**一、浏览器**

浏览器如何渲染页面的？

**整体流程**

浏览器从HTTP服务器获取html文档，到呈现页面给用户，会经过以下几个步骤：

1、html被html解析器解析成DOM树

2、css被css解析器解析成CSS规则树

3、将DOM树和CSS规则树合成render树（其实就是将CSS样式附在对应的节点上。

4、生成布局(flow),即将所有渲染树的所有节点进行平面合成

5、将布局绘制(repaint)到屏幕上

**重绘(repaint)**

当元素样式的改变不影响页面布局时，比如元素的颜色，浏览器将对元素进行的更新，称之为重绘。位置大小不改变。

**重排(reflow)**

也叫做重排。当元素的尺寸或者位置发生了变化，就需要重新计算渲染树，这就是回流，比如元素的宽高、位置，浏览器会重新渲染页面，称为回流，又叫重排（layout）。位置、大小改变

**关系**

重排必定会触发重绘，重绘不一定会触发重排。

重绘的开销较小，重排的代价较高。

**如何减少回流和重绘？**

使用class类名来控制样式的改变;

对于复杂的动画效果可以使其脱离文档流;

免循环操作DOM/避免循环读取offsetLeft等属性

**DOM树上有10个节点，渲染树上一定有10个节点吗？**

不一定，理论上来讲，dom树的节点会多一些

* DOM 树与 HTML 标签一一对应，包括 head 和隐藏元素
* 渲染树不包括 head 和隐藏元素

参考链接：

<https://blog.csdn.net/weixin_45820444/article/details/109013996>

cookies、sessionStorage 、localStorage

三者最大的不同在于生命周期、大小的不同。

**cookie**

生命周期为人为设置，在过期时间之前都是有效的。每次http请求都会携带cookie。大小约为4kb

**sessionStorage**

生命周期为仅在当前浏览器窗口关闭之前有效，关闭页面或者浏览器会被清除。不参与和服务器的通信。大小约5M或者更大

**localStorage**

永久有效，窗口或者浏览器关闭也会一直保存，除非手动永久清除。不参与和服务器的通信。大小约5M或者更大

removeItem()/setItem()/clear()/getItem()

Get和post的区别

1、get请求是获取数据的，而post请求是提交数据的。

2、get请求能被缓存，post请求不能被缓存

3、get请求对数据长度的限制；当发送数据时，GET 方法向 URL 添加数据get请求的传送数据会拼接在url后面

4、GET在浏览器回退时是无害的，而POST会再次提交请求。

5、GET比POST更不安全，因为参数直接暴露在URL上，所以不能用来传递敏感信息。

6、GET参数通过URL传递，POST放在Request body中。

辅助记忆：

Url传参-长度限制-缓存-不安全

同源策略

同源策略是浏览器最核心也最基本的安全功能，目的是保护信息安全。

如何保护？禁止浏览器跨域请求数据。

如何禁止？通过识别域名、协议、端口号。

跨域

域名、协议、端口号三者有一个不同的话都是属于一个跨域行为

如何解决跨域？

nginx代理跨域，代理跨域的原理

通过jsonp跨域

postMessage跨域

document.domain + iframe跨域

location.hash + iframe

window.name + iframe跨域

输入URL之后会发生什么？

1.DNS域名解析；

2.建立TCP连接；

3.发送HTTP请求；

4.服务器处理请求并返回响应结果；

5.关闭TCP连接；

6.浏览器解析HTML并布局渲染；

浏览器缓存机制

强缓存和协商缓存

==>是浏览器在本地磁盘对用户最近请求过的文档进行存储，当访问者再次访问同一页面时，浏览器就可以直接从本地磁盘加载文档。

三次握手

第一次握手：建立连接时，客户端发送syn包（syn=j）到服务器，并进入SYN\_SENT状态，等待服务器确认；SYN：同步序列编号（Synchronize Sequence Numbers）。

第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个SYN包（syn=k），即SYN+ACK包，此时服务器进入SYN\_RECV状态；

第三次握手：客户端收到服务器的SYN+ACK包，向服务器发送确认包ACK(ack=k+1），此包发送完毕，客户端和服务器进入ESTABLISHED（TCP连接成功）状态，完成三次握手。

**二、JavaScript**

什么是JavaScript？

ECMAScript是一种标准化语言规范，基于ES规范，然后设计了BOM模型、DOM模型，最终形成JavaScript这样一个脚本语言。也可以说js使ES的实现和扩展。

Js是弱类型、动态的，弱类型体现在定义变量不需要声明类型，动态体现在变量类型随时可以发生转化。

JavaScript的数据类型有哪些？

一共有11种。

值类型(基本类型)：字符串（String）、数字(Number)、布尔(Boolean)、对空（Null）、未定义（Undefined）、Symbol。其中Symbol由ES6引入，表示独一无二的值。

引用数据类型：对象(Object)、数组(Array)、函数(Function)、map、set。其中map、set为ES6新增加的数据类型。

arguments

arguments 是一个对应于传递给函数的参数的类数组对象。

arguments对象是所有（非箭头）函数中都可用的局部变量。

//转换成真正的数组

var args = Array.from(arguments)

var args = [...arguments]

判断数据类型的方法？

**typeof x：**

返回值:数据类型

缺点:无法判断null和array,数组可以使用Array原型上的.isArray()方法，返回一个布尔值

**x instanceof y：**

原理：需要查找x的原型链，从而判别y是否在x的原型链上

返回值:boolean

缺点:无instanceof只能用来检测两个对象是否在一条原型链上，并不能检测出对象的具体类型

**x.constructor.name：**

返回值:构造函数的名称（可以代表数据类型）

原理：JS规定，每个构造函数都会有一个prototype属性，代表构造函数的原型对象，而原型对象中会有一个constructor属性指向构造函数。创建一个实例，这个实例会继承原型的constructor。

**Object.prototype.toString.call(x)：**

返回值:"[object 类型]"

原理：调用Object原型的方法

==和===的区别

===比较类型和值

==只比较值

js在比较的时候如果是 == 会先做类型转换，再判断值是否相等

Null和undefined的区别

console.log(null==undefined)//true

console.log(null===undefined)//false

Null类型，代表“空值”，代表一个空对象指针，使用typeof运算得到 “object”，所以你可以认为它是一个特殊的对象值。

undefined： Undefined类型，当一个声明了一个变量未初始化时，得到的就是undefined。

null:

1作为函数的参数，表示该函数的参数不是对象。

2作为对象原型链的终点。

undefined：

1变量被声明了，但没有赋值时，就等于undefined。

2调用函数时，应该提供的参数没有提供，该参数等于undefined。

3对象没有赋值的属性，该属性的值为undefined。

4函数没有返回值时，默认返回undefined。

事件委托/事件代理

事件委托就是利用事件冒泡，只指定一个事件处理程序，就可以管理某一类型的所有事件。

是利用事件冒泡的特性，将本应该注册在子元素上的事件处理程序注册在了父组件身上，

这样点击子元素时，发现其没有相应事件就会到父元素上寻找做做出响应，这样做的优势有：

１、减少dom操作，提高性能

２、随时都可以添加子元素，子元素也有相应的事件处理程序

事件流模型

1. 事件冒泡(fasle/不写)：当触发一个节点的事件是，会从当前节点开始，依次触发其祖先节点的同类型事件，直到DOM根节点。

2. 事件捕获(true)：当初发一个节点的事件时，会从DOM根节点开始，依次触发其祖先节点的同类型事件，直到当前节点自身。

3. 什么时候事件冒泡？什么时候事件捕获？

① 当使用addEventListener绑定事件，第三个参数传为true时表示事件捕获；

② 除此之外的所有事件绑定均为事件冒泡。

阻止事件冒泡和默认事件

阻止事件冒泡：

① IE10之前，e.cancelBubble = true;

② IE10之后，e.stopPropagation();

阻止默认事件：

① IE10之前：e.returnValue = false;

② IE10之后：e.preventDefault();

事件循环机制

**背景**

由于JavaScript 是一门单线程语言，所有的任务默认会按顺序执行，且无法并发地运行多个任务，因此一些非常耗时的任务就会影响后续任务的执行。基于此原因，提出一种事件循环机制，来解决这个问题。

**解决的思路**

将这些需要等待的、非常耗时的任务，挂起来，优先执行后续不需要等待的任务，当挂起来的任务具备继续执行的条件时（比如，收到服务器响应的数据），再回调这些任务继续执行。

