# js

## JS数据类型：

基本数据类型：Number、String、Boolean、Null、 Undefined、Symbol（ES6），这些类型可以直接操作保存在变量中的实际值。

引用数据类型：Object（在JS中除了基本数据类型以外的都是对象，数据是对象，函数是对象，正则表达式是对象）。

基本数据类型是指存放在栈中的简单数据段，数据大小确定，内存空间大小分配，它可以们是直接按值存放的，所以可以直接按值访问。

引用数据类型也叫对象数据类型，包括function,object,array,date,RegExp等可以可以使用new创建的数据，又叫对象类型，他们是存放在堆(heap)内存中的数据。

## JS中的作用域：

在ES5中，没有块级作用域，JS的作用域分为函数作用域和全局作用域两种。内部环境可以通 过作用域链访问所有的外部环境，但是外部环境不可以访问内部环境中的任何变量和函数。这些环境的访问顺序是线性有次序的。在ES6中，新增了let和const，这两个类型的变量只存在于块级作用域中，并且不存在变量提升。

## 谈谈原型和原型链：

原型：每个函数都有 prototype 属性，该属性指向原型对象；使用原型对象的好处是所有对象实例共享它所包含的属性和方法。

原型链：主要解决了继承的问题；每个对象都拥有一个原型对象，通过proto 指针指向其原型对象，并从中继承方法和属性，同时原型对象也可能拥有原型，这样一层一层，最终指向 null。

## 重绘和回流

**什么是重绘？**

由于几何属或样式发生改变不影响布局而需要重新渲染的过程，就叫重绘。

**什么是回流？**

由于布局发生改变需要重建就叫做回流

**什么原因会导致回流**

添加或者删除可见的\*\*DOM元素；\*\*

元素位置改变；

元素尺寸改变\*\*——边距、填充、边框、宽度和高度\*\*

页面渲染初始化；

浏览器窗口尺寸改变\*\*——resize事件发生时；\*\*

回流必将引起重绘，而重绘不一定会引起回流。

## JS中判断基本数据类型的方法

**1.typeof：**

typeof'';// string 有效

typeof 1;// number 有效 typeof Symbol();// symbol 有效

typeof true;//boolean 有效 typeof undefined;//undefined 有效

typeof null;//object 无效 typeof [] ;//object 无效

typeof new Function();// function 有效 typeof new Date();//object 无效

typeof new RegExp();//object 无效

**2.instanceof**

instanceof 是用来判断 A 是否为 B 的实例，表达式为：A instanceof B，如果 A 是 B 的实例，则返回 true,否则返回 false。

在这里需要特别注意的是：instanceof 检测的是原型 3.constructor 当一个函数 F被定义时，JS引擎会为F添加 prototype 原型，然后再在 prototype上添加一个 constructor 属性 4.toString toString() 是 Object 的原型方法，调用该方法，默认返回当前对象的 [[Class]] 。这是一个内部属性，其格式为 [object Xxx] ，其中 Xxx 就是对象的类型。

在浏览器中输入url发生了什么？

第一步：浏览器查找域名的IP地址——DNS解析。

第二步：与服务器建立连接。

第三步：发送HTTP请求：http协议，HTTP请求报文。

第四步：服务器处理请求并返回HTTP报文。

第五步：浏览器解析渲染页面。

第六步：断开TCP连接

## 数组去重：

1: 思路：定义一个新数组，并存放原数组的第一个元素，然后将元素组一一和新数组的元素对比，若不同则存放在新数组中。Methods

2: 思路：先将原数组排序，在与相邻的进行比较，如果不同则存入新数组。Methods

3: 利用对象属性存在的特性，如果没有该属性则存入新数组。

4: 利用数组的indexOf下标属性来查询。Methods

5: 利用数组原型对象上的includes方法。Methods

6: 利用 ES6的set 方法。

## 深浅拷贝以及深浅拷：

浅拷贝只复制指向某个对象的指针而不复制对象本身，新旧对象还是共享同一块内存。但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 （在某些引用类型值不更新的情况下用深拷贝）

