 单层循环，定义一个空数组，往这个空数组中插入数据，插入之前使用indexOf进行判断，判断是否已经存在当前数据。
- 如果元素中有字符串、数字、对象和数字等不同元素的话，使用lodash插件进行去重。

对象中的属性按照属性名进行字典排序?

```
let obj = {b:'hello',name:'Tom',a:'vue!'};
let keys = Object.keys(obj).sort();
let newObj={};
keys.forEach(item => newObj[item] = obj[item]);
```

比较两个对象是否相等?

```
let obj = {name:'tom',age:10};
let obj2 = {name:'tom',age:10};
function(obj1,obj2){
    return JSON.stringfy(obj) == JSON.stringfy(obj2);
}
//如果对象属性顺序不一样，那就先把对象排序，使用上面的排序算法先进行字典排序。
//lodash插件进行比较
```

Promise与async/await有什么区别?

1. Promise是ES6中处理异步请求的语法，使用.then()来实现链式调用,使用.catch()来捕获异常。

2. async/await 是对Promise的升级，async用于声明一个函数是异步的，awaait是等待一个异步方法执行完成（await一个Promise对象），async/await的捕获异常可以使用try/catch语法。（也可以使用.catch语法）

http三次握手以及常见的状态码?

- 第一次握手是客户端给服务端发送请求，请求建立连接。
- 第二次握手是服务端给客户端发送请求，表示已经收到客户端的请求，并且同意建立连接。
- 第三次握手是客户端向服务端发送请求，表示确认收到服务端发送的信息。
- 三次握手的原因是为了确认客户端和服务端都有收发信息的能力，少一次确认不了，多一次浪费资源。

常见状态码：

- 200：服务器已经成功处理了请求。
- 3XX：表示重定向。
- 401：未授权，服务器请求身份验证。
- 404：服务器找不到请求的网页。
- 500：服务器内部错误，无法完成请求。
- 503：服务器目前无法使用。

head标签中有一个viewport标签，这是干什么用的?

- 是用来适配移动端屏幕的，该标签的作用是让当前viewport的宽度等于设备的宽度，同时不允许用户手动缩放。

js中常见的兼容性问题?

1. 关于获取行外样式currentStyle和getComputedStyle出现的兼容问题。

```
ie：currentStyle()
其他：getComputedStyle('元素名'，属性名)
```

图片懒加载是怎么实现的?

- lazy-load
- 使用占位符来保存图片真实的地址，然后监听滚动条的滚动事件，当图片出现在可视区域内时，将图片的src属性替换成图片的真实地址即可。

ES6中class语法中的constructor和super是什么

- super作为函数调用时，代表父类的构造函数。ES6 要求，子类的构造函数必须执行一次super函数

```
class A {}
class B extends A {
  constructor() {
    super();
  }
}
```

- 上面代码中，子类B的构造函数之中的super()，代表调用父类的构造函数。

- constructor是类的构造函数,如果子类没有定义constructor方法，这个方法会被默认添加，代码如下:

```
class ColorPoint extends Point {
}
// 等同于
class ColorPoint extends Point {
  constructor(...args) {
    super(...args);
  }
}
```

- 另一个需要注意的地方是，在子类的构造函数中，只有调用super之后，才可以使用this关键字，否则会报错。这是因为子类实例的构建，基于父类实例，只有super方法才能调用父类实例。

js中如何准确判断一个数据的数据类型?

- 如果是基本数据类型，使用typeof来判断，但是typeof不能用来判断引用数据类型的数据。

```
typeof null                   // 'object'
typeof undefined              // 'undefined'
typeof 123                    // 'number'
typeof Infinity               // 'number'
typeof NaN                    // 'number'
typeof true                   // 'boolean'
typeof 'abc'                  // 'string'
typeof function(){}           // 'function'
typeof {name: 'uolcano'}      // 'object'
typeof [1,2,3]                // 'object'
typeof new (function F(){})   // 'object'
```

- 正确判断一个数据的数据类型应该选择

```
 Object.prototype.toString.call( )
```

cookie，localstorage，sessionstorage的区别?