**总方针**

先同步再异步，异步中先微任务再宏任务。

**具体机制**

将任务分为两种，同步任务、异步任务。

* 同步任务，立即执行的任务，也可以说排队依次执行的任务，同步任务一般会直接进入主执行栈，在主线程中执行。
* 异步任务，需要等待一段时间再继续执行的任务，比如ajax网络请求，setTimeout 定时函数等都属于异步任务，一般放在任务队列（task queue）。

同步和异步任务分别进入不同的执行环境，同步的进入主线程，即主执行栈，异步的进入任务队列。

Loop

{

在主线程上执行同步任务；

异步任务有了结果，就放入任务队列；

一旦主线程内（主执行栈）的任务执行完毕，主执行栈为空，就去任务队列读取对应的异步任务，推入主线程执行。

}

**补充说明：**

*任务队列存放的是异步任务的事件。虽然说是去任务队列读取任务，实际上任务队列存储的使一系列的IO事件、点击事件、页面滚动事件。*

*推入执行栈的是回调函数。所谓"回调函数"（callback），就是那些会被主线程挂起来的代码。异步任务必须指定回调函数，当主线程开始执行异步任务，就是执行对应的回调函数。*

*setTimeout函数，指定时间到达之后，会将回调函数放入任务队列中，具体什么时候执行要看主线程是否有空，所以实际的延时往往大于预先定义的事件。*

参考链接：

<https://www.ruanyifeng.com/blog/2014/10/event-loop.html>

<https://www.jianshu.com/p/aa6108c9bff6>

如何理解闭包？

闭包是JavaScript的一个特色、一个难点，一些js高级应用需要依赖闭包来实现。

**变量的作用域：**

Js中有全局变量、也有局部变量，他们处在不同的作用域。当前作用域可以访问上层作用域的函数与变量，上层作用域不能访问当前作用域。其实也就是说，全局变量的作用域更大一些，局部变量的作用域小一些，局部变量只能在局部使用，而全局变量可以在全局使用。

**Js链式作用域：**

Js遇到变量或者函数，首先在当前作用域进行寻找，找到则直接使用，找不到则到上层作用域寻找，如果还找不到就以此类推向上层搜索。

**处于全局作用域，如何使用一些函数内部的局部变量？**

刚才在[变量的作用域]部分已经说了，默认情况下，在当前作用域外部无法访问当前作用域的变量。但是，如果非要访问，还是存在一些办法的。可以在函数内部再定义一个函数,让子函数去访问父函数的变量，然后父函数return子函数。

闭包的简单理解就是：

父函数内部定义子函数，从而实现访问父函数的变量。

一个子函数在父函数内部被定义，然后被父函数返回。

**闭包的优势：**

* 实现在全局中使用局部变量（如读取函数内部变量），且不造成变量污染。
* 读取函数内部变量，并使长期驻留在内存中，可以重复使用，避免垃圾回收机制。（原理，子函数被赋给一个全局变量，而全局变量位于内存中，因此子函数会停留在内存，由于子函数的存在依赖于父亲函，因此间接导致父函数及其父函数的变量也会停留在内存而不会被清理）
* 可以用来定义私有属性和私有方法。

*全局变量，可以重复使用，容易造成变量污染。*

*局部变量，不可以重复使用，不造成变量污染。*

*闭包结合了两者的优势。*

**闭包的劣势（2个风险）：**

* 内存泄露风险。因为闭包使得一些变量可以驻留在内存，因此会导致内存消耗比较大，所以不可以滥用。应对方法，退出函数之前，将不使用的局部变量全部删除。
* 函数局部变量被修改的风险。由于闭包可以间接地操纵函数的私有变量，因此也要谨慎使用闭包。

Js如何避免全局变量污染？

变量污染是什么？

当我们定义全局变量时，在a.js和b.js中都定义了变量a，那么这个时候就会出现问题，比如在使用a的时候会互相覆盖。这种情况，就称之为：全局变量污染。也可以理解为全局变量的命名冲突。

原因是什么？

过多地定义全局变量所导致

如何避免？

* 尽量少定义全局变量，多使用局部变量
* 使用namespace模式，即使用命名空间将变量封装在对象里边
* 使用IIFE模式，即匿名函数自调用/自执行。IIFE中定义的任何变量和函数，都会在执行结束时被销毁。这种做法可以减少闭包占用的内存问题，因为没有指向匿名函数的引用。只要函数执行完毕，就可以立即销毁其作用域链了。
* 使用闭包

函数内部声明变量，不加var会如何？

这个变量会自动成为一个全局变量（不加var潜台词就是在说，这个变量已经定义过了，不需要再定义了，因此会成为全局变量），加上var才能表示局部变量。

IIFE是什么？

**全称：**

immediately invoked function expression

**写法：**

（funciton（）{}（））；

（funciton（）{}）（）；第一个括号代表独立的作用域，第二个括号使立即执行。

因为函数在创建的时候就被执行了，函数名的存在没有意义，因此一般都会省略函数名。

当加上括号的那一刻起，这个声明函数就会被当作表达式来对待。

这种写法告诉解释器这不是一个声明函数，且需要立即执行。

执行流程：

声明函数-函数表达式-立即执行-销毁 / 声明-执行-销毁

**出现的背景：**

在ES6提出块级作用域之前，ES5中只有全局作用域和函数作用域，js只能通过函数来实现作用域的隔离。

为了实现作用域的隔离，经常会创建一些函数，而一些函数往往仅被调用一次就再也不用。而这是有违函数的初衷的，因为一般来讲，设计函数是为了实现代码复用。因此，既然该函数仅被调用一次，那么在其声明的时候立即执行然后销毁就可以了。实现这种立即执行的编码模式称之为IIFE模式。

**实现的目的：**

隔离作用域，防止对全局命名空间的污染。

垃圾回收机制和内存泄漏

**内存泄露？**

是指当一块内存不再被应用程序使用的时候，由于某种原因，这块内存没有返还给操作系统或者内存池的现象。内存泄漏的原因：

* 闭包常驻内存
* 大量使用全局变量
* 忘记清除的定时器
* Dom的引用。比如dom树已经删除某个节点，但是该节点依然被引用。

**垃圾回收机制？**

JavaScript的垃圾回收机制就是为了应对内存泄漏

**哪些东西要回收？**

在JavaScript中，一般来说没有被引用的对象就是垃圾，就是要被清除；有个例外如果几个对象引用形成一个环，互相引用，但根访问不到它们，这几个对象也是垃圾，也要被清除。

**怎么回收？**

JavaScript 引擎中有一个后台进程称为垃圾回收器，它会通过“标记&清除算法”监视所有对象，并删除那些不可访问的对象。

标记清除算法将“不再使用的对象”定义为“无法到达的对象”。即从根部（在JS中就是全局对象）出发定期（周期性）扫描内存中的对象，凡是能从根部到达的对象，保留。那些从根部出发无法触及到的对象被标记为不再使用，回收。

Javascript 原型链之原型对象、实例和构造函数三者之间的关系

**原型是什么？**

① 所有引用类型都有一个\_\_proto\_\_(隐式原型)属性，属性值是一个普通的对象   
② 所有函数对象都有一个prototype(原型)属性，属性值是一个普通的对象。  
③ 所有引用类型的\_\_proto\_\_属性指向它构造函数的prototype

④ 所有对象都有\_\_proto\_\_

⑤ 最顶层的原型对象Object.prototype，这个对象中保存了最常用的方法，如toString、valueOf、hasOwnProperty等，因此我们才能在任何对象中使用这些方法。

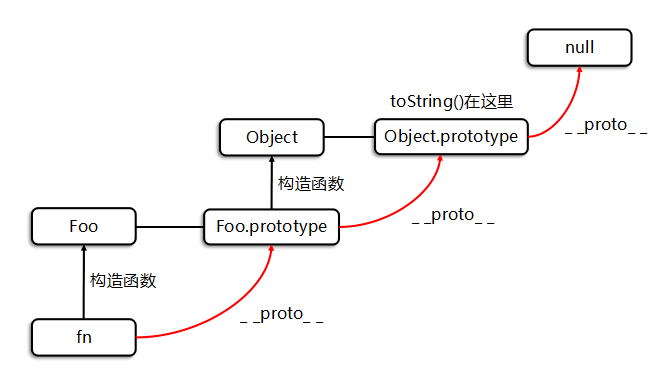
⑥ prototype是函数才具有的属性，默认情况下，new 一个函数创建出的对象，其原型都指向这个函数的prototype属性。

