JS中使用typeof能得到哪些类型

变量类型 值类型 和 引用类型

值类型：变量本身就是含有赋予给它的数值的，它的变量本身及保存的数据都存储在栈的内存块当中

引用类型：引用类型当然是分配到堆上的对象或者数据变量，根据官方一点的解释就是引用类型的变量只包括对其所表示的数据引用

对象、数组、函数

无限扩展属性，可能出现内存占用比较大的情况

typeof abc //"undefined" typeof NaN //"number" typeof true //"boolean" typeof {} //"object" typeof [] //"object" typeof null //"object" typeof console.log //"function"

typeof只能区分值类型，对引用类型无能为力，只能区分函数function

NaN表示特殊的非数字值，null是空指针，并没有指向任何一个地址

typeof能区分的五种基本类型：string、boolean、number、undefined、object和函数function

变量计算

可能发生强制类型转换的情况：

字符串拼接

== 运算符

100 == '100' //true 0 == "" //true null == undefined //true 复制代码

if语句

var c = ''; if(c){ //.... } 复制代码

逻辑运算

console.log(10 && 0); //0 console.log('' || 'abc'); //'abc' console.log(!window.abc); //true //判断一个变量会被当做true还是false 双非判断 var a = 100; console.log(!!a);

在js逻辑运算中，0、NaN、""、null、false、undefined都会判为false，其他都为true

var add\_level = (add\_step == 5 && 1)||(add\_step == 10 && 2)||(add\_step == 12 && 4)||(add\_step==15 && 5 )|| 0;

var add\_level = {'5':1,'10':2,'15':5,'12':4}[add\_step]||0; //更精简

何时使用 === 何时使用 ==

if(obj.a == null){ //这里相当于obj.a === null || obj.a === undefined,简写 //这是jquery源码中推荐的写法，除此之外其他全用 === //主要是用于判断这个属性是否存在 }

//判断函数参数是否存在 function (a,b){ if(a == null){ ... } }

//这种写法不能用 if(xxx == null){ //可能会报错，这个参数未定义 not defined }

JS中有哪些内置函数-数据封装类对象

Object

Array

Boolean

Number

String

Function

Date

RegExp

Error

Math内置对象

JS变量按照存储方式区分为哪些类型，并描述其特点

如何理解json

JSON是一个JS内置对象，常用两个API

JSON.stringify(a:10,b:20) JSON.parse('{"a":10,"b":20}') 复制代码

同时也是一种数据格式

二、原型和原型链

题目：

如何判断一个对象是数组类型？

写一个原型链继承的例子

描述new一个对象的过程

zepto(或其他框架)源码中如何使用原型链相关知识点：

知识点：

构造函数

构造函-扩展

原型规则和示例

原型链

构造函数

function Foo(name,age){ this.name = name this.age = age this.class = 'class - 1' //return this // 默认有这一行 } var f = new Foo('zhangsan',20) var f1 = new Foo('lisi',22) //创建多个对象 复制代码

特点

构造函数要以大写字母开头

new一个对象的过程

1、把参数穿进去

2、new执行的时候，this会先变成一个空对象，

3、然后会把this.xxx赋值，默认会把 this

return function Foo(name,age){ this.name = name; this.age = age; } 复制代码

构造函数-扩展

var a = {} 其实是var a = new Object()的语法糖

var a = [] 其实是var a = new Array()的语法糖

function foo(){...} 其实是var Foo = new Function(...)的语法糖

使用instanceof+构造函数 判断一个函数是否是另一变量的构造函数

原型规则和示例

所有的引用类型(数组、函数、对象)都具有对象特性，即可自由扩展属性(null除外)

