# JavaScript面试题

### 1 js数据类型

基本数据类型：number string Boolean null undefined

引用数据类型：Object Function Array

区别：

值类型：

1、占用空间固定，保存在栈中

2、保存与复制的是值本身

 3、使用typeof检测数据的类型

4、基本类型数据是值类型

引用类型：

1、占用空间不固定，保存在堆中

 2、保存与复制的是指向对象的一个指针

3、使用instanceof检测数据类型

4、使用new()方法构造出的对象是引用型

### ==和===的区别：

“==”：叫相等运算符

“===”：叫严格运算符

它们两有什么区别呢？

　　== ：表示值相等则都为true

　　===：表示 不仅要值相等，双方类型也要相等才为true

### 2 什么是原型

定义：一个对象，也称为原型对象，被构造函数所拥有。在JavaScript中，我们创建一个函数A(就是声明一个函数), 那么浏览器就会在内存中创建一个对象B，而且每个函数都默认会有一个属性 prototype 指向了这个对象( 即：prototype的属性的值是这个对象 )。这个对象B就是函数A的原型对象，简称函数的原型。这个原型对象B 默认会有一个属性 constructor 指向了这个函数A

### 3 什么是原型链及作用

每个对象都可以有一个原型\_proto\_，这个原型还可以有它自己的原型，以此类推，形成一个原型链。查找特定属性的时候，我们先去这个对象里去找，如果没有的话就去它的原型对象里面去，如果还是没有的话再去向原型对象的原型对象里去寻找...... 这个操作被委托在整个原型链上，这个就是我们说的原型链了。

\_\_proto\_\_ 是原型链查询中实际用到的，它总是指向 prototype，换句话说就是指向构造函数的原型对象，它是对象独有的

原型链指向：构造函数创建的对象实例 → 函数.prototype → Object.protype → null ;

每个函数都有一个原型对象，该原型对象有一个constructor属性，指向创建对象的函数本身。

**主要作用是继承**

### 4 闭包

闭包就是指有权访问另一个函数作用域中的变量的函数。

**闭包的作用**：延伸了变量的作用范围，外部函数调用之后其变量本应该被销毁，但闭包的存在使我们仍然可以访问外部函数的变量对象，这就是闭包的重要概念

**闭包的缺陷：**闭包的缺点就是常驻内存会增大内存使用量，并且使用不当很容易造成内存泄露。如果不是因为某些特殊任务而需要闭包，在没有必要的情况下，在其它函数中创建函数是不明智的，因为闭包对脚本性能具有负面影响，包括处理速度和内存消耗

闭包会造成内存泄漏，就是因为它把外部的包含它的函数的活动对象也包含在自己的作用域链中了，会比一般的函数占用更多内存。

闭包案例：

循环注册点击事件：

一个ul里有四个li，需要点击li获取当前li的索引号，先将这些li全部取出来，放到一个数组里面，然后通过for循环，在for循环里面加上立即执行函数，在立即执行函数里为每一个li绑定点击事件打印索引号

**作用域链**

当代码在一个环境中执行时，会创建变量对象的一个作用域链（作用域形成的链条）

* 作用域链的前端，始终都是当前执行的代码所在环境的变量对象
* 作用域链中的下一个对象来自于外部环境，而再下一个变量对象则来自下一个外部环境，一直到全局执行环境
* 全局执行环境的变量对象始终都是作用域链上的最后一个对象

**内部环境可以通过作用域链访问所有外部环境，但外部环境不能访问内部环境的任何变量和函数。**

### 5 split() join() 的区别

前者是将字符串切割成数组的形式，后者是将数组转换成字符串

### 6 js中的new做了什么

1 创建一个新对象

2 将新对象的\_proto\_指向构造函数的prototype对象

3 将构造函数的作用域赋值给新对象 （也就是this指向新对象）

4 执行构造函数中的代码（为这个新对象添加属性）

5 返回新的对象

### 7 DOM事件的级别和DOM事件模型

· DOM0: element.onclick=function(){}

· DOM2: element.addEventListener('click',function(){},false)(true表示在捕获阶段触发，false表示在冒泡阶段触发，默认是false)；

· DOM3: element.addEventListener('keyup',function(){},false)(true表示在捕获阶段触发，false表示在冒泡阶段触发，默认是false)，增加了很多事件类型；  
  
