JavaScript：核心：JavaScript执行过程，事件循环，上下文，执行栈，变量对象VO（先找function声明的，再找var声明的） 活动对象AO（进入执行阶段后变量对象转为活动对象）

创建函数上下文时先找function声明的，再找var声明的，执行时如果有赋值，不再考虑声明形式（没有等号就无效，因为此时变量对象（活动对象）属性名已确定，只能赋值不能声明），而是直接看赋的值的类型（只看等号），即执行时只能赋值不能声明！！！

JavaScript类型转换：Number(str)

面试前看array哪些方法是在原数组操作，哪些是在新数组操作

常用数组方法：forEach遍历，slice复制数组到另一个数组，splice删除或添加值到某元素后面，map(fun(item))返回一个数组，数组元素被fun函数处理，charAt查找字符串中指定位置的字符，indexOf查找指定字符首次出现的位置，lastIndexOf查找指定字符最后出现的位置，使用charAt+for循环遍历字符串，toString转换为字符串，substring(start,end)获取指定位置之间的字符，后面加.length获取长度（！！！indexOfsubstring都是包括查找的位置），&、|、>>、<<运算符，concat()连接多个数组，返回连接后的值。更新数组值的时候不要只用赋值的方式！！！要习惯用Array的各种方法

反转字符串：String的split（’’）方法把字符串转换为字符数组，参数为‘’可分开每个字符，再用Array的reverse方法反转数组，用join方法连接起来并转换为字符串，join参数也是‘’

深拷贝实现：JSON实现（不能处理循环对象），浅拷贝：展开运算符

1、JavaScript原生对象通过大括号定义，原生类通过函数构造，this都是指向实例对象，函数构造的类可以用var定义

私有变量，实例化的对象不能直接访问改变量，需要通公有方法访问，实例化对象的prototype对象有一个constructor属性，指向构造函数，

类通过.属性创建的属性或方法，实例化对象不能访问，类通过.prototype.属性创建的属性或方法，实例化对象可以访问

2、（1）写好缩进 （2）少些if else，用函数代替，代替不了写注释 （3）数字和字符串用常量或枚举代替 （4）React Native

一个js文件是一个类

3、（1）JavaScript上下文是当前执行环境中的变量、this等信息（即当前的代码执行环境），全局上下文和函数上下文两种

（2）JavaScript引擎会创建一个上下文栈存储上下文 （3）创建上下文确定this指向

（4）函数上下文是每次调用时创建的，并不是声明时创建的，创建时在上一层上下文中找该函数

（5）函数声明存在变量提升，函数表达式不存在变量提升，因为在

变量对象创建时function声明的变量值直接赋值给变量对象的属性，var声明的变量属性值是undefined，let和const不进行处理，具体值执行代码时赋值

（6）函数只有被调用时才创建上下文，return返回函数时不创建上下文 ，返回函数时相当于定义了一个函数

（7）\*\*\*函数上下文创建时，在当前上下文（函数内部）找this、变量并作为变量对象的属性，所以存在变量提升

（8）变量对象{

解决问题：创建上下文时的过程+创建上下文时如何找变量+变量提升原理+函数声明覆盖问题

ping带有this.state.props的函数，props是在上层上下文中找的，不是在函数的变量对象中找的

}

（9）作用域链：当前上下文的变量对象属性中没有的变量，会在创建该函数的地方找该变量（闭包），作用域链是定义时确定的，不是创建上下文时确定的

4、JavaScript变量提升是可以先赋值使用再声明，声明必须在使用（输出等操作）之前，JavaScript变量不赋值无法使用

5、React Native中，类中定义箭头函数，this指向类，不使用箭头函数不会自动绑定this，需要手动绑定this，文件中单独定义函数，this指向待定，并不是指向文件模块

