## javascript篇

### 一、数据类型

**JavaScript 有几种类型**

* 基本数据类型：number、string、boolean、undefined、null、symbol(es6的新数据类型)
* 引用数据类型：object、array、function(统称为object)

**数据类型检测**

typeof对于基本数据类型来说，除了 null 都可以显示正确的类型，typeof 对于对象来说，除了函数都会显示 object

**typeof**5*// 'number'*

**typeof**'5'*// 'string'*

**typeofundefined***// 'undefined'*

**typeoffalse***// 'boolean'*

**typeof**Symbol()*// 'symbol'*

console.log(**typeofnull**)*//object*

console.log(**typeofNaN**)*//number*

**typeof**[]*// 'object'*

**typeof**{}*// 'object'*

**typeof**console.log*// 'function'*

instanceof通过原型链来判断数据类型的

p1**=new**Person()

p1**instanceof**Person*// true*

Object.prototype.toString.call()可以检测所有的数据类型，算是一个比较完美的方法了。

**var**obj**=**{}

**Var**arr**=**[]

console.log(Object.prototype.toString.call(obj))*//[objectObject]*console.log(Object.prototype.toString.call(arr))*//[object Array]*

**深浅拷贝**

浅拷贝

Object.assign()*//es6的方法*

Object.assign会合并对象生成一个新对象。如果对象的属性是普通类型改变之后新对象不会改变，如果是引用类型改变后新对象也会改变，所以Object.assign实际上还是浅拷贝。

**var**obj**=**{aa**:**1,b**:**{item**:**'45'}};

**var**newObj**=**Object.assign({},obj);

obj.aa**=**2;

obj.b.item**=**'kk';

console.log(newObj.aa);*//1*

console.log(newObj.b.item);*//kk*

深拷贝

JSON.parse(JSON.stringify(obj))

利用JSON.stringify(obj)将对象先转为json字符串，再JSON.parse(）转回为json对象可以实现深拷贝，这也是比较常用的一种方法。

**用js实现一个深拷贝**

其实深拷贝可以拆分成 2 步，浅拷贝 + 递归，浅拷贝时判断属性值是否是对象，如果是对象就进行递归操作，两个一结合就实现了深拷贝。

**Function** cloneDeep(source){

**if**(**!**isObject(source))**return**source;*// 非对象返回自身***var**target**=**Array.isArray(source)**?**[]**:**{};

**for**(**var**key**in**source){

**if**(source.hasOwnProperty(i)){

**if**(isObject(source[key])){

target[key]**=**cloneDeep(source[key]);

*// 注意这里*

} **else**{

target[key]**=**source[key];

}

}}

**Return** target;

}

**function**isObject(obj){

**return typeof**obj**===**'object'**&&** obj**!=null**;

}

### 二、作用域

**变量声明提升**

* 在 JavaScript 中，函数声明（function aa(){}）与变量声明（var）经常被 JavaScript 引擎隐式地提升到当前作用域的顶部。
* 函数声明的优先级高于变量，如果变量名跟函数名相同且未赋值，则函数声明会覆盖变量声明
* 声明语句中的赋值部分并不会被提升，只有变量的名称被提升

**作用域链**

因为函数的嵌套形成作用域的层级关系。当函数执行时，从当前作用域开始搜，没有找到的变量，会向上层作用域查找，直至全局函数，这就是作用域链。

* 在 JavaScript 中，作用域为 function(){}内的区域，称为函数作用域。
* 全局函数无法查看局部函数的内部细节，但局部函数可以查看其上层的函数细节，直至全局细节

**闭包**

闭包的原理就是作用域链，比函数F内部有一个函数G，函数 G可以访问到函数F中的变量，那么函数G就是闭包。

**Function** F(){

**let**a**=**1

window.G**=function**(){

console.log(a)

}}F()*// 1*

## 三、原型和继承

**js 创建对象的几种方式**

对象字面量的方

**var**obj**=**{};

new一个构造函数

**function**Pel(){}

**var**p**=new**Pel();

p.name**=**"hu";

p.age**=**"25";

p.address**=function**(){

}

new一个内置对

**var**obj**=new**Object();

Object.create（）创建对象

**var**test**=**Object.create({x**:**1});

给大家留一道思考题，new Object() 、 Object.create(）、{}，这三种方式创建对象有什么区别。

**JS 如何实现一个类**

构造函数法

缺点：用到了 this 和 prototype，编写复杂，可读性差

**function**P(name,age){

**this**.name**=**name;

**this**.age**=**age;

}

P.prototype.sal**=function**(){

}

**var**pel**=new**P("jj",1);

pel.sell()

ES6 语法糖 class

class Point {

constructor(x, y) {

this.x = x;

this.y = y;

