**JS 面试题**

1. **Get 请求 传递参数长度的误区**

答：1) HTTP并未规定Get和Post所传递参数的最大长度

1. Get不允许传递太长的参数是因为浏览器限制了Url的最大长度。

不同的浏览器规定的最大长度不同。

1. **说一下get和post请求在缓存方面的区别**

答：1. Http缓存一般只适用于查询请求

2. Get一般作查询请求，不影响数据库的操作，所以一般都对get请求进行缓存

**3. 闭包**

答：1. 闭包就是一个能够读取其他函数内部变量的函数

2. 作用：缓冲数据，延长变量的作用域链

3. 常以两种形式出现：

函数作为返回值

函数作为参数传递

4. 可能会造成变量的泄露

**3. 如何创建一个类，请列出你知道的所有方法**

答：1. 通过自定义构造函数创建

2.使用es6 class关键字创建

**4. 如何创建一个类，请列出你知道的所有方法**

答：1. 通过自定义构造函数创建

2.使用es6 class关键字创建

3.通过工厂模式进行创建

**5. 如何实现类的继承，请列出你知道的所有方法.它们的优缺点分别是什么？**

答：1. 原型继承

关键代码：

Son.prototype = new Father

Son.prototype.constructor = Son

弊端: 1) 所有对象的原型属性都是一样的，无法按需更改

2.构造函数继承

关键代码：

Function Son(){

Father.call(this,args) //args是要传给父类用于为父类实例属性赋值的

}

弊端：所有的属性和方法都是实例的，无法进行函数的复用。

3. 组合继承

function SubType() {

SuperType.call(this,name)

}

SubType.prototype = new SuperType()

SubType.prototype.constructor = SubType

我们把公用的方法和不会修改的属性放在原型对象里，其他属性由实例自行定义

4. es6 中extends关键字继承

Class Father{}

Class Son extends Father{

constructor(){

super(args) // 必须执行一次super函数

}

}

**6. 如何解决回调地狱？**

答：1. 使用Promise

2. 接收多个异步请求的返回参数请使用

Promise.all

**7. 请说明以下前端的事件流**

答：事件流包括3个阶段

事件捕获阶段、处于目标阶段、事件冒泡阶段

通过事件对象的eventPhase 来查看属于什么阶段

**7. 如何让事件先冒泡后捕获？**  
根据w3c标准，应先捕获再冒泡。若要实现先冒泡后捕获，给一个元素绑定两个addEventListener，其中一个第三个参数设置为false（即冒泡），另一个第三个参数设置为true（即捕获），调整它们的代码顺序，将设置为false的监听事件放在设置为true的监听事件前面即可

**8. 说一下事件委托？**  
1.事件委托就是利用事件冒泡原理，把子元素的事件冒泡给父元素代为执行

2.这样做的好处有：避免了为众多子元素一一绑定事件而造成的消耗，动态创建的事件也自动绑定了响应的事件

**9. mouseover/mousemove/mouseenter 区别？**  
1.mouseover：当指针进入元素或其子元素时触发，只触发一次

2.mousemove ：指针在元素内移动时重复触发

3.mouseenter: 当指针进入元素(不包括其子元素)时触发

**10. js的new操作符做了哪些工作？**  
1.创建一个空对象

2.将this设置为当前对象

3.该对象继承该函数的原型

4.把属性和方法加入到this引用的对象中

5.新创建的对象由this引用，最后隐式地返回this。

**10. 请阐述bind/call/apply的使用方法，作用及区别**  
1.作用都是改变函数内部的this的指向的

2.bind和call,apply的区别是，前者是返回一个修改过this指针的函数供用户之后使用

后者均为立刻调用并返回结果

3.bind/call:function.bind/call(this,args)

apply:function.apply(this,argsArr)

**10. 如何异步加载JS？**  
1. 动态创建script标签，在window.onload事件触发后执行，否则依然会造成页面阻塞

2. script 标签添加defer属性，仅支持ie

3.script 标签 添加async属性  async="async"：脚本相对于页面的其余部分异步地执行（当页面继续进行解析时，脚本将被执行），因此被执行的脚本不要影响页面结构

**11. ==/===/Object.is()的区别是什么**  
1. == 会进行默认类型转换

2. === /Object.is() 不会进行默认的类型转换

3.===/Object.is()区别在于===会将+0等于-0,认为NaN不等于NaN，Object.is则不会

**11. 数组去重有什么方法？**

1.使用es6的Set类型：

Array.from(new Set(arr))

**12. ES6的暂时性死区能讲讲吗？**

但我们在块级作用域内用let声明一个变量时，就已经把这个变量绑定到了当前作用域。由于不存在变量提升，所以当我们在给其声明之前，该变量均处于不可用的状态，直至我们声明它后。

**13. 实现一个简单的深拷贝**

1.通过json对象进行深拷贝

let copy = JSON.parse(JSON.stringify(obj))

