JS面试题

一、js的数据类型 - Boolean、Number、String、Undefined、Null、Object、Symbol、BigInt。

附加题：js 中的基础类型和对象类型有什么不一样？- 基础类型存储在栈中，对象类型存储在堆中。

附加题：原始数据类型直接存储在栈，占据空间小、大小固定，由编译器自动分配释放，引用数据类型存储在堆占据空间大、大小不固定。由开发者分配释放程序结束时可能由垃圾回收机制回收

附加题：什么是 JavaScript 中的包装类型？

在调用基本类型的属性或方法时 JavaScript 会在后台隐式地将基本类型的值转换为包装类型的对象，var的过程相当于new的过程，也可以使用valueOf方法将包装类型倒转成基本类型

附加题：字符串是基本数据类型,基本类型是没有方法的，但为什么字符串还有很多方法？

创建字符串的过程实际上是创建String类的过程，在String类上具有很多方法，所以字符串会继承自String

附加题：null空对象用于赋值可能会返回对象的变量作为初始化；undefined: 声明了变量但是没有赋值

附加题：null === undefined => false null == undefined => true

附加题：为什么会有BigInt的提案？为了解决js超出最⼤安全数字这个范围出现精度丢失计算不准确情况

附加题：如何获取安全的undefined - 可以用 void 0 来获得 undefined附加题：typeof NaN === 'number'

附加题：es6中的声明 - var let const class function import

附加题；数据类型检测的方式有哪些？typeof、instanceof、constructor、Object.prototype.toString.call

附加题：construct作用 - 判断数据的类型和对象访问它的构造函数，对象来改变它的原型就访问不到了

附加题：instanceof？能正确判断引用数据类型，而不能判断基本数据类型，其内部运行机制是判断在其原型链中能否找到该类型的原型，instanceof 运算符可以用来测试一个对象在其原型链中是否存在一个构造函数的 prototype 属性

附加题：Obj.toString的结果和Object.prototype.toString.call的结果不一样，这是为什么？

obj.toString将obj转换为字符串类型，另外一个得到对象的具体类型，所以结果不一样

附加题：isNaN 和 Number.isNaN区别？isNaN 接收参数转换为数值而另一个判断传入参数是否为数字如果是数字再判断是否NaN 不会进行数据类型的转换

附加题：什么是安全整数 - IEEE-754双精度数字，Number.MAX\_SAFE\_INTEGER

附加题：如何判断一个整数是安全整数 - Number.isSafeInteger

附加题：如何判断一个数值为整数 - Number.isInteger / typeof num == "number" && num % 1 == 0

二、0.1 + 0.2 为什么不等于 0.3？

数据都是以二进制存储的计算时数据先转换成二进制然后计算结果转换成十进制所以0.1+0.2时实际上是两个数二进制的和。0.1的二进制是1100循环0.2的二进制是1100循环，这两个数的二进制都是无限循环的数，IEEE 754 标准中的尾数位保存 52 位有效数字，所以 二进制就会发生舍入，所以就产生了误差

附加题：判断 0.1+0.2 = 0.3 - 设置误差范围EPSILON即2-52，Math.abs(arg1 - arg2) < Number.EPSILON

，在ES6中，提供了Number.EPSILON属性，而它的值就是2-52，只要判断0.1+0.2-0.3是否小于Number.EPSILON，如果小于，就可以判断为0.1+0.2 === 0.3

附加题：怎么能使 0.1 + 0.2 = 0.3 - 1. 转为整数计算，计算后再转回小数 2. 转成字符串相加（效率较低）

附加题：js对二进制小数的存储方式? IEEE 754标准共64位共有1个符号位11指数位52尾数位

附加题：Number的存储空间是多大？- Math.pow(2, 53) 如果后台发送了一个超过最大数字？- 截断

附加题：二进制小数的存储为什么尾数位只有 52 位，但是要从第一个 1 后面开始保留52位 有效数字？

数部分的整数部分一定是一个 1，表示更高的精确度可以这个整数部分省略，52 位 都用来表示小数

三、toString类型，toNumber类型，toBoolean

1. null ,undefined,,boolean,Number对应加’’ number极小和极大的数字会使用指数形式Symbol直接转换，只允许显式使用隐式会报错obj自己的toString会调用使用返回值否则会调用 Object.prototype.toString

2. undefined-NaN,null-0,true-1, false-0,String使用Number非数字NAN，Symbol会报错对象转换为基本类型值,obj使用valueOf/toString返回基本类型值new RegExp() - /(?:)/,new date - 时间戳 {}-{} []-[]

3. undefined null false +0-0 NaN均为false，其他的都是都应该是真值

附加题：介绍一下JavaScript的ToPrimitive

JavaScript 中每个值隐含的ToPrimitive用来将值转换为基本类型值, 如果值为基本类型直接返回值如果值为对象ToPrimitive(obj,type)，type-number时调用valueOf原始值返回否则toString；type-string时调用toString原始值返回否则valueOf；除date外使用Number(value.valueOf().toString())

附加题：JavaScript中如何进行隐式类型转换？

JavaScript 中的隐式类型转换主要发生运算符而运算符只能操作基本类型值，所以使用ToPrimitive转换

1. +: 至少一个string类型就toString；其他情况下toNumber 注：1+Symbol() 会报错

2. -、\*、\操作符：1 / 'aa' => NaN 3. 对于==操作符：操作符两边的值都尽量转成number

4. 对于<和>比较符：如果两边都是字符串则比较字母表顺序其他情况下toNumber再比较

附加题：将字符串转换为数字的方法 - +和Number

四、函数的参数的传递规则 - 原始类型就是按值传递，引用类型是临时复制成副本，但并不是引用传递

五、typeof和instanceof的区别

typeof 会返回一个运算数的基本类型不能正确判断原始数据类型，instanceof返回的是布尔值准确判断引用数据类型，typeof 虽然可以判断原始数据类型null 但是无法判断引用数据类型function除外

附加题：typeof为什么检测null的时候会是object？

js在开发初期为了性能使用低位1-3位存储变量约定000开头为对象null二进制刚好设置为全0

附加题：typeof document.all === undefined // true - V8 中js对象都有一个Map对象描述对象类型相关信息类似元数据，Map使用16 bit instance\_type来描述，他俩的is\_undetectable bit都是1

附加题：typeof 1/0 等于什么和为什么？typeof运算符优先级高于除法会先执行typeof 1返回'number'`执行 `number' / 0会toNumer成NAN。最后执行NaN / 0的操作所以结果是NaN

附加题：typeof NaN => "number" NaN !== NaN -> true typeof null => object

六、Object.is与比较操作符===、==的区别？

使用==如果两边的类型不一致会强制类型转化后再比较。使用===如果两边的类型不一致时返回false

Object.is和三等号的判断相同具备特殊处理比-0和+0不再相等，两个NaN是相等的

附加题：object.assign和扩展运算法是深拷贝还是浅拷贝，两者区别 - 浅拷贝，Object.assign(目标对象,源对象)把源对象合并到目标对象会触发 ES6 setter。…不复制继承的属性或类的属性会复制symbols属性

七、new操作符是什么？- 用于创建一个给定构造函数的实例对象

1. new通过构造函数创建出来的实例可以访问到构造函数内部的属性和构造函数原型链内部的属性

2. new内部会执行构造函数，构造函数无返回会自动返回new内部创建的新对象，构造函数如果返回原始值没有任何意义，如果返回一个对象就会被正常引用，就访问不到构造函数内部以及原型链当中的属性了

附加题：new操作符的实现原理 - 1. 创建一个新的空对象 2. 设置原型，将对象的原型指向构建函数的 prototype 对象就可以访问到构造函数原型上的属性和方法 3. 让函数的this指向这个对象执行构造函数的代码就为这个新对象添加构造函数上的属性和方法 4. 判断函数的返回值类型，如果是值类型，返回创建的对象。如果是引用类型，就返回这个引用类型的对象，返回的对象将作为构造函数的实例

八、对原型、原型链的理解

当使用构造函数新建一个对象后，在这个对象的内部将包含一个指针指向构造函数的 prototype 属性对应的值这个指针就是对象的原型。构造函数的prototype属性包含了所有实例所共享的属性和方法，当访问一个对象的属性时，如果这个对象内部不存在这个属性，那么它就会去它的原型对象里找这个属性，这个原型对象又会有自己的原型，于是就这样一直找下去，这样就形成了一个原型查找链条，也就是原型链。