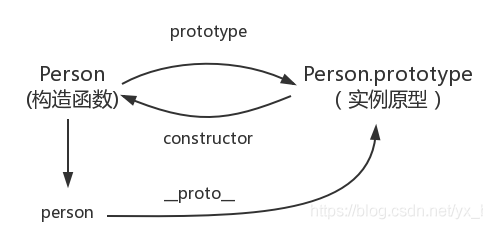
**原型链是什么？**

①当访问一个对象的某个属性时，会先在这个对象本身属性上查找→对象的\_\_proto\_\_隐式原型上查找→原型的\_\_proto\_\_中查找，这样一层一层向上查找就会形成一个链式结构，我们称为原型链。（本身-原型-原型的原型-undefined）

②一直往上层查找，直到到null还没有找到，则返回undefined   
③Object.prototype.\_\_proto\_\_ === null

java中有父类、子类之分，但是JavaScript没有父类、子类这些概念。本质上来讲，JavaScript利用原型链描述了对象之间的继承关系。





构造函数被创建的时候，自动分配一个prototype指针，指向它的原型对象；原型对象的constructor指向构造函数本身。由此形成一个小闭环。

构造函数创建一个实例，实例会自动分配一个\_\_proto\_\_指针，指向构造函数的原型对象。可见，构造函数和实例之间无法直接读取，需要通过\_\_proto\_\_指针间接访问。需要注意的是：实例没有prototype指针，只有\_\_proto\_\_指针。

因此可以得出关系：instance.constructor.prototype = instance.\_\_proto\_\_

**prototype指针的引入，有两个好处：**

识别实例对象的构造函数（间接地判断了本身类型）

由实例出发，构造一个同类型的实例。实例2 = 实例1.constructor（）

参考链接：

<https://www.cnblogs.com/thonrt/p/5900510.html>

<https://blog.csdn.net/xiaoermingn/article/details/80745117>

<https://blog.csdn.net/yx_back/article/details/91470917>

补充：

原型对象好比一个公共区域，所有实例都可以访问这个公共区域，因此可以把实例对象中共有的内容放到原型对象，节省资源。

prototype叫做显式原型

\_\_proto\_\_叫做隐式原型

如果当前实例不存在属性，那么就一直沿着它的隐式原型向上寻找。

如何创建对象？

**字面量**

var a = new Object()

var a = {}

a的原型就是Object.prototype

Object.getPrototypeOf(a)===Object.prototype===a.\_\_proto\_\_

**构造函数**

var xiaoming = new Person（）

xiaoming 的原型就是Person.prototype

**Object.create（）**

var obj1 = {};

var obj2 = Object.create(obj);

obj2.\_\_proto\_\_ === obj1;

obj2的原型（\_\_proto\_\_）obj1

**class**

使用class关键字创建，ES6特性

复制对象的方法（深拷贝与浅拷贝）

**浅拷贝**

引用指针。数据类型为引用类型的时候，仅仅复制内存地址，而不是复制对象的内容。源对象改变内容，浅层复制来的对象内容也会随之改变。

* obj1 = Object.assign(obj);
* obj2 = Object.create(obj);
* 引用类型直接赋值

**深拷贝**

* **序列化（深）**

var newObj = JSON.parse(JSON.stringify(obj));

Function、undefined会被忽略掉

* **遍历赋值（视情况而定）**

将把所有的属性遍历一遍，赋值给新对象。

* **扩展运算符（视情况而定）**

var { ...obj2 } = obj

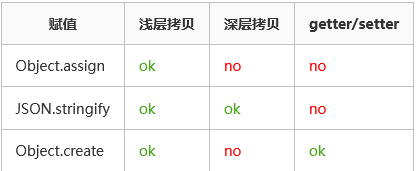
* **递归实现（最优解）**

**补充：**

**针对数组，可以使用concat或者slice 拷贝（视情况而定）**

var a = [1,2,3];

var c = [].concat(a);



判断对象为空的方法

**Object.keys()**

返回一个包含所有给定对象自身可枚举属性名称的数组。

**Object.getOwnPropertySymbols()**

返回一个数组，它包含了指定对象自身所有的符号属性。

**转为字符串**

JSON.stringify(jsonobj);

Json对象转化为json**字符串**，再判断该字符串是否为"{}"

**遍历**

for in 循环判断

普通函数和箭头函数的区别

写法不一样

箭头函数This取决于上一级this

箭头函数没有原型，向上无法继承，向下无法构造实例

箭头函数必然为匿名函数

变量提升、函数提升

js引擎会在代码执行之前进行预编译，把声明式函数、var做定义的变量提到当前作用域顶部进行声明。函数提升的优先级最高，对函数的声明会提升至作用域最顶端。

**注意：**

函数提升，提升的是整个函数体。

变量提升，仅提升定义，赋值操作不会被提升。

作用域和作用域链

**作用域**

JavaScript有三种作用域：全局作用域、函数作用域、块级作用域

利用作用域隔离，可以起到隔离变量的作用，避免了变量的命名冲突，不同作用于下可以定义同名变量，互不影响。

所有未定义直接赋值的变量，都会自动声明为全局变量。

作用域是分层的，内层可以访问外层变量，外层一般不可以使用内层变量除非使用闭包。

**作用域链**

寻找变量的时候，首先在当前作用域寻找，没有则去上层作用域寻找，如果还没有就继续向上层寻找，如果最终到全局作用域仍然没找到需要的变量，那么就说明找不到了直接放弃寻找。这样从内层向外层或者说从下层至上层这样一层一层的搜索链条称之为作用域链。

**补充：**

作用域一经定义就会确定下来，不会再改变。

执行上下文，在代码执行时才会确定，随时可以改变。

//ES6新特性

Promise

class类

let,const

结构赋值

箭头函数

函数参数的默认值default

Spread / Rest 操作符

==>rest操作符让函数的所有参数可由一个变量统一接收 ，帮助我们创建更加灵活的函数

==>spread运算符比较经常数组的解析与构造for...of 和 for...in

==>for…in是遍历数组(尽量不用)、对象的key:不仅遍历数字键名，还会遍历手动添加的其它键，甚至包括原型链上的键

==>for…of是遍历数组,类数组，Map,Set的value

对象超类：使用super关键字可以调用父级的方法

==>Object.setPrototypeOf(child,parent)：设置置child的对象的原型（即内部 [[Prototype]] 属性）

数字前面添加 0o 或者0O 即可将其转换为八进制值：

//Promise的理解

let、var 和 const 的区别

**变量提升**。Var可以自动，而let、const不会被提升，只能手动提升到顶部。

**作用域。**Var作用于所定义的函数/全局作用域，let 和 const 声明的仅作用于块级作用域。

**声明、修改、初始化方面。**Var可以重复声明、可以修改、可以初始化也可以不初始化；let不能重复声明、可以修改、可以初始化也可以不初始化；const一经声明不可修改，且必须初始化。

补充：

块级作用域由{}包括，比如一个for循环就是一个块级作用域，在循环体内部用let声明的变量只在该循环体内部有效

This指向

1.浏览器，在全局范围内this 指向window对象

2.函数，this永远指向最后调用他的那个对象；

3.构造函数，this指向new出来的那个新的对象；

4.call、apply、bind中的this被强绑定在指定的那个对象上；

js给函数内置的一些API，调用他们可以为函数指定this的执行,同时也可以传参。

5.箭头函数中this比较特殊,箭头函数this为父作用域的this，不是调用时的this.要知道前四种方式,都是调用时确定,也就是动态的,而箭头函数的this指向是静态的,声明的时候就确定了下来；

script 标签后加 async 属性和 defer 属性有什么区别

1.<script src="example.js"></script>

没有defer或async属性，浏览器会立即加载并执行相应的脚本。也就是说在渲染script标签之后的文档之前，不等待后续加载的文档元素，读到就开始加载和执行，此举会阻塞后续文档的加载；

2.<script async src="example.js"></script>

有了async属性，表示后续文档的加载和渲染与js脚本的加载和执行是并行进行的，即异步执行；

3.<script defer src="example.js"></script>

有了defer属性，加载后续文档的过程和js脚本的加载(此时仅加载不执行)是并行进行的(异步)，js脚本的执行需要等到文档所有元素解析完成之后，DOMContentLoaded事件触发执行之前。

AMD、CMD、ES6、CommonJS 的区别

js规范

1. 第一种是 CommonJS 方案，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. 第二种是 AMD 方案，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. 第三种是 CMD 方案，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. 第四种方案是 ES6 提出的方案，使用 import 和 export 的形式来导入导出模块。

==>1.commonjs输出的，是一个值的拷贝，而es6输出的是值的引用；

==>2.commonjs是运行时加载，es6是编译时输出接口；

防抖与节流

**背景**

当某个事件频繁触发时，事件处理函数会频繁执行。如果处理函数不消耗资源、不消耗时间，那么也不会出现什么异常现象；如果处理函数有一些费时、耗性能的操作，就会导致页面出现卡顿甚至浏览器崩溃，为了解决这个现象，就需要节流和防抖。简单来说，立即执行函数。