2.递归复制

//使用递归的方式实现数组、对象的深拷贝

//使用递归的方式实现数组、对象的深拷贝

function deepClone(obj) {

//判断拷贝的要进行深拷贝的是数组还是对象，是数组的话进行数组拷贝，对象的话进行对象拷贝

let objClone = Array.isArray(obj) ? [] : {};

//进行深拷贝的不能为空，并且是对象或者是

if (obj && typeof obj === "object") {

for (let key in obj) {

if (obj.hasOwnProperty(key)) {

if (typeof obj[key] === "object") {

objClone[key] = deepClone(obj[key]);

} else {

objClone[key] = obj[key];

}

}

}

}

return objClone;

}

**14. 说一下scrollLeft/offsetLeft/clientLeft 以及 scrollHeight/offsetHeight/clientHeight的区别**

答：1.

offsetLeft:元素的边框的外边缘距离与已定位的父容器（offsetparent）的左边距离（不包括元素的边框和父容器的边框）

附：offsetparent的确定：

元素自身有fixed定位，offsetParent的结果为null

元素自身无fixed定位，且父级元素都未经过定位，offsetParent的结果为<body>

元素自身无fixed定位，且父级元素存在经过定位的元素，offsetParent的结果为离自身元素最近的经过定位的父级元素

<body>元素的parentNode是null

clientLeft:元素的内边距的外边缘和边框的外边缘的距离，实际就是边框的左边框宽度

scrollLeft：水平滚动条左边卷去的宽度

2.

offsetHeight: 描述元素外尺寸高度，是指 元素内容高度+内边距高度(上下两个)+边框(上下两个)，不包括外边距和滚动条部分。

clientHeight:用于描述元素内尺寸高度，是指 元素内容+内边距 大小，不包括边框、外边距、滚动条部分

scrollHeight:元素的实际高度，包括被卷起来的部分

**15. JS如何实现元素的拖动？**

window.onload = function () {

let isDrag = false

let dragBox = document.querySelector('.drag')

let innerX

let innerY

dragBox.onmousedown = function (e) {

e = e || window.event

isDrag = true

innerX = e.clientX - this.offsetLeft

innerY = e.clientY - this.offsetTop

}

dragBox.onmouseup = function () {

isDrag = false

}

document.onmousemove = function (e) {

if (!isDrag) return

e = e || window.event

let windowWidth = window.innerWidth

let windowHeight = window.innerHeight

console.log(windowWidth + '==' + windowHeight)

let left = e.clientX - innerX

let top = e.clientY - innerY

if (left < 0) {

left = 0

} else if (left > windowWidth - dragBox.offsetWidth) {

left = windowWidth - dragBox.offsetWidth

}

if (top < 0) {

top = 0

} else if (top > windowHeight - dragBox.offsetHeight) {

top = windowHeight - dragBox.offsetHeight

}

dragBox.style.left = left + 'px'

dragBox.style.top = top + 'px'

}

}

**16 Ajax解决浏览器缓存问题**

1.设置请求头

xhr.setRequestHeader("If-Modified-Since","0")

xhr.setRequestHeader("Cache-Control","no-cache")

2.在url后添加一个随机数或时间戳的查询字符串

**17. 说一下Commonjs、AMD和CMD？**

1.为了在js中方便进行模块化和使用其进行服务端开发，出现了commonjs。

Nodejs就是基于其开发的。

2.AMD:客户端也想用模块化开发的概念。但是不可以直接使用commonjs。所以使用AMD创建适合前端的模块化规范，使得AMD可以异步加载模块

3、CMD作用与AMD类似，但是AMD是依赖前置，而CMD是依赖就近

**18. 说一下eval？**

eval就是能够解析并执行执行其内部js代码字符串的命令。在严格模式下，它不可以访问命令外的变量。

关于它内部的this指向：仅当直接使用eval()时内部才会指向当前对象

**19.JS中的垃圾回收机制？**

1.JS采用的垃圾回收机制有两种：标记清除和引用计数

2.最常见的是标记清除

3.大部分浏览器以标记清除方式进行垃圾回收，当变量进入执行环境（函数中声明变量）的时候，垃圾回收器将其标记为“进入环境”，当变量离开环境的时候（函数执行结束）将其标记为“离开环境”，在离开环境之后还有的变量则是需要被删除的变量。

**20.如果已经有三个promise，A、B和C，想串行执行，该怎么写？**

**21. 如何实现一个私有变量，用getName方法可以访问，不能直接访问？**

可以使用es6中的Proxy进行代理

let user = {

`name:’cc’

}

Let p = new Proxy(user,{

get:function(){

return user[name

}

})

**22.请自己实现一个bind函数**

Function.prototype.bind=function(newThis){

//接收默认参数

let args = Array.prototype.slice.call(arguments,1)

let that = this //保存this指针

return function(){

return that.apply(newthis,Array.prototype.concat.call(args,arguments))