附加题：获取对象的原型 - Object.getPrototypeOf

附加题：为什么新建的对象能够使用 toString - 原型链的尽头一般来说都是 Object.prototype

附加题：如何获得对象非原型链上的属性 -使用hasOwnProperty方法来判断属性是否属于原型链的属性

附加题:原型链的终点是什么如何打印出原型链的终点 - 原型链的终点是null，Object.prototype.\_\_proto\_\_

九、什么是闭包

函数内部嵌套函数，内层函数调用外层的变量也就是内层作用域链包含了外层的，内层函数被引用导致内层函数不会被释放这个时候就形成了闭包。闭包的作用域链包含了自身作用域，被定义所在函数作用域以及全局作用域

附加题：闭包的本质 - 作用域对象是在堆中被创建的，函数在执行的时候会放到一个执行栈上当函数执行完毕之后就从执行栈上移除，但是堆上作用域成员因为被外部引用而不能被释放，因此内部函数依然可以访问外部函数的成员

附加题：怎样产生一个闭包 - 在一个函数内部定义另一个函数

附加题：闭包的注意事项 - 1. 产生了闭包的函数作用域会一直存在等闭包不存在了之后才会被销毁2. 闭包只能取回包含函数中任何变量的最后一个值3. 如果内层函数返回匿名函数该函数作用域指向window

附加题：闭包的应用 - 定义函数的私有变量、可以做计数器、实现块级作用域、函数柯里化

附加题：闭包是什么时候被销毁的？- 当它不被任何其他的对象引用的时候。

附加题：闭包的优点 - 缓存变量将变量隐藏起来不被GC回收

附加题：闭包的缺点 - 内存消耗，闭包产生的变量无法被销毁；性能问题，由于闭包内部变量优先级高于外部变量，所以需要多查找作用域链的一个层次，一定程度影响查找速度

附加题：js闭包是如何处理父函数中let形成块级作用域中定义的变量的？

区别在于每次for循环里面都是一个单独的块极作用域，而let就是块极作用域里面声明变量，可以理解为每次执行for的时候都是一个新的i，所以在for循环的块中i不一样可以理解为每次循环都保存了一个变量i这个i属于对应的那次执行的块极作用域，而var是函数作用域，for循环无论执行多少次，都是去最近的堆栈里面找，而不是块中找，所以只有一个i，现在的i是10

十、函数柯里化Currying

柯里化是把接受多个参数的函数转变为单一参数的函数，返回接受余下的参数且返回结果的新函数的技术。主要是通过闭包管理，支持链式调用且每次运行返回一个function

附加题：柯里化有什么好处？ - 参数复用、提前确认、延迟运行

十一、对作用域的理解 - 作用域分为三种：全局、函数、块级的

附加题：什么是作用域链

使用一个变量时，JS引擎会尝试在当前作用域下寻找该变量，如果没找到，再到它的上层作用域寻找，以此类推，直至找到该变量或是查找至全局作用域，如果在全局作用域里仍然找不到该变量，它就会在全局范围内隐式声明该变量(非严格模式下)或是直接报错。整个查找形成的链条就是作用域链

附加题：作用域链的作用？ - 可以访问到外层环境的变量和函数

十二、执行上下文类型？- 全局-this指向全局对象window；函数-调用创建 eval-执行在eval函数创建

附加题：执行上下文的生命周期？创建阶段 → 执行阶段 → 回收阶段

附加题：什么是执行上下文栈？程序执行默认进入全局执行上下文中，此时如果在全局环境中调用了一个函数, 就会为整个函数创建一个函数执行上下文并且将这个上下文压入执行栈之中。只要有新的执行上下文，它被压入现有栈的顶部。浏览器永远会执行当前栈中顶部的执行上下文

附加题：如何创建执行上下文？

首先会创建一个window对象，并将this进行绑定，如果是在全局执行上下文中，this指向window，如果是函数执行上下文，那么this的指向取决于函数如何调用，如果是被一个引用对象调用那么this就指向整个引用对象，否则就指向window/undefined；然后创建词法环境，用来存储变量/函数与实际对象或者原始数据的映射，然后创建变量环境，将持有变量声明语句在执行上下文中创建的绑定关系，最后完成对变量的分配，然后执行代码

附加题：JS预解析？预解析分为变量预解析和函数预解析，这两种预解析都是将当前的声明提升至作用域的最前面，只提升声明不提升赋值/调用；函数表达式的写法不存在函数提升；函数提升优先级高于变量提升，且不会被同名变量声明时覆盖，但是会被同名变量赋值后覆盖

1. 为什么要有垃圾回收？V8引擎会自动申请分配内存不需要手动管理内存

附加题：垃圾回收产生的缺点 - 在写代码的时候不注意，造成循环引用等情况，导致内存泄露。

附加题：内存结构分配 -V8最大内存在64位系统下大约为1.4GB，在32位系统下大约为700MB。

附加题：v8 中的垃圾回收机制分为三种 - 1.Scavenge工作在新生代把from space中的存活对象移至to space 2.Mark-Sweep标记清除，新生代的某些对象由于过度活跃会被移至老生代，此时对老生代中活对象进行标记，并清理死对象3. Mark-Compact标记整理。

附加题：标记清除 - 当变量进入执行环境时，就标记这个变量进入环境是不能被回收的，因为他们正在被使用。当变量离开环境时就会被标记为离开环境会被内存释放。垃圾收集器在运行的时候会给存储在内存中的所有变量都加上标记。然后它会去掉环境中的变量以及被环境中的变量引用的标记。而在此之后再被加上标记的变量将被视为准备删除的变量，因为环境中的变量已经无法访问到这些变量了。最后垃圾收集器完成内存清除工作，销毁那些带标记的值，并回收他们所占用的内存空间。

附加题：引用计数 - 就是跟踪记录每个值被引用的次数。当声明了一个变量a并赋值引用类型b，b引用次数就是1。如果此时 a=c，那么b引用次数就减1变为0，说明这个变量已经没有价值，在垃圾回收期下次再运行时，这个变量就会被释放。引用计数会引起循环引用的问题：例如：obj1和obj2通过属性进行相互引用，两个对象的引用次数都是2

附加题：node 的内存管理跟垃圾回收机制有了解过吗？- 首先分两种情况：V8 将内存分成新生代空间和老生代空间， 新生代空间用于存活较短的对象内部细化为from空间与to空间当from空间被占满时会启动Scavenge GC算法将存活的对象从from space转移到to space然后清空from space然后from space 与 to space 互换然后完成一次新生代 GC老生代空间用于存活时间较长的对象从新生代空间转移到老生代空间的条件是经历过一次以上 Scavenge GC 的对象当 to space 体积超过 25%，标记清除算法标记存活的对象未被标记的则被释放，标记的方式分为增量标记即小模块标记在代码执行间隙执GC会影响性能以及并发标记不阻塞 js 执行

附加题：WeakMap与垃圾回收有何关系 - WeakSet 不能遍历，是因为成员都是弱引用，随时可能消失，遍历机制无法保证成员的存在，很可能刚刚遍历结束，成员就取不到了。WeakSet 的一个用处是储存 DOM 节点，而不用担心这些节点从文档移除时，会引发内存泄漏。

附加题:哪些情况会导致内存泄漏（6种）

1. 全局变量。对于全局变量很难判断什么时候不用这些变量无法正常回收。2. 闭包引起的内存泄露。解决办法：将事件处理函数定义在外部，解除闭包。3. 定时器setInterval或者setTimeout不再需要使用时，且没有被清除，导致定时器的回调函数及其内部依赖的变量都不能被回收，就会造成内存泄漏。4. 事件监听。垃圾回收机制不好判断事件是否需要被解除导致callback不能被释放，此时需要手动解除绑定。5. 元素引用没有清理。解决办法：移除元素后，手动设置元素的引用为null。6. console。传递给console.log的对象是不能被垃圾回收，可能会存在内存泄漏。解决办法：清除不必要的console。

附加题：如何减少垃圾回收？- 清空一个数组length=0赋值会创建新对象，对象尽量复用不用=null，在循环中的函数表达式，如果可以复用尽量放在函数的外面。

十四、列出对数组产生副作用和没有副作用的方法

产生副作用pop、push、shift、unshift、reverse、sort、splice、map

没有副作用concat、reduce、some、every、filter、forEach、slice、join、toString、toLocaleString、valueOf

附加题：常见的类数组document.getElementsByTagName、document.querySelectorAll、arguments

附加题：在 js 中如何把类数组转化为数组1. Array.prototype.slice.call(arrayLike); 2. Array.prototype.splice.call(arrayLike, 0)3. Array.prototype.concat.apply([], arrayLike)4. Array.from(arrayLike);5.扩展运算符

附加题：为什么函数的arguments参数是类数组而不是数组？如何遍历类数组?

arguments是一个对象，它的属性是从0开始依次递增的数字，还有callee和length等属性，与数组相似；但是它却没有数组常见的方法属性，如forEach, reduce等，所以叫它们类数组

1. 生成100个元素为1的数组Array.from(Array(100), x => 1)Array(100).fill(1)