}

toString() {

return '(' + this.x + ', ' + this.y + ')';

}

}

var point = new Point(2, 3);

**原型链**

一句话解析什么是原型链

遍历一个实列的属性时，先遍历实列对象上的属性，再遍历它的原型对象，一直遍历到Object

任何一个类（函数）都有原型对象，原型对象至少有两个属性（constructor,**proto**）。constructor指向函数本身，**proto**指向父类原型对象。

函数上有一个prototype属性，指向原型对象，通过它可以访问原型对象

函数的实列可以直接访问原型对象(因为实列上有**proto**指向构造函数的原型对象)

**function**Dog(){}*//类* **var** obj**=new** Dog();*//实列*obj.name**=**'沪江'; Dog.prototype.name**=**"旺财";Dog.prototype.eat**=function**(){

console.log(**this**.name);};

console.log(Dog.prototype.name);*//旺财*console.log(obj.prototype);*//undefined,prototype是类上才有的，实列上没有*obj.eat();*//沪江（先遍历实列对象上的属性，再遍历它的原型对象）*

### 继承

**Js如何实现继承？**

构造函数绑定：使用 call 或 apply 方法，将父对象的构造函数绑定在子对象上

**function**Cat(name,color){

　Animal.apply(**this**,arguments);

**this**.name**=**name;

**this**.color**=**color;}

实例继承：将子对象的 prototype 指向父对象的一个实例

Cat.prototype**=new**Animal();Cat.prototype.constructor**=**Cat;

拷贝继承：如果把父对象的所有属性和方法，拷贝进子对象

**function**extend(Child,Parent){

**var**p**=**Parent.prototype;

**var**c**=**Child.prototype;

**for**(**var**i**in**p){

　　　 c[i]**=**p[i];

　　　}

　　　c.uber**=**p;}

原型继承：将子对象的 prototype 指向父对象的 prototype

**function**extend(Child,Parent){

**var**F**=function**(){};

　F.prototype**=**Parent.prototype;

　Child.prototype**=new**F();

　Child.prototype.constructor**=**Child;

　Child.uber**=**Parent.prototype;}

ES6 语法糖 extends继承

**class**ColorPoint**extends**Point{

constructor(x,y,color){

**super**(x,y);*// 调用父类的constructor(x, y)***this**.color**=**color;

}

toString(){

**returnthis**.color**+**' '**+super**.toString();*// 调用父类的toString()*}}

### ❤️篇幅有限，更详细的内容可以查看，下面这篇文章❤️

[由浅入深：彻底弄懂JS原型与继承​mp.weixin.qq.com](https://link.zhihu.com/?target=https://mp.weixin.qq.com/s?__biz=MzI5MjU0Mjk5MA==&mid=2247483768&idx=1&sn=4579b7bd22420d18009505a5a7454a9d&chksm=ec7e8e19db09070f8d529cb6a3f9bb425c20ac3bd4d936f4eb9e918facbe3d7e6eecdd719c4c&token=816776520&lang=zh_CN%23rd)

## 四、new和this

**new 操作符具体干了什么？**

当我们new一个数据的时候，new操作符到底做了什么？

* 首先是创建实例对象{}
* this 变量引用该对象，同时还继承了构造函数的原型
* 其次属性和方法被加入到 this 引用的对象中
* 并且新创建的对象由 this 所引用，最后隐式的返回 this

**new的模拟实现**

**function**objectFactory(){

**var**obj**=new**Object(),*//从Object.prototype上克隆一个对象*

Constructor**=**[].shift.call(arguments);*//取得外部传入的构造器*

**var**F**=function**(){};

F.prototype**=**Constructor.prototype;

obj**=new**F();*//指向正确的原型*

**var**ret**=**Constructor.apply(obj,arguments);*//借用外部传入的构造器给obj设置属性*

**returntypeof**ret**===**'object'**?**ret**:**obj;*//确保构造器总是返回一个对象*};

**this 对象的理解**

普通函数

* this 总是指向函数的直接调用者
* 如果有 new 关键字，this 指向 new 出来的实例对象
* 在事件中，this 指向触发这个事件的对象
* IE 下 attachEvent 中的 this 总是指向全局对象 Window
* 箭头函数中，函数体内的this对象，就是定义时所在作用域的对象，而不是使用时所在的作用域的对象。

**function**foo(){

console.log(**this**.a)}**var**a**=**1foo()*//1* ​**const**obj**=**{

a**:**2,

foo**:**foo}obj.foo()*//2*​**const**c**=new**foo()*//undefined*

* 对于直接调用 foo 来说，不管 foo 函数被放在了什么地方，this 一定是window
* 对于 obj.foo() 来说，我们只需要记住，谁调用了函数，谁就是 this，所以在这个场景下 foo 函数中的 this 就是 obj 对象
* 对于 new 的方式来说，this 被永远绑定在了 new出来的对象上，不会被任何方式改变 this