 事件捕获：从上往下找到目标元素

事件冒泡：从目标元素往上查找

### 8 DOM事件流

浏览器为当前页面和用户做交互的过程中发生的过程，一个完整的事件流分三个阶段。

1 事件捕获

2 目标阶段

3 事件冒泡

事件通过捕获到达目标元素，这个时候就是目标阶段，第三个阶段就是从目标元素上传到window对象。

### 9 描述DOM事件捕获的具体流程

window --> document --> html(用js表示html节点的方法：document.documentElement;) --> body --> ··· --> 目标元素

### 10 Event对象的常见应用

· event.preventDefault()：阻止默认行为；

· event.stopPropation()：阻止事件冒泡；

· event.stopImmeidiatePropagation()：阻止其他事件（若一个按钮绑定了两个或多个事件：事件a、事件b、……。当下场景想要实现点击按钮的时候只执行事件a，其他事件不执行，则在事件a的函数中加上event.stopImmeidiatePropagation()就能实现）；

· event.currentTarget：当前所被绑定的事件；

· event.target：当前被点击的元素（早期的ie版本不支持，早期ie用event.srcElement）；

### 11 call apply bind总结

相同点：都可以改变函数内部this指向

区别点：1 call和apply会调用函数，并且改变函数内部this指向，bind不会调用函数

2 call和apply传递参数不一样，call可以传递多个参数，apply只能传递两个，所以第二个参数必须是数组形式

主要应用场景：

1 call经常做继承

2 apply经常跟数组有关系

3 bind不调用函数，还想改变this指向

**Call比apply快：call 方法的参数格式正是内部方法所需要的格式**

### 12 继承

### 13 数组去重的几种方法

<https://www.cnblogs.com/zsp-1064239893/p/11196501.html>

### 14 深浅拷贝

浅拷贝 shadow copy   
 仅仅只拷贝对象自身，内部元素引用的其他对象只拷贝一个引用。

深拷贝 deep copy   
 外围和内部元素都拷贝了对象本身，而不是引用。因此深拷贝产生的副本可以随意修改，而不用担心会影响原始值。

**深拷贝**：

**var** obj = {

            id: 1,

            name: 'zhangsan',

            sex: {

                age: 18,

                sex: 'nan'

            },

            color: ['pink', 'red']

        };

**var** o = {};

**function** deepCopy(newobj, oldobj) {

            for (**var** k in oldobj) {

**var** item = oldobj[k];

                if (item instanceof Array) {

                    newobj[k] = [];

                    deepCopy(newobj[k], item)

                } else if (item instanceof Object) {

                    newobj[k] = {};

                    deepCopy(newobj[k], item)

                } else {

                    newobj[k] = item;

                }

            }

        }

        deepCopy(o, obj);

        console.log(o);

**深拷贝：**  
 通过利用JSON.parse(JSON.stringify(Object))来达到深拷贝的目的， 就是利用JSON.stringify 将js对象序列化（JSON字符串），再使用JSON.parse来反序列化(还原)js对象；序列化的作用是存储。很多问题：

如果obj里面有时间对象，则JSON.stringify后再JSON.parse的结果，时间将只是字符串的形式。而不是时间对象；

如果obj里有RegExp、Error对象，则序列化的结果将只得到空对象；

如果obj里有函数，undefined，则序列化的结果会把函数或 undefined丢失；

如果obj中的对象是有构造函数生成的， 则使用JSON.parse(JSON.stringify(obj))深拷贝后，会丢弃对象的constructor；  
如有需要可以自己手动封装函数来达到目的，  
**浅拷贝：** 通过ES6新特性Object.assign()与扩展运算符来达到浅拷贝的目的

### 15 let和const

Let： （1）使用let关键字声明的变量具有块级作用域

1. 必须先声明再使用，不存在变量提升
2. 暂时性死区：在代码块内，使用let命令声明变量之前，该变量都是不可用的。这在语法上，称为“暂时性死区”
3. let声明的全局变量不是全局对象的属性。不可以通过window.变量名的方式访问

Const: （1）具有块级作用域

1. 声明变量时必须赋值
2. 常量赋值后值不能更改

### 16 js判断数据类型

**（1）Typeof：**number, boolean, string, undefined, object, function,symbol. typeof null 返回类型错误，返回object，引用类型，除了function返回function类型外，其他均返回object。