附加题：如何过滤数组中的 falsy value？比如false, null, 0, "", undefined, NaN, 使用array.filter(Boolean)

附加题：如何判断一个数组是否包含某个值 - Array.prototype.includes()

附加题：如何判断某一个值是数组?(共5种) 1. Object.prototype.toString.call(obj).slice(8,-1) === 'Array'

2.原型链做判断obj.\_\_proto\_\_ === Array.prototype 3. Array.isArray 4. a instanceof Array 5. Array.prototype.isPrototypeOf

附加题：如何把一个数组 Array 转化为迭代器 Iterable？list[Symbol.iterator]()

附加题：数组里面有10万个数据，取第一个元素和第10万个元素的时间相差多少

几乎没有时间相差，数组是在计算机内存空间中分配一段连续的内存空间，并会记录下索引为`0`的内存地址。当需要访问索引为`10万`的数据，计算机会进行计算，先找到索引为`0`的内存地址，在此基础上 `+ 10万` 即可以拿到索引为`10万`的数据。时间复杂度为常数级别： `O(1)`，这点计算时间对于内存来讲相当于没有，所以几乎没有时间相差

1. 如何在 url 中传递数组 `a=3&a=4&a=5` 或者 `a[]=3&a[]=4&a[]=5` 或者 `a[0]=3&a[1]=4&a[2]=5`
2. Array.from() 和 Array.of() 的使用及区别？

Array.from将伪数组对象或可遍历对象转换为真数组。Array.of将一系列值转换成数组，会创建一个包含所有传入参数的数组，而不管参数的数量与类型，解决了new Array行为不统一的问题。

附加题：forEach和map方法有什么区别？1. forEach针对每一个元素执行提供的函数对数据的操作会改变原数组无返回值；2. map方法不会改变原数组的值返回新数组内值为原数组调用函数处理之后的值；

附加题：let unary = fn => val => fn(val)；['1.1', '2', '0.3'].map(unary(parseInt))；- [1,2,0]

附加题：for...in和for...of的区别 - for...of 遍历获取的是对象的键值，for…in 获取的是对象的键名；for...in会遍历对象的整个原型链，性能非常差而 for...of 只遍历当前对象不会遍历原型链；对于数组的遍历for…in 会返回数组中所有可枚举的属性(包括原型链上可枚举的属性)，for...of 只返回数组的下标对应的属性值；

总结： for...in 循环主要是为了遍历对象而生，不适用于遍历数组；for...of 循环可以用来遍历数组、类数组对象，字符串、Set、Map 以及 Generator 对象。

1. JavaScript 中数组是如何存储的？

同种类型数据的数组分配连续的内存空间，存在非同种类型数据的数组使用哈希映射分配内存空间

十八、什么是继承

继承主要是利用构造函数以及原型链实现继承效果。它可以让实例使用父类的所有属性以及方法并且可以直接在子类上扩展新的属性或方法，使用继承可以提高我们代码的复用性减少代码量降低开发成本。

附加题：js实现继承的原理? - 复制父类的属性和方法来重写子类的原型对象

十九、ES5继承的方式 1. 原型链继承new2. 借用构造函数继承call/apply执行构造函数改变this指向问题3. 组合继承call/apply + new4. 原型式继承Object.create5. 寄生继承6. 寄生组合式继承7. class

附加题：原型链继承new将父类的实例作为子类的原型继承另一个原型对象的所有属性和方法最终形成原型链Cat.prototype = new Animal()Cat.prototype.constructor = Animal原型对象的实例丢失手动矫正

缺点：1. 来自原型对象的所有属性被所有实例共享2. 创建子类实例时，无法向父类构造函数传参

附加题：借用构造函数继承call/apply即执行构造函数改变this指向问题等于是复制父类实例属性给子类

function Cat () { Animal.call(this) } 优点：1. 解决了原型链继承的子类实例共享父类引用属性的问题2. 创建子类实例时，可以向父类传递参数3. 可以实现多继承（call多个父类对象）缺点：1. 只能继承父类的实例属性和方法（定义在this上的），不能继承原型属性/方法（定义在prototype上的）2. 无法实现函数复用，每个子类都有父类实例函数的副本，影响性能

附加题：组合继承call/apply + new本质：将原型链和借用构造函数的技术组合到一块。使用原型链实现对原型属性和方法的继承，而通过构造函数来实现对实例属性的继承。借用构造函数 + 原型链

缺点：1. 调用了两次父类的构造函数，导致父类通过this声明的属性/方法，生成两份的问题, 造成了一定的性能消耗, 导致内存占⽤过多2. 原型链上下文丢失：子类和父类通过prototype声明的属性/方法都存在于子类的prototype上

附加题：原型式继承 Object.create()本质不自定义类型的情况下，临时创建一个构造函数借助已有的对象作为临时构造函数的原型，然后在此基础实例化对象，并返回；通过Object.create返回一个对象其\_\_proto\_\_指向传入的参数，也就实现返回的对象继承参数对象 优点：继承可以让原型链丰富，根据需求定制不同的原型链，不会存在内存浪费的情况，原型只会保留一份，用到的时候调用就行还能节省空间

缺点：原型的引用类型属性会在各实例之间共享

附加题：寄生继承本质在原型式继承得到对象的基础上，在内部再以某种方式来增强对象，然后返回；通过Object.creat(original)的方式 优点：寄生式继承在主要考虑对象而不是自定义类型和构造函数的情况下

非常有用 缺点：寄生式继承为对象添加函数不能做到函数复用，因此效率降低

附加题：手动实现一个Object.create()；function create (obj) { function A () {} A.prototype = obj; // 将被继承的对象作为空函数的prototype ；return new A();返回new期间创建的新对象此对象的原型为被继承的对象,通过原型链查找可以拿到被继承对象的属性

附加题：寄生组合式继承 组合继承无论什么情况下都会调用两次超类型的构造函数，并且创建的每个实例中都要屏蔽超类型对象的所有实例属性。

附加题: class继承都做些什么？es6继承的本质1. 将子类Child的\_\_proto\_\_属性指向了父类的Parent 2. 子类的原型对象Child.prototype的\_\_proto\_\_属性指向了父类的原型对象Parent.prototype 3. 通过关键字super调用（实际上类似于使用call/apply执行父类的构造函数并且改变this的指向), 使得子类具有父类的方法与属性  
附加题：ES6继承, 构造函数是什么时候执行的 - 构造函数是在实例化的时候执行的

附加题：ES6继承静态方法/属性和实例方法/属性是什么时候挂载 - 在外界只能创建实例后通过实例取

附加题:ES5/ES6 的继承除了写法以外还有什么区别？

1. ES5的继承机制实际上先创建子类实例对象然后在将父类的属性和方法绑定在子类上通过apply/call的方式但是ES6的继承机制是先创建父类的实例对象this即先调用super方法，然后再用子类的构造函数修改this 2. ES5继承时通过原型或构造函数机制来实现；ES6通过class关键字定义类，类之间通过extends关键字实现继承里面有构造方法constructor，子类必须在constructor方法中调用super方法，否则新建实例报错。因为子类没有自己this对象而是继承了父类this对象然后对其进行加工。如果不调用super方法子类得不到this对象 3. ES6在继承语法上不仅继承了类原型对象还继承了类的静态属性和静态方法

附加题：什么方法可以设置对象的\_\_proto\_\_属性？new操作符，Obejct.create, Object.setPrototypeOf

附加题：为什么⼀定要通过桥梁的⽅式让 Child.prototype 访问到Parent.prototype？因为是引用类型，在给 Child.prototype 添加新的属性或者⽅法后，Parent.prototype 也会随之改变

附加题：如何实现Date的继承 要调用Date上方法的实例对象必须通过Date构造出来，否则不允许调用Date的方法，主要是因为规则规定日期对象只能通过构造函数来实例化， v8引擎底层代码中有限制如果调用对象的[[Class]]不是Date，则抛出错误

十九、对象创建的方式有哪些？- 通用字面量的形式直接创建对象，

1. 工厂模式。用函数来封装创建对象的细节，从而通过调用函数来达到复用的目的。缺点：创建出来的对象无法和某个类型联系起来，它只是简单的封装了复用代码，而没有建立起对象和类型间的关系。

2. 构造函数模式。优点是：所创建的对象和构造函数建立起了联系 缺点：造成了不必要的函数对象的创建浪费了不必要的内存空间

3. 原型模式，因为函数具备prototype 属性它包含了通过构造函数创建的所有实例都能共享的属性和方法。优点：解决了函数对象的复用问题 缺点：无法传入参数来初始化值，引用类型的话所有的实例将共享对象