所有的引用类型(数组、对象、函数)有一个\_\_proto\_\_ 属性，属性值是一个普通的对象

所有的函数都有一个prototype属性，属性值也是一个普通的对象

所有的引用类型(数组、对象、函数),\_\_proto\_\_属性指向它的构造函数的prototype属性

当试图得到一个对象的某个属性时，如果这个对象本身没有这个属性，那么回去他的\_\_proto\_\_(即它的构造函数的prototype中寻找)

function Foo(name,age){ this.name = name } Foo.prototype.alertName = function(){ alert(this.name) } //创建示例 var f = new Foo('zhangsan'); f.printName = function(){ console.log(this.name); } f.printName(); f.alertName(); 复制代码

this

function Foo(name,age){ this.name = name; } Foo.prototype.alertName = function(){ alert(this.name) } //创建示例 var f = new Foo('zhangsan') f.printName = function(){ console.log(this.name) } //测试 f.printName() f.alertName() 复制代码

for-in

enter image deｓｃｒｉｐｔion here

var item; for (item in f){ //高级浏览器已经在for in中屏蔽了来自原型的属性 //但是在这里建议大家还是加上这个判断，保证程序的健壮性 if(f.hasOwnProperty(item)){ console.log(item); } } 复制代码

原型链

f.toString() //要去f.\_\_proto\_\_.\_\_proto\_\_中查找 复制代码

Object 的 \_proto\_指向的是null

instanceof

instanceof：用于判断引用类型属于哪个构造函数的方法

f instanceof Foo的判断逻辑是：

f的\_\_proto\_\_一层一层往上，能否对应到Foo.prototype

再试着判断f instance Object

面试题

笔试题一：如何准确判断一个变量是否是数组类型

var arr = [] arr instance Array //true typeof arr //object,typeof是无法判断是否是数组的

笔试题二：写一个原型链继承的例子

要写接近于实战的例子

function Animal(){ this.eat = function(){ console.log('Animal eat'); } } function Dog(){ this.bark = function(){ console.log("dog bark"); } } Dog.prototype = new Animal(); var hashiqi = new Dog();

function Elem (id){ this.elem = document.getElementById(id); } Elem.prototype.html = function(val){ var elem = this.elem; if(val){ elem.innerHtml = val; return this; //链式操作 }else{ return elem.innerHtml; } } Elem.prototype.on = function(type,fn){ var elem = this.elem elem.addEventListener(type,fn) } var div1 = new Elem('content\_wrapper') //div1.html('<p>ds<p>') //div1.on('click',function(){ // alert('clicked') // } //) div1.html('<p>放大放大放大发范德萨大点的我请萨</p>').on('click',function(){ alert('clicked'); }).html('<p>javaｓｃｒｉｐｔ</p>') 复制代码

笔试题三：描述创建一个对象的过程

创建一个新对象

this指向这个新对象

执行代码，即对this赋值

返回this

笔试题四：zepto源码中如何使用原型链

阅读源码是高效提升技能的方式

但不能"埋头苦读"有技巧在其中

搜索"zepto设计和源码分析"

三、作用域和闭包

题目：

说一下对变量提升的理解

说明this几种不同的使用场景

创建10个 < a>标签，点击的时候弹出对应的序号

如何理解作用域

实际开发中闭包的应用

知识点：

执行上下文

this

作用域

作用域链

闭包

执行上下文

范围：一段<ｓｃｒｉｐｔ>或者一个函数

全局：变量定义、函数声明、一段<ｓｃｒｉｐｔ>

函数：变量定义、函数声明、this、arguments

console.log(a) //undefined var a = 100 fn('zhangsan') //'zhangsan' 20 function fn(name){ age = 20 console.log(name,age) var age } 复制代码

函数声明会被提出来，函数表达式会和变量一样，以undefined占位

在函数执行之前,变量定义、函数声明、this、arguments都要先提取出来

函数声明和函数表达式的区别

fn() //不会报错 function fn(){ //声明 } 复制代码

fn() //Uncaught TypeError:fn1 is not a function var a = function(){ //表达式 } 复制代码

变量提升： var a ; undefined,并不知道是一个函数

函数提升： 把函数声明提前

fn('zhangsan') function fn(name){ //函数 console.log(this) console.log(arguments) age = 20 console.log(name,age) var age bar(100) function bar(num){ console.log(num) } } 复制代码