## 深浅拷贝如何实现

浅拷贝：1.通过拷贝构造方法实现浅拷贝

2.通过重写clone（）方法进行浅拷贝

深拷贝：1.通过重写clone（）方法进行深拷贝

2.通过对象序列化实现深拷贝

## 箭头函数和普通函数的区别

1.相比普通函数更简洁的语法

2.没有this,捕获其所在上下文的 this 值，作为自己的 this 值

3.不能使用new,箭头函数作为匿名函数,是不能作为构造函数的,不能使用new

4.不绑定arguments，用rest参数...解决 let test3=(...a)=>{console.log(a[1])} //22

5.使用call()和apply()调用:由于 this 已经在词法层面完成了绑定，通过 call() 或 apply() 方法调用一个函数时，只是传入了参数而已，对 this 并没有什么影响：

6.箭头函数没有原型属性

7.不能简单返回对象字面量 let fun5 = ()=>({ foo: x }) //如果x => { foo: x } //则语法出错

8.箭头函数不能当做Generator函数,不能使用yield关键字

9.箭头函数不能换行

## 防抖节流

防抖节流就是使用定时器来实现我们的目的。

防抖(debounce)：

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。

典型的案例就是输入框搜索：输入结束后n秒才进行搜索请求，n秒内又输入的内容，则重新计时。

节流(throttle)：

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。

典型的案例就是鼠标不断点击触发，规定在n秒内多次点击只生效一次。

## 为什么要掌握防抖和节流

函数节流与函数防抖都是可以限制函数的执行频次，根据不同的场景来对

执行频率进行限制，避免了函数触发频率过高导致的响应速度跟不上触发频率，出现延迟，假死或卡顿的现象。

## 数组添加删除的方法？

unshift,push ; pop,shift，splice

## for和forEach的区别？

forEach不能用break退出循环

## 数组去重

1.双层循环，外层循环元素，内层循环时比较值，如果有相同的值则跳过，不相同则push进数组

2.双层循环，外层循环元素，内层循环时比较值，值相同时，则删去这个值，注意点:删除元素之后，需要将数组的长度也减1.

3.利用对象的属性不能相同的特点进行去重

4.组递归去重：运用递归的思想，先排序，然后从最后开始比较，遇到相同，则删除

5.利用indexOf以及forEach

6.利用ES6的set：Set数据结构，它类似于数组，其成员的值都是唯一的。利用Array.from将Set结构转换成数组

## 判断对象是否为数组或者函数的方法

方法一： instanceof: var arr=[]; console.log(arr instanceof Array) //返回true

方法二： constructor: console.log(arr.constructor == Array); //返回true

方法三： Array.isArray() console.log(Array.isArray(arr)); //返回true

## http和https的区别

HTTP的URL由 http://起始且默认使用端口80，而HTTPS的URL由https://起始且默认使用端口443

HTTP是超文本传输协议，信息是明文传输，HTTPS则是具有安全性的 SSL 加密传输协议

HTTP的连接很简单，是无状态的，HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，比 http 协议安全

## http优化

1.减少DNS查找：每次主机名的解析都需要一次网络往返，从而增加了请求的延迟时间，同时还会阻塞后续的请求

2.重用TCP连接，尽可能的使用持久连接，以消除因TCP握手和慢启动导致的延迟

3.减少http的重定向

## 跨域问题如何解决

CORS

服务器代理

nginx

jsonp

## 跨域解决方法

1、JSONP 使用简单且兼容性不错，但是只限于 get 请求。利用 src 不受同源策略的影响 => 利用 script 标签会把所有内容当作 JS 代码来执行 => 我们就绕开了浏览器同源策略的影响

2、CORS(跨域资源共享) 需要浏览器和后端同时支持。IE 8 和 9 需要通过 XDomainRequest 来实现。

3、服务器代理(常用) 切换到 nginx 服务器, 使用 nginx 服务器进行代理配置 -> 配置好以后, 请求的时候只要请求代理标识符就可以了

## 跨域共享cookie的方法：

设置cookie.setDomain(".aabb.com");

场景如下：

A机所在的域：home.abc.com,A有应用web\_a

B机所在的域：map.aabb.com，B有应用web\_b

1 在web\_a下面设置cookie的时候，增加cookie.setDomain(".aabb.com");这样在web\_b下面就可以取到cookie。

2 注意这个参数必须以“.”开始。

3 输入url访问web\_b的时候，必须输入域名才能解析。

## 回调地狱

多个回调函数嵌套出现的代码书写结构叫做 回调地狱 + 代码的阅读性和维护性要低了, 代码书写的不是不对, 是不好

## 什么是路由，路由的两种模式

Hash和history区别

Hash:

Url：有#，很Low

回车刷新：可以加载到hash值对应页面

支持版本：支持低版本浏览器和IE浏览器

History：

Url：无#，好看

回车刷新：一般就是404掉了

支持版本：HTML5新推出的API

## get post区别

1、post是通过HTTP post机制，⽤户看不到这个过程 。如果想要看到数据可以从控制台NetWork中的form Data中进⾏查看。

2、get进⾏数据请求的时候会将传递的参数信息通过url进⾏传递。在地址的？后⾯按照key=val进⾏传递如果需要传递多个数据的时候⽤&符进⾏分隔

3、 get传送的数据量较⼩，不能⼤于2KB。post传送的数据量较⼤，⼀般被默认为不受限制。

4、 get安全性⾮常低，post安全性相⽐较get来说较⾼。但是执⾏效率却⽐Post⽅法好。get与post的安全性取决于http协议或者https协议

## 了解浏览器缓存嘛？ 说一下具体有什么区别

缓存分为强缓存和协商缓存。

强缓存不过服务器，协商缓存需要过服务器，协商缓存返回的状态码是304。两类缓存机制可以同时存在，强缓存的优先级高于协商缓存。当执行强缓存时，如若缓存命中，则直接使用缓存数据库中的数据，不再进行缓存协商。

## 什么是闭包，在业务中什么情况下会用到闭包

当一个函数内部嵌套另一个函数，子函数可以访问到父级函数中的变量，就

是闭包。严格来说，闭包产生的三个条件

1.必须是嵌套关系

2.内部函数访问其所在的作用域

3.在所在作用域外被调用

闭包的作用与应用场景

作用一 使变量在内存中保存不被销毁。

作用二，延长了函数内局部变量的作用范围。

优缺点：不销毁的话占用内存内存泄漏不能多用

## 闭包的理解

全局变量生存周期是永久，局部变量生存周期随着函数的调用介绍而销毁。 闭包就是 在函数中定义且成为该函数内部返回的函数的自由变量 的变量，该变量不会随着外部函数调用结束而销毁。 （注：不光是变量，函数内声明的函数也可以形成闭包） 当函数可以记住并访问所在的词法作用域，即使函数是在当前词法作用域之外执行，这时就产生了闭包。

## 继承有哪些方式

1、call 、apply:不建议使⽤浪费内存

2、原型对象继承

3、原型拷⻉继承

4、原型链继承

5、混合继承

6、继承继承

7、ES6 类继承

## 第二次打开网页为什么比第一次快？

主要是因为第一次加载页面过程中，缓存了一些数据，之后再加载就直接从缓存中获取而不用请求服务器，所以速度更快，也减轻了服务器的压力

## 内存泄漏

内存泄漏（Memory Leak）是指程序中已动态分配的堆内存由于某种原因程序未释放或无法释放，造成系统内存的浪费，导致程序运行速度减慢甚至系统崩溃等严重后果。

1、意外声明全局变量是最常见也最容易修复的内存泄漏问题，比如：

function fn() {

name = '张三';

}

解释器在解释上面的函数时，会把name当做全局变量，即window.name = '张三'。只要window对象没有被清理，那么name属性和属性值将一直存在，造成内存泄露。

解决方法：

(1)只要在变量声明前面加上var、let或const关键字即可，这样变量就会在函数执行完毕后离开作用域。

(2)使用this关键字

function fn() {

this.name = '张三';

}

(3)可以在 JavaScript 文件开头添加 “use strict”，使用严格模式。这样在严格模式下解析 JavaScript 可以防止意外的全局变量

(4)在使用完之后，对其赋值为null或者重新分配

2、 定时器导致的泄露

let name = '张三';

setInterval(() => {

console.log(name);

}, 100);

上面的代码中，只要定时器一直运行，回调函数中引用的name就会一直占用内存。

3、闭包、控制台日志、循环(在两个对象彼此引用且彼此保留时，就会产生一个循环),下面我们看一个JavaScript闭包导致的内训泄露例子

let fun = function() {

let name = '张三';

return function() {

return name;