生命周期:

- cookie：可设置失效时间，没有设置的话，默认是关闭浏览器后失效
- localStorage：除非被手动清除，否则将会永久保存。
- sessionStorage： 仅在当前网页会话下有效，关闭页面或浏览器后就会被清除。

存放数据大小:

- cookie：4KB左右
- localStorage和sessionStorage：可以保存5MB的信息。

http请求:

- cookie：每次都会携带在HTTP头中，如果使用cookie保存过多- 数据会带来性能问题
- localStorage和sessionStorage：仅在客户端（即浏览器）中保存，不参与和服务器的通信

易用性:

- cookie：需要程序员自己封装，源生的Cookie接口不友好

- localStorage和sessionStorage：源生接口可以接受，亦可再次封装来对Object和Array有更好的支持

应用场景：

- 从安全性来说，因为每次http请求都会携带cookie信息，这样无形中浪费了带宽，所以cookie应该尽可能少的使用，另外cookie还需要指定作用域，不可以跨域调用，限制比较多。但是用来识别用户登录来说，cookie还是比storage更好用的。其他情况下，可以使用storage，就用storage。
- storage在存储数据的大小上面秒杀了cookie，现在基本上很少使用cookie了，因为更大总是更好的，哈哈哈你们懂得。
  localStorage和sessionStorage唯一的差别一个是永久保存在浏览器里面，一个是关闭网页就清除了信息。localStorage可以用来夸页面传递参数，sessionStorage用来保存一些临时的数据，防止用户刷新页面之后丢失了一些参数。

typeof null和typeof undefined的结果什么，为什么?

undefined

- 当你声明一个变量但未对其初始化时，这个变量的值就是undefined

null

- 表示一个空对象指针，表示没有指向任何值。

```
typeof null         // 'object'
typeof undefined    // 'undefined'
#因为undefined本身就是一种基本数据类型，所以typeof undefined == undefined
```

- 简单来说是js历史遗留问题，typeof null的结果为Object的原因是一个bug。在 js 的最初版本中，使用的 32位系统，使用低位来存储变量的类型信息。
  而null的机器码标识为全0，而对象的机器码低位标识为000。所以typeof null的结果被误判为Object。

```
//null 和 undefined 的值相等，但类型不等：

typeof undefined             // undefined
typeof null                  // object
null === undefined           // false
null == undefined            // true
```

filter和map以及reduce的区别及用法?

- map() 方法映射并返回一个新的数组，不会改变原数组。

```
array.map(function(value,index,arr))
 /*value表示当前值
 index表示当前下标
 arr表示当前数组*/
```

- filter方法表示过滤，返回过滤后的新数组。

```
var ages = [32, 33, 16, 40];

function checkAdult(age) {
    return age >= 18;
}
```

- reduce方法表示数组归并（累加），返回累加后的值。

```
var arr = [1, 2, 3, 4];
var sum = arr.reduce(function(prev, cur, index, arr) {
    return prev + cur;
},init)
/*
prev表示上一次调用回调函数时的返回值
cur表示当前元素
index表示当前元素下标
arr表示当前数组
*/
```

什么是语义化标签？有什么好处？

- H5新增的语义化标签有main、section、nav、footer、header、article等。
- 好处：通过标签名字就可以知道标签的作用或者用途，语义化的标签可读性更强，便于维护，也便于seo优化。

什么是seo优化？

- seo就是搜索引擎优化，国内主流是百度，竞价排名。
- 前端开发的seo技巧：
- 设置meta标签，设置网站的keywords 和desctiption
- 不要在页面中出现空链接
- 为图片添加title和alt属性
- 设置页面中的h1-h6标签
- 优化网站性能，提高网站加载速度
- 购买外链