**实现**

防抖（防止抖动），闭包加定时器。如果某个频繁触发的事件在规定的时间内没有再次触发，则执行事件处理程序；如果在这段时间内事件再次触发了，则重新计时准备延时执行。简单来说，延迟N秒执行函数。

节流（节省流量），使用时间戳。如果频繁触发事件，就每隔一段时间处理一次。

**共同点**

两者都限制了事件处理程序的执行频率，有效防止事件处理函数的频繁执行。

**不同点**

在实现思路上有所不同。

防抖，与上次执行的间隔小于指定间隔时间，那么延时执行。假如，设置间隔为1秒，在一秒内连续点击多次，只会执行最后一次。

节流，与上次执行的间隔大于指定间隔时间，执行，隔一段时间才会触发一次。假如，设置间隔为1秒，在一秒内连续点击多次，只会执行第一次。

延迟N秒执行，规定时间内连续触发事件，响应最后一次事件。

立即执行，规定时间内连续触发事件，响应第一次事件。

==> JSONP

核心原理：script 标签不受同源策略约束，所以可以用来进行跨域请求，优点是兼容性好，但是只能用于 GET 请求；

==>img 标签，link 标签，script 标签

原生 js 实现 ajax(手写代码)

封装 axios(手写代码)

图片懒加载的实现原理（手写代码）

关于call和apply

这两个方法的作用是一样的：都是在特定的作用域中调用函数，等于设置函数体内this对象的值，以扩充函数赖以运行的作用域。

一般来说，this总是指向调用某个方法的对象，但是使用call()和apply()方法时，就会改变this的指向。

call,apply,bind

**相同之处：**

它们在功能上没有区别的,都是改变this的指向，扩充了函数的作用域。

**不同之处：**

返回值不同

call和apply方法都是在调用之后该函数会立即执行。

bind调用之后是返回该函数，可以稍后执行。如果需要立即执行，需要添加()。

传参不同

call，第一个参数为this指向的对象，其他参数逗号隔开

apply，第一个参数为this指向的对象，其他参数放入一个数组

bind，第一个参数为this指向的对象，其他参数逗号隔开

此外，

bind一旦绑定，函数将会永远绑定在第一个对象上。

bind无法改变构造函数的指向

Function.prototype.call（）

Function.prototype.apply（）

Function.prototype.bind（）

参考连接：

<https://www.cnblogs.com/Shd-Study/p/6560808.html>

箭头函数

()=>{}

箭头函数也是匿名函数，不能作为构造函数，不能使用new

箭头函数不绑定argumets，取而代之用rest参数解决

箭头函数会捕获父类的上下文来作为自己的this值

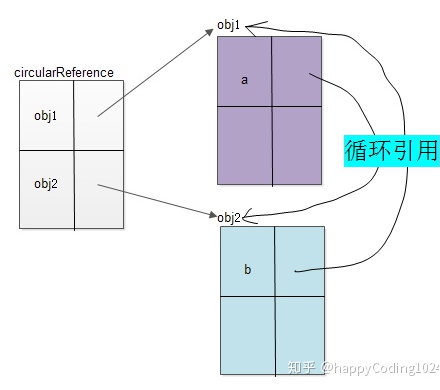
箭头函数没有原型

箭头函数的this取决于父级作用域，而且一经定义this就会确定下来，是静态的，任何方法（比如 call、apply、bind）都改变不了指向。

Js对象循环引用

本质：某一个属性值或者内部对象的属性值是引用型，而且指向某一个外层对象，从而形成一个环。

例子：当对象 1 中的某个属性指向对象 2，对象 2 中的某个属性指向对象 1 就会出现循环引用。当然不止这一种情况，不过原理是一样的。



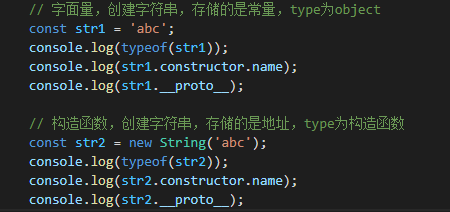
出现循环引用的对象，JSON序列化的时候会报错。

注意点

全局作用域中声明的数字，会自动转化为字符串类型，但在其他作用域中不会发生自动转化。

var name = 1;

创建字符串的两种方式



new的具体过程

总体来讲，分为以下四个步骤：

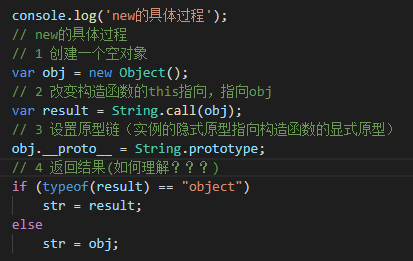
创建空对象

改变构造函数的指针，指向空对象

修改原型

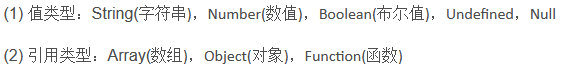
返回

最后的返回不知道要如何理解？？？



基本数据类型和引用类型的区别

定义、存储、销毁、直接复制



基本数据类型，会将变量存放在栈内存中。函数执行完毕自动销毁。直接赋值是深复制。

引用类型，会将变量名（地址）存放在栈内存中，变量的值存放在堆内存。不会自动销毁，通过垃圾回收机制进行回收。直接赋值是浅复制。



页面有一个iframe，将主页面的一个数组传到iframe中再用 instanceof 判断数组是否可行？

不可行，在iframe中无法判断，因为iframe和window的作用域不一样， instanceof 在iframe无法判断一个数组是不是构造函数的实例。

在iframe中要想正确判断只有利用iframe中的构造函数来判断

参考链接

<https://blog.csdn.net/qq_21896051/article/details/117605506>

const 声明生成对象的时候，如何使其不可更改

const声明的基本数据类型，值不可改变

引用类型，由于变量名存储的仅仅是地址，地址不变，但是引用类型内部的属性/值是可以修改的。

如何使其绝对不可修改？

* 如果没有嵌套的引用类型，Object.freeze(obj)冻结即可。（只能冻结一层）
* 如果有嵌套的引用类型，需要使用DFS深度优先遍历，将所有的引用类型都冻结一遍。

<https://www.runoob.com/js/js-let-const.html>

Promise,手写Promise.all()

背景：

有N个异步任务，需要按照顺序执行。假如按照以往的写法，那么就需要在第i个回调函数里边调用第i-1个函数，形成一个嵌套的结构，可维护性较弱，可读性差。

ES6的异步解决方案：

为了解决这个问题，引入了promise对象，有了promise对象，可以一定程度上避免层层嵌套的不良结构。另外，使用Promise.all（）可以批量操作异步任务。

ES7的异步解决方案：

async / await

**三、VUE**

mvvm框架

M：model 数据源

V:view 视图（页面）

VM：viewmodle（将数据源与视图关联起来）这也是VUE的实例化对象

一是将【模型】转化成【视图】，即将后端传递的数据转化成所看到的页面。实现的方式是：数据绑定。  
二是将【视图】转化成【模型】，即将所看到的页面转化成后端的数据。实现的方式是：DOM 事件监听。

同时实现了一个双向绑定，页面数据改变，model自动更新。Model改变，页面上也会自动更新。

MVC的思想：

一句话描述就是Controller负责将Model的数据用View显示出来，换句话说就是在Controller里面把Model的数据赋值给View

与MVC相比，最大的区别就是MVVM实现了自动的双向数据绑定。

v-if和v-show的区别

v-if是条件渲染，如果条件为假，那么这个元素会直接不被渲染

v-show，如果条件为假，那么这个元素会被渲染，只是不被显示出来。Display=none

如果显示与不显示需要频繁的切换，那么这种情况下使用v-show更合适，开销更小。

1. **HTML**

阿松大

啊大苏打

1. **CSS**

css单位px，rem，em，vw，vh的区别

px，像素的基本单位，相对长度单位，网页设计常用的基本单位，像素px是相对于显示器屏幕分辨率而言的。

%,相对长度单位，相对于父元素的百分比值

em，相对于父元素的字体大小而言

rem，相对于html根元素的字体大小而言

vh，相对于浏览器的高度。假如浏览器高1002px，那么1vh = 10.02px

vw，相对于浏览器的宽度。

vmin，相对于浏览器高度、宽度中较小值

vmax，相对于浏览器高度、宽度中较大值

备注：

浏览器默认字体为16px

vh,vw,vmin，vmax他们都是动态的，时刻依赖于当前视图的大小，当浏览器尺寸变化时，他们的单位值也会跟着改变。

移动端的布局与倍率问题

CSS选择器的权重

css的权重可以分为6级

1.权重 = 无限大/10000

!important 无穷大；它的权重最高；

2.权重 = 1000

内嵌样式(style)；

3.权重 = 100

id选择器；

4.权重 = 10

类选择器

伪类选择器(:hover, :link, :active, :visited)