4. 组合使用构造函数模式和原型模式。通过构造函数来初始化对象的属性，通过原型对象来实现函数方法的复用。缺点：对于代码的封装性不够好。

5. 动态原型模式：将原型方法赋值的创建过程移动到了构造函数的内部，通过对属性是否存在的判断，可以实现仅在第一次调用函数时对原型对象赋值一次的效果。

6. 寄生构造函数模式，基于一个已有的类型在实例化时对实例化的对象进行扩展。无法实现对象的识别。

二十、Object.keys与Object.getOwnPropertyNames有何区别

前者返回一个对象可枚举属性的字符串数组后者列出所有属性值(包括可枚举与不可枚举)

二十一、 如何遍历一个对象1. Reflect.ownKeys 2. Object.keys 3. for...in 4. object.entries

附加题：如何遍历对象的属性？Object.keys，Object.getOwnPropertyNames for ... in

二十二、在 JS 中如何监听 Object 某个属性值的变化 - new Proxy

二十三、什么是箭头函数

箭头函数不绑定this会捕获其所在上下文的this作为自己的this不能用作构造函数不绑定arguments使用rest参数解决同时没有super和new.target，使用call/apply/bind并不会改变箭头函数中this指向，只会传入参数并调用函数，调用bind时只会返回一个预设参数的新函数并不会改变这个新函数的this指向。

箭头函数和普通函数的区别

箭头函数没有自己的this对象，不可以当作构造函数会抛错误，不可以使用arguments对象，该对象在函数体内不存在可以用rest代替，不可以使用yield命令因此箭头函数不能用作Generator函数

二十四、什么是纯函数 - 输出仅由输入决定，每一个固定的输入总是返回相同的输出不产生副作用

二十五、什么是尾调用，使用尾调用有什么好处？

尾调用指的是函数的最后一步调用另一个函数。代码执行是基于执行栈的，所以当在一个函数里调用另一个函数时，会保留当前的执行上下文，然后再新建另外一个执行上下文加入栈中。使用尾调用的话，因为已经是函数的最后一步，所以这时可以不必再保留当前的执行上下文，从而节省了内存，这就是尾调用优化。但是 ES6 的尾调用优化只在严格模式下开启，正常模式是无效的。

二十六、this的四种绑定规则 - 默认，隐式，显示，new，箭头函数

1.默认绑定：在非严格模式下this指向window在严格模式下this指向undefined

2.隐式绑定：函数在调用位置是否具有上下文对象，如果有，this就会隐式的绑定在这个对象上

this指向`最后一层`调用函数的对象即就近原则 `obj1.obj2.foo() => 此时this指向obj2`；隐式绑定丢失：实际上就是在函数调用过程中并没有上下文对象只是对函数引用就会造成隐式绑定丢失此时this=window

附加题：有两种情况容易发生隐式丢失问题：1.使用另一个变量来给函数取别名 2.将函数作为参数传递时会被隐式赋值，回调函数丢失this绑定 3. 回调函数`的隐式绑定丢失 test(obj2.foo)`

3. 显示绑定：在某个对象上面强制绑定this 使用call,apply, bind方法

注意：传入的不是对象：传入原始值（string,number,boolean） 将传入的原始值转换成为对象

传入null， undefined，call，apply，call将会将其忽略，this绑定方式采用默认的方式

4. new绑定: 使用函数调用的过程中，this会自动绑定在new期间创建的对象上

附加题：this绑定规则优先级 - 显示 > 隐式 > 默认 new > 隐式 > 默认 new与显示不能相比较会报错

附加题：箭头函数的this指向不会使用上述的四条规则

1. 箭头函数继承于上层第一个不是箭头函数的this的指向2. 箭头函数一旦绑定的了上下文你就不会改变

二十七、异步编程的实现方式？

1.回调函数 缺点是会造成回调函数地狱代码耦合度太高不利于代码的可维护。2. Promise可以链式调用。缺点：造成多 then 的链式调用可能会造成代码的语义不够明确。3. generator当遇到异步函数执行的时候，将函数执行权转移出去，当异步函数执行完毕时再将执行权给转移回来。可以以同步的顺序来书写。缺点额外需要一个自动执行 generator 的机制 4. async/await：generator+promise 实现的一个自动执行的语法糖内部自带执行器await相当于返回promise 函数将会等待promise状态变为 resolve 再继续向下执行。

二十八、setTimeout、Promise、Async/Await 的区别？

1. setTimeout异步任务，当同步任务执行完毕之后才会被调用执行

2. Promise是一个微任务，本身是同步的立即执行函数， 当在executor中执行resolve或者reject的时候, 此时是异步操作， 会先执行then/catch等，当主栈完成后，才会去调用resolve/reject中存放的方法执行

3. async/await：本质上是promise的语法糖，await通过返回一个Promise对象来实现同步的效果

二十九、为什么 setTimeout 有最小时延 4ms ?

根据规范如果设置的timeout小于0，则设置成0，如果setTimeout的嵌套层级超过了五层，并且 timeout 小于 4ms，则设置 timeout 为 4ms，各大浏览器的厂商对于setTimeout都有自己的设计chrome在不满足嵌套层级时最小延迟时间设置为 1ms。设置 minimumInterval = 4ms，是因为chromemium希望设置更低的延迟时间但是由于一些网站对于setTimeout的不良使用过小的延迟时间会造成CPU-spinning计算机没有办法进入低功耗模式。所以chromium做了些 benchmark 测试选定了 4ms

附加题：追求低延迟为什么不设置0ms？

如果浏览器允许 0ms，会导致 JavaScript 引擎过度循环，也就是说如果浏览器架构是单进程的，那么可能网站很容易无响应。因为浏览器本身也是建立在 event loop 之上的，如果速度很慢的 JavaScript engine 通过 0ms timer 不断安排唤醒系统，那么 event loop 就会被阻塞。用户同时遇到 CPU spinning 和基本挂起的浏览器，没有任何响应

附加题：JavaScript中setTimeout的实现原理

javascript是单线程执行的，无法同时执行多个任务，当一段代码正在执行的时候，所有的后续的任务都必须进行等待，形成一个队列，当前任务执行完毕之后在从队列当中取出一个任务进行执行，也就是阻塞式运行。如果代码中设定了一个 setTimeout，那么浏览器便会在合适的时间，将代码插入任务队列，如果这个时间设为 0，就代表立即插入队列，但不是立即执行，仍然要等待前面代码执行完毕。所以 setTimeout 并不能保证执行的时间，是否及时执行取决于 JavaScript 线程是拥挤还是空闲。

三十、什么是eventloop（事件循环）

js单线程当遇到异步事件会将这个事件放在事件队列当中，继续执行执行栈中的其他任务。当前执行栈中所有任务都执行完主线程处于闲置状态时，主线程会去查找事件队列是否有任务，主线程从中取出排在第一位的事件并把这个事件对应回调放入执行栈中，然后执行其中的同步代码如此反复就被称为事件循环

异步任务分两类：微任务（micro task）和宏任务（macro task）

附加题：微任务包括： promise 的回调、node -process.nextTick 、MutationObserver

宏任务包括： script，setTimeout 、setInterval 和setImmediate 还有如 I/O 操作，UI 渲染等。

附加题：宏任务和微任务的优先级? - 在微任务优先级高于宏任务

附加题：用单线程模式工作的原因？

js使用单线程主要是为了避免复杂的线程同步问题，如果在javaScript线程当中有多个任务需要执行，其中一个线程修改了某一个dom元素，另外一个线程又删除了某一个Dom元素，那此时浏览器就不知道该以那一次的线程结果为准，这样设计的优点：安全简单，缺点：遇到耗时的任务就会导致假死的情况

附加题：JS 中异步任务为什么分为微任务与宏任务

主要是为了微任务当中注册的微任务可以插队，如果不区分任务类型，新注册的任务就要等下一个宏任务执行完毕之后在执行，中间需要的状态就没办法在下一个任务中同步

附加题：请用一句话描述 try catch 能捕获到哪些 JS 异常?

