1. 0.1 + 0.2 === 0.3吗？为什么 https://zhuanlan.zhihu.com/p/95318421 false js表示的数字范围：-2 ^ 53 + 1 ~ 2 ^ 53 - 1

js使用Number类型来表示数字(整数或浮点数)，遵循IEEE 754标准，通过64位来表示一个数字（1 + 11 +52）

- 1 位符号位，0表示正数，1表示负数
- 11位指数位(e)
- 52位尾数，小数部分(即有效数字)

最大安全数字： Number.MAX_SAFE_INTEGER = Math.pow(2,53) -1，转换成整数就是16位，所以0.1=== 0.1是因为通过toPrecision(16)去除有效位之后两者是相等的

在两位数相加时，会先转换成二进制，0.1和0.2转换成二进制的时候尾数会发生无限循环，然后进行对阶运算，js引擎对二进制进行截断，所以会造成精度丢失 所以总结：精度丢失可能出现在进制转换和对阶运算中

十进制转二进制：整数部分除以2取余，倒序排列， 小数部分乘2取整数部分，顺序排列 0.1 转化为二进制 0.0 0011 0011 0011 0011 0011 0011 ... （0011循环） 0.2 转化为二进制 0.0011 0011 0011 0011 0011 0011 0011 ... （0011循环）

然后我们用之前说过的IEEE 754 双精度64位浮点数（需翻墙）来表示： // 0.1 e = -4; m = 1.1001100110011001100110011001100110011001100110011010 (52位)

// 0.2 e = -3; m = 1.1001100110011001100110011001100110011001100110011010 (52位)

然后我们把它相加，这里有一个问题，就是指数不一致时，应该怎么处理，一般是往右移，因为即使右边溢出了，损失的精度远远小于左移时的溢出。 e = -4; m = 1.1001100110011001100110011001100110011001100110011010 (52位) + e = -3; m = 1.1001100110011001100110011001100110011001100110011010 (52位) 转化

e = -3; m = 0.1100110011001100110011001100110011001100110011001101 (52位) + e = -3; m = 1.1001100110011001100110011001100110011001100110011010 (52位) 得到

e = -3; m = 10.0110011001100110011001100110011001100110011001100111 (52位)

2. js整数是怎么表示的

使用Number类型来表示，遵循IEEE754标准，通过64位来表示一个数字（1+11+52） 符号位决定了一个数的正负，指数部分决定了数值的大小，小数部分决定了数值的精度

3. Number()存储的空间是多大 如果后台发送了一个超过最大数字的数字怎么办

Math.pow(2,53),如果超过最大数字，会发生截断

4. 有3个函数可以把非数值转换成数值：Number()、parseInt()和parseFloat()。其中Number()可以将任意类

型的值转化成数值，而parseInt()和parseFloat()只应用于字符串向数字的转换

Number('') => 0 Number(null) => 0 parseInt('') => NaN

5. js对于数据的处理方法

- Number Number()可以将任意类型的值转换为数值 // 数值：十进制数字 console.log(Number(11),Number(011),Number(0x11));//11 9 17 // undefined：转成 NaN Number(undefined) // NaN // null：转成0 Number(null) // 0

// 布尔值：true 转成1，false 转成0 console.log(Number(true),Number(false));//1 0

Number()函数解析字符串时会识别出字符串的前置空格并去掉

【1】若字符串只包含十进制或十六进制数字，则转成十进制的数字

[注意1]Number()不识别八进制数字的字符串，会按照十进制数字处理

[注意2]字符串'1.2.'不会报错，但数字1.2.会报错 【2】若字符串为空字符串或空格字符串，则转成0 【3】其他情况的字符串，则转成NaN

Number()函数解析对象时，会按照以下步骤进行处理

【1】调用对象的valueOf()方法，如果返回原始类型的值，则直接对该值使用Number()函数

【2】如果valueOf()方法返回的还是对象，则调用对象的toString()方法，如果返回原始类型的值，则对该值使用Number()函数

【3】如果toString()方法返回的依然是对象，则结果是NaN

在第一步中，由于只有时间Date()对象返回的是原始类型的值数字，所以Number(new Date())返回现在到1970年1月1日00:00:00的数值类型的毫秒数

Number(new Date())//1465976459108

在第二步中，数组Array类型返回由数组中每个值的字符串形式拼接而成的一个以逗号分隔的字符串，如果字符串中只存在数字，则返回数字，其他情况返回NaN；由于其他对象的toString()方法返回的字符串中不只包括数字，所以返回NaN Number([]);//0 Number([0]);//0 Number([-0]);//0 Number([10]);//10 Number([1,2]);//NaN Number(其他对象);//NaN