**（2）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。

对于 Object 对象，直接调用 toString()  就能返回 [object Object] 。而对于其他对象，则需要通过 call / apply 来调用才能返回正确的类型信息。

Object.prototype.toString.call('') ; // [object String]

Object.prototype.toString.call(1) ; // [object Number]

Object.prototype.toString.call(true) ; // [object Boolean]

Object.prototype.toString.call(Symbol()); //[object Symbol]

（3）**constructor**：obj.constructor === Array

**（4）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，instanceof 只能用来判断两个对象是否属于实例关系， 而不能判断一个对象实例具体属于哪种类型。

### 17 Promise async await

回调函数实现异步编程如果碰到了完成多个异步操作的情况就需要嵌套使用回调函数，异步操作越多，嵌套得就越深，这样非常不利于代码的维护，代码阅读起来也很困难，回调地狱带来的负面作用有以下几点：

代码臃肿。

1 可读性差。

2 耦合度过高，可维护性差。

3 代码复用性差。

4 容易滋生 bug。

5 只能在回调里处理异常。

Promise 是异步编程的一种解决方案，比传统的解决方案——回调函数和事件——更合理和更强大。解决来之前在请求中回调请求产生的回调地狱，使得现在的代码更加合理更加优雅，也更加容易定位查找问题。

Promise主要解决的问题就是异步回调嵌套过深造成代码难以维护和理解。

Promise构造函数内的代码是同步执行的，而之后then或catch方法是异步执行的，构造函数接受两个函数参数resolve和reject，它们执行时接受的参数分别会传递给then和catch表示成功的回调以及失败回调接受到的值。

· Promise一共有三种状态pending等待状态、resolved已完成状态、rejected已拒绝状态，状态的改变只能由等待转为已完成或等待转为已拒绝状态，而且状态的改变只会发生一次。

· 必须要实现then方法且方法里必须要返回一个Promise对象，如果是返回其他的类型会尝试包装成Promise对象；

· then可以被链式的调用。

· 缺点是Promise链中途无法取消；错误需要通过回调函数捕获。

如何中断promise：

原Promise对象的状态将跟新对象保持一致。利用这一特性，当新对象保持“pending”状态时，原Promise链将会中止执行。

Promise.resolve().then(() => {

console.log('ok1')

return new Promise(()=>{}) // 返回“pending”状态的Promise对象

}).then(() => {

// 后续的函数不会被调用

console.log('ok2')

}).catch(err => {

console.log('err->', err)

})

**Generator函数：**

Generator函数是ES6提供的一种异步编程解决方案，形式上也是一个普通函数，但有几个显著的特征：

  --  function关键字与函数名之间有一个星号 "\*" （推荐紧挨着function关键字）

  --  函数体内使用 yield 表达式，定义不同的内部状态 （可以有多个yield）

  --  直接调用 Generator函数并不会执行，也不会返回运行结果，而是返回一个遍历器对象（Iterator Object）

  --  依次调用遍历器对象的next方法，遍历 Generator函数内部的每一个状态

使用场景： 延迟执行

generator调用后，其状态是挂起的，通过调用next()来让其继续执行，从而达到延迟执行的效果

#### async await

**async function语法:**

async/await实际上是Generator的语法糖。顾名思义，async关键字代表后面的函数中有异步操作，await表示等待一个异步方法执行完成。async 函数返回一个Promise对象（如果指定的返回值不是Promise对象，也返回一个Promise，只不过立即 resolve ，处理方式同 then 方法），因此 async 函数通过 return 返回的值，会成为 then 方法中回调函数的参数

**await用法：**

  await 就是异步等待，它等待的是一个Promise，因此 await 后面应该写一个Promise对象，如果不是Promise对象，那么会被转成一个立即 resolve 的Promise。 async 函数被调用后就立即执行，但是一旦遇到 await 就会先返回,先执行func()后面的同步任务，同步任务执行完后，接着await等待的位置继续往下执行。

**async优点：**

1. ****内置执行器****。 Generator 函数的执行必须靠执行器，所以才有了 co 函数库，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。
2. ****更好的语义****。 async 和 await，比起星号和 yield，语义更清楚了。async 表示函数里有异步操作，await 表示紧跟在后面的表达式需要等待结果。
3. ****更广的适用性****。 co 函数库约定，yield 命令后面只能是 Thunk 函数或 Promise 对象，而 async 函数的 await 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）。