线程执行已经进入try catch但try catch未执行完的时候抛出来的。Promise没有异常，Promise 的异常都是由reject和Promise.prototype.catch来捕获是因为 Promise 在执行回调中都用try catch 包裹起来了，其中所有的异常都被内部捕获到了，并未往上抛异常。

三十一、为什么又模块化？- 方便维护，高内聚低耦合，变量冲突问题，

1. CommonJS，它通过 require 来引入模块，通过 module.exports 定义模块的输出接口。这种模块加载方案是服务器端的解决方案，它是以同步的方式来引入模块的，因为在服务端文件都存储在本地磁盘，所以读取非常快，所以以同步的方式加载没有问题。但如果是在浏览器端，由于模块的加载是使用网络请求，因此使用异步加载的方式更加合适。

2. AMD，这种方案采用异步加载的方式来加载模块，模块的加载不影响后面语句的执行，所有依赖这个模块的语句都定义在一个回调函数里，等到加载完成后再执行回调函数。require.js 实现了 AMD 规范

3. CMD，这种方案和 AMD 方案都是为了解决异步模块加载的问题，sea.js 实现了 CMD 规范。它和require.js的区别在于模块定义时对依赖的处理不同和对依赖模块的执行时机的处理不同。4. 自执行函数iife 5. ES6 提出的方案，使用 import 和 export 的形式来导入导出模块

三十二、什么是前端工程化？

以前端的视角与前端能力去完成软件工程的一个实战，并将前端工程方法系统化的应用在前端开发中（也就是实际的落地），以工程方法（系统、严谨、可量化的方法）开发、运营和维护前端应用程序，基于业务诉求，梳理出最符合当前需要的架构设计

目前我是做了技改的owner负责人，从业务视角对前端工程化进行了划分和整合，针对当前公司的业务场景做了以下的划分：

1. 提取公共配置设计了针对vue3业务场景的脚手架，目前设想已经完成

2. 将公共组件库划分为两大部分：UI类公共组件库、业务功能组件库、大屏组件库、低代码平台系统前台展示库，地图库

3. 引用开源框架wujie作为此次技改的技术支撑，为更换技术栈新老业务代码可以并存提供基础

4. 封装公共主应用以供业务场景支撑，并抽离出公共配置项以供其他两条业务线后续快速接入做准备

5. 抽离出公共业务微服务支持定制，做业务侧支撑

6. 每一个大的功能类别划分为一个工程，支持打包售卖，业务粒度扩大，更适合定制地方部署场景

三十三、前端工程化发展史

1. 前后端分离： B/S架构兴起，有了前后端之分

2. 模块化：模块复用

3. 自动化：前端框架，自动化，构建系统（webpack）

4. 最佳实践： 开箱即用dva

5. vite, esm, wasm, 低代码能力

三十四、如何设计一个脚手架

主要是分为3方面：准备阶段、开发阶段以及发布流程阶段

准备阶段：1. 技术选型：首先要对公司业务进行技术选型，考虑到既有人员技术栈以及即往项目技术栈的基础，我们采取 vue3全家桶 的技术栈 2. 资源划分：项目的初始资源划分为：埋点、地图、UI组件库的代码植入 3.打包构建在无性能瓶颈的前提下采用vue-cli的构建方法

开发阶段：1. 制定开发模式，微前端子应用可以实现独立运行，具备代码植入 2. 本地mock服务，提取公共的接口服务 3. 单元测试 + e2e测试

发布流程- git commit规范、changeLog规范、打包构建、部署验收

ES6面试题

一、let、const、var的区别

1. let和const具有块级作用域，var不存在 2.var存在变量提升，let和const不存在变量提升

3. var声明的变量为全局变量，并且会将该变量添加为全局对象的属性，但是let和const不会

4. var声明变量时可以重复声明变量后面的覆盖前面的。const和let不允许重复声明变量。

5.在使用let、const命令声明变量之前，该变量都是不可用的也就是暂时性死区。var不存在

6. 在变量声明时，var 和 let 可以不用设置初始值。而const声明变量必须设置初始值。

7. let创建的变量是可以更改指针指向（可以重新赋值）。但const声明的变量是不允许改变指针的指向。

附加题：块级作用域解决了ES5中的什么问题？

1.内层变量可能覆盖外层变量 2. 用来计数的循环变量泄露为全局变量

附加题：const对象的属性可以修改吗？

const保证的并不是变量的值不能改动，而是变量指向的那个内存地址不能改动。但对于引用类型的数据变量指向数据的内存地址，const只能保证这个指针是固定不变的，其他的不能控制

二、如果new一个箭头函数的会怎么样？

箭头函数没有prototype也没有this指向，更不可以使用arguments参数，所以不能New一个箭头函数。

附加题：箭头函数的this指向哪⾥？- 箭头函数没有⾃⼰的this需要捕获所在上下⽂this作为⾃⼰的 this

三、扩展运算符的作用及使用场景

1. 对象扩展运算符用于取出参数对象中的所有可遍历属性，拷贝到当前对象之中

2. 数组的扩展运算符可以将一个数组转为用逗号分隔的参数序列，且每次只能展开一层数组，可以用于将数组转换为参数序列，复制数组，合并数组，将字符串转为真正的数组

四、如何提取高度嵌套的对象里的指定属性？ - 使用解构赋值

附加题：对 rest 参数的理解?

扩展运算符被用在函数形参上时，它还可以把一个分离的参数序列整合成一个数组, ...rest运算符它可以把函数的多个入参收敛进一个数组里。经常用于获取函数的多余参数

五、模板字符串的优势

1. 易读，代码整体的质量都变高 2. 在模板字符串中，空格、缩进、换行都会被保留 3.支持“运算”式

六、ES6中还新增了一系列的字符串方法 - includes、startsWith、endsWith、repeat

七、ES6模块与CommonJS模块有什么异同？

commonJS是模块的浅拷贝，es6是对模块的引用只存只读，CommonJS和ES都可以对引⼊的对象进⾏赋值，即对对象内部属性的值进⾏改变。

八、什么是 Iterable 对象，与 Array 有什么区别

实现了 [Symbol.iterator] 属性的对象即是 Iterable 对象，然后可以使用操作符 for...of 进行迭代

九、es6新增了那一些新特性

1. 新增关键词：let / const / async / await / yield 2. 新增关键特性 ：模版字符串、解构赋值 / 展开操作符、promise / ESModule、for...of、async/await、迭代器接口 iterator、可选链 3. 新增方法、对象等内容Proxy / Reflect 和Set / Map

十、generator 是如何做到中断和恢复的

利用协程，协程是一种比线程更加轻量级，可以把协程看成是跑在线程上的任务一个线程上多个协程但是在线程上同时只能执行一个协程。它的运行流程大致如下：协程A开始执行、协程A执行到某个阶段，进入暂停，执行权转移到协程B、协程B执行完成或暂停，将执行权交还A、协程A恢复执行，所以协程遇到yield命令就暂停，等到执行权返回，再从暂停的地方继续往后执行。它的最大优点，就是代码的写法非常像同步操作，如果去除yield命令，简直一模一样。

十一、promise解决什么样的问题

1. 解决回掉地狱2. 解决异步并发问题promise.all 3. 错误处理变得更加容易

附加题：什么是promise？

Promise是异步编程的一种解决方案它是一个构造函数，接收一个函数作为参数，返回一个Promise实例。有三种状态，分别是pending、resolved 和 rejected。实例的状态只能由 pending 转变 resolved 或者rejected 状态，并且状态一经改变无法再被改变了。状态的改变是通过 resolve和 reject函数来实现的，可以在异步操作结束后调用这两个函数改变 Promise 实例的状态，它的原型上定义了一个 then 方法，使用这个 then 方法可以为两个状态的改变注册回调函数。回调函数属于微任务会在本轮事件循环末尾执行。

附加题：Promise的缺点

无法取消Promise，一旦新建它就会立即执行，无法中途取消。如果不设置回调函数，Promise内部抛出的错误，不会反应到外部。当处于pending状态时无法得知目前进展到哪一个阶段

附加题：Promise的特点

对象的状态不受外界影响。只有异步操作的结果可以决定是哪一种状态，其他的行为都不能。一旦状态改变就不会再变，任何时候都可以得到这个结果。promise对象的状态改变与事件完全不同，事件的特点是：如果你错过了它，再去监听是得不到结果的

附加题：为什么promise的then方法要返回一个 promise的实例 - 因为then方法实现了链式调用

附加题：为什么不效仿jquery方法的return this来实现链式调用？

因为promise是具有状态的，且在promise的excutor的回掉函数当中可能会具有异步任务，且promise的状态都是从pending状态转变成成功状态或者失败状态且不可逆，所以此处返回一个promise的实例刚好，因为新的promise的实例具有新的状态转变

附加题：为什么在新返回的promise的实例当中要不断的调用resolve函数？

因为then方法在返回值不是一个promise的时候，除抛出异常外，都会走到下一个promise的resolve函数当中，所以我们需要将此次的返回值直接传递给resolve函数使得此次的promise走向成功状态

附加题：excutor是被包裹在trycatch代码块当中执行的，所以会捕获最外层的异常，一旦promise内部存在异步任务且抛出异常的情况下，就不能够捕获到了，怎么办？在then方法继续包裹trycatch代码块

附加题：let promise2 = p.then(() => { then: function () {} })` 这是一个promise吗？是有then函数就是

附加题：promise的测试方法 promisea+ 当中提供了测试方法会调用deferred方法获取到promise实例

附加题：Promise then 第二个参数和catch的区别是什么? - 1. reject是用来抛出异常的，catch是用来处理异常的；2. reject是Promise的方法，而then和catch是Promise实例的方法