说明this几种不同的使用场景

作为构造函数执行

作为对象属性执行

作为普通函数执行(window)

call、apply、bind

var a = { name : 'A', fn:function(){ console.log(this.name) } } a.fn() //this === a a.fn.call({name:'B'}) //this === {name:'B'} var fn1 = a.fn fn1() //this === window 复制代码

this要在执行的时候才能确认值，定义时无法确认 ,因为js不是编译型语言而是解释性语言

作用域

没有块级作用域

if(true){ var name = 'zhangsan' } console.log(name) 复制代码

函数作用域和全局作用域

var a = 100 function fn(){ var a = 200 console.log('fn',a) } console.log('global',a) fn() 复制代码

作用域链

如何理解作用域：

自由变量

作用域链，即自由变量的查找

闭包的两个场景

var a = 100 function fn(){ var b = 200 //当前作用域没有定义的变量，即“自由变量” console.log(a) console.log(b) } fn() 复制代码

自由变量作用域链，一直往父级作用域找，一直找到全局作用域

var a = 100 function F1(){ var b = 200 function F2(){ var c = 300 console.log(a) //a是自由变量 console.log(b) //b是自由变量 console.log(c) } F2() } F1() 复制代码

闭包

作用域是定义时的作用域，而不是执行时的作用域

闭包的使用场景:

函数作为返回值

function F1(){ var a = 100 //返回一个函数(函数作为返回值) return function(){ console.log(a) } } var f1 = F1() var a = 200 f1() 复制代码

函数作为参数传递

function F1(){ var a = 100 return function(){ console.log(a) //自由变量，父作用域寻找 } } var f1 = F1() function F2(fn){ var a = 200 fn() } F2(f1) 复制代码

面试题

题目1、说一下对变量提升的理解

在某一作用域中，声明变量的语句会默认解析为在该作用域的最开始就已经声明了

题目2、说明this几种不同的使用场景

blog.csdn.net/w\_q\_1025/ar…

题目3、创建10个 < a>标签，点击的时候弹出对应的序号

错误写法

var i,a for(i = 0;i<10;i++){ a = document.createElement('a') a.innerHTMl = i + '<br>' a.addEventListener('click',function(e){ e.preentDefault() alert(i) }) document.body.appendChild(a) } 复制代码

正确写法

var i,a for(i = 0;i < 10;i++){ (function(i){ var a = document.createElement('a') a.innerHTML = i + '<br>' a.addEventListener('click',function(e){ e.preventDefault() alert(i) }) document.body.appendChild(a) })(i) } 复制代码

题目4、如何理解作用域

自由变量

作用域链，即自由变量的查找

闭包的两个场景

题目5、实际开发中闭包的应用

//闭包实际应用中主要用于封装变量，收敛权限 function isFirstLoad(){ var \_list = []; return function(id){ if(\_list.indexOf(id) >= 0){ return false; }else{ \_list.push(id); return true; } }} var firstLoad = isFirstLoad(); firstLoad(10); 复制代码

四、异步和单线程

题目:

同步和异步的区别是什么？分别举一个同步和异步的例子

一个关于setTimeout的笔试题

前端使用异步的场景有哪些

知识点：

什么是异步(对比同步)

前端使用异步的场景

异步和单线程

什么是异步

是否阻塞程序的运行

何时需要异步

在可能发生等待的情况

定时任务：setTimeout,setTimeInterval

网络请求：ajax请求，动态<img>加载

事件绑定

等待过程中不能像alert一样阻塞程序运行

因此所有的“等待的情况”都需要异步

异步和单线程

console.log(100) setTimeout(function(){ console.log(200) },1000) console.log(300) 复制代码

执行第一行，打印100

执行setTimeout后，传入setTimeout的函数会被暂存起来，不会立即执行（单线程特点，不能同时干两件事）

执行最后一行，打印300

待所有程序执行完，处于空闲状态时，会立马看有没有暂存起来的要执行

发现暂存起来的setTimeout中的函数无需等待时间，就立即来过来执行

面试题

笔试题1： 同步和异步的区别是什么？分别举一个同步和异步的例子

同步会阻塞代码执行，而异步不会

笔试题2：一个关于setTimeout的笔试题

console.log(1) setTimeout(function(){ console.log(2) },0) console.log(3) setTimeout(function(){ console.log(4) },1000) console.log(5) 复制代码