属性选择器；

5.权重 = 1

标签选择器

伪元素选择器（::after，::before, ::first-line, ::first-letter);

6.权重 = 0

继承的样式，

\*通配符的样式

它们的权重依次降低。

Flex**布局**详解

参考链接

<https://blog.csdn.net/jarvan5/article/details/115361945>

<https://blog.csdn.net/qq_23260455/article/details/115325654>

flex全称Flexible Box，弹性盒子布局，是一种布局方式。

**解决的问题**

* 解决元素居中问题
* 自动伸缩，适配屏幕。自动弹性伸缩，合适适配不同大小的屏幕，以及在移动端方面也有广泛的应用。

**两条轴**

主轴，决定了元素的排列方式。

交叉轴，垂直于主轴的轴。

**容器的6个重要属性**

1. flex-direction 主轴方向

2. flex-wrap 主轴一行满了换行

3. flex-flow 1和2的组合

4. justify-content 主轴元素对齐方式

5. align-items 交叉轴元素对齐方式//单行

6. align-content 交叉轴行对齐方式//多行

**与容器相关的元素属性**

1. flex-grow： 长大
2. flex-shrinik: 缩小
3. align-self: 覆盖container align-items 属性
4. order 排序
5. flex-basis: 有效宽度

BFC

BFC(Block formatting context)，直译为"块级格式化上下文"，它规定了内部的Block-level Box如何布局，并且与这个区域外部毫不相干。

BFC是一个独立的布局环境，其中的元素布局是不受外界的影响，并且在一个BFC中，块盒与行盒（行盒由一行中所有的内联元素所组成）都会垂直的沿着其父元素的边框排列。

**作用：**

1.避免外边距margin重叠。多个带有外边距的元素排列，默认情况下，外边距会发生折叠。通过激活每个元素的BFC，使其相互独立，彼此不受外界影响。

2.包含子元素margi带跑父元素。子元素的margin可能会把父元素也带跑，为了防止带跑父元素，激活父元素BFC，既可防止坍塌。

3.防止兄弟元素的遮盖问题，一般是非浮动元素被浮动元素遮盖。为了使非浮动元免受浮动元素影响，激活非浮动元素的BFC，即可消除影响。应用场景：自适应两栏布局。

4.包含浮动元素，防止高度坍塌。如果父元素内有浮动子元素，那么父元素的高度可能会发生坍塌。为了免受浮动子元素影响，激活父元素的BFC，将子元素全部包裹起来，消除了子元素的浮动影响。

*前两个都是关于margin*

*后两个都是关于浮动*

**三个重要的要点：**

* BFC的区域不会与float box重叠。
* BFC就是页面上的一个隔离的独立容器，容器里面的子元素与外面的元素互不影响。
* 计算BFC的高度时，浮动元素也参与计算

**如何激活BFC？**

有还能多条件都可以激活BFC，一般可以用overflow：hidden来激活

**备注：**

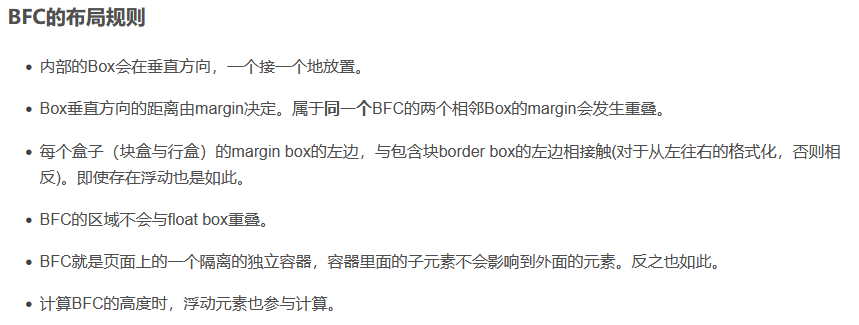
此外，Inline formatting context (简称IFC)

**参考链接：**

<https://blog.csdn.net/sinat_36422236/article/details/88763187>

<https://www.itcast.cn/news/20201016/16152387135.shtml>

<https://blog.csdn.net/riddle1981/article/details/52126522>



display:none,overflow:hidden,visibility:hidden之间的区别

display:none，看不见，占据空间。

overflow:hidden，看不见，占据空间。

visibility:hidden，隐藏溢出，一般可以用来激活BFC。

1. **数据结构与算法分析**

js实现函数

function isArray(){}

function isNaN(){}

// constructor instanceof Object.prototype.toString.call

**其他**

设计模式

一、单例模式

==>确保只有一个实例，并且提供全局访问

==>var a = {},但是会造成全局污染

二、策略模式

==>策略模式的目的就是将算法的使用与算法的实现分离开来。

==>定义一系列的算法，把它们一个个封装起来，并且使它们可以相互替换。

三、代理模式

==>图片懒加载实现的过程就是通过代理模式中的虚拟代理来实现的

==>为其他对象提供一种代理以控制对这个对象的访问。在某些情况下，一个对象不适合或者不能直接引用另一个对象，而代理对象可以在客户端和目标对象之间起到中介的作用。

四、迭代器模式

==>迭代器模式是指提供一种方法顺序访问一个聚合对象中的各个元素，而又不需要暴露该对象的内部表示。

==>for循环

五、发布—订阅模式

==>js事件就是经典的

==>发布-订阅模式里面包含了三个模块，发布者，订阅者和处理中心。

六、命令模式

==>命令模式中的命令(command)指的是一个执行某些特定事情的指令。

==>通过不同的命令做不同的事情，常含有（关联）接收者。

==>将一个请求封装为一个对象，从而使你可用不同的请求对客户进行参数化

七、组合模式

==>组合模式将对象组合成树形结构，以表示“部分-整体”的层次结构。

==>虚拟dom：将真实的DOM的数据抽取出来，以对象的形式模拟树形结构

八、模板方法模式

==>一种只需使用继承就可以实现的非常简单的模式。

==>模板方法由两部分组成，一是抽象的父类，二是具体实现的子类。通常在抽象父类中封装了子类的算法框架，包括实现一些公共方法以及封装子类中所有方法的执行顺序。

==>ypescript中就有abstract中的抽象类和抽象方法就是使用的模板方法模式

九、装饰者模式

==>允许向一个现有的对象添加新的功能，同时又不改变其结构。

==>为对象添加新功能；不改变其原有的结构和功能

==>ecorator 是 ES7 的一个新语法;装饰器语法

十、职责链模式

十一、中介者模式

十三、状态模式

十四、适配器模式

十五、外观模式

十六、享元模式

字节

字节

with 的了解

EventLoop JS事件循环队列、宏任务和微任务

$set vue响应式更新

this.$set('a','b',1)

this.a.b = 2

// 视图如何变化

输出什么？

复制代码

1

2

3

4

5

6

7

for(let i = 0; i < 10; i++) {

this.a = i;

this.$nextTick(() => {

console.log(i);

console.log(this.a)

})

}

WebSocket 在服务端是怎么处理消息的？

· 利用 CORS 解决跨域问题，除了 CORS 还有其他什么方案？（JSONP 、反向代理......）

· JSONP原理

栈的特点？[链表](https://www.nowcoder.com/jump/super-jump/word?word=%E9%93%BE%E8%A1%A8" \t "https://www.nowcoder.com/discuss/_blank)和数组的对比，有什么区别

作者：一点点很美味

链接：https://www.nowcoder.com/discuss/827563?source\_id=discuss\_experience\_nctrack&channel=-1

来源：牛客网

强缓存和协商缓存,什么情况下会用到协商缓存？

为什么要有协商缓存，什么东西会用到

请求头的Cookie

http://a.com cookie: b:1

-ajax 发送到 http://b.a.com. (能不能收到这个cookie)

18. 如果我要跨域发送cookie，怎么发

19. cookie的属性有哪些

20. httponly 是什么，怎么用，为什么要用

21. 代码题：

复制代码

1

2

// 力扣，最小栈

// 螺旋打印矩阵

字节飞书

栈和队列的区别

如何用两个栈实现队列？

进程间通信的方式？

共享内存的优缺点

八种[排序](https://www.nowcoder.com/jump/super-jump/word?word=%E6%8E%92%E5%BA%8F" \t "https://www.nowcoder.com/discuss/_blank)简述一下？冒泡、快速、归并、希尔、堆、之类的反正都说了，说了大概思路和怎么实现的