区别：如果resolve当中抛出去的异常会由catch捕获，但是reject函数当中抛出去的异常catch捕获不到，then的第二个参数和catch捕获错误信息的时候会就近原则，如果是promise内部报错，reject抛出错误后，then的第二个参数和catch方法都存在的情况下，只有then的第二个参数能捕获到，如果then的第二个参数不存在，则catch方法会捕获到。

附加题：return promise 与 return await promise 有何区别

return promise: 返回结果为 Promise {fulfilled 或者 rejected}

return await promise:返回结果为 Promise {pending}, 因为 async 函数总是返回一个 promise

十二、对async/await 的理解？

async/await其实是Generator 的语法糖，它能实现的效果都能用then链来实现，它是为优化then链而开发出来的。async 用于申明一个 function 是异步的，而 await 用于等待一个异步方法执行完成。async 函数返回的是一个 Promise 对象，如果在函数中 return 一个直接量，async 会把这个直接量通过 Promise.resolve封装成 Promise 对象。在没有 await 的情况下执行 async 函数，它会立即执行，返回一个 Promise 对象，并且绝不会阻塞后面的语句。这和普通返回 Promise 对象的函数并无二致。

附加题：await 到底在等啥？

一般来说，都认为 await 是在等待一个 async 函数完成。按语法说明await 等待的是一个表达式，这个表达式的计算结果是 Promise 对象或者其它值。因为 async 函数返回一个 Promise 对象也就是await 在等 async 函数，它等的实际是一个返回值。如果它等到的不是一个 Promise 对象，那 await 表达式的运算结果就是它等到的东西。如果它等到的是一个 Promise 对象，await 就忙起来了，它会阻塞后面的代码，等着 Promise 对象 resolve，然后得到 resolve 的值，作为 await 表达式的运算结果。

附加题：async/await的优势 代码表现上同步的方式实现异步

附加题：async/await对比Promise的优势 - 代码读起来更加同步写法优雅、错误处理友好、调试友好，

附加题：async/await 如何捕获异常 - 使用try catch代码块

十三、Map与WeakMap的区别

1. Map: 可使用任何数据类型作为 key，但因其在内部实现原理中需要维护两个数组，存储 key/value，因此垃圾回收机制无法回收

2. WeakMap: 只能使用引用数据类型作为 key。弱引用，不在内部维护两个数组，可被垃圾回收，但因此无法被遍历！即没有与枚举相关的 API，如 keys、values、entries 等

十四、 在 ES6 Class 中，super 的过程中做了什么？ 本质就是 call（this）

Ts面试题

1. 说说你对 TypeScript 的理解？与 JavaScript 的区别？

ts是js的类型的超集，支持ES6语法支持面向对象编程的概念，如类、接口、继承、泛型等

ts特性：类型批注和编译时类型检查以及类型推断、在编译过程中批注的内容和接口会在运行时利用工具擦除，可以使用接口来定义对象类型，可以使用枚举、Mixin、元组、泛型编程等手段

与js区别：ts是js的超集扩展了js的语法、ts可处理已有的js代码并只对其中的ts代码进行编译，在编写 ts的文件的时候就会自动编译成 js 文件

1. 说说 typescript 的数据类型有哪些

在js的基础之上拓展了tuple元组类型、enum枚举类型、any任意类型、void类型、never类型

附加题：高级类型

1.交叉类型&：将两个类型合并在一起 2. 联合类型｜：相当于或 3. 类型别名 4. 类型索引keyof 类似于 Object.keys 5. 类型约束extends 不同于在 class 后使用 extends 的继承作用，泛型内使用的主要作用是对泛型加以约束 6. 映射类型通过 in 关键字做类型的映射，遍历相当于forEach已有接口的 key 或者是遍历联合类型 7. 条件类型相当于三目运算符 T extends U ? X : Y

附加题：interface和type的区别

两者最大的区别在于，interface只能用于定义对象类型，而 type 的声明方式除了对象之外还可以定义交叉、联合、原始类型等，类型声明的方式适用范围显然更加广泛

附加题：用来获取对象的值 - K extends keyof T

附加题：P in keyof T - 相当于先拿到T当中所以的key得到F，在forEach遍历F得到P

1. 说说你对 TypeScript 中枚举类型的理解？应用场景？

枚举就是一个对象的所有可能取值的集合，类型可以分成：数字枚举、字符串枚举和异构枚举

1. 说说你对 TypeScript 中泛型的理解？应用场景？

泛型允许我们在强类型程序设计语言中编写代码时使用一些以后才指定的类型，在实例化时作为参数指明这些类型 在typescript中，定义函数，接口或者类的时候，不预先定义好具体的类型，而在使用的时候在指定类型的一种特性，例如interface ReturnItemFn<T> { (para: T): T }，泛型的索引类型和约束类型两种：1. 索引类型 keyof T 把传入的对象的属性类型取出生成一个联合类型，泛型 U 被约束在这个联合类型中

1. 多类型约束适应继承extend
2. 说说你对 TypeScript 装饰器的理解？应用场景？

@expression 是一种在不改变原类和使用继承的情况下，动态地扩展对象功能，是Object.defineProperty的语法糖，typescript是一个实验性特性，若要使用，需要在tsconfig.json文件启动experimentalDecorators，

类的装饰器可以装饰类、方法/属性、参数和访问器；当装饰器作为修饰类的时候，会把构造器constructor传递进去Person = addAge(function P() { ...})

1. 说说对 TypeScript 中命名空间与模块的理解？区别

模块：任何包含顶级 import 或者 export 的文件都被当成一个模块；命名空间目的就是解决重名问题

区别：命名空间是位于全局命名空间下的一个普通的带有名字的js对象。但就像其它的全局命名空间污染一样，它很难去识别组件之间的依赖关系，尤其是在大型的应用中。模块可以包含代码和声明。 不同的是模块可以声明它的依赖。在正常的TS项目开发过程中并不建议用命名空间，但通常在通过 d.ts 文件标记 js 库类型的时候使用命名空间，主要作用是给编译器编写代码的时候参考使用

1. 在vue当中怎么使用ts - 使用库函数vue-property-decorator
2. const和readonly的区别

const可以防止变量的值被修改，readonly可以防止变量的属性被修改。

1. 枚举和常量枚举的区别？

枚举：一个对象的所有可能取值的集合；常量枚举：只能使用常量枚举表达式并且不同于常规的枚举，它们在编译阶段会被删除。 常量枚举成员在使用的地方会被内联进来常量枚举不允许包含计算成员。

1. ts中undefined和void类型的区别

undefined：表示一个没有设置值的变量；void：用于标识方法返回值的类型，表示该方法没有返回值。

null和undefined只能赋值给void和它们各自

1. typescript 实现一个字符串类型去左侧空格 - type B = LeftTrim<A>;

十二、Pick 和Omit区别 - pick从对象中取出某些属性、oimit从对象中去除某些属性

Git版本管理

1. 常用的版本管理工具 - 集中式 svn和cvs 分布式git和 hg

附加题：版本控制系统的优点 - 记录文件所有历史变化、随时恢复到任意时间点、支持多功能并行开发

二、git工作原理

当我们通过git init创建或者git clone一个项目的时候，项目目录会隐藏一个.git子目录，其作用是用来跟踪管理版本库的，git并不只提取最新版本的文件快照，而是把代码仓库完整地镜像下来。Git 中所有数据在存储前都计算校验和然后以校验和来引用，Git用以计算校验和的机制叫做 SHA-1散列（hash，哈希）， 这是一个由 40 个十六进制字符0-9 a-f）组成字符串，基于 Git 中文件的内容或目录结构计算出来，当我们修改文件的时候，git就会修改文件的状态modified、staged、committed，文件状态对应的，不同状态的文件在Git中处于不同的工作区域。

三、fork, clone,branch区别

1. fork 只能对代码仓进行操作，且 fork 不属于 git 的命令，通常用于代码仓托管平台的一种“操作”

2. clone 是 git 的一种命令，它的作用是将文件从远程代码仓下载到本地，从而形成一个本地代码仓

3. branch与fork 很类似，fork得到的是一个新的自己的代码仓，而 branch是一个代码仓的一个新分支

四、说说Git 中 HEAD、工作树和索引之间的区别？

1. HEAD指针指向所在的分支当在某个分支上创建新的提交时，分支指针总是会指向当前分支的最新提交

2.工作树是查看和编辑的（源）文件的实际内容

3. 索引是放置要提交给git仓库文件的地方，如工作树的代码通过 git add 则添加到 git 索引中，通过git commit 则将索引区域的文件提交到 git 仓库中