### 18 $documnent.ready()  VS  window.onload() 两种方式的区别（哪个先执行）

document.ready 和 window.onload 的区别是：上面定义的document.ready方法在DOM树加载完成后就会执行，而window.onload是在页面资源（比如图片和媒体资源，它们的加载速度远慢于DOM的加载速度）加载完成之后才执行。也就是说$(document).ready要比window.onload先执行。

### 19 ES6的新特性

1. const和let：let表示声明变量，而const表示声明常量，两者都为块级作用域；const 声明的变量都会被认为是常量，意思就是它的值被设置完成后就不能再修改了
2. 模板字符串：基本的字符串格式化。将表达式嵌入字符串中进行拼接。用${}来界定；
3. ES6反引号(``)直接搞定；
4. 箭头函数
5. 对象和数组解构
6. for...of 和 for...in：for...of 用于遍历一个迭代器，如数组
7. for...in 用来遍历对象中的属性
8. ES6中的类：extends 允许一个子类继承父类，需要注意的是，子类的constructor 函数中需要执行 super() 函数。  
   当然，你也可以在子类方法中调用父类的方法，如super.parentMethodName()。

### 20 箭头函数和普通函数区别

1 箭头函数是匿名函数，不能作为构造函数，不能使用new

2 箭头函数不能绑定arguments，必须通过命名参数和不定参数这两种形式访问函数的参数3 箭头函数没有原型属性

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

5 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

### 21 this指向

1. 如果 new 关键词出现在被调用函数的前面，那么JavaScript引擎会创建一个新的对象，被调用函数中的this指向的就是这个新创建的函数。
2. 如果通过apply、call或者bind的方式触发函数，那么函数中的this指向传入函数的第一个参数。
3. 如果函数使用了明确对象来调用（对象方法），this为这个调用对象。
4. 默认，严格模式this为undefined，非严格模式为全局对象（ES6的箭头函数会默认使用外层作用域的this）

（普通函数：window

构造函数：实例对象，原型对象里面的方法也指向实例对象

对象方法：该方法所属对象

事件绑定对象：绑定事件对象

定时器函数以及立即执行函数：window）

### 22 绑定事件的几种方式

1. **在DOM元素中直接绑定：**

**1 原生函数**

<input onclick="alert('谢谢支持')" type="button" value="点击我，弹出警告框" />

**2自定义函数**

<input onclick="myAlert()" type="button" value="点击我，弹出警告框" />

<script type="text/javascript">

function myAlert(){

alert("谢谢支持");

}

</script>

1. **在JavaScript代码中绑定**：

document.getElementById('btn').onclick = function(){ console.log('hello'); }

1. **绑定事件监听函数**：绑定事件监听函数addEventListener()或attachEvent()

### 23 js中哪些操作会导致内存泄露以及如何找到内存泄露

1. ****意外的全局变量：****未定义的变量会在全局对象创建一个新变量。在浏览器中，全局对象是 window
2. ****被遗忘的计时器或回调函数：****当不需要setInterval或者setTimeout时，定时器没有被clear，定时器的回调函数以及内部依赖的变量都不能被回收，造成内存泄漏
3. **闭包引起的内存泄露**
4. **dom清空或删除时，事件未清除导致的内存泄漏**
5. **循环引用以及死循环**
6. **如果在mounted/created  
   钩子中使用了on，需要在beforeDestroy中做对应解绑(on，需要在beforeDestroy 中做对应解绑(*on*，需要在*beforeDestroy*中做对应解绑(off)处理**

如何找到内存泄露**：**

Chrome 提供了一套很棒的检测 JavaScript 内存占用的工具。与内存相关的两个重要的工具：timeline 和 profiles。Profiles 是你可以花费大量时间关注的工具，它可以保存快照，对比 JavaScript 代码内存使用的不同快照，也可以记录时间分配。每一次结果包含不同类型的列表，与内存泄露相关的有 summary（概要） 列表和 comparison（对照） 列表。

summary（概要） 列表展示了不同类型对象的分配及合计大小：shallow size（特定类型的所有对象的总大小），retained size（shallow size 加上其它与此关联的对象大小）。它还提供了一个概念，一个对象与关联的 GC root 的距离。