从上面可以看到，Number转换的时候是很严格的，只要有一个字符无法转成数值，整个字符串就会被转为NaN

- parseInt 将给定的字符串以指定的基数解析为整数 parseInt相比Number，就没那么严格了，parseInt函数逐个解析字符，遇到不能转换的字符就停下来

parseInt('32a3') //32

parseInt(string,radix) // 第二个参数表示进制 【1】parseInt()专门用于把字符串转换成整数。在转换字符串时，会忽略字符串前面的空格，直到找到第一个非空格字符。如果第一个字符不是数字字符或者负号，parseInt()就会返回NaN。如果是，则继续解析，直到解析完成或者遇到非数字字符

5. 实现深度克隆

浅克隆 function shallowClone(obj) { let cloneObj = {} for(let i in obj) { cloneObj[i] = obj[i] } } var obj = Object.assign(obj,o1,o2)

深克隆 // 这样拷贝的结果 时间只是字符串形式 而不是事件对象 // 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； // 如果对象中存在循环引用的情况也无法正确实现深拷贝； JSON.parse(JSON.stringify(obj))

深克隆要考虑：解决循环引用

- 考虑基本类型
- 引用类型
- 一般写法 并没有解决循环引用 function deepClone(target) { let newObj = Array.isArray(target) ? [] : {} if(target === null) return target if(typeof target !== 'object') return target for(let i in target) { newObj[i] = deepClone(target[i]) } return newObj }
- 解决循环引用的写法 function deepClone(target) { const map = new Map() function clone() { if(target === null || typeof target !== 'object') { return target } var result = Array.isArray(target) ? [] : {} if(map.get(target)) { return map.get(target) }

```
map.set(target, result)
for(let key in target) {
    if(target.hasOwnProperty(key)) {
        result[key] = clone(result[key])

    }
}
return result
```

} return clone(target) }

6.事件流 事件流是网页元素接收事件的顺序，分为三个阶段：事件捕获、到达目标和事件冒泡

https://www.cnblogs.com/Leophen/p/11405579.html

- DOM0级事件：onClick 赋值函数 赋值函数是元素的方法，因此事件处理程序会在元素的作用域中运行，this等于元素 移除 onClick = null

同一个元素 onClick 赋值 后面的事件会覆盖掉前面的事件，DOM2级不会

DOM0级事件有很好的兼容性，绑定速度快

- DOM2级事件 addEventListener/ removeEventListener

target.addEventListener(type,listener, [useCapture])

listener: 事件处理方法 useCapture: 布尔参数，默认false，表示在事件冒泡阶段处理，如果是true，则表示在捕获阶段调用事件处理程序

- 只有DOM2级事件有三个阶段 事件捕获、处于目标阶段、事件冒泡阶段

ie兼容DOM2 attachEvent detachEvent

- 事件委托/事件代理 利用事件冒泡，例如为整个document指定一个onClick点击事件

DOM3: 模拟事件，自定义DOM事件，增加了事件类型：如UI事件，鼠标事件

7. 事件是如何实现的

基于发布订阅模式，就是在浏览器加载的时候会读取事件相关的代码，但是只有等实际触发的时候才执行

7.new一个函数发生了什么

new运算符创建一个用户定义的对象类型的实例或具有构造函数的内置对象的实例

1. 创建一个空的简单javascript对象即{}
2. 链接该对象到另一个对象（即设置该对象的构造函数）
3. 将步骤1新创建的对象作为this的上下文
4. **如果该函数没有返回对象或返回基本类型则返回this**

- 当代码new Foo(...)执行时，会发生以下事情

    1. 一个继承自Foo.prototype的新对象被创建
    2. 使用指定的参数调用构造函数Foo，并将this绑定到新创建的对象，new Foo等同于new Foo(),也就是没有指定参数列表，Foo不带任何参数调用的情况
    3. 由构造函数返回的对象就是new表达式的结果。如果构造函数没有显式返回一个对象，则使用步骤1返回的对象。（一般情况下，构造函数不返回值，但是用户可以选择主动返回对象，来覆盖正常的对象创建步骤）

  实现一个new 函数 function _new(func,...args) { let obj = Object.create(func.prototype) var res = func.call(obj,...args) if(res!== null && typeof res === 'object' || typeof res === 'function'){ return res } return obj }

8. symbol有什么好处

可以用来表示一个独一无二的变量防止命名冲突。 symbol不会被常规方法遍历到，除了Object.getOwnPropertySymbol外，所以可以用来模拟私有变量

提供遍历接口 布置了Symbol.iterator的对象才可以使用for of循环

9.闭包是什么

当函数可以记住并访问所在的词法作用域时，就产生了闭包，即使函数是在当前词法作用域之外执行。

- 闭包的本质：当前环境中存在指向父级作用域的引用
- 一般如何产生闭包?