说完了以上几种情况，其实很多代码中的 this 应该就没什么问题了，下面让我们看看箭头函数中的 this

**function**a(){

**return**()=>{

**return**()=>{

console.log(**this**)

}

}}a()()()*//Window*

* 首先箭头函数其实是没有 this 的，箭头函数中的 this 只取决包裹箭头函数的第一个普通函数的 this。在这个例子中，因为包裹箭头函数的第一个普通函数是 a，所以此时的 this 是 window。另外对箭头函数使用 bind这类函数是无效的。

### 五、apply、call、bind

call、apply和bind是Function对象自带的三个方法，都是为了改变函数体内部 this 的指向。  
apply 、 call 、bind 三者第一个参数都是 this 要指向的对象，也就是想指定的上下文；  
apply 、 call 、bind 三者都可以利用后续参数传参；  
bind 是返回对应 **函数**，便于稍后调用；apply 、call 则是立即调用 。

**function**fruits(){}

fruits.prototype**=**{

color**:**'red',

say**:function**(){

console.log(**this**.color);

}};

**var**apple**=new**fruits();

apple.say();*// red, 此时方法里面的this 指的是fruits*banana**=**{color**:**'yellow'};apple.say.call(banana);*//yellow,此时的this的指向已经通过call（）方法改变了，指向的是banana，this.color就是banana.color='yellow';*apple.say.apply(banana);*//yellow,同理，此时的this的指向已经通过apply（）方法改变了，指向的是banana，this.color就是banana.color ='yellow';*apple.say.apply(**null**);*//undefined, null是window下的，此时，this 就指向了window ，但是window下并没有clolr这个属性，因此this.clolr就是window.color=undefined;*

call 传入参数列表  
apply 传入数组

**var**array1**=**[12,'foo'];**var**array2**=**['Doe',100];

Array.prototype.push.call(array1,'Doe',100)Array.prototype.push.apply(array1,array2)

bind() 方法会创建一个 新函数，当调用这个新函数时，新函数会以创建它时传入 bind() 方法的第一个参数 作为 this，传入 bind() 方法的 第二个以及以后的参数加上绑定函数运行时本身的参数按照顺序作为原函数的参数来调用原函数。

**var**bar**=function**(){

console.log(**this**.x);};**var**foo**=**{

x**:**3};bar();*// undefined***var**func**=**bar.bind(foo);

func();*// 3*

## 六、数据处理

**数组去重**

**var**arr**=**['12','32','89','12','12','78','12','32'];

*// 最简单数组去重法***function**unique1(array){

**var**n**=**[];*//一个新的临时数组***for**(**var**i**=**0;i**<**array.length;i**++**){*//遍历当前数组***if**(n.indexOf(array[i])**==-**1)

n.push(array[i]);

}

**return**n;

}

arr**=**unique1(arr);

**排序**

*/\*\* \* 按 sort 及 id 排序 \* @param {Object} a \* @param {Object} b \*/*

**function**sortFun(a,b){

**return**a.sort**-**b.sort**==**0**?**a.id**-**b.id**:**a.sort**-**b.sort

};

arr.sort(sortFun)*//从小到大排序*

**递归求和**

**function**add(num1,num2){

**var**num**=**num1**+**num2;

**if**(num2**+**1**>**100){

**return**num;

}**else**{

**return**add(num,num2**+**1)

}

}

**var**sum**=**add(1,2)

**计算数组各项的重复次数**

**var**arr**=**['胡将','胡将','hujiang','胡将','胡江','hujiang'];**var**obj**=**{};arr.sort();*//先排序***for**(**var**i**=**0;i**<**arr.length;){

**var**con**=**0;

**for**(**var**j**=**i;j**<**arr.length;j**++**){

**if**(arr[i]**===**arr[j]){

con**++**

}

}

obj[arr[i]]**=**con;

i**=**i**+**con;*//跳过重复的值*}console.log(obj);*//{ hujiang: 2, '胡将': 3, '胡江': 1 }*

## 七、Event Loop

**宏任务/微任务**

除了广义的同步任务和异步任务，我们对任务有更精细的定义：

1. macro-task(宏任务)：当前调用栈中执行的任务称为宏任务。包括：script全部代码、setTimeout、setInterval、setImmediate（浏览器暂时不支持，只有IE10支持，具体可见[MDN](https://link.zhihu.com/?target=https://developer.mozilla.org/zh-CN/docs/Web/API/Window/setImmediate)）、I/O、UI Rendering。
2. .micro-task(微任务)： 当前（此次事件循环中）宏任务执行完，在下一个宏任务开始之前需要执行的任务为微任务。包括：Process.nextTick（Node独有）、Promise、Object.observe(废弃)、MutationObserver
3. 不同类型的任务会进入对应的Event Queue，宏任务中的事件放在callback queue中，由事件触发线程维护；微任务的事件放在微任务队列中，由js引擎线程维护。