网站性能的优化

- js和css文件压缩合并
- 减少http请求次数
- 对网页中的图片资源进行压缩
- css中使用精灵图
- 使用cdn资源加速
- 开启服务器的gzip压缩
- 开启服务器的缓存

浏览器兼容性

- 国内主要兼容360安全浏览器，极速模式和兼容模式，分别对应谷歌的blink内核和IE的trident内核
- 电脑中浏览器的兼容性（360、qq浏览器、搜狗）
- 手机浏览器的浏览器兼容性（一般手机，小米手机，ios手机），微信浏览器。
- 你是如何解决这些兼容性问题的？
- 答：我们都是浏览器挨个去调试，或者加浏览器前缀，或者jQuery来解决js的兼容问题，避免使用h5和css3新增的标签和样式。

如何实现页面自适应?

- 使用rem单位
- 使用媒体查询
- 使用flex结合百分比布局

普通函数和箭头函数的区别?

- 箭头函数没有自己的this指向，它的this指向来源于它的上级，并且继承而来的this指向是无法改变的。
- 箭头函数由于没有自己的this，所以不能作为构造函数。
- 箭头函数中没有arguments（形参数组）

函数声明和函数表达式的区别?

- 函数声明会被预解析，函数表达式不会。

for...in 和for...of的区别?

- for...in只能获得对象的键名，对数组来说是下标，对象是属性名。并且手动添加的属性也能遍历到。
- for...of只能获得键值（数组），遍历对象会报错。

js中Map和Object有什么区别?

- Map是es6新增的一种数据结构，继承自Object，并对Object做了一些拓展。
- Map的键可以是任意的数据类型，Object不行。
- Map是可迭代（iterable）的，可以使用for...of遍历，而Object不行。
- Map的长度可以通过size直接拿到，Object不行。
- Map使用拓展运算符是可以直接展开的，Object不行。

什么是简单对象？如何判断一个对象是不是简单对象?

- 简单对象又称纯粹对象或者朴素对象
- 是指用{ }或者new Object运算符创建出来的对象

```
function isPlainObject(obj) {
  if (typeof obj !== 'object' || obj === null) return false
  let proto = obj
  while (Object.getPrototypeOf(proto) !== null) {
    proto = Object.getPrototypeOf(proto)
  }
  return Object.getPrototypeOf(obj) === proto
}
```

css中的transition和animate的用法以及区别?

- transition表示过渡，表示一个元素从一种状态到另一种状态所需要的时间。

1. transition需要事件触发，所以没法在网页加载时自动发生。
2. transition是一次性的，不能重复发生，除非一再触发。
3. transition只能定义开始状态和结束状态，不能定义中间状态，也就是说只有两个状态。

- animate表示动画。

```
//创建名为myfirst的动画
@keyframes myfirst
{
    from {background: red;}
    to {background: yellow;}
}
//执行动画
keyframes myfirst
{
    from {background: red;}
    to {background: yellow;}
}
```

1. 动画不需要事件触发。
2. 动画可以通过设置infinite来循环播放。
3. 可以通过animation-duration来设置动画的持续时间，animation-delay来设置动画开始前的时间等等。

为什么会存在对象的深拷贝问题？如何实现对象的深拷贝？

- 深浅拷贝是因为引用数据类型数据存储的问题，引用数据类型会把指针存放在栈内存中，真正的值存放在堆内存中，浅拷贝只是拷贝了地址，而深拷贝则是完全拷贝，并且拷贝后的对象和原对象没有任何关联。
- 使用递归实现深拷贝。
- 使用lodash插件中的deepclone来实现深拷贝。
- 使用JSON.stringfy()和JSON.parse()来实现。

```
let obj = {name:'zhangsan',age:18}
let newObj = JSON.parse(JSON.stringfy(obj))
```