}

}

**23.你了解函数柯里化吗？**

见jsstudy

**24.数组扁平化**

function flat(originArr) {

let res = []

const makeFlat = function (arr) {

for(let val of arr.values()){

if(Array.isArray(val)){

makeFlat(val)

}else{

res.push(val)

}

}

}

makeFlat(originArr)

return res

}

**25.用setTimeout来实现setInterval**

function mySetInterval(interval, fn) {

const args = Array.prototype.slice.call(arguments, 2)

setTimeout(function () {

fn(...args)

mySetInterval(interval, fn, ...args)

}, interval)

}

function mySetInterval(interval, fn) {

let times = 0

const args = Array.prototype.slice.call(arguments, 2)

const begin = function (interval, fn) {

setTimeout(function () {

fn(...args)

console.log(++times)

begin(interval, fn, ...args)

}, interval)

}

begin(interval, fn)

}

**26.原生js实现promise**

见jsstudy

**27.如何实现sleep的效果（es5/es6）**

见jsstudy

**28.请你实现一个简单的Vue**

但我们在块级作用域内用let声明一个变量时，就已经把这个变量绑定到了当前作用域。由于不存在变量提升，所以当我们在给其声明之前，该变量均处于不可用的状态，直至我们声明它后。

**29.JS如何实现超大整数的相加？**

但我们在块级作用域内用let声明一个变量时，就已经把这个变量绑定到了当前作用域。由于不存在变量提升，所以当我们在给其声明之前，该变量均处于不可用的状态，直至我们声明它后。

**30.请说说js的单线程和事件队列机制？**

但我们在块级作用域内用let声明一个变量时，就已经把这个变量绑定到了当前作用域。由于不存在变量提升，所以当我们在给其声明之前，该变量均处于不可用的状态，直至我们声明它后。

**前端优化专题**

**1. 图片如何进行懒加载和预加载？**

答：

**2. 谈一下节流与防抖，请你具体地实现一下**

答：

**2. 谈一下重排和重绘，它们的关系和区别是什么？**

答：首先讲一下浏览器非工作机制：

1、构建DOM树（parse）：渲染引擎解析HTML文档，首先将标签转换成DOM树中的DOM node（包括js生成的标签）生成内容树（Content Tree/DOM Tree）；

2、构建渲染树（construct）：解析对应的CSS样式文件信息（包括js生成的样式和外部css文件），而这些文件信息以及HTML中可见的指令（如<b></b>），构建渲染树（Rendering Tree/Frame Tree）；

3、布局渲染树（reflow/layout）：从根节点递归调用，计算每一个元素的大小、位置等，给出每个节点所应该在屏幕上出现的精确坐标；

4、绘制渲染树（paint/repaint）：遍历渲染树，使用UI后端层来绘制每个节点。

**重绘**是指一个元素外观的改变所触发的浏览器行为，浏览器会根据元素的新属性重新绘制，使元素呈现新的外观。

**触发重绘的条件：改变元素外观属性。如：color，background-color等。**

重绘（回流）：渲染树中的一部分(或全部)因为元素的规模尺寸，布局，隐藏等改变而需要重新构建, 这就称为回流(reflow)。每个页面至少需要一次回流，就是在页面第一次加载的时候。

触发重排的条件：任何页面布局和几何属性的改变都会触发重排，比如：

　　1、页面渲染初始化；(无法避免)

　　2、添加或删除可见的DOM元素；

　　3、元素位置的改变，或者使用动画；

　　4、元素尺寸的改变——大小，外边距，边框；

　　5、浏览器窗口尺寸的变化（resize事件发生时）；

　　6、填充内容的改变，比如文本的改变或图片大小改变而引起的计算值宽度和高度的改变；

　　7、读取某些元素属性：（offsetLeft/Top/Height/Width,　clientTop/Left/Width/Height,　scrollTop/Left/Width/Height,　width/height,　getComputedStyle(),　currentStyle(IE)　)

重排必定会引发重绘，但重绘不一定会引发重排。

重绘和重排的开销是非常昂贵的，如果我们不停的在改变页面的布局，就会造成浏览器耗费大量的开销在进行页面的计算，这样的话，我们页面在用户使用起来，就会出现明显的卡顿。

如何避免不必要的开销，提高性能？

首先，浏览器会维护1个队列，把所有会引起回流、重绘的操作放入这个队列，等队列中的操作到了一定的数量或者到了一定的时间间隔，浏览器就会flush队列，进行一个批处理。这样就会让多次的回流、重绘变成一次回流重绘。

1）合并DOM操作。避免不必要的修改。

2) 变修改样式为修改类名

3)将需要多次重排的元素，position属性设为absolute或fixed，元素脱离了文档流，它的变化不会影响到其他元素

4）使用cloneNode(true or false) 和 replaceChild 技术，引发一次回流和重绘，用于代替删除及添加