对比不同的快照的 comparison list 可以发现内存泄露。

对于更大型的网站，Chrome 还提供了 Record Heap Allocations 功能。

### 24 有哪些方法可以处理 JS 中的异步代码？

1 回调

2 Promise

3Generator函数

3 async/await

### 25 什么是缓存及它有什么作用？

缓存是建立一个函数的过程，这个函数能够记住之前计算的结果或值。使用缓存函数是为了避免在最后一次使用相同参数的计算中已经执行的函数的计算。这节省了时间，但也有不利的一面，即我们将消耗更多的内存来保存以前的结果。

### 26 . 什么时候不使用箭头函数?

* 当想要函数被提升时(箭头函数是匿名的)
* 要在函数中使用this/arguments时，由于箭头函数本身不具有this/arguments，因此它们取决于外部上下文
* 使用命名函数(箭头函数是匿名的)
* 使用函数作为构造函数时(箭头函数没有构造函数)

### 27 如何在 JS 中创建对象？

**使用对象字面量：即consta= {}**

**使用构造函数：用new调用**

**Object.create方法：创建没有原型的对象**

### 28 什么是回调函数？

**回调函数**是一段可执行的代码段，它作为一个参数传递给其他的代码，其作用是在需要的时候方便调用这段（回调函数）代码。

在JavaScript中函数也是对象的一种，同样对象可以作为参数传递给函数，因此函数也可以作为参数传递给另外一个函数，这个作为参数的函数就是回调函数。

### 29 原型的继承，ES5和ES6继承与不同点

#### 1 ES5：

**原型链：**通过原型链将子类的原型指向父类的实例

// 原型链继承

function Person(){

this.name = 'xiaopao';

}

Person.prototype.getName = function(){

console.log(this.name);

}

function Child(){

}

Child.prototype = new Person();

var child1 = new Child();

child1.getName(); // xiaopao

**缺点**：

引用类型的属性被所有实例共享

在创建Child 的实例时， 不能向Person传参

2 **借用构造函数：**其核心思想是：在子类型构造函数的内部调用父类构造函数，使用apply()和call()方法将this指向子类

function FarType(name) {

this.name = name

this.sex = 'male'

this.color = ['red', 'blue', 'yellow']

}

FarType.prototype.showName = function() {

console.log(this.name)

}

FarType.prototype.showSex = function() {

console.log(this.sex)

}

function SonType(age, name) {

FarType.call(this, name)

this.age = age

}

SonType.prototype.showAge = function() {

console.log(this.age)

}

缺点：  
1.只是子类的实例，不是父类的实例  
2.方法都在构造函数中定义，每次创建实例都会创建一遍方法

3 **组合继承：**将原型链继承和借用构造函数继承组合到一起。使用原型链继承实现对原型属性和方法的继承，用借用构造函数继承实现对父实例属性的继承。这样既通过在原型上定义方法实现了函数复用，又能保证每个实例都有自己的属性

function FarType(name) {

this.name = name

this.sex = 'male'

this.color = ['red', 'blue', 'yellow']

}

FarType.prototype.showName = function() {

console.log(this.name)

}

FarType.prototype.showSex = function() {

console.log(this.sex)

}

function SonType(age, name) {

FarType.call(this, name)

this.age = age

}

SonType.prototype = new FarType();

SonType.prototype.showAge = function() {

console.log(this.age)

}

缺点：

调用了两次父类构造函数  
（组合继承最大的问题是无论什么情况下，都会调用两次超类型构造函数：一次是在创建子类型原型的时候，另一次是在子类型构造函数内部）

4 **原型式继承：**原型式继承的基本思想：

1个基础对象，object.create创建1个新对象，把基础对象传进去作为原型对象

原型式继承解决了原型链无法传参的问题，并且无需使用构造函数（避免了构造函数的问题）。因此在没必要使用构造函数时可以采用这种方法。

但引用类型问题依旧存在

//基础对象

var person = {

name: "一灯",

arr: [1,2,3]

}

//Object.create()创建新对象，传入基础对象

var son1 = Object.create(person)

son1.name = "AAA"

son1.arr.push(4)

console.log(son1.name) //AAA

var son2 = Object.create(person)

son2.name = "BBB"

console.log(son2.name) //BBB

console.log(son2.arr) //1,2,3,4，引用类型问题依然存在

5 寄生式继承：寄生式继承可以理解为是原型式继承的增强。在原型式继承中我们创建了一个新对象，寄生式继承便是在新对象中添加方法，以增强对象

6 **寄生组合式**：

组合式：子级的prototype继承父级的prototype——通过new Father()