6、想把一个函数组合成多个函数的结合，各部分函数需要只负责单独功能，联系要少

7、JavaScript定义的对象this自动绑定，对象构造器this自动绑定，es6类的方法不会自动绑定到类上，需要手动绑定或使用箭头函数自动绑定

8、JavaScript代码执行分为两个阶：翻译阶段和执行阶段，翻译阶段确定作用域规则，执行阶段创建上下文、执行代码，没遇到一个函数调用就进行翻译和执行两个阶段，而不是先把整个文件翻译一下，这就是解释型语言

9、浏览器并不是单线程的，浏览器内核是渲染进程，包含了多个线程，其中的GUI线程负责渲染页面，解析html、css、构建

DOM树等，js引擎线程和GUI线程互斥，因此浏览器加载页面时页面渲染会和脚本执行冲突

10、For in：遍历的是数组下标或对象索引

For of：遍历的是数组元素值

11、typeof：基本类型正确返回（number，string，Boolean），NULL返回object，引用类型实例返回object，function实例（function声明和var声明都是）返回function（Array等构造函数返回function，包括Object）

A Instanceof B：判断B的原型对象的某个属性是否存在于A中，返回布尔值，根据原型链判断

Object对象是所有对象的根对象，包括window对象

12、基本数据类型：number，string，boolen，undefined

Undefined是变量声明后未赋值时的状态，unll是变量没有指向对象的状态

引用数据类型的变量也保存在栈中，只不过它保存的数据是堆中的一个对象的地址

13、js引擎解析脚本时变量就入栈（必须进入内存才能进行其它操作！！！）

14、**JavaScript主线程对应执行栈（调用栈），主线程外有一个任务队列（又叫消息队列，是一个数据结构不是线程），存放异步任务的回调，执行栈中任务执行完后会读取任务队列，将任务队列中的任务调进执行栈中**

**15、异步操作完成时，把回调函数压进任务队列中，执行栈执行完毕后将任务队列中任务压进执行栈中，任务队列可看作是回调函数的队列（任务队列和消息队列是一回事）**

**16、事件监听也是一种异步操作，发生时将回调压进任务队列，类似异步任务执行完。**

**17、setTimeout只是指定在规定毫秒后将回调函数压进任务队列中，并不能保证规定毫秒后执行，因为执行栈中的任务量可能会比较大**

**JavaScript执行过程：1、主进程对应执行栈，任务队列是回调函数的队列 2、异步操作完毕把回调压进执行栈 3、事件监听也是一种异步操作，发生时刻对应异步操作完成时刻 4、异步操作完毕后把回调压进执行栈，并不能保证立刻执行 5、JavaScript主线程之外有异步处理线程（Ajax请求、定时器、事件触发线程），处理异步事件，结束后把回调传进任务队列中**

****

**Function.prototype.a = () => alert(1);**

**Object.prototype.b = () => alert(2);**

**function A() {};**

**var a = new A();**

**a.a();**

**a.b();**

**a instanceof A**

**a instanceof Function**

**a instanceof Object**

**for (var i = 0; i < 3; i++) {**

**document.body.addEventListener('click', function(){console.log(i)})**

**}**

**var name = '123';**

**function a() {**

**var name = 'abc'**

**function b() {**

**console.log(name);**

**}**

**b();**

**}**

**a();**

**18、JavaScript原型继承是对象实例可继承父类或Object对象的原型对象，即使实例化对象实例后添加Object的原型对象的属性也能访问新属性，因为是通过原型链访问的**

**19、子对象的原型对象设置的属性不能通过父类或Object对象访问**

**原型链：（1）对象实例通过new方法和父类建立原型关系，不是继承父类的原型！！！（不是指向父类的原型，也不是复制父类的原型） （2）访问实例对象的属性时会沿着原型链往上找，所以即使实例化后也能访问Object的原型对象的新属性 （3）因为是沿着原型链查找属性，所以父类或Object对象不能访问实例对象的原型对象的属性**