手写快速[排序](https://www.nowcoder.com/jump/super-jump/word?word=%E6%8E%92%E5%BA%8F" \t "https://www.nowcoder.com/discuss/_blank)

设计模式有没有了解？简单说一下？我说了两种

vue父子组件间的通信方式？

字节跳动

· 事件循环

· 虚拟DOM

HTTP2.0/3.0

· 前端安全，CSRF 如何防护？

算法：

假设一共有 n 个人，给出一个数组指出哪些人可以把东西传给哪些人？最大传递次数为k。计算传给第 n-1 个人一共有几种方式，传不到返回0  
Example: n = 5, arr = [[0, 2], [0, 4], [2, 1], [1, 3], [4, 1], [2, 3], [3, 4]], k = 3

一道闭包

一道promise

一道[二叉树](https://www.nowcoder.com/jump/super-jump/word?word=%E4%BA%8C%E5%8F%89%E6%A0%91" \t "https://www.nowcoder.com/discuss/_blank)求路径和

字节本地生活一面

手写jsonp

关于作用域链

关于this

字节-飞书

进程和线程

线程共享为什么比进程共享容易？

SPA是什么？

路由

绝对定位 position：absolute

TCP是怎么判断丢包的？

Cookie Session Token的原理

平时怎么学习

类选择器和伪类的区别及优先级？

CSS选择器的优先级

算法：斐波那契（尾递归优化）

如何判断数据类型，语法糖是啥

Vite原理

字节国际化

输入url到页面渲染完成

4.TCP/IP协议分层,TCP在哪一层

5.三次握手 四次挥手,为什么挥手多一次

6.浏览器缓存,强缓存、协商缓存,为什么cache-control优先级更高

7. no-cache,no-store分别代表什么

8. html、css、js文件会放在磁盘缓存还是内存缓存

9. DOM树和css om树是互斥的还是同时的

10. JS脚本阻塞DOM构建,js脚本会不会对css om树影响

11.重绘、重排概念

看题说输出

var name = 'window'

const obj = {

name: 'obj',

sayName:function() {

console.log(this.name)

},

}

obj.sayMyName = () => {

console.log(this.name)

}

const fn1 = obj.sayName()

const fn2 = obj.sayMyName()

fn1() // window

obj.sayName() // obj

fn2() // window

obj.sayMyName() // window

有哪些类型的作用域

1. var let const区别

eventbus实现一下

http2.0和http1.0的区别

3.多路复用怎么实现的

4.https和http之间的区别，https握手过程

5.中间人攻击如何实现？怎样防范？

6.position取值,reletive相对谁

7.flex属性,flex布局

第二个子元素的高度是多少

<div class="container">

<div style="height: 100px"></div>

<div style="min-height: 10px"></div>

</div>

<style>

.container{

display: flex;

}

.container > div {

width: 100px;

}

</style>

事件循环讲解

看代码说输出

async function async1(){

console.log('async1')

await async2()

console.log('async1 end')

}

async function async2(){

console.log('async2')

}

console.log('script start')

setTimeout(() =>{

console.log('setTimeOut')

},0)

async1()

new Promise((resolve) => {

console.log('promise')

resolve()

}).then(() =>{

console.log('promise2')

})

console.log('script end')

继承有哪些方法

Vue常见的组件通信方法

13. 兄弟组件通信你会考虑用哪些方法

14. Vue MVVM实现思路

15. proxy对比defineproperty的优势

如何理解闭包

7.如何理解面向对象

阿里菜鸟

判断回文序列

数组相邻k个元素最大子序列求和

正则表达式的运用

青蛙跳台

CSS各种定位、每种定位的特点

Flex布局介绍及其意义

ES6新特性

Promise,Generator原理，优缺点

怎么保护cookie不被泄露

介绍token，具体原理，怎么应用

介绍XSS，CSRF,怎么防范，它们之间的区别

Vue3相比于Vue2做了哪些改变？

为什么要废弃一些api？

Vue中的通信方式，传参方式，传参的底层原理？

Vuex的设计原理

奇安信

防抖和节流区别，手写防抖

箭头函数和普通函数区别

有关this的输出

项目中的难点以及解决方案

echarts中数据的差距比较大，导致最小数据那个图的看不到，如何去解决，有一种设计模式可以抽象的去解决，了解过吗？

项目中观察者和发布订阅模式区别

websocket的连接方式，如何去建立，连接过程，服务端要进行那些设置，不支持websocket的浏览器的解决方案

cookie,session登录鉴权和token差别，csrf如何取防范

左侧固定，右侧自适应的两栏布局都有哪些

对笔试中的复盘，最后的括号匹配解决思路

手写Promise.prototype.finally

CVTE

html5 语义化标签 （真不知道是什么）label的作用 header

promise是什么 promise 有哪些方法，promise.all 和 promise.race 的区别（这两者里面的promise的执行顺序） 实现一个20个请求的串行和并行 promise（我讲了链式调用 但是面试官并不太满意，更优雅的方式）

浏览器本地化存储 cookie sessionStorage localStorage indexDB

自己常用哪些布局

flex 布局有哪些属性

双飞翼和圣杯布局（只讲了用 flex 实现的，浮动的真难记）如果中间内容宽度超过实际宽度怎么办

ES6 新属性

箭头函数的 this

对 JavaScript 里面的面向对象怎么理解 讲了类 组合继承 （问我还有什么其他方式实现继承）

项目里面用到的技术难点 websocket

首屏加载速度可以怎么改善

如何提高页面加载速度 （讲了缓存 代码压缩 ）

http缓存机制 强缓存的时候状态码是什么

cookie 有什么值（似乎把 secure 和 httponly 弄混了）

图片的懒加载和预加载（忘了）

代码压缩是怎么配置

怎么拿到 dom 节点 如何算出页面中所有节点的数目

手动实现一个 bind（忘了）

怎么改变 this 的指向

session 是什么

事件循环机制

文件传输怎么实现

图片展示有哪几种实现方式

讲到了export和import，那么还有了解其他的模块化加载机制吗？他们之间有什么区别

阿里

为什么要用Virtul Dom，Virtual Dom的效率一定比直接操作Dom高吗，你说Virtual Dom减少不必要的Dom操作，让你来设计如何实现这个效果

flex布局水平垂直居中

less和sass有用过吗

作者：注意run时机  
链接：[https://www.nowcoder.com/discuss/756937?channel=-1&source\_id=discuss\_terminal\_discuss\_history\_nctrack&ncTraceId=f36a988fea6d4201bfad304aa83a2438.3857.16474138896851978](https://www.nowcoder.com/discuss/756937?channel=-1&source_id=discuss_terminal_discuss_history_nctrack&ncTraceId=f36a988fea6d4201bfad304aa83a2438.3857.16474138896851978" \t "_blank)  
来源：牛客网

1、句子单词逆序

how old are you => you are old how

\*/

/\*

2、金额格式化

1000000 => 1,000,000

10.24 => 10.24

\*/

/\*

3、数字查找和排序 -  找出数组中重复最多的元素

var arr = [1, 2, 3, 1, 2, 3, 4, 3, 3, 5, 3];

a、找出数组中重复最多的数字。

b、重复最多的数字最先开始的位置。

c、重复最多的数字的数量。

// a) 3

// b) 2

// c) 5

\*/

/\*

4、CSS颜色转换

#0000FF => rgb(0, 0, 255)

#A37 => rgb(170, 51, 119)

#huahs => invalid

\*/

/\*

5、运用你所掌握的数据结构，设计和实现一个  LRU (最近最少使用) 缓存机制 。

实现 LRUCache 类

\*/

项目中遇到的难点？

为什么用vue，vue的优点

vue数据双向绑定原理

1分、2分、3分、5分硬币若干枚，有多少种拼凑到100分的可能

前端性能优化？从输入url到浏览器渲染的流程中的每个部分都阐述一遍优化思路

口述快排思路

不同浏览器引擎之间的差异及原理

跨端系统实现方案、原理

抖音、支付宝、微信、QQ小程序之间的区别

问了浏览器引擎和小程序实现之间的区别

VIVO

JS对象的定义有几种方式？

经典浏览器输入URL后发生了什么，浏览器怎么渲染页面

MVC和MVVM的区别与联系？

同花顺

Vue和原生js有什么区别

Vue生命周期

用了Vuex还要用使用全局事件总线？

怎么用Vuex来实现组件跨级通信？

Vuex中mapState、mapGetters、mapMutations、mapActions

Vue父子组件通信

Vue.set()和vm.$set()

