## 前端面试题及答案（JS）

**1、如何将浮点数点左边的数每三位添加一个逗号，如12000000.11转化为『12,000,000.11』?**

function commafy(num){

return num&&num

.toString()

.replace(/(\d)(?=(\d{3})+\.)/g,function($1,$2){

return $1 + ',';

})

}

1. **null和undefined的区别？**

null 表示一个对象是“没有值”的值，也就是值为“空”；

undefined 表示一个变量声明了没有初始化(赋值)；

undefined不是一个有效的JSON，而null是；

undefined的类型(typeof)是undefined；

null的类型(typeof)是object；

注意：

在验证null时，一定要使用　=== ，因为 == 无法分别 null 和　undefined

null == undefined // true

null === undefined // false

1. **typeof 和instanceof的区别？**

如果想要判断一个变量是否存在，可以使用typeof：不能使用if(a) 若a未声明，报错)

**typeof**对于基本类型变量和函数、对象会返回正确的值，会返回变量的基本类型，

但是不适合用来判断数组，数组、对象和null都会返回object

typeof [1,2,3] //Object

**instanceof**返回的是布尔值，只能用来判断**数组、对象和函数、用内置对象String创建的字符串**，不能判断数字和字符串

null instanceof Object//false

1. **哪些操作会造成内存泄露？**

* setTimeout 的第一个参数使用字符串而非函数的话，会引发内存泄漏。
* 循环（在两个对象彼此引用且彼此保留时，就会产生一个循环）
* 闭包

1. **看看以下的代码会输出什么？**

var undefined;

undefined == null; // true

0 == false; // true

0 == ''; // true

NaN == NaN; // false

[] == false; // true

[] == ![]; // true

一个是number一个是string时，会尝试将string转换为number

尝试将boolean转换为number，0或1

尝试将Object转换成number或string，取决于另外一个对比量的类型

所以，对于0、空字符串的判断，建议使用 “===” 。“===”会先判断两边的值类型，类型不匹配时为false。

**6、写一个函数，去除字符串首尾空格？**

String.prototype.test=function(){

return this.replace(/^\s+/, "").replace(/\s+$/,"")

}

**7、阅读以下代码，请分析出结果：**

var arr = new Array(1 ,3 ,5);

arr[4]='z';

arr2 = arr.reverse();

arr3 = arr.concat(arr2);

alert(arr3);

弹出提示对话框：z,,5,3,1,z,,5,3,1

原因：reserves会改变原数组

1. **iframe的优缺点?**

优点：

1. 解决加载缓慢的第三方内容如图标和广告等的加载问题

2. Security sandbox(安全沙箱)

3. 并行加载脚本

缺点：

1. iframe会阻塞主页面的Onload事件

2. 即时内容为空，加载也需要时间

3. 没有语意

**9、判断一个字符串中出现次数最多的字符，统计这个次数**

方法一：

var str = 'asdfssaaasasasasaa';

var json = {};

for (var i = 0; i < str.length; i++) {

        if(!json[str.charAt(i)]){

                json[str.charAt(i)] = 1;

        }else{

                json[str.charAt(i)]++;

        }

};

var iMax = 0, iIndex = '';

for(var i in json){

    if(json[i]>iMax){

    iMax = json[i];

    iIndex = i;

    }

}

alert("出现次数最多的是:"+iIndex+"出现"+iMax+"次");

方法二：借助数组的reduce方法

var str="abcdcdabb",

arr=str.split("");

arr.reduce(function(prev,next){

prev[next]=(prev[next]+1)||1;

return prev;

},{});

1. **写一个获取非行间样式的函数**

(注意：currentStyle和getComputedStyle只能用于获取页面元素的样式，不能用来设置相关值。如果要设置相应值，应使用style。)

function getStyle(obj,attr,value)

{

if(!value)

{

if(obj.currentStyle)//IE

{

return document.currentStyle(obj).getPropertyValue;

}

else{

window.getComputedStyle(dom,false).getPropertyValue.;//FF、opera、safari、chrome

}

}

else

{

obj.style[attr] = value;

}

}

1. **解释jsonp的原理，说说与ajax有什么不同？**

* 带src 属性的iframe script等标签都可以实现跨域
* ajax和jsonp其实本质上是不同的东西。ajax的核心是通过XmlHttpRequest获取非本页内容，而jsonp的核心则是动态添加<script>标签来调用服务器提供的js脚本。
* 其实ajax与jsonp的区别不在于是否跨域，ajax通过服务端代理一样可以实现跨域，jsonp本身也不排斥同域的数据的获取。

## 

## 12、去掉数组中重复的数字

方法一：

//思路：每遍历一次就和之前的所有做比较，不相等则放入新的数组中！

//这里用的原型 个人做法；

Array.prototype.unique = function(){

    var len = this.length,

       newArr = [],

       flag = 1;

    for(var i = 0; i < len; i++, flag = 1){

        for(var j = 0; j < i; j++){

            if(this[i] === this[j]){

                flag = 0;        //找到相同的数字后，不执行添加数据

            }

        }

        flag ? newArr.push(this[i]) : '';

    }

    return newArr;

}

方法二：

var arr=[1,2,3,3,4,4,5,5,6,1,9,3,25,4];

Array.prototype.unique2 = function()

{

var n = []; //一个新的临时数组

for(var i = 0; i < this.length; i++) //遍历当前数组

{

//如果当前数组的第i已经保存进了临时数组，那么跳过，

//否则把当前项push到临时数组里面

if (n.**indexOf**(this[i]) == -1) n.push(this[i]);

}

return n;

}

var newArr2=arr.unique2(arr);

alert(newArr2); //输出1,2,3,4,5,6,9,25

方法三：ES6

[...new Set(array)]