**var ans={**

**age:5**

**}**

**Object.prototype.x=8;**

**ans.x;**

**8//输出**

**20、JavaScript布尔型可用0、1表示，true==1为true**

**21、“==”是先把二者转换为同一类型再用“===”比较（判断是否相等），“===”直接判断是否相等，引用类型判断时判断的是地址**

**22、this：JavaScript this始终指向全局对象，即window对象和global对象，只不过JavaScript原生函数中this是在调用时绑定的，es6箭头函数this是在定义时绑定的，定义时this指向全局对象，所以箭头函数的this不指向定义的对象**

const o1 = {

    text: 'o1',

    fn: function() {

        return this.text;

    }

}

const o2 = {

    text: 'o2',

    fn: o1.fn

}

console.log(o2.fn());

输出o2，调用时绑定this，o1.fn只是赋给了o2的fn，this是在调用时绑定的

**23、JavaScript上下文：this、变量对象、作用域链，对应this、词法环境组件、变量环境组件，词法和变量环境组件中有环境记录和外部环境引入记录（即是作用域链的原理），外部环境引入记录是根据定义时的位置确定的，不是调用时确定的（闭包原理）**

**24、JavaScript “+”作用：全部是数字时做算术运算，非数字相加都转换成字符串再拼接，“-”作用：只做算术运算，非数字会转换成数字再运算**

**25、JavaScript NAN是Number类型，表示一个Number类型的变量没有确切值，每个值是NAN的变量不一样，判等结果是false**

**26、undefined是变量声明后没有赋值，此时没有类型，null是引用类型变量没有指向，NAN是Number类型，每个NAN不一样**

**27、不使用var声明变量也可以使用（此时是设置了全局变量），原理是全局对象设置了属性**

****

**28、function person(name,age){**

**this.name=name;**

**this.age=age;**

**this.eat=function(){**

**console.log(1);**

**}**

**}**

**var student=new person('student1',2);**

**student.name;**

**研究\_\_proto\_\_对象**

****

**JavaScript原型继承关系：JavaScript原型对象有一个constructor属性，指向构造函数或父类父对象（不是构造函数的原型对象！！！）JavaScript原型不是构造函数本身！！！**

**JavaScript对象->prototype->constructor->构造函数**

**29、变量对象相关：JavaScript创建上下文时会找出函数体内所有function和var声明的变量，不管是在哪声明的，即使是在条件语句内声明的，条件语句内声明的并不是等执行时再处理**

**30、function functionFunction(str) {**

**function fun(str2){**

**return str+', '+str2;**

**}**

**return fun;**

**}**

**functionFunction()()返回str+str2，函数多个括号调用必须是外层函数内返回一个函数，第一个函数执行后，得到一个函数，因此会继续执行返回的函数**

**31、arguments对象是函数的一个属性，包含函数实际参数，（1）arguments.callee指向当前函数（正在执行的函数，严格模式下不可用，基本上指向该函数），（2）arguments.callee.length表示函数形参个数，等同于函数名.length，arguments.length表示函数实际参数个数**

**31、JavaScript作用域只有全局作用域和函数作用域，没有块级作用域和对象作用域，对象中的属性在对象方法中访问时需要加上this**

**32、function makeClosures(arr, fn) {**

**var result=new Array();**

**for(let i=0;i<arr.length;i++){**

**result[i]=function(){**

**return fn(arr[i]);**

**}**

**}**

**return result;**

**}**

**JavaScript每次循环都会创建一个作用域，每个函数都有自己的作用域链**

****

**33、var num=2; num.toString(2)将num转化为num对应二进制的字符串**

**34、JavaScript执行前寻找function和var声明的变量，function声明的值直接赋值，var声明的为undefined，在执行过程中，遇到function声明不会再进行赋值**

**function functions(flag) {**

**if (flag) {**

**function getValue() { return 'a'; }**

**} else {**

**function getValue() { return 'b'; }//并不是执行时赋值，而是创建上下文时创建变量对象时赋值，getValue的值始终是第二个**

**}**

**return getValue();**

**}**

****

****

**35、promise是一个容器，用来封装异步操作，好处在于提供便于操作的接口，promise也是一个function，typeof判断结果是function。Then方法就是接收两个回调，分别对应resolve和reject两种状态下的回调**