什么是闭包？闭包的使用场景？

- 闭包就可以在全局函数里面操作另一个作用域的局部变量！
- 在函数外部可以访问函数内部的变量。
- 使用场景：一时半会想不到了
- 虽然可以保护私有变量，但用多了可能造成内存泄露。

说出几个数组去重的方法，说说哪个性能比较好

1. 利用数组的indexOf方法，新建一个空数组，循环遍历旧数组，判断空数组中是否有当前的元素，如果有就跳过，如果没有就执行push方法。

```
let arr = [1, 1, 2, 2, 3, 3, 4, 5];
      let newArr = [];
      arr.forEach(item => {
        if (newArr.indexOf(item) < 0) {
          newArr.push(item);
        }
      });
```

2. 利用数组的splice方法，先利用sort方法对数组进行排序，然后循环遍历数组，比较前一项与后一项是否相同，如果相同就执行spilce方法删除当前项。
3. 利用ES6中Set不能存放相同元素的特点，配合...展开运算符进行去重。

4. lodash插件也可以去重。

数组排序

```
/*冒泡排序*/
var arr1=[7,12,4,23,21,2,17,13,6,33]//处理的数组
for(var j=0;j<arr1.length;j++){
    for(var i=0;i<arr1.length;i++){
    if(arr1[i]>arr1[i+1]){
        var temp=arr1[i]
        arr1[i]=arr1[i+1]
        arr1[i+1]=temp
    }
  }
}
<!--选择排序-->
var arr1=[7,12,4,23,21,2,17,13,6,33]
var min=null,temp,index=1
for (var i=0;i<arr1.length;i++){
    min=arr1[i]
    for(var j=i+1;j<arr1.length;j++){
        if(arr1[j]<min){
            min=arr1[j]
            index=j
        }
    }
    temp=arr1[i]
    arr1[i]=min
    arr1[index]=temp
}
/*sort排序 */
arr.sort((a,b)=>{
return a-b;
})
```

数组中常见方法

- find 查找符合条件的项
- findIndex 查找符合条件项目的下标
- filter过滤数组，返回符合条件的项
- map遍历数组，映射并返回一个新数组
- forEach循环数组，无返回值
- reduce数组的累加方法
- splice截取数组
- concat数组合并
- join数组转字符串

js中的事件传播流程

- 捕获阶段-从外往里传播
- 事件源阶段-目标阶段
- 冒泡阶段-从里往外传播

对象的相关操作

- Object.keys获取所有属性的属性名，返回一个数组
- for...in遍历对象，获取属性名
- Object.assign对象的浅拷贝
- ...展开运算符
- 结构赋值

字符串的相关操作

- substr 从下标几开始，截取几位
- substring 闭区间，截取指定区间的字符串
- replace替换为指定的字符串
- replaceAll替换全部
- split分割成数组

es6中的新语法

- 箭头函数
- class类的继承
- const、let变量定义
- 模板字符串
- 结构赋值
- Promise对象
- 字面量简写（当属性值与属性名一样时）
- 拓展运算符
- import和export，之前是commonJS的规范

Promise是什么，它的API有哪些?

- Promise是es6新增的语法，用来处理异步请求，解决了es5中回调地狱的问题
- Promise的回调函数中接收两个参数，第一个表示请求成功的回调，第二个表示请求失败的回调，分别是resolve和reject。
- 使用.then（）去获取结果，使用.catch（）去捕获异常，并且上一个Promise的执行结果会返回一个Promise对象。
- Promise.all()和Promise.race()

介绍http请求报文

```
请求行
    请求行由请求方法、URL和HTTP协议版本3个字段
请求头（携带数据）
    cookies
    token
    content-type
请求体
    传输数据

数据传输的格式
    json        '{"name":"zhangsan"}'
    urlencoded  name=Tom&&age=18
    form-data   此格式数据常常用于文件上传
get和post的区别
    get请求参数明文传输，在url中传递
    post请求在请求体中传输
    post请求相对get请求更安全一些
```

```
    post请求传输的参数大小比get请求大
    get请求可以加入收藏夹
    get请求常常用于获取数据，请求速度快

get请求传递的参数为什么有大小限制？
    get请求在url中传递，不同浏览器对url长度有限制
```

dataset数据集?