五、说说对git pull 和 git fetch 的理解？有什么区别？

git fetch是将远程主机的最新内容拉到本地，用户在检查了以后决定是否合并到工作本机分支中，更新代码本地的库中master的commitID不变，远程的origin/master会发生变化

git pull 命令用于从另一个存储库或本地分支获取并集成(整合) git pull = git fetch + git merge 这样可能会产生冲突，需要手动解决

六、说说你对git stash 的理解？应用场景？

git stash保存当前工作进度会把暂存区和工作区的改动进行保存，这些修改会保存在一个栈上，后续可以在任何时候任何分支重新将某次的修改推出来，重新应用这些更改的代码；git stash默认会缓存添加到暂存区的修改staged changes和跟踪的但并未添加到暂存区的修改unstaged changes，不会缓存在工作目录中新的文件untracked files和被忽略的文件ignored files，都想缓存就使用-a

应用场景：解决多人协作日常更新代码、在迭代开发中紧急解决线上问题

七、说说你对git rebase 和 git merge的理解？区别？

merge合并分支会新增一个merge commit，然后将两个分支的历史联系起来，是一种非破坏性的操作，对现有分支不会以任何方式被更改，但是会导致历史记录相对复杂

rebase会将整个分支移动到另一个分支上有效整合了所有分支上的提交，主要的好处是历史记录更加清晰，是在原有提交的基础上将差异内容反映进去，消除了 git merge所需的不必要的合并提交

八、说说你对git reset 和 git revert 的理解？区别

reset用于回退版本，可以遗弃不再使用的提交，执行遗弃时，需要根据影响的范围而指定不同的参数，可以指定是否复原索引或工作树内容

revert在当前提交后面，新增一次提交，抵消掉上一次提交导致的所有变化，不会改变过去的历史，主要是用于安全地取消过去发布的提交

区别：1. git revert是用一次新的commit来回滚之前的commit，git reset是直接删除指定的commit 2. git reset 是把HEAD向后移动了一下，而git revert是HEAD继续前进，只是新的commit的内容和要revert的内容正好相反，能够抵消要被revert的内容 3. 在回滚这一操作上看，效果差不多。但是在日后继续 merge 以前的老版本时有区别4. git revert是用一次逆向的commit“中和”之前的提交，因此日后合并老的branch时，之前提交合并的代码仍然存在，导致不能够重新合并 5. 但是git reset是之间把某些commit在某个branch上删除，因而和老的branch再次merge时，这些被回滚的commit应该还会被引入 6. 如果回退分支的代码以后还需要的情况则使用git revert， 如果分支是提错了没用的并且不想让别人发现这些错误代码，则使用git reset

九、npm/yarn install 原理

执行npm/yarn install之后，包如何到达项目node\_modules当中。其次，node\_modules内部如何管理依赖。执行命令后，首先会构建依赖树，然后针对每个节点下的包，会经历下面四个步骤:

1. 将依赖包的版本区间解析为某个具体的版本号 2.下载对应版本依赖的 tar 包到本地离线镜像

3. 将依赖从离线镜像解压到本地缓存4. 将依赖从缓存拷贝到当前目录的 node\_modules 目录

然后，对应的包就会到达项目的node\_modules当中。

CSS面试题

一、盒模型宽度计算

普通的box-sizing: content-box; width只包含内容宽度=width + padding + border

怪异box-sizing: border-box; width = offsetWidth = (width + padding + border

二、margin纵向重叠 - margin纵向重叠取重叠区最大值，不进行叠加

三、margin的负值问题 - margin-top：元素会向上移动margin-left：元素会向左移动 margin-right: 右侧元素左移，自身不受影响margin-bottom: 下侧元素上移，自身不受影响

四、BFC是什么? 哪些属性可以构成一个BFC呢?

块级格式化上下文，一块独立渲染的区域，内部元素的渲染不会影响边界以外的元素

形成BFC的条件：1. float不设置成none 2. position是absolute或者fixed 3. overflow 不是visible 4. display是flex 或者 inline-block 5. 应用清除浮动

五、float

六、CSS 中解决浮动中高度塌陷的方案有哪些？（手写clearfix）

般不含包裹属性的div内部有浮动元素的话，那么这个浮动元素会让此div的高度塌陷。1. 父级定义伪类目前适用于所有浏览器的通用写法.clearfix::after { content: ''; display: table; clear: both; }.clearfix { \*zoom: 1; /\* 兼容IE6/7浏览器 \*/ } 2. .clearfix { overflow: hidden; } 3. 添加空的div标签.clearfix { clear: both; }

附加题样式单位1. em: 相对于自身字体大小的单位2. rem: 相对于html标签字体大小的单位3. vh： 相对于视口高度大小的单位20vh == 视口高度/10020 4. vw相对于视口宽度大小的单20vh == 视口/10020

附加题单位区别 px：绝对单位页面按精确像素展示；em：相对单位，基准点为父节点字体的大小，如果自身定义了font-size按自身来计算，整个页面内1em不是一个固定的值；rem：相对单位，可理解为root em, 相对根节点html的字体大小来计算；vh、vw：用于页面视口大小布局在页面布局上更加方便简单

附加题：说说设备像素、css像素、设备独立像素、dpr、ppi 之间的区别？

无缩放情况下，1个CSS像素等于1个设备独立像素 ；设备像素由屏幕生产之后就不发生改变，而设备独立像素是一个虚拟单位会发生改变；PC端中，1个设备独立像素 = 1个设备像素 （在100%，未缩放的情况下）；在移动端中，标准屏幕（160ppi）下 1个设备独立像素 = 1个设备像素；设备像素比（dpr） = 设备像素 / 设备独立像素；每英寸像素（ppi），值越大，图像越清晰

七、重排（回流）、重绘和合成

回流⼀定触发重绘，重绘不⼀定触发回流，重绘开销⼩，回流代价⾼

重排（回流）：渲染树中部分或者全部元素的尺寸、结构或属性变化，浏览器就重新渲染部分或全部文档

触发回流的操作行为：1. 初次渲染2. 窗口大小改变（resize事件）3. 元素属性、尺寸、位置、内容改变

4. 元素字体大小发生变化5. 添加/删除可见dom元素6. 激活css伪类（如 hover）7. 查询某些属性/调用某些方法，会刷新渲染队列，如果要使用最好将值缓存起来Client... offset... scroll... getComputedStyle getBoundingClientRect scrollTo

重绘: 某些元素的样式如颜色改变，但是不影响它在文档流中的位置，浏览器会对元素重新绘制，不再执行布局阶段，直接进入绘制阶段

附加题：如何进行绘制优化 1. 最小化重绘和重排：样式集中处理，使用添加新样式类名2. 使用absolute或者fixed脱离文档流3. 开启GPU加速，利用css属性transform opacity will-change等，tansform属性不会触发重绘和重排，使浏览器为元素创建⼀个GPU图层，这使得动画元素在⼀个独立的层中进行渲染，当元素内容没有改变就没必要进⾏渲染4. 使用visibility替换display:none，前者重绘，后者回流5. 尽量不使用table布局6. 动画实现酥服越快，回流次数越多，也可以选择使⽤requestAnimationFrame 7. 通过documentFragment创建⼀个DOM在它上⾯批量操作DOM，完成后再添加到文档中，只触发⼀次回流

八、垂直居中

1. 元素定宽定高

+ absolute + margin：0 auto；

+ absolute + calc

+ 利用绝对定位 + 设置 left和top 50% + margin-left和margin-top 以子元素一半宽高进行负值赋值

2. 元素不定宽高

+ 利用绝对定位 + 设置 left和top 50% + transform：translate(-50%, -50%)

+ line-height: initial

+ writing-mode: vertical-lr 改变文字的显示方向

+ 使用flex + align-item: center;

+ grid + align-items

3. 内联元素 - 单行文本父元素确认高度 height === line-height多行文本父元素确认高度 display: table-cell; vertical-align:middle

4. 块级元素

1. `display: flex; align-items: center;`, 不适用IE浏览器

2. 绝对定位`position: absolute; left: 0; right: 0; top: 0; bottom: 0;`+ `margin: auto;`, 不适用移动端使用, 但是对IE友好

3. 父元素 `display: table;`、子元素 `display: table-cell; vertical-align: middle;`

4. 父元素 `display: table-cell; vertical-align: middle;`

5. 多行文字垂直居中：父元素 `display: table-cell; vertical-align: middle;` 子元素 `display: inline-block;vertical-align: middle;`

6. child固定尺寸使用绝对定位：父元素 `position: relative;` 子元素 `position: absolute; height: 20px; width: 20px; top: 50%;left: 50%; margin-top: -10px;margin-left: -10px;`

7. 元素尺寸不定：父元素 `position: relative;` 子元素 `position: absolute; top: 50%;left: 50%; transform: translate(-50%, -50%);`