笔试题3：前端使用异步的场景有哪些

需要等待，因为js是单线程语言

五、常见对象

题目

获取2017-06-10格式的日期

获取随机数，要求是长度一直的字符串格式

写一个能遍历对象和数组的通用forEach函数

Date

Date.now() //获取当前时间毫秒数 var dt = new Date() dt.getTime() //获取毫秒数 dt.getFullYear() //年 dt.getMonth() //月(0-11) dt.getDate() //日(0-31) dt.getHours() //小时(0-23) dt.getMinutes() //分钟(0-59) dt.getSeconds() //秒(0-59) 复制代码

Math

Math.random()：可以用来清除缓存

Array

forEach 遍历所有数据

every 判断所有元素是否都符合条件

some 判断是否有至少一个元素符合条件

sort 排序

map 对元素重新组装，生成新数组>- 过滤符合条件的元素

array.forEach

arr.forEach( function(item,index){ console.log(index,item); });

array.every

var arr = [1,2,3,4,5]; var result = arr.every(function(index,item){ //用来判断所有的数组元素，都满足一个条件 if(item<6) return true }); console.log(result);

array.some

var result = arr.some(function(index,item){ if(item<5) return true; }); console.log(result); arry.sort

var arr2 = arr.sort(function(a,b){ //从小到大排序 return a-b; //从大到小排序 //return b-a; })

array.map

arr.map(function(item,index){ return '<br>'+index+':'+item+'<br>' })

array.filter

var arr2 = arr.filter(function (item,index){ //通过某一个条件过滤数组 if(item >= 2) return true })

对象API

var obj = { x:100, y:200, z:300 } var key for(key in obj){ if(obj.hasOwnProperty(key)){ console.log(key,obj[key]) } }

面试题

解答1:获取2016-06-10格式的日期

function formatDate(dt){ if(!dt){ dt = new Date() } var year = dt.getFullYear() var month = dt.getMonth() + 1 var date = dt.getDate() if(month < 10){ //强制类型转换 month = '0' + month } if(date < 10){ //强制类型转换 date = '0' + date } //强制类型转换 return year + '-' + month + '-' + date } var dt = new Date() var formatDate = formatDate(dt) console.log(formatDate) 复制代码

解答2:获取随机数，要求是长度一致的字符串格式

var random = Math.random() + '0000000000'; var random = random.slice(0,10); 复制代码

解答3:写一个能遍历对象和数组的通用forEach函数

function forEach(obj,fn){ var key if(obj instanceof Array){ //准备判断是不是数组 obj.forEach(function(item,index){ fn(index,item) }) }else{ //不是数组就是对象 for(key in obj){ fn(key,obj[k]) } } }

五、JS-Web-API：

知识点：

js基础知识：ECMA262标准

JS-Web-API：W3C标准

W3C标准中关于JS的规定有：

DOM操作

BOM操作

事件绑定

ajax请求(包括http协议)

存储

DOM操作

DOM的本质 Document、Object、Model 浏览器把拿到的html代码，结构化一个浏览器能够识别并且js可操作的一个模型而已

DOM的节点操作

获取DOM节点

Attribute 和 properity

attribute：是HTML标签上的某个属性，如id、class、value等以及自定义属性，它的值只能是字符串，关于这个属性一共有三个相关的方法，setAttribute、getAttribute、removeAttribute； 注意：在使用setAttribute的时候，该函数一定接收两个参数，setAttribute（attributeName,value）,无论value的值是什么类型都会编译为字符串类型。在html标签中添加属性，本质上是跟在标签里面写属性时一样的，所以属性值最终都会编译为字符串类型。

property：是js获取的DOM对象上的属性值，比如a，你可以将它看作为一个基本的js对象。这个节点包括很多property，比如value，className以及一些方法onclik等方法。 一个js对象有很多property，该集合名字为properties，properties里面有其他property以及attributies，attributies里面有很多attribute。 而常用的attribute，id、class、name等一般都会作为property附加到js对象上，可以和property一样取值、赋值