- data-xxx表示数据集，选中元素，然后使用dataset来获取。

ajax如何做一个loading效果?

- 借助 jQuery 提供的 ajaxSetup() 方法来对 ajax 进行全局的设置。当请求开始时自动将loading 框显示出来，等请求结束后自动将其隐藏。
- beforeSend：发送请求前我们可以在此显示 loading框。
  complete：请求完成时会执行(不管成功失败)。我们可以在此隐藏loading框。

什么是虚拟dom？有什么优势?

- 不是真实的dom，是用js表述的一个dom结构
- 实际渲染的时候将js表述的这个dom结构渲染成真实的dom
- 优势：渲染效率高，修改结点数据时只做局部改变。
- 可以针对不同的终端执行不同的渲染函数

diff算法是什么?

- 虚拟dom改变时的节点替换算法

OSI七层模型?

OSI七层网络模型	TCP/IP四层概念模型	对应网络协议
应用层（Application）	应用层	HTTP、TFTP, FTP, NFS, WAIS、SMTP
表示层（Presentation）		Telnet, Rlogin, SNMP, Gopher
会话层（Session）		SMTP, DNS
传输层（Transport）	传输层	TCP, UDP
网络层（Network）	网络层	IP, ICMP, ARP, RARP, AKP, UUCP
数据链路层（Data Link）	数据链路层	FDDI, Ethernet, Arpanet, PDN, SLIP, PPP
物理层（Physical）		IEEE 802.1A, IEEE 802.2到IEEE 802.11

js的执行机制？微任务和宏任务

1. js是单线程的

所谓单线程就是一次只能执行一段代码，在执行的时候如果遇到比较耗时的操作，默认就会停下来等待这个操作完成之后继续走。这种情况下，就会出现页面卡在那里不动。为了解决这个问题js一般把比较耗时的操作做异步处理

2. js中的异步处理
   js中存在一个异步队列，所有比较耗时的操作都会被丢在这个异步队列中。当主线程空闲（同步代码）之后会执行异步队列中的代码,这个就是js中的eventloop(事件循环)

- 宏任务，是运行环境提供的异步操作,例如:setTimeout
- 微任务，是js语言自身的异步操作,例如:Promise
  在一个宏任务周期中会执行当前周期中的所有微任务，当所有的微任务都执行完成之后会进入下一个宏任务周期

CSS常见的布局方式?

- 流式布局，盒子自上而下排列。
- 弹性布局，flex弹性盒。
- grid网格布局。
- 自适应（响应式布局），使用rem单位。

介绍一下css的盒模型?

1. W3C 标准盒模型：
   属性width,height只包含内容content，不包含border和padding。
2. IE （怪异）盒模型：
   属性width,height包含border和padding，指的是content+padding+border。

介绍一下XSS攻击和CSRF?

- XSS（跨站脚本攻击），是一种代码注入攻击，是通过在网站中注入恶意代码达到劫持用户cookie或其他信息的一种攻击。
- 客户端：限制输入长度
- 上线前：使用扫描工具自动检测 XSS 漏洞

- CSRF（Cross-site request forgery）跨站请求伪造
- 服务端做一些防御操作

判断数据类型的几种方法?

- typeof
- instanceof
- Object.prototype.toString().call()

call/apply/bind区别和原理?

- call方法call()是apply()的一颗语法糖，作用和apply()一样，同样可实现继承，唯一的区别就在于call()接收的是参数列表，而apply()则接收参数数组。
- bind方法bind()的作用与call()和apply()一样，都是可以改变函数运行时上下文，区别是call()和apply()在调用函数之后会立即执行，而bind()方法调用并改变函数运行时上下文后，返回一个新的函数，供我们需要时再调用.