寄生组合式：子级的prototype继承父级的prototype——通过赋值

#### 2 ES6

Class son extends father

Super关键字用于访问和调用父类上的函数，可以是父类的构造函数，也可以是普通函数，

子类在构造函数中使用super，必须放到this前面

### 30 Set,Map解构

ES6 提供了新的数据结构 Set。  
它类似于数组，但是成员的值都是唯一的，没有重复的值。Set 本身是一个构造函数，用来生成 Set 数据结构。  
 ES6 提供了 Map 数据结构。它类似于对象，也是键值对的集合，但是“键”的范围不限于字符串，各种类型的值（包括对象）都可以当作键。

### 31 服务器端渲染（SSR）和预渲染

SSR：如果服务器恰好是nodejs，那么可以让服务器提前运行“第一次渲染”。**（服务器端在返回html之前，在html特定的区域特定的符号里用数据填充，再给客户端，客户端只负责解析HTML。）**浏览器只需要显示就好了，如果需要，再进行以后的渲染。

预渲染：无需服务器实时动态编译，采用预渲染，在构建时针对特定路由简单的生成静态HTML文件

优点：

1前端耗时少，首次渲染快，更快的内容到达时间利于SEO

2加载应用程序的路由，将结果保存在一个静态的HTML文件中  
 3 无需更改代码或添加服务器端

服务端渲染的过程为：解析执行JS => 构建HTML页面 => 输出给浏览器

预渲染：直接输出HTML页面给浏览器

### 32 图像懒加载的原理是什么？

页面加载时，将图片加载的链接，保存在img标签的自定义属性中，src属性为空，并监听窗口的scroll事件。当img标签出现在视口中时，利用js将图片加载编写填写至src属性中，实现动态加载图片

### 33 Js事件循环

### 34 长轮询和短轮询是什么？

短轮询指浏览器向服务器发送一个请求，询问是否有数据更新，服务里立刻返回响应。一段时间后浏览器又发起一个到服务器的新请求。

长轮询指浏览器向服务器发送一个请求，服务一直保持连接打开，直到有数据可发送，发送完数据后，浏览器关闭连接，随即又发起一个到服务器的新请求。（使用XHR对象和setTimeout()实现）

### 35  jQuery获取的dom对象和原生的dom对象有何区别

js原生获取的dom是一个对象，jQuery对象就是一个数组对象，其实就是选择出来的元素的数组集合，所以说他们两者是不同的对象类型不等价

原生DOM对象转jQuery对象：

**var** box = document.getElementById('box');**var** $box = $(box);

jQuery对象转原生DOM对象：

**var** $box = $('#box');**var** box = $box[0];

### 36  jQuery如何扩展自定义方法

(jQuery.fn.myMethod=**function** () {

alert('myMethod');

})

### 36 script标签的defer、async的区别

defer是在HTML解析完之后才会执行，如果是多个，按照加载的顺序依次执行

加载脚本和渲染后续文档元素并行进行，但脚本的执行会等到 html 解析完成后执行

async是在加载完成后立即执行，如果是多个，执行顺序和加载顺序无关

加载脚本和渲染后续文档元素并行进行，脚本加载完成后，暂停html解析，立即解析js脚本

### 37 函数防抖

****一个需要频繁触发的函数，在规定时间内，只让最后一次生效，前面的不生效。****

其原理就第一次调用函数，创建一个定时器，在指定的时间间隔之后运行代码。当第二次调用该函数时，它会清除前一次的定时器并设置另一个。如果前一个定时器已经执行过了，这个操作就没有任何意义。然而，如果前一个定时器尚未执行，其实就是将其替换为一个新的定时器，然后延迟一定时间再执行。



**应用场景：**

文本输入的验证（连续输入文字后发送 AJAX 请求进行验证，验证一次就好）

### 38 函数节流

有个需要频繁触发函数，出于优化性能角度，在规定时间内，只让函数触发的第一次生效，后面不生效。

其原理是用时间戳来判断是否已到回调该执行时间，记录上次执行的时间戳，然后每次触发 事件执行回调，回调中判断当前时间戳距离上次执行时间戳的间隔是否已经到达 规定时间段，如果是，则执行，并更新上次执行的时间戳，如此循环