DOM结构操作

面试题1：DOM是那种基本的数据结构？

树

面试题2：DOM操作的常用API有哪些？

获取DOM节点，以及节点的property和Attribute

获取父节点，获取子节点

新增节点，删除节点

面试题3：DOM节点的attr和property有何区别

property只是一个JS对象的属性的修改

Attribute是对html标签属性的修改

BOM操作

问题：

如何检测浏览器的类型

拆解URL各部分

知识点

navigator

screen

location

history

navigator & screen

//navigator var ua = navigator.userAgent var isChrome = ua.indexOf('chrome') console.log(isChrome) //screen console.log(screen.width) console.log(screen.height) //location console.log(location.href) console.log(location.protocol) console.log(location.pathname) console.log(location.search) //?之后的参数 console.log(location.hash) //#号之后 //history history.back() history.forward() 复制代码

六、事件

题目：

编写一个通用的事件监听函数

描述时间冒泡流程

对于一个无限下拉加载图片的页面，如何给每个图片绑定事件

知识点:

通用事件绑定

事件冒泡

代理

知识点

通用事件绑定

var btn = document.getElementById('btn1') btn.addEventListener('click', function(event) { console.log('clicked') }) function bindEvent(elem, type, fn) { elem.addEventListener(type, fn) } var a = document.getElementById('link1') bindEvent(a, 'click', function(e) { e.preventDefault(); //阻止默认行为 alert('clicked') })

关于IE低版本的兼容性

IE低版本使用attachEvent绑定事件，和W3C标准不一样

IE低版本使用量已非常少，很多网站早已不支持

建议对IE低版本的兼容性：了解即可，无须深究

如果遇到对IE低版本要求苛刻的面试，果断放弃

事件冒泡

e.stopPropatation() //取消冒泡

代理

<div id="div1"> <a href = "#">a1</a> <a href = "#">a2</a> <a href = "#">a3</a> <a href = "#">a4</a> <!--会随时新增更多 a 标签--> </div> var div1 = document.getElementById('div1') div1.addEventListener('click',function(e){ var target = e.target if(target.nodeName === 'A'){ alert(target.innerHTMl) } }) 复制代码

完整写法

function bindEvent(elem, type, selector, fn) { if (fn == null) { fn = selector selector = null } elem.addEventListener(type, function(e) { var target if (selector) { target = e.target if (target.matches(selector)) { fn.call(target, e) } } else { fn(e) } }) } //使用代理 var div1 = document.getElementById('div1') bindEvent(div1, 'click', 'a', function(e) { console.log(this.innerHTML) }) //不使用代理 var a = document.getElementById('a1') bindEvent(div1, 'click', function(e) { console.log(a.innerHTML) }) 复制代码

面试题

面试题1：编写一个通用的事件监听函数

面试题2：描述事件冒泡流程

DOM树形结构

事件冒泡

组织冒泡

冒泡的应用

面试题3：对于一个无限下拉加载图片的页面，如何给每个图片绑定事件

使用代理

知道代理的两个优点

七、Ajax

题目：

手动写一个ajax，不依赖第三方库

跨域的几种实现（原理）

知识点:

xmlHttpRequest

状态码说明

跨域

XMLHttpRequest

var xhr = new XMLHttpRequest() xhr.open("GET","/api",false) xhr.onreadystatechange = function(){ //这里的函数异步执行，可参考之前JS基础中的异步模块 if(xhr.readyState == 4){ if(xhr.status == 200){ alert(xhr.responseText) } } } xhr.send(null) 复制代码

IE兼容性问题

IE低版本使用ActiveXObject，和W3C标准不一样

IE低版本使用量已经非常少，很多网站早已不支持

建议对IE低版本的兼容性：了解即可，无需深究

如果遇到对IE低版本要求苛刻的面试，果断放弃

.