};

};

调用fun()会导致分配给name的内存被泄漏。以上代码执行后创建了一个内部闭包，只要返回的函数存在就不能清理name，因为闭包一直在引用着它。

**箭头函数和普通函数的区别**

1、外形不同：箭头函数使用箭头定义，普通函数中没有。

2、 箭头函数全都是匿名函数：普通函数可以有匿名函数，也可以有具名函数

3、箭头函数不能用于构造函数：普通函数可以用于构造函数，以此创建对象实例。

4、箭头函数中 this 的指向不同：在普通函数中，this 总是指向调用它的对象，如果用作构造函数，它指向创建的对象实例。

5、箭头函数不具有 arguments 对象：每一个普通函数调用后都具有一个

arguments 对象，用来存储实际传递的参数。但是箭头函数并没有此对象。

6、其他区别：箭头函数不具有 prototype 原型对象。箭头函数不具有 super。

箭头函数不具有 new.target

**bom和dom**

1.BOM是浏览器对象模型提供了独立于内容而与浏览器窗口进行交互的对象。描述了与浏览器进行交互的方法和接口，可以对浏览器窗口进行访问和操作，譬如可以弹出新的窗口，改变状态栏中的文本，对Cookie的支持，IE还扩展了BOM，加入了ActiveXObject类，可以通过js脚本实例化ActiveX对象等等）

2.DOM是文档对象模型DOM是针对XML的基于树的API。描述了处理网页内容的方法和接口，是HTML和XML的API，DOM把整个页面规划成由节点层级构成的文档。DOM本身是与语言无关的API，它并不与Java，JavaScript或其他语言绑定。

## 宏任务和微任务

1.宏任务：当前调用栈中执行的代码成为宏任务。（主代码快，定时器等等）。

2.微任务： 当前（此次事件循环中）宏任务执行完，在下一个宏任务开始之前需要执行的任务,可以理解为[回调](https://so.csdn.net/so/search?q=%E5%9B%9E%E8%B0%83&spm=1001.2101.3001.7020" \t "https://note.youdao.com/newEditorV1/_blank)事件。（promise.then，proness.nextTick等等）

## 虚拟DOM

虚拟dom原理：

1. 为什么需要虚拟DOM

先介绍浏览器加载一个网页需要经历那些过程；我们只讨论页面解析流程，不考虑网络请求过程。

浏览器内核拿到html文件后，大致分为一下5个步骤：

2.解析html元素，构建dom 树

3.解析CSS，生成页面css规则树（Style Rules）

4.将dom树 和 css规则树关联起来，生成render树

5.布局（layout/ reflow），浏览器会为Render树上的每个节点确定在屏幕上的尺寸、位置

6.绘制Render树，绘制页面像素信息到屏幕上，这个过程叫paint

当你用原生js 或jquery等库去操作DOM时，浏览器会从构建DOM树开始讲整个流程执行一遍，所以频繁操作DOM会引起不需要的计算，导致页面卡顿，影响用户体验。而Virtual DOM能很好的解决这个问题。它用javascript对象表示virtual node(VNode)，根据VNode 计算出真实DOM需要做的最小变动，然后再操作真实DOM节点，提高渲染效率。

## 改变this指向的方法

1. call （抗~）方法

语法：函数名.call(调用者, 参数1, …)

作用：函数被借用时，会立即执行，并且函数体内的this会指向借用者或调用者

结果：普通函数的this指向window，利用call方法将this指向了obj

2. apply（俺普拉唉）方法

语法：函数名.apply(调用者, [参数, …])

作用：函数被借用时，会立即执行，并且函数体内的this会指向借用者或调用者

结果：普通函数的this指向window，利用apply方法将this指向了obj

3. bind（白的）方法

语法：函数名.bind(调用者, 参数, …)

作用：函数被借用时，不会立即执行，而是返回一个新的函数。需要自己手动调用新的函数来改变this指向

结果：普通函数的this指向window，利用bind方法将this指向了obj

相同点： 三者都可以把一个函数应用到其他对象身上，注意不是自身对象

不同点：

○ call，apply是直接执行函数调用。bind是绑定，执行需要再次调用。

○ call，bind接收逗号分隔的无限个参数列表；apply接收数组作为参数。