**一句话解析下什么是event loop**

* 主线程运行的时候会生成堆（heap）和栈（stack）；
* js 从上到下解析方法，将其中的同步任务按照执行顺序排列到执行栈中；
* 当程序调用外部的 API 时（比如 ajax、setTimeout 等），会将此类异步任务挂起，继续执行执行栈中的任务。等异步任务返回结果后，再按照顺序排列到事件队列中；
* 主线程先将执行栈中的同步任务清空，然后检查事件队列中是否有任务，如果有，就将第一个事件对应的回调推到执行栈中执行，若在执行过程中遇到异步任务，则继续将这个异步任务排列到事件队列中。
* 主线程每次将执行栈清空后，就去事件队列中检查是否有任务，如果有，就每次取出一个推到执行栈中执行，这个循环往复的过程被称为“Event Loop 事件循环”

### 八、浏览器页面渲染过程

浏览器渲染页面的一般过程：

1.浏览器解析html源码，然后创建一个 DOM树。并行请求 css/image/js在DOM树中，每一个HTML标签都有一个对应的节点，并且每一个文本也都会有一个对应的文本节点。DOM树的根节点就是 documentElement，对应的是html标签。

2.浏览器解析CSS代码，计算出最终的样式数据。构建CSSOM树。对CSS代码中非法的语法它会直接忽略掉。解析CSS的时候会按照如下顺序来定义优先级：浏览器默认设置 < 用户设置 < 外链样式 < 内联样式 < html中的style。

3.DOM Tree + CSSOM --> 渲染树（rendering tree）。渲染树和DOM树有点像，但是是有区别的。

DOM树完全和html标签一一对应，但是渲染树会忽略掉不需要渲染的元素，比如head、display:none的元素等。而且一大段文本中的每一个行在渲染树中都是独立的一个节点。渲染树中的每一个节点都存储有对应的css属性。

4.一旦渲染树创建好了，浏览器就可以根据渲染树直接把页面绘制到屏幕上。

以上四个步骤并不是一次性顺序完成的。如果DOM或者CSSOM被修改，以上过程会被重复执行。实际上，CSS和JavaScript往往会多次修改DOM或者CSSOM。

## 九、浏览器缓存

这块内容平时面试中问到比较多的就是**强缓存和协商缓存**了，下面这篇文章写的很好，我就不班门弄斧了，大家直接看这篇文章吧

## css篇

## 一、盒模型

盒模型的组成，由里向外content,padding,border,margin.

**在IE盒子模型中，width表示content+padding+border这三个部分的宽度**

**在标准的盒子模型中，width指content部分的宽度**

box-sizing的使用

box-sizing**:**content-box是W3C盒子模型

box-sizing**:**border-box是IE盒子模型

box-sizing的默认属性是content-box

## 二、居中

**水平居中**

* 行内元素: text-align: center
* 块级元素: margin: 0 auto
* position:absolute +left:50%+ transform:translateX(-50%)
* display:flex + justify-content: center

**垂直居中**

* 设置line-height 等于height
* position：absolute +top:50%+ transform:translateY(-50%)
* display:flex + align-items: center
* display:table+display:table-cell + vertical-align: middle;

//不知道宽高

width**:**78px**;**

height**:**78px**;**

position**:**absolute**;**

left**:**50**%;**

top**:**50**%;**

transform**:**translateX**(**-50**%)**translateY**(**-50**%);**

//知道宽高

height**:**100px**;**

width**:**100px**;**

position**:**absolute**;**

left:50**%;**

top**:**50**%;**

margin-left**:**-50px**;**

margin-top**:**-50px**;**

display:flex**;**

justify-content**:**center**;**

align-content**:**center**;**

写在最后：

面试是一个双向选择的过程，我们面试的时候不要把自己当成一个被挑选的弱势者。前端知识杂却多，有时面试结果不理想，不要怀疑自己。只是在某个方面自己还有欠缺，之后补上就可以了。我们不是在祈求别人给我们一份工作，我们只是在找寻一个适合我们的公司。不妄自菲薄更不骄傲自负，整理好心态，你一定能找到适合自己的公司！！！加油啊

前端知识点较杂且多，在学习过程中需要不断的总结。。。