状态码说明

0 - (未初始化) 还没调用send()方法

1 - (载入) 已调send() 方法，正在发送请求

2 - (载入完成) send()方法执行完成，已经接收到全部相应内容

3 - (交互) 正在解析响应内容

4 - (完成) 响应内容解析完成，可以在客户端调用了

.

status说明

2XX - 表示成功处理请求。如200

3XX - 需要重定向，浏览器直接跳转

4XX - 客户端请求错误，如404

5XX - 服务器端错误

跨域

什么是跨域

浏览器有同源策略，不允许ajax访问其他域的接口

跨域条件：协议、域名、端口，有一个不同就算跨域

可以跨域的三个标签

<img src=''>

<ｓｃｒｉｐｔ src=''>

<link href=''>

三个标签的场景

<img>用于打点统计，统计网站可能是其他域(而且没有任何兼容性问题)

<link> <ｓｃｒｉｐｔ>可以使用CDN，CDN也是其他域

<ｓｃｒｉｐｔ>可以用于JSONP

跨越注意的问题

所有的跨域请求都必须经过信息提供方允许

如果未经允许即可获取，那是浏览器同源策略出现漏洞

JSONP实现原理

加载 coding.m.imooc/classindex.…

不一定服务器端真正有一个classindex.html文件

服务器可以根据请求，动态生成一个文件，返回

通理于<ｓｃｒｉｐｔ src="http://coding.m.imooc.com/api.js">

//例如你的网站要跨域访问慕课网的一个接口 //慕课给你一个地址 http://coding.m.imooc.com/api.js //返回内容格式 callback({x:100,y:200}) <ｓｃｒｉｐｔ> window.callback = function(data){ //这是我们跨域得到信息 console.log(data) } </ｓｃｒｉｐｔ> <ｓｃｒｉｐｔ src="http://coding.m.imooc.com/api.js"></ｓｃｒｉｐｔ> 复制代码

服务器端设置http header

cookie、sessionStorage、localStorage的区别

容量

是否会携带到ajax中

API易用性

ios safari隐藏模式下，localStorage.getItem会报错，建议统一用try-catch封装 复制代码

八、开发环境

关于开发环境：

IDE(写代码的效率)

git（代码管理工具，多人协作开发）

JS模块化

打包工具

上线回滚的流程

IDE

GIT

- git add - git checkout xxx - git commit -m""xxx" - git push origin master - git pull origin master - git branch - git chekcout -b xxx/git checkout xxx - git merge xxx 复制代码

模块化

构建工具

九、运行环境

怎么去设计一个组件封装

组件封装的目的是为了重用，提高开发效率和代码质量

低耦合，单一职责，可复用性，可维护性

前端组件化设计思路

js 异步加载的方式

渲染引擎遇到 script标签会停下来，等到执行完脚本，继续向下渲染

defer 是“渲染完再执行”，async 是“下载完就执行”，defer 如果有多个脚本，会按照在页面中出现的顺序加载，多个async 脚本不能保证加载顺序

加载 es6模块的时候设置 type=module，异步加载不会造成阻塞浏览器，页面渲染完再执行，可以同时加上async属性，异步执行脚本（利用顶层的this等于undefined这个语法点，可以侦测当前代码是否在 ES6 模块之中）

css 动画和 js 动画的差异

代码复杂度，js 动画代码相对复杂一些

动画运行时，对动画的控制程度上，js 能够让动画，暂停，取消，终止，css动画不能添加事件

动画性能看，js 动画多了一个js 解析的过程，性能不如 css 动画好

XSS 与 CSRF 两种跨站攻击

xss 跨站脚本攻击，主要是前端层面的，用户在输入层面插入攻击脚本，改变页面的显示，或则窃取网站 cookie，预防方法：不相信用户的所有操作，对用户输入进行一个转义，不允许 js 对 cookie 的读写

csrf 跨站请求伪造，以你的名义，发送恶意请求，通过 cookie 加参数等形式过滤

我们没法彻底杜绝攻击，只能提高攻击门槛

事件委托，目的，功能，写法

把一个或者一组元素的事件委托到它的父层或者更外层元素上

优点，减少内存消耗，动态绑定事件

target 是触发事件的最具体的元素，currenttarget是绑定事件的元素(在函数中一般等于this)