V-if和V-show的区别

如果用户没有进行登录，vue-router如何进行拦截？

vue-router中push和replace的区别？

axios发起请求时，如果登录失效怎么办

axios发送get和post请求有什么区别

三栏布局以及怎么实现三列等高

详细说一下http状态码

JSON字符串转换成JSON对象

跨域解决办法

原生JS对DOM操作的方法

Vue父子组件生命周期执行顺序

斐波那契数列(递归和非递归解法)

蚂蚁金融

笔试题目见链接

<https://www.nowcoder.com/discuss/860063?source_id=discuss_experience_nctrack&channel=-1>

如何实现节流？Js写代码

虚拟DOM和DIFF[算法](https://www.nowcoder.com/jump/super-jump/word?word=%E7%AE%97%E6%B3%95" \t "https://www.nowcoder.com/discuss/_blank)

有没有用过iframe？

有做过h5的手机端的页面吗

在手机端一般用什么单位？我就说比较喜欢用rem、em、rpx这些，面试官接着又让我解释一下rem

Promise用的比较多哈，你能解释一下Promise是什么样的一个概念吗？

我解释了一下Promise的主要用途和用法，也同时讲了一下async/await的概念，以及它们和Promise之间的关系。

在以往的项目里有遇到什么困难的点吗？

亿联网络

div水平垂直居中布局（说了绝对定位、flex、网格），

事件循环机制（说了为什么需要，怎么样），

项目中遇到的困难，你的贡献，

http状态码（4xx、5xx），

跨域、同源机制

如何加快页面渲染速度（懒加载、CDN、压缩、雪碧图，treeshrinking），

ES6新增的东西（let、const、解构赋值、箭头函数，块级作用域），

websocket协议（原生与sock.io、wss），

express和koa的最主要的区别（koa中间件洋葱模型），

vue双向绑定响应式原理（语法糖、数据劫持订阅发布），

webpack（问用过什么插件，loader），

vue2、3区别，为什么要用proxy替换原响应式方案。（v2方案缺陷，v3亮点），

vue数据传递的方案（注入，store，props，事件总线）

网络七层模型及各层应用，

前端性能优化，

CDN原理，

项目部署相关，

HTTP HTTPS区别，

知乎前端

css选择器权重值和优先级

css怎么画一个大小为父元素面积一半的正方形

flex实现8个元素分两行排列 每行4个平均分布

css样式隔离

promise有几个状态

promise.catch后 后面的.then还会执行吗

浏览器同源策略

跨域 如何解决

给一个dom同时绑定两个点击事件，一个用捕获，一个用冒泡，说下会执行几次事件，然后会先执行冒泡还是捕获

解 addEventListener绑定几次就执行几次 先捕获，后冒泡

http状态码   
vue组件通信   
es6有哪些新特性

箭头函数和普通函数的区别

同源策略 跨域 解决

vue路由history和hash两种模式的区别

vue响应式原理

webpack的loader和plugin区别

手写防抖

手写斐波那锲数列

手写翻转[二叉树](/jump/super-jump/word?word=%E4%BA%8C%E5%8F%89%E6%A0%91" \t "_blank)

手写发布订阅模式

小米前端

// 第一题

function sayHi() {

console.log(name); // undefined

console.log(age); // ReferenceError: Cannot access 'age' before initialization

var name = 'Lyric';

let age = 19;

}

sayHi();

闭包

作用域

宏任务、微任务

变量声明提升

事件循环

作者：TwinkleM

链接：https://www.nowcoder.com/discuss/854687?source\_id=discuss\_experience\_nctrack&channel=-1

来源：牛客网

讲了讲有哪些宏任务和微任务，以及它们的循环方式，之后思考对于宏任务和微任务队列的实现方式（给了几个思路，1. 二维数组 2. 哈希散列 3. 数组链表）

垂直水平居中方式（4种 + 了解的两种，后面两种没说实现思路，只说了有这么个方式 grid 和table）要求手写，但是牛客网面试房间 HTML 标签也不能直接 tab 键补全，导致我写的很慢，面试官就说直接讲思路就好了

Set、Map、WeakMap、WeakSet 区别（Map 和 Object 的效率问题，各自的 API，WeakMap 和 WeakSet 可迭代吗，JS 垃圾回收机制对 WeakMap 和 WeakSet 的影响）

本地存储了解哪些？Cookie、localStorage、sessionStorage 区别？（Cookie 详细介绍，包括 HTTP 无状态、domain、path、secure、httpOnly、SameSite）

Vue 组件间通信方式（9种，1. props父传子 2. emit 子传父 3. bus公共事件总线 4. Vuex 5. parent/children 6. Storage 7. provide/inject 8. attrs 9. ref 和 refs）

Vue 中 key 的作用，为什么有高效性？（就地复用、Diff 算法）

Webpack相关，Proxy 代理跨域的实现原理（本地 -> 代理服务器 -> 跨域接口），并要求用 node.js 手写出来一个简单的例子【不会！(理直气壮)】

响应式布局，媒体查询 + 百分比布局 + rem，以及 rem 和 em 的区别（em 面试官说我搞错了，下去要学）

单行文本省略，多行文本省略分情况实现（按行数或高度两种情况，面试官提醒注意分情况！）

判断数组的方式（五种，1. typeof 2. constructor 3. instanceof 4. Object.prototype.toString.call 5. Array.isArray），然后说说 3. 4. 5. 方式的区别，顺便手写了 instanceof [ 省的讲一遍原型原型链了——从构造函数到寄生式继承 ]

ES6 和 ES7 了解吗，我没先说 ES6，先讲了下 ES7 的 async 和 await，面试官就顺着这个开始问 setTimeout、Promise、async 和 await，然后实现原理 generator

· 防抖节流，原理介绍+应用场景，手写防抖和节流

· 介绍一下原型链，从构造函数到原型，再到原型链，盗用构造函数，原型式继承，寄生式继承，寄生式继承如何弥补了普通原型链的缺点。手写了原型式继承，以及手写寄生式继承如何连接父子构造函数。

· 做了一道事件循环代码顺序题目，关于Promise，setTimeout等，和平时见到的不太一样，有点绕，但是应该也不算难

· 问了 Vue 响应式原理，先手写了观察者模式（网上说是发布订阅模式，我觉得还是有区别的，发布订阅模式是 Vue 的 emit和 $on 的实现原理），之后讲 Vue 如何递归去把数据添加到响应式系统 【Object.defineProperty】，然后讲 Dep 和 Watcher 类，Vue 在 Compile 阶段如何识别 data 数据，实例化 Watcher 的过程【Dep.target】

· 问了问了不了解 computed 源码，不懂，面试官耐心的给我讲了讲

· HTTPS 和 HTTP 区别，就是讲公开密钥加密还有共享密钥加密，权威机构认证证书那些东西

· 浏览器缓存机制，强缓存、协商缓存等，还有个启发式缓存机制

· Vue 组件通信方式（9种）

腾讯CSIG

替换url中key的值

// 输入

url = 'https://www.nowcoder.com/discuss/v2/post?type=0&key=test'

key = 'abcd'

// 输出

res = 'https://www.nowcoder.com/discuss/v2/post?type=0&key=abcd'

写代码用到了for in，问与for of的区别，答到了原型链上，顺着就让手写了个es5的类的继承；又问prototype.constructor的作用，不写会发生什么

二叉树最近公共祖先https://leetcode-cn.com/problems/lowest-common-ancestor-of-a-binary-tree/

用递归写的，面试官问不用递归怎么做

没想出来，讨论良久，提到存储父节点用到map，又问了问es6

前端设计模式？（说了解发布订阅模式）手写一下

AfterShip前端

说一下哈希表的哈希算法怎么实现

进程间的通讯有哪些方式

数据库的三大范式

typescript中的泛型

缓存有了解吗

讲了内存缓存、硬盘缓存，和强缓存、协商缓存（Cache-control）

深信服日常实习一面(已OC)

get和post请求区别

Vue中key值为什么不能用索引(diff算法)

6.JS异步发展历史

7.Async和Await原理

8.手写Promise.all

webpack工作原理

1. webpack的loader和plugin区别

蚂蚁集团

Webpack 和 vite 有什么区别？

平时通过什么方式来学习前端？

http1.1 和 http2.0 的区别

垂直水平居中

为什么会选择前端

百度作者：一点点很美味

链接：https://www.nowcoder.com/discuss/827565?source\_id=discuss\_experience\_nctrack&channel=-1