用到了闭包做防抖

### 39 [Object.keys( )与 for in 区别](https://www.cnblogs.com/yuzhongyu/p/10831561.html)

### ****Object.keys()** 方法会返回一个由一个给定对象的自身可枚举属性组成的数组，数组中属性名的排列顺序和使用 for...in 循环遍历该对象时返回的顺序一致**

### **两者之间最主要的区别就是Object.keys( )不会走原型链，而for in 会走原型链；**

### 40 arguments是什么，有什么特性

在js中在调用一个函数的时候，我们经常会给这个函数传递一些参数，js把传入到这个函数的全部参数存储在一个叫做arguments的东西里面，实际上是所在函数的一个内置类数组对象，可以用数组的[i]和.length

特性：

js是一种弱类型的语言，没有重载机制，当我们重写函数时，会将原来的函数直接覆盖，这里我们能利用arguments，来判断传入的实参类型与数量进行不同的操作，然后返回不同的数值。

### 41 map数据结构相对于object的优势

Map数据结构类似于对象，也是键值对的集合，但是“键”的范围不限于字符串，各种类型的值（包括对象）都可以当作键。也就是说，Object 结构提供了“字符串—值”的对应，Map 结构提供了“值—值”的对应，是一种更完善的 Hash 结构实现。如果你需要“键值对”的数据结构，Map 比 Object 更合适。

### 42 如果Object.create(null)会怎么样

直接用 var m = {}的情况下，虽然我们得到的也是{}，但是他会继承对象的各种内置属性，包括各种处理对象的方法，而如果直接用Object.create(null)得到的{}，就只是一个空对象，并没有继承对象的基本属性。

**这样的好处是什么呢？**

在vue组件中我们经常会需要自己动态添加属性，这样做的好处在于避免了自加属性与默认属性的重名，除去了不必要的属性，显得较为轻便。

### 43 JS事件循环

event loop它的执行顺序：

1 一开始整个脚本作为一个宏任务执行

2 执行过程中同步代码直接执行，宏任务进入宏任务队列，微任务进入微任务队列

3 当前宏任务执行完出队，检查微任务列表，有则依次执行，直到全部执行完

4 执行浏览器UI线程的渲染工作

5 检查是否有Web Worker任务，有则执行

6 执行完本轮的宏任务，回到2，依此循环，直到宏任务和微任务队列都为空

**微任务包括：**MutationObserver、Promise.then()或reject()、Promise为基础开发的其它技术，比如fetch API、V8的垃圾回收过程、Node独有的process.nextTick。

**宏任务包括**：script、script 、setTimeout、setInterval 、setImmediate 、I/O 、UI rendering。

**注意**：在所有任务开始的时候，由于宏任务中包括了script，所以浏览器会先执行一个宏任务，在这个过程中你看到的延迟任务(例如setTimeout)将被放到下一轮宏任务中来执行。

### 44 进程和线程

进程是 CPU 资源分配的最小单位；线程是 CPU 调度的最小单位

一个进程由一个或多个线程组成，线程是一个进程中代码的不同执行路线，一个进程的内存空间是共享的，每个线程都可用这些共享内存。

### 45 多进程与多线程

多进程：在同一个时间里，同一个计算机系统中如果允许两个或两个以上的进程处于运行状态。多进程带来的好处是明显的，比如你可以听歌的同时，打开编辑器敲代码，编辑器和听歌软件的进程之间丝毫不会相互干扰。

多线程：程序中包含多个执行流，即在一个程序中可以同时运行多个不同的线程来执行不同的任务，也就是说允许单个程序创建多个并行执行的线程来完成各自的任务。

### 46 浏览器与node事件循环区别

**宏任务和微任务的执行方式在浏览器和 Node 中有差异。**

浏览器环境下，microtask 的任务队列是每个 macrotask 执行完之后执行。而在 Node.js 中，microtask 会在事件循环的各个阶段之间执行，也就是一个阶段执行完毕，就会去执行 microtask 队列的任务。

**process.nextTick**

这个函数其实是独立于 Event Loop 之外的，它有一个自己的队列，当每个阶段完成后，如果存在 nextTick 队列，就会清空队列中的所有回调函数，并且优先于其他 microtask 执行。