8. 利用伪元素： 父元素 `text-align: center;` 子元素 `display: inline-block; vertical-align: middle;` 父元素的伪类 `.p::before {content: ''; height: 100%; display: inline-block; vertical-align: middle;}`

9. child固定尺寸使用计算属性calc: `width: 100px;height: 100px; padding: -webkit-calc((100% - 100px) / 2); padding: -moz-calc((100% - 100px) / 2); padding: -ms-calc((100% - 100px) / 2); padding: calc((100% - 100px) / 2); background-clip: content-box;`

10. 改成块内元素：`display: inline-block; vertical-align: middle;`

九、水平居中 1. absolute + margin：0 auto；2. 利用绝对定位 + 设置 left和top 50% + transform：translate(-50%, -50%) 3. flex 4. grid

十、垂直水平居中1. flex2. 绝对定位的方式3. grid .container { display: grid; place-content: center; }

十一、文本溢出单行省略 - overflow: hidden; white-space: nowrap;-文本不换行 text-overflow: ellipsis;

多行省略 display: -webkit-box; -webkit-line-clamp: 2;-webkit-box-orient: vertical;overflow: hidden;

十二、父子级z-index层级关系

1. 当父元素设置了z-index,无论子元素设多小都会浮在父元素上面

2. 父元素的z-index >= 兄弟组件的z-index， 子元素z-index比父元素同级组件高会浮在所有元素上面

3. 父元素的z-index < 兄弟组件的z-index, 子元素z-index无论多高都会在最下面

十三、z-index压盖顺序

1. 只有定位元素(position属性明确为absolute、fixed或relative)可使⽤z-index

2. 如果z-index值相同，html结果在后⾯的ܴ盖֘在前⾯的

3. ⽗⼦都有z-index，⽗亲z-index数值⼩时，⼉⼦数值再⼤也没用

十四、margin和padding的值为百分比

当 padding padding 属性值为百分⽐的时候，如果⽗元素有宽度，相对于⽗元素宽度，如果没有，找其⽗辈元素的宽度，均没设宽度时，相对于屏幕的宽度。

附加题：为什么是相对width？

正常流中的⼤多数元素都会⾜够⾼以包含其后代元素（包括外边距），如果⼀个元素的上下外边距时⽗元素的 height 的百分数，就可能导致⼀个⽆限循环，⽗元素的 height 会增加，以适应后代元素上下外边距的增加，⽽相应的，上下外边距因为⽗元素 height 的增加也会增加，如此循环。

十四、link和@import的区别

1.@import是 CSS 提供的语法规则，只有导⼊样式表的作⽤；link是HTML提供的标签，不仅可以加载 CSS文件还可以定义RSS和ref连接属性 2. link和html同时加载，@import用到组件才会加载 4. 兼容性区别：@import是 CSS2.1 才有的语法，可在 IE5+ 才能识别；link标签作为 HTML 元素，不存在兼容性问题 5. DOM的操作性：可以通过 JS 操作 DOM ，插⼊link标签来改变样式；由于 DOM ⽅法是基于⽂໩的，⽆法使⽤@import的⽅式插⼊样式。6. 权重：link⽅式的样式权重⾼于@import的权重

十五、src和href的区别

src：⽤于替换当前元素将内容将会嵌⼊到文档中当前标签，会阻碍其他资源的下载 href⽤于在当前⽂档和引⽤资源之间确⽴联系浏览器遇到 href 就会并⾏下载资源并且不会停⽌对当前文档的处理。

十六、scss的理解 1. scss支持属性嵌套 2. 支持定义变量3. 支持混合语法 定义@mixin， 在属性中使用@include 4. 支持继承 @extend .A

十七、隐藏元素的方法 display: none visibility: hidden opacity: 0 clip/clip-path使⽤元素裁剪⽅法来实现元素的隐藏transform: scale(0,0 )都不会影响监听事件 position: absolute z-index: 负值

十八、display: none与 visibility: hidden的区别

1. display 会让元素完全从渲染树中消失，渲染时不会占据任何空间；visibility则相反2.display是⾮继承属性，⼦孙节点会随着⽗节点从渲染树消失，通过修改子孙节点的属性也⽆法显示；visibility相反，⼦孙节点消失是由于继承了hidden通过设置 visibility: visible可以让子孙节点显示；

3. 修改常规文档中元素的 display 通常会造成文档的重排但是修改 visibility 属性只会造成本元素的重绘

十九、使chrome支持12px以下的文字 - 1. 使用zoom：50% / 0.5; 2. 使用 -webkit-transform: scale(0.3)

二十、flex布局

弹性布局，用来为盒状模型提供最大的灵活性，采用flex布局的元素叫做flex container，它所有的子元素自动成为容器元素叫做flex item，容器默认是存在两条轴，水平主轴（main axis）和垂直的交叉轴（cross axis），项目默认沿主轴排列

附加题：最经常用的flex的容器属性就是：

+ flex-direction: 决定主轴的排列方向，row(水平)/column(垂直)/row-reverse/column-reverse

+ flex-wrap: flex item都在一根轴线上，也就是要不要换行， wrap/nowrap/wrap-reverse

+ flex-flow = flex-direction + flex-wrap，默认是 flex-grow: row nowrap;

+ justify-content: 主轴上的对齐方式 flex-start(左对齐)/flex-end(右对齐)/center(居中)/space-between(两端对齐)/space-around(每个项目两侧的间隔相等)

+ align-items: 交叉轴的对齐方式, flex-start | flex-end | center | baseline(第一行文字的基线对齐) | stretch(默认值)

+ align-content: 多根轴线的对齐方式,只有一根轴线，该属性不起作用

flex的项目属性：

+ order：项目的排列顺序，数值越小，排列越靠前，默认是0

+ flex-grow：项目的放大比例，默认为0，即如果存在剩余空间，也不放大

+ flex-shrink：项目的缩小比例，默认为1，即如果空间不足，该项目将缩小。负值对该属性无效；为1的时候等比缩小

+ flex-basis：在分配多余空间之前，项目占据的主轴空间，也就是项目占据的固定空间

+ flex = flex-grow + flex-shrink + flex-basis = 0 1 auto；快捷值：auto (1 1 auto) 和 none (0 0 auto)。

附加题：flex：1 代表什么 - flex: 1 1 auto; 也就是根据容器空间等比放大/缩小，项目不具备固定的空间

二十一、grid布局

Grid 布局即网格布局，是一个二维的布局方式，由纵横相交的两组网格线形成的框架性布局结构，能够同时处理行与列，擅长将一个页面划分为几个主要区域，以及定义这些区域的大小、位置、层次等关系

grid容器属性：grid-template-columns属性设置列宽，grid-template-rows属性设置行高

二十二、postion属性大概讲一下, static是什么表现? static在文档流里吗?

默认有5个值relative absolute fixed static inherit；static 为 position 属性的默认值，static 元素会遵循正常的文档流，且会忽略 top,bottom,left,right 等属性

二十三、css3新增的选择器属性 - :root :last-child :nth-child :not

附加题：CSS3新增了哪些新特性？transform 转换animation 动画 transition 过渡word-wrap

附加题：css实现动画方式 1. transition实现渐变动画 2.transform转变动画 3.animation实现自定义动画

附加题：CSS选择器的优先级 - 内联 > ID选择器 > 类选择器 > 标签选择器

附加题：CSS继承属性 - 字体系列属性、文本系列属性、元素可见性、列表属性、光标

二十四、什么是响应式设计？响应式设计的基本原理是什么？如何做？

响应式网站设计是一种网络页面设计布局，页面的设计与开发应当根据用户行为以及设备环境(系统平台、屏幕尺寸、屏幕定向等)进行相应的响应和调整；响应式设计的基本原理是通过媒体查询检测不同的设备屏幕尺寸做处理，为了处理移动端，页面头部必须有meta声明viewport；

附加题：实现响应式布局的方式 1.媒体查询@media screen and 2.百分比 3.vw/vh 4. rem

二十五、CSS提高性能的方法 - 1.内联首屏关键CSS 2.异步加载CSS 3.资源压缩4.合理使用选择器5.减少使用昂贵的属性 6.不要使用@import

二十六、css形式实现视觉差滚动效果 1. background-attachment 2. transform:translate3D

二十七、CSS如何画一个三角形？原理是什么？

实心三角形width: 0; height: 0; border-style:solid; border-width: 0 50px 50px; border-color: transparent transparent #d9534f; 空心三角形：利用伪类新建一个小一点的三角形定位上去

原理：边框是实现三角形的部分，边框实际上并不是一个直线，将四条边设置不同的颜色，将边框逐渐放大，可以得到每条边框都是一个梯形，取消一条边的时候，与这条边相邻的两条边的接触部分会变成直的当仅有邻边时， 两个边会变成对分的三角当保留边没有其他接触时，极限情况所有东西都会消失