**用function返回Promise的方式，可以做到得到一个Promise并且能传入参数，**

**Promise也是使用回调函数，在then中传入回调**

**promise链式回调，后面的then就是为前面返回的promise注册回调，默认执行，不用考虑resolve还是reject状态**

**36、任务队列中的任务是一个一个调进执行栈中执行的**

**37、**

**构造函数类的var私有变量不是静态变量，每个实例化对象都有自己的私有变量，私有变量不能直接访问，需要定义函数访问，.创建的静态变量也不能直接访问，只能通过.访问**

**38、**

1. **函数上下文的作用域链是在定义时确定的，不是执行时确定的**
2. **函数内的变量使用return返回时：var声明的仅仅返回值，仍是函数内的私有变量，外界不能访问，没有使用var声明的变量返回后成为全局变量的属性，外界可访问。没有使用var声明的变量不返回的话外界仍无法访问**

**39、JavaScript实例对象不能给原型对象添加属性**

**40、JavaScript new的作用：创建空对象，把构造函数的this指向该空对象，空对象的原型对象的constructor指向构造函数**

**41、fun.call(obj)，call就是把fun的this指向obj中this的指向，然后执行fun**

**42、JavaScript对象的方法中如果不加this则this指向全局对象**

**43、JavaScript闭包中的变量不是存在栈中，而是存在堆中。Js引擎生成一个scope对象存储闭包中的变量。闭包中的变量属于被捕获变量**

**44、JavaScript let和const声明的变量不会作为全局对象的属性，只有var声明的变量才作为全局对象的属性（也就是const和let声明的变量不会作为变量对象的属性）**

**45、引用类型赋值是赋地址仅适用于对象（包括数组和把对象赋值给另一个对象的属性），不适用函数（虽然函数也是对象），函数赋值给另一个函数，更改一个函数另一个不会受到影响**

**46、JavaScript for of循环用于可迭代或可遍历对象，array，string和…，不包括对象、函数，for in循环用于对象（有属性的），包括函数、数组、字符串。**

**47、JavaScript 每个NAN都不一样，判等结果是false（== 和 ===都是false），null判等结果是true**

****

**fn返回2，typeof fn()是判断返回结果**

**48、JavaScript捕获变量：**

**49、promise的回调和其他回调一样，都是压进任务队列后再进入执行栈中执行，所以即使不是耗时操作也在顺序程序执行执行后执行。promise中的代码是顺序执行的，和JavaScript正常代码一样，不是异步的，是在执行栈内执行的，只是回调是异步的而已。**

**Promise只是集中管理了回调操作，可以把多种情况的回调集中管理，promise对象内部的代码和正常代码执行时没什么区别，只是多了一个then来指定回调（转移回调的地方）。Resolve和reject只是起到传值作用，不会结束promise内部代码的执行**

**50、JavaScript in：x in obj，x是一个key值，如果obj中有对应key的属性，返回true，没有则返回false。**

**51、JavaScript对象可以用[key]的方式访问属性值（函数也可当对象使用，需要通过 . 赋属性值）**

**52、注意字符串内使用字符串要用单引号和双引号分开**

**53、async function async1() {**

**console.log('async1 start');**

**await async2();**

**console.log('async1 end');**

**}**

**async function async2() {**

**new Promise(function(resolve) {**

**console.log('promise1');**

**resolve();**

**}).then(function() {**

**console.log('promise2');**

**});**

**}**

**console.log('script start');**

**setTimeout(function() {**

**console.log('setTimeout');**

**}, 0)**

**async1();**

**new Promise(function(resolve) {**

**console.log('promise3');**

**resolve();**

**}).then(function() {**

**console.log('promise4');**

**});**

**console.log('script end');**

**async2函数中promise需要执行完才算结束，async1中的await后面的代码才能执行**

**async返回一个promise对象，是async修饰的函数执行后返回一个promise对象，如：var result = myAsync(); result是一个promise对象，也可以在myAsync函数中返回一个promise对象，显示返回。**