### 47 浏览器内核多线程

GUI 渲染线程

JavaScript 引擎线程

定时触发器线程

事件触发线程

异步 http 请求线程

### 48 用js写动画如果发生掉帧的情况怎么办

#### **requestAnimationFrame代替setTimeout和setInterval**

****为什么**setTimeout**和**setInterval**不好？****  
由于js是单线程执行，所以为了防止某个任务执行时间过长而导致进程阻塞，js中存在异步队列的概念，对于如setTimeout和ajax请求都是把进程放到了异步队列中，当主进程为空时才执行异步队列中的任务。所以 setTimeout和setInterval无法保证回调函数的执行时机，可能会在一帧之内执行多次导致多次页面渲染，浪费CPU资源甚至产生卡顿，或者是在一帧即将结束时执行导致重新渲染，出现掉帧的情况。  
**requestAnimationFrame**是怎么优化的？****

* CPU节能，当页面被隐藏或最小化时，暂停渲染。
* 函数节流，其循环间隔是由屏幕刷新频率决定的，保证回调函数在屏幕的每一次刷新间隔中只执行一次。

**requestAnimationFrame和setTimeout的区别：**

* 1 间隔时间不精确，可能被阻塞。计时器的间隔指的是将回调函数推入任务队列的间隔时间，任务队列中的任务只有在主线程任务执行完毕后才会被执行。
* 2 计时器动画的间隔时间如果设定过短就会出现过度渲染占用大量资源，如果设定过长就会影响动画的流畅度。只能够估计合适的时间间隔。
* 3 多数浏览器对于计时器动画没有优化。

使用 requestAnimationFrame 方法来定义动画就不会出现以上这些问题。

* requestAnimationFrame 动画的帧数是由系统根据当前页面是否可见，CPU 的占用情况等等来决定的，可以最大化地利用系统性能。
* 浏览器对 requestAnimationFrame 动画进行了优化。

### 49 如何进行垃圾回收

1). 标记清除算法  
 从全局作用域开始，一层一层往下标记，所有被标记到的就是不清除的，标记完成后把所有没被标记的删掉。  
2). 引用计数  
 记录每个对象被引用的次数，每次被创建引用的时候加1，有不引用它的就减1，当减到0的时候这个对象就应该被垃圾回收。

50 js事件模型  
IE事件模型

IE事件模型共有两个过程:

事件处理阶段(target phase)。事件到达目标元素, 触发目标元素的监听函数。

事件冒泡阶段(bubbling phase)。事件从目标元素冒泡到document, 依次检查经过的节点是否绑定了事件监听函数，如果有则执行。

W3C标准模型

在该事件模型中，一次事件共有三个过程:

事件捕获阶段(capturing phase)。事件从document一直向下传播到目标元素, 依次检查经过的节点是否绑定了事件监听函数，如果有则执行。

事件处理阶段(target phase)。事件到达目标元素, 触发目标元素的监听函数。

事件冒泡阶段(bubbling phase)。事件从目标元素冒泡到document, 依次检查经过的节点是否绑定了事件监听函数，如果有则执行。

### 51 谈谈对Proxy的理解 ？

和Object.defineProperty有些类似，它的作用是用来自定义对象中操作。Proxy的构造函数接受两个参数，第一个参数是需要代理的对象，第二个参数是一个对象，里面会定义get和set方法，当代理对象中的某个值被访问或重新赋值就会触发相应的get和set方法。vue3.0就抛弃了Object.defineProperty而拥抱了Proxy，它的优点是只需要代理一次，对象内的值发生了改变就会被感知到，不再需要像以前为对象的每个值进行数据劫持；而且以前对象的新增，数组的下标设置0清空等情况都可以被感知到，在响应式里也不在需要为数组和对象收集两次依赖，相信会大大提升性能。

### 52 ES5的继承和ES6的继承有什么区别？

ES5的继承时通过prototype或构造函数机制来实现。ES5的继承实质上是先创建子类的实例对象，然后再将父类的方法添加到this上（Parent.apply(this)）。

ES6的继承机制完全不同，实质上是先创建父类的实例对象this（所以必须先调用父类的super()方法），然后再用子类的构造函数修改this。

### 53 如何判断空数组

1 arr.length === 0?

2 JSON.stringify(obj) === ‘[]’