来源：牛客网

为什么想做前端？

一般遇到问题如何解决？

简单介绍下 WebSocket？

做一个游戏的时候，选择一个网络协议会选择什么？

OSI 七层模型 应用层、表示层、会话层、传输层、网络层、数据链路层、物理层

应用层都有哪些？

TCP 流量控制、拥塞控制？

TCP 头部大小是固定的吗？

WebSocket 如何连接？服务端给客户端发送的http状态码是什么？

ES6 用过哪些？

如何使用 Reflect，它提供了什么？

箭头函数和普通函数的区别？

箭头函数有 super 吗？

什么是类数组？

箭头函数可以用call或者apply改变this指向吗？

比较擅长什么？

什么是MVVM？

MVVM 的特性？

Vue什么特性表示出MVVM特性？

手写 promise.all()

手写 Array.prototype.reduce()

通常在浏览页面时候，图片经过压缩还是有位置的，在图片加载完成之前会有占位符，让他显示出一个缩略图（轮廓），如何做？ 缩略图如何获取？

浏览器多线程多线程？

进程和线程的区别？

进程间是否共享内存？

Worker线程之间如何通信？

滴滴

作者：一点点很美味

链接：https://www.nowcoder.com/discuss/827564?source\_id=discuss\_experience\_nctrack&channel=-1

来源：牛客网

JS 基本数据类型

Symbol 和 Bigint 简单介绍

undefined 和 null 的区别

typeof(null) 为什么返回的是 'object'

== 和 === 的区别？

判断数据类型的方法

Instanceof 的原理

手写 call()

typeof(NaN) 返回什么

Class 和 new

手写 new

对 Map 和 Set 的理解

Promise 的理解

Promise 新增的一个方法

Object 和 Map 相互转换

Map.entries是什么样的？

事件循环机制

const 定义的值一定是不能改变的吗？

基本数据类型和引用数据类型的根本区别？

手写深拷贝

有了 promise 为什么还需要 async/await

Http 常见状态码

· cookie session token 的认证机制

小米

介绍项目

对 post 和 get 请求的理解

post 和 get 数据包发送方式

对 CSS 样式的了解（选择器权重）

描述 块级元素 和 行内元素 的区别

隐藏一个元素（visibility: hidden opacity: 0 display: none z-index)

一个按钮，一直固定在在窗口的右边（浮动，flex，定位position， margin）

对闭包的了解

描述用过的 ES6 的语法

用过ES6哪些数组的方法，简述他们的特性

Map 和 Set （说到了 WeakMap 和 WeakSet)

对 Vue3 的理解

Vue3 双向数据绑定的原理

在项目当中，用到的 Vue 的周边

回答以下代码输出结果

1

2

3

4

5

6

var a = 1;

function f() {

console.log(a); // undefined

var a = 2;

}

f()

1

2

3

4

5

6

7

8

9

10

11

12

// 说出代码输出并改正错误

function A(a) {

this.x = a;

var get = function() {

return this.x;

}

this.print = function() {

console.log(get()); // undefined

}

}

a = new A(1);

a.print();

算法题

1

统计字符串 `aaaabbbccccddfgh`中字母的个数

复制代码

1

2

3

4

5

6

7

8

9

10

11

12

13

14

// 根据 id 比较数组的差异，返回差异项，要求找出增加项，删除项，变更项

// id 不一定是有序的

const arr1 = [

{id: 1, value: 100},

{id: 2, value: 400},

{id: 3, value: 200},

]

const arr2 = [

{id: 1, value: 100},

{id: 2, value: 200},

{id: 5, value: 200}

{id: 3, value: 300},

{id: 4, value: 700}

]

作者：一点点很美味

链接：https://www.nowcoder.com/discuss/827562?source\_id=discuss\_experience\_nctrack&channel=-1

来源：牛客网

事件循环机制，浏览器渲染机制

浏览器架构，都有什么进程和线程

实现 instanceof 的 polyfill

节流

this指向使用场景

怎么理解JS 静态作用域

如何使其拥有动态作用域的特性

作用域链

输出题：

1

2

3

4

5

6

7

8

9

10

11

12

13

Object.prototype.a = 10;

var s = Symbol();

var obj = {

[s]: 20,

b: 30

}

Object.defineProperty(obj, 'c', {

enumerable: true，

value: 40

})

for(let var in obj) {

console.log(val)

}

作者：一点点很美味

链接：https://www.nowcoder.com/discuss/827562?source\_id=discuss\_experience\_nctrack&channel=-1

来源：牛客网

事件循环机制，浏览器渲染机制

浏览器架构，都有什么进程和线程

实现 instanceof 的 polyfill

节流

this指向使用场景

怎么理解JS 静态作用域

如何使其拥有动态作用域的特性

作用域链

输出题：

1

2

3

4

5

6

7

8

9

10

11

12

13

Object.prototype.a = 10;

var s = Symbol();

var obj = {

[s]: 20,

b: 30

}

Object.defineProperty(obj, 'c', {

enumerable: true，

value: 40

})

for(let var in obj) {

console.log(val)

}

Https 与 http 的区别，https 比 http 多了哪些？

非对称加密和对称加密

https是非对称还是对称

浏览器输入一个url,会发生什么？到页面的展示过程

lazyMan('name').eat('apple').sleep(1000).eat('orange')

// 打印 name，打印 apple，延迟 1s 打印 orange

// 写一个自己熟练的排序算法

快手前端实习

<https://www.nowcoder.com/discuss/821047>

阿里

<https://www.nowcoder.com/discuss/819176>

Html

<https://www.nowcoder.com/discuss/809654>

CSS

<https://www.nowcoder.com/discuss/808087>

<https://www.nowcoder.com/discuss/806261>

深信服

作者：HSHhhh  
链接：[https://www.nowcoder.com/discuss/802842?source\_id=discuss\_experience\_nctrack&channel=-1](https://www.nowcoder.com/discuss/802842?source_id=discuss_experience_nctrack&channel=-1" \t "_blank)  
来源：牛客网

CSS选择器

CSS position

absolute relative的区别是什么 在文档流中

var let const的区别

js类型判断

怎么跨域

前端项目工程化

webpack

JS看代码写输出

去重后[排序](/jump/super-jump/word?word=%E6%8E%92%E5%BA%8F" \t "_blank) [1,1,2,'1',55,-1,0]

(当时写了个双重for循环...）

说一下数组和[链表](https://www.nowcoder.com/jump/super-jump/word?word=%E9%93%BE%E8%A1%A8" \t "https://www.nowcoder.com/discuss/_blank) 两者的优缺点

单向[链表](https://www.nowcoder.com/jump/super-jump/word?word=%E9%93%BE%E8%A1%A8" \t "https://www.nowcoder.com/discuss/_blank)反转 (没想到出这么简单，迭代秒了)

function reverse(p) {

let pre = null

let cur = p

while(cur) {

const next = cur.next

cur.next = pre

pre = cur

cur = next

}

return pre

}

有写过正则表达式吗

有了解过前端工程化吗

一点小建议

明确就业方向。最重要的是选择适合自己的方向，而不是看其他人选择的方向，其他人的选择只能是一种参考

明确希望去的平台。互联网大厂？传统行业龙头？国企？考公？等等

明确工作地点。北山广深？二三线城市还是家乡？

尽可能去实习。这一点主要看个人时间

一定要参加提前批。实践检验真理，早点发现自己有多菜，以后的offer就不会是烂白菜

多刷题，多看书，多看面经，多总结回顾。不要浮于表面还安然自若

美团

两个长度不等的有序数组合并成一个有序数组

前端路由

手撕代码, isEqual(a, b) 实现深度比较

手撕代码: 【VDOM 模拟事件捕获机制】一个用数组存储的树, 查找树中的元素并输出查找路径

<https://www.nowcoder.com/discuss/783864>

深信服

30道不定项（60分）：

3道编程（40分）：

滴滴

<https://www.nowcoder.com/discuss/781312>

还有很多题目可以看看

为什么常量比较是false，实验发现不管赋值为多少，比较输出总是false

const o1 = {a:'a'};

const o2 = {a:'a'};

console.log(typeof(o1));

console.log(typeof(o2));

console.log(o1 == o2);

console.log(o1 === o2);

const o1 = 1;

const o2 = 2;

console.log(typeof(o1));

console.log(typeof(o2));

console.log(o1 == o2);

console.log(o1 === o2);

JS万字总结

<https://zhuanlan.zhihu.com/p/428670721>

前端工程师精选面经合集

<https://www.nowcoder.com/discuss/experience?tagId=644>