**\*\*\*promise可以不要resject，只要reslove**

**\*\*\*promise的reslove可以不传值，一样可以使用then回调**

**//误解**

**54、函数柯里化：一个函数接收多个参数，柯里化后：（1）只接收第一个参数，（2）返回一个函数，（3）返回的函数接收其余参数，并返回结果**

**55、立即执行函数就像promise，声明后立即执行代码**

**56、async中如果有await，则等待await后面的语句执行。Promise中resolve和reject并不会阻塞后面的代码执行，只是把参数传递给了回调函数**

**57、es6展开运算符用于展开对象的属性**

**58、es6 static设置静态方法，在类中使用，类通过.语法设置静态属性**

**59、函数柯里化必须是每次只接收一个参数**

**60、扩展运算符：把数组或对象分解成逗号隔开的参数序列，确实是逗号隔开的，用console.log输出时没有输出逗号是因为逗号被console当成了分隔符，赋值时也一样，逗号被当成了数组元素之间的分隔符**

**61、promise状态只能改变一次，从初始的pending变成reslove或reject后就不能再改变**

**61、（1）promise的then确实是传递两个状态下的回调函数的，如果只使用reject一个，resolve的回调需要设置为null.（2）catch作用：传递rejected状态下的回调函数，如果是resolve状态，catch捕获resolve回调执行中的异常。（3）只写一个回调默认是resolve的回调，类似react redux的provider**

**62、promise的意义在于统一管理多个异步操作的结果，并随时使用和多次使用，并且每次使用的方式（指定的回调函数）可以不一样**

**63、DOM是文档对象，页面是固定的，而BOM是浏览器对象，是浏览器的一个窗口，其中的网页内容无所谓是哪个，可以随意更改（通过location对象设置），一切操作都是针对该窗口的**

**64、BOM就是一些钩子函数和一些属性。Location：url相关属性和操作，navigator：浏览器版本等信息，history：历史记录相关信息和**

**65、location对象：（1）设置href属性会跳转到新的url，会刷新页面，即使是同一个url 。（2）设置href属性本身不会传递参数，需要设置search属性。（3）跳转页面传参原理：location是BOM对象，管理浏览器窗口，获取url中的参数**

**66、路由传参方式：params方式是动态路由（即动态请求路径），发送请求时参数不可少，否则无法处理请求，服务端通过req.params.参数名获取，react中通过match对象获取，match.params.参数名。 query方式是传统的get请求参数，即url中的参数（?参数名=值&参数名=值），可以随意设置，发送请求时只根据路径判断（即query参数不作为请求路径判断的依据），服务端通过req.query.参数名获取**

**二者主要区别：params作为请求路径判断的依据，query不作为判断路径的依据；获取方式不同；params需要在路径中设置，query不需要设置，前端自己发过来**

**67、BOM的location.reload()方法的true参数（强制刷新）：浏览器像服务器发送http请求时会获取文件在服务器上最后的修改时间，再次发送请求时会先判断修改时间是否一致，一致则在缓存中找，不一致则从服务器中获取。**

**Location常用方法：assign(重定向)，replace(替换当前url)，reload(刷新页面)**

**68、本地存储：BOM对象的sessionStorage存储，生命周期为浏览器窗口打开到关闭，关闭后数据消失（刷新不会消失，刷新仅仅是刷新文档内容，BOM对象并没有变），并且仅限于同一窗口访问。LocalStorage生命周期永久有效（除非手动删除），浏览器所有页面都可访问**