Javascript 事件委托详解

线程，进程

线程是最小的执行单元，进程是最小的资源管理单元

一个线程只能属于一个进程，而一个进程可以有多个线程，但至少有一个线程

负载均衡

当系统面临大量用户访问，负载过高的时候，通常会使用增加服务器数量来进行横向扩展，使用集群和负载均衡提高整个系统的处理能力

服务器集群负载均衡原理？

什么是CDN缓存

CDN 是一种部署策略，根据不同的地区部署类似nginx 这种服务服务，会缓存静态资源。前端在项目优化的时候，习惯在讲台资源上加上一个 hash 值，每次更新的时候去改变这个 hash，hash 值变化的时候，服务会去重新取资源

(CDN)是一个经策略性部署的整体系统，包括分布式存储、负载均衡、网络请求的重定向和内容管理4个要件

CDN\_百度百科

闭包的写法，闭包的作用，闭包的缺点

使用闭包的目的——隐藏变量，间接访问一个变量,在定义函数的词法作用域外，调用函数

闭包的内存泄露，是IE的一个 bug，闭包使用完成之后，收回不了闭包的引用，导致内存泄露

「每日一题」JS 中的闭包是什么？

闭包造成内存泄露的实验

跨域问题，谁限制的跨域，怎么解决

浏览器的同源策略导致了跨域

用于隔离潜在恶意文件的重要安全机制

[jsonp ，允许 script加载第三方资源]https://segmentfault.com/a/1190000008445998

nginx 反向代理（nginx 服务内部配置 Access-Control-Allow-Origin \*）

cors 前后端协作设置请求头部，Access-Control-Allow-Origin 等头部信息

iframe 嵌套通讯，postmessage

javascript 中常见的内存泄露陷阱

内存泄露会导致一系列问题，比如：运行缓慢，崩溃，高延迟

内存泄露是指你用不到（访问不到）的变量，依然占居着内存空间，不能被再次利用起来

意外的全局变量，这些都是不会被回收的变量（除非设置 null 或者被重新赋值），特别是那些用来临时存储大量信息的变量

周期函数一直在运行，处理函数并不会被回收，jq 在移除节点前都会，将事件监听移除

js 代码中有对 DOM 节点的引用，dom 节点被移除的时候，引用还维持

Javascript 中 4 种常见的内存泄露陷阱

babel把ES6转成ES5或者ES3之类的原理是什么

它就是个编译器，输入语言是ES6+，编译目标语言是ES5

babel 官方工作原理

解析：将代码字符串解析成抽象语法树

变换：对抽象语法树进行变换操作

再建：根据变换后的抽象语法树再生成代码字符串

Promise 模拟终止

当新对象保持“pending”状态时，原Promise链将会中止执行。

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

promise 放在try catch里面有什么结果

Promise 对象的错误具有冒泡性质，会一直向后传递，直到被捕获为止，也即是说，错误总会被下一个catch语句捕获

当Promise链中抛出一个错误时，错误信息沿着链路向后传递，直至被捕获

网站性能优化

http 请求方面，减少请求数量，请求体积，对应的做法是，对项目资源进行压缩，控制项目资源的 dns 解析在2到4个域名，提取公告的样式，公共的组件，雪碧图，缓存资源，

压缩资源，提取公共资源压缩，提取 css ，js 公共方法

不要缩放图片，使用雪碧图，使用字体图表（阿里矢量图库）

使用 CDN，抛开无用的 cookie

减少重绘重排，CSS属性读写分离，最好不要用js 修改样式，dom 离线更新，渲染前指定图片的大小

js 代码层面的优化，减少对字符串的计算，合理使用闭包，首屏的js 资源加载放在最底部

js 自定义事件实现

原生提供了3个方法实现自定义事件

createEvent，设置事件类型，是 html 事件还是 鼠标事件

initEvent 初始化事件，事件名称，是否允许冒泡，是否阻止自定义事件

dispatchEvent 触发事件

angular 双向数据绑定与vue数据的双向数据绑定

二者都是 MVVM 模式开发的典型代表

angular 是通过脏检测实现，angular 会将 UI 事件，请求事件，settimeout 这类延迟，的对象放入到事件监测的脏队列，当数据变化的时候，触发 $diget 方法进行数据的更新，视图的渲染