如何清除浮动?

- clear：both（给空元素添加）

- 使用伪类:after

```
:after{
    content:'';
    clear:both;
    width:0;
    height:0;
    visbility:hidden;
}
```

- 父元素添加overflow:hidden;

判断一个数据是不是数组?

- 使用Array.isArray()

```
let arr = [];
console.log(Array.isArray(arr))
//true
或者:
console.log(Object.prototype.toString.call(arr))
//[Object,Array]
```

防抖和节流?

- 防抖：在事件被触发n秒后再执行回调，如果在这n秒内又被触发，则重新计时（本质是一个定时器）。
- 例如：搜索框的联想功能，不断输入值的时候，使用防抖来节约资源。
- 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。

共同点:

- 函数防抖和函数节流都是防止某一时间频繁触发，但是这两兄弟之间的原理却不一样。
- 函数防抖是某一段时间内只执行一次，而函数节流是单位时间执行一次。

js中的设计模式?

- 工厂模式
- 单例模式
- 原型模式
- 组合模式
- 发布-订阅模式

浏览器缓存?

- 强制缓存
- 协商缓存
- 不是很了解。

js会不会阻塞dom的解析？css会吗?

- js会（js中有能够操纵dom的语句）
- css不会（css会和html同时加载）

跨域解决方案

- 通过jsonp跨域
- postMessage跨域
- 跨域资源共享（CORS）
- nginx代理跨域
- nodejs中间件代理跨域
- WebSocket协议跨域

JS中的继承?

原型链继承

```
//父类
function Super() {
  this.flag = true;
}
//给父类原型添加方法
Super.prototype.getFlag = function () {
  return this.flag;
};
//子类
function Sub() {
  this.subFlag = false;
}
//将父类的实例作为子类的原型
Sub.prototype = new Super();
//给子类添加子类特有的方法，注意顺序在继承之后
Sub.prototype.getSubFlag = function () {
  return this.sunFlag;
};
//构造实例，实现继承
var es5 = new Sub();
console.log(es5);

 /*优点：
  父级新增的属性和方法子级都能够访问的到
  缺点：
  就是多个子类共享一个原型的属性和方法，无法实现多个继承
 */
```

构造函数继承

```
#构造函数继承就是在子级构造函数里面执行父级的构造函数并且改变this的指向

  function Person () {
    this.name = 'zhangsan'
    this.age = 23
  }
  Person.prototype.count = 50
  function Son () {
    Person.apply(this)
    this.num = 18
  }

  var son = new Son()
```

```
console.log(son.name) // zhangsan
console.log(son.age)  // 23
console.log(son.count)// undefined

/*构造函数继承的优点：
  子级直接继承父级构造函数的属性和方法

  构造函数继承的缺点：
  子类无法继承父类原型链上的属性和方法
*/
```

混合继承

```
// 混合继承
  function Person () {
    this.name = 'zhangsan'
    this.age = 23
  }
  Person.prototype.count = 50
  function Son () {
    Person.apply(this)
    this.num = 18
  }
  Son.prototype = new Person()
  var son = new Son()

  console.log(son.name) // zhangsan
  console.log(son.age)  // 23
  console.log(son.count)// 50

  /*优点：
    集合了原型链继承和构造函数继承的优点集合在一起
  */
```

ESlass继承，constructor()和super()关键字。

- super()方法用来调用父类的构造函数。
- constructor是原型对象上的一个属性，默认指向这个原型的构造函数。

你的项目如何实现支付功能的?

网页上：

- 支付宝：后端返回我一个URL，我直接做跳转就行了。
- 微信：如果是小程序，调用微信的支付接口。

手机App：

- 支付宝：调用原生开发人员给我提供的方法就行了。
- 微信：使用微信小程序提供的支付接口。

你的项目如何获取位置信息?

- 网页上：使用百度地图做定位
- App中：调用原生开发人员提供的方法就行