**69、跨域解决：jsonp把服务端响应的内容以字符串的形式导入，经过eval处理在本地执行**

**70、**

**解构出来的tail是array类型的**

**71、JavaScript函数传参时赋值，参数值将是这个值且每次调用时赋值一次，每次赋值不一定相同（该值可能变化）**

**72、parseInt函数接受两个参数，第一个参数是字符串，第二个参数是数字，把第一个参数按照第二个参数的值解析为数字。**

**注意：parseInt和map结合使用时，map传入函数参数应是方法名，回调函数都是传入方法名或方法定义，加括号的话传入的就是函数执行结果了。另外，map的函数参数接受的参数值是固定的，parseInt接收的参数值并不是固定的，只是具有特殊作用，二者的参数并不冲突**

**73、防抖是：事件触发后n秒后执行，如果重复触发则重新计算时间，所以表现出来效果就是以最后一次事件触发为准。**

**节流是：一段时间内函数只执行一次，事件触发后立即执行回调**

**74、object是字符串——值的映射，Map是值——值的映射，键可以是任意类型，包括对象。**

**75、JavaScript强引用：一个对象有引用时，如果不再需要这个使用对象，需要手动删除该对象的引用，否则垃圾回收机制不会回收该对象的内存。**

**弱引用：一个对象有引用时，如果不再需要这个对象，垃圾回收机制会回收该对象的内存，此时引用也失效**

**WeakMap的键只能是对象类型，且是弱引用类型**

**使用对象引用时，要习惯手动删除对象的引用**

**76、es6 Class实例化：使用构造函数时的步骤是：创建空对象，把属性通过this赋到新对象上，把新对象的protopyte指向构造函数。而es6 的类的继承是先创建父类的实例对象，再通过super把实例的this修改到子类中**

**77、！！！！！重点：async await中await后面的执行后会返回一个promise对象，后面的代码就会阻塞，并不是继续执行。注意！！！：后面的代码相当于promise的then的回调，await后面的语句相当于promise的传入的函数，await阻塞的含义就是await后的语句像promise中的函数一样同步执行。**

**用同步，不要用异步（用一个变量接收返回的promise对象再添加then回调）**

**78、Generator：（1）使用function\*声明一个Generator函数，（2）函数执行返回一个迭代状态机对象，（3）每个状态不是函数执行时返回的，是状态机对象使用next函数时执行yield语句之间的程序**

**79、DOM的nodeType属性返回节点类型，如果是元素节点返回1，如果是属性节点返回2**

**80、DOM的childNodes属性返回元素子节点**

**81、数组扁平化：**

**(1)使用flat()函数，参数是扁平化的层数，默认一层，传入Infinity扁平化所有层**

**(2)手动扁平化，通过some判断是否存在数组元素，存在则通过扩展运算符把元素展开，然后通过[]连接展开后的数组并赋给原数组**

**82、**

**1、函数中定义变量，不使用var，则默认为全局变量（作为window对象的属性）**

**2、把对象作为参数传进函数中，argument对象存进去参数的地址，使用new Object时，创建了空对象，申请了新地址并存进参数变量的栈内存中，this指向了当前最高对象（window对象），原对象webSite的引用和属性并没有变化**

**83、基本类型作为参数传给函数，传递的是值，函数中操作参数，原变量不变。**

**对象类型作为参数传给函数，传递的是地址，函数中操作参数，原对象改变**

**84、！！！！！数组的forEach方法遍历数组，回调的第一个参数是元素，改变它不会改变原数组的元素，必须用下标操作原数组**

**！！！！！并且，如果是空数组，forEach不会执行回调！！！！！**

**！！！！！undefine进行算术运算，结果是NAN**

**85、**var a = 10;

(function () { console.log(a)

a = 5

console.log(window.a)

var a = 20;

console.log(a)})()

判断运行结果的代码就考虑函数执行上下文、变量对象等问题

立即执行函数的上下文的变量对象中不包含window环境的变量