vue 通过数据属性的数据劫持和发布订阅的模式实现，大致可以理解成由3个模块组成，observer 完成对数据的劫持，compile 完成对模板片段的渲染，watcher 作为桥梁连接二者，订阅数据变化及更新视图

get与post 通讯的区别

Get 请求能缓存，Post 不能

Post 相对 Get 安全一点点，因为Get 请求都包含在 URL 里，且会被浏览器保存历史纪录，Post 不会，但是在抓包的情况下都是一样的。

Post 可以通过 request body来传输比 Get 更多的数据，Get 没有这个技术

URL有长度限制，会影响 Get 请求，但是这个长度限制是浏览器规定的，不是 RFC 规定的

Post 支持更多的编码类型且不对数据类型限制

有没有去研究webpack的一些原理和机制，怎么实现的

解析webpack配置参数，合并从shell传入和webpack.config.js文件里配置的参数，生产最后的配置结果。

注册所有配置的插件，好让插件监听webpack构建生命周期的事件节点，以做出对应的反应。

从配置的entry入口文件开始解析文件构建AST语法树，找出每个文件所依赖的文件，递归下去。

在解析文件递归的过程中根据文件类型和loader配置找出合适的loader用来对文件进行转换。

递归完后得到每个文件的最终结果，根据entry配置生成代码块chunk。

输出所有chunk到文件系统。

ES6模块与CommonJS模块的差异

CommonJs 模块输出的是一个值的拷贝，ES6模块输出的是一个值的引用

CommonJS 模块是运行时加载，ES6模块是编译时输出接口

ES6输入的模块变量，只是一个符号链接，所以这个变量是只读的，对它进行重新赋值就会报错

模块加载AMD，CMD，CommonJS Modules/2.0 规范

这些规范的目的都是为了 Javascript 的模块化开发，特别是在浏览器端的

对于依赖的模块，AMD 是提前执行，CMD 是延迟执行

CMD 推崇依赖就近，AMD 推崇依赖前置

Node 事件循环，js 事件循环差异

Node.js 的事件循环分为6个阶段

浏览器和Node 环境下，microtask 任务队列的执行时机不同

Node.js中，microtask 在事件循环的各个阶段之间执行

浏览器端，microtask 在事件循环的 macrotask 执行完之后执行

3. 递归的调用process.nextTick()会导致I/O starving，官方推荐使用setImmediate()

浅拷贝和深拷贝的问题

深拷贝和浅拷贝是只针对Object和Array这样的复杂类型的

也就是说a和b指向了同一块内存，所以修改其中任意的值，另一个值都会随之变化，这就是浅拷贝

浅拷贝， ”Object.assign() 方法用于将所有可枚举的属性的值从一个或多个源对象复制到目标对象。它将返回目标对象

深拷贝，JSON.parse()和JSON.stringify()给了我们一个基本的解决办法。但是函数不能被正确处理

开放性问题

开放性问题主要是考察候选人业务积累，是否有自己的思考，思考问题的方式，没有标准答案。不过有些问题挺刁的，哈哈哈哈，比如：” 你见过的最好的代码是什么？ “总之提前准备下没错。

先自我介绍一下，说一下项目的技术栈，以及项目中遇到的一些问题

从整体中，看你对项目的认识，框架的认识和自己思考

项目中有没有遇到什么难点，怎么解决

如果你在创业公司你怎么从0开始做（选择什么框架，选择什么构建工具）

说一下你项目中用到的技术栈，以及觉得得意和出色的点，以及让你头疼的点，怎么解决的

一个业务场景，面对产品不断迭代，以及需求的变动该怎么应对，具体技术方案实现

你的学习来源是什么

你觉得哪个框架比较好，好在哪里

你觉得最难得技术难点是什么

你见过的最好的代码是什么