1. 返回函数
2. 函数当作参数传递

- **闭包的应用(todo深入理解)**

1. 函数柯里化 接收多个参数的函数变换成接收单一参数的函数，嵌套返回直到所有的参数都被使用并返回最终的结果。f(a,b,c) => f(a)(b)(c)
2. 模块

缺点：使用闭包会占有内存资源，过多的闭包会导致内存溢出 内存泄漏处理：简单的说就是把那些不需要的变量，但是垃圾回收又收不走的那些赋值为null，然后让垃圾回收走 闭包就是能够读取其他函数内部变量的函数，通俗的讲就是函数a的内部函数b被函数a外部的一个变量引用的时候，就创建了一个闭包，最常见的是函数封装的时候，再就是使用定时器的时候

闭包的优点： 减少全局变量，减少传递函数的参数量，封装 缺点：占用内存资源，过多的闭包会导致内存溢出 内存泄漏解决： 不需要的变量赋值为null

- 垃圾回收机制： js具有垃圾收集器，垃圾收集器会按照固定的时间间隔周期性的执行 最常见的垃圾回收方式有两种：

1. 标记清除 运行时候给内存变量加标记 去掉环境变量以及引用的变量的标记，剩下的带标记的视为准备删除的，销毁带标记的值并回收内存空间
2. 引用计数 跟踪记录每个值被引用的次数 次数变为0 就释放

10. 什么是作用域

es5有两种作用域：全局作用域和函数作用域，可以理解为是一套规则，用了管理引擎如何在当前作用域及嵌套作用域根据变量标识符进行变量查找 es6 块作用域

11. 什么是作用域链

当访问一个变量时候，会从当前的作用域进行查找，如果没有找到会去父级作用域查找，如果父级没找到，继续往上查找，直到找到全局作用域，如果是取值会报错 a is not defined 如果是赋值则在全局设置一个值 a = 3

12. NaN是非有效数字 跟任何值都不相等包括其自身

typeof NaN === 'number'

13. js隐式转换，显式转换

一般非基础类型进行转换时会先调用valueOf，如果valueOf无法返回基本类型值，就会调用toString()

- 1. 字符串和数字 +操作符：如果有一个为字符串，那么都转换为字符串然后执行字符串拼接 -操作符：转换为数字，相减 (-a, a * 1 a/1) 都能进行隐式强制类型转换 [] + {} => "[object Object]" {} + [] => 0 {}在前被当作区块语句 相当于 +[] 为一元运算符，直接转换为Number类型，相当于Number() 所以 +[] => Number([]) => 0

undefined + undefined = NaN null + null = 0 undefined + null = undefined [] + [] = '' 因为数组[].toString() => '' [1, 2, 3].toString() => "1,2,3" {} + {} = "[object Object][object Object]"

(function() {}).length => 0 形参的个数

+操作符：数字+数字 =》数字 字符串+字符串=>字符串 其他类型的值相加最终都会隐式转换为上述两种情况

布尔值到数字： 1+ true = 2 1+ false = 1 '1' + true = '1true' '1' + false = '1false' 数字与字符串相加结果是字符串 数字字符串与数字相减 数字字符串会变成数字，结果就是数值了 非数值字符串与数字相减 结果是NaN '' -1 = -1

- 转换为布尔值的情况 for中的第二项 while if 三元 || && 左边的操作数
- === 和 == ==会进行强制类型转换

1. 字符串与数字 转换成数字然后比较
2. 其他类型与布尔类型 先把布尔类型转换成数字，然后进行后续比较
3. 对象与非对象 将对象转换为原始值，然后进行比较

true - true = 0 (!+[]+[]+![]) => "truefalse"

! + [] => !+(+[]) => true

[] == false ![] == false [] == ![]

[]==[] //false

[]==![] //true 原因就是进行比较的时候，等式右边的空数组被转换为了一个布尔值，空数组是true，取反是false；数组左边与布尔值进行比较，需要将二者都转换为数字，左侧空数组转换为0， 右侧相当于false转换为数字，也是0， 所以二者相等

{}==!{} //false

{}==![] //VM1896:1 Uncaught SyntaxError: Unexpected token ==

![]=={} //false

[]==!{} //true

[] == false // true

undefined==null //true {} == false // 报错

一、对象转化成字符串： 规则：

1、如果对象有toString方法，则调用该方法，并返回相应的结果；（代码通常会执行到这，因为在所有对象中都有toString方法）

2、如果对象有valueOf方法，则调用该方法，并返回相应的结果；

3、否则抛出异常。

二、对象转化成数字 需要转化成数字的两种主要情况： 函数里边的参数需要是数字，如： Math.sin(obj) / isNaN(obj) 以及算术运算符： +obj； 用于比较，如：obj == 'John' PS:下面两种比较不会发生类型转换，

> a)在严格比较（===）中，不会发生任何的类型转换，
>
> b)在非严格比较中，如果参数都是对象，不会发生类型转换，通常，如果两个对象引用统一对象，则返回true.

转化成数字的规则：

1、如果对象有valueOf方法，则调用该方法，并返回相应的结果；

2、当调用valueOf返回的依然不是数字，则会调用对象的toString方法，并返回相应的结果；

3、否则抛出异常。

- 假值列表 undefined null false +0，-0， NaN ""