**13、加减运算**

alert('5'+3); //53 string

alert('5'+'3'); //53 string

alert('5'-3); //2 number

alert('5'-'3'); //2 number

**14、结果是什么？(代码运行有问题)**

function foo(){

    foo.a = function(){alert(1)};

    this.a = function(){alert(2)};

    a = function(){alert(3)};

    var a = function(){alert(4)};

};

foo.prototype.a = function(){alert(5)};

foo.a = function(){alert(6)};

foo.a(); //6

var obj = new foo();

obj.a(); //2

foo.a(); //1

**15、输出结果**

var a = 5;

function test(){

    a = 0;

    alert(a);

    alert(this.a); //没有定义 a这个属性

    var a;

    alert(a)

}

test(); // 0, 5, 0

new test();//0, undefined,0//由于类它自身没有属性a，所以是undefined

## 16、bind(), live(), delegate()的区别

* bind： 绑定事件，对新添加的事件不起作用，方法用于将一个处理程序附加到每个匹配元素的事件上并返回jQuery对象。
* live： 方法将一个事件处理程序附加到与当前选择器匹配的所有元素（包含现有的或将来添加的）的指定事件上并返回jQuery对象。
* delegate： 方法基于一组特定的根元素将处理程序附加到匹配选择器的所有元素（现有的或将来的）的一个或多个事件上。
* 无论使用bind、on、delegate、click(function())都是重复绑定，即绑定的同类型事件被放到一个事件队列中，依次执行，后绑定的事件不会替换之前绑定的，对于on使用off，delegate用undelegate，bind及click使用unbind来解除绑定，例如unbind(type)传递为事件类型，如果不传type则解出所有事件绑定；需要注意的是元素本身自带的事件无法unbind（如button1）

1. **以下代码中end字符串什么时候输出**

var t=true;

setTimeout(function(){

console.log(123);

t=false;

},1000);

while(t){}

console.log(‘end’);

永远不输出

1. **以下代码会输出什么？**

var User = {

count = 1，

getCount：function（）{

return this.count;

}

}

console.log(User.getCount());

var func = User.getCount;

console.log(func());

1 undefined（因为是window对象执行了func函数）;

**19、下列JavaScript代码执行后，iNum的值是**

var iNum = 0;

for(var i = 1; i< 10; i++){

if(i % 5 == 0){

continue;

}

iNum++;

}

答案：8 continue后iNum++就不执行了 共循环9次 有一次跳过

**20、以下代码输出结果？**

var obj = {proto: {a:1,b:2}};

function F(){};

F.prototype = obj.proto;//通过原型继承

var f = new F();

obj.proto.c = 3;

obj.proto = {a:-1, b:-2};//执行完这一步 指向关系结束

//alert(f.a);

alert(f.c);

delete F.prototype['a'];

alert(f.a);

alert(obj.proto.a);

输出 ：1 3 undefined -1

1. **以下代码输出？**

console.log(a);//是一个函数

var a = 3;

function a(){}//在加载的时候就已经初始化了

console.log(a);////3

**22、以下代码输出（NFE命名函数表达式）**

bar();//报错

var foo = function bar(name) {

console.log("hello"+name);

console.log(bar);//bar函数

};

alert(typeof bar);//undefined

foo("world");//"hello world"

console.log(bar);//undefined

console.log(foo.toString());//bar函数

bar();//报错

**函数表达式和函数声明的区别**

* 函数声明必须要有函数名或者标识符（可以在自身作用域内递归调用）
* 函数表达式可以省略标识符或函数名

FunctionDeclaration :

function Identifier ( FormalParameterList opt ){ FunctionBody }

FunctionExpression :

function Identifier opt ( FormalParameterList opt ){ FunctionBody }

* 若identifier存在怎样区分是函数表达式还是函数声明？ECMAScript ：根据上下文，如果function是赋值表达式的一部分，那么它就是函数表达式，若function被包含在一个函数体内或者本身是一个函数体，那么它就是函数声明。
* function foo(){} // declaration, since it's part of a <em>Program</em>
* var bar = function foo(){}; // expression, since it's part of an assignmentExpression
* new function bar(){}; // expression, since it's part of a NewExpression
* (function(){// expression

function bar(){} // declaration, since it's part of a FunctionBody

})();

* 使用上或者说行为上有什么差异呢？

**函数声明**：

* 不管出现在什么位置上，都会提升到作用域的最前面，并进行了解析，例如：

alert(fn);//输出hello world

function fn(){return “hello world”};

* 声明不能在条件语句下，只能在程序里或函数体里：

下面是错误的演示

if (true) {

function foo() {

return 'first';

}

}else {

function foo() {

return 'second';

}

}

正确的做法是使用函数表达式

var foo;

if (true) {

foo = function() {

return 'first';

};

}else {

foo = function() {

return 'second';

};}

* var bar=function foo(）{}因为是赋值语句，所以这是一个函数表达式，而这个函数的名字是foo,但是这个名字只能在这个新定义的自身的作用域内可被调用，（用来代替arguments.callee 递归调用），这样即使在外部定义同名函数，也不会被覆盖。

var f = function foo(){

return typeof foo; // "foo" is available in this inner scope

};

// `foo` is never visible "outside"

typeof foo; // "undefined"

f(); // "function"

**23、["1", "2", "3"].map(parseInt) 答案是多少？**

parseInt() 函数能解析一个字符串，并返回一个整数，需要两个参数 (val, radix)，

其中 radix 表示要解析的数字的基数。【该值介于 2 ~ 36 之间，并且字符串中的数字不能大于radix才能正确返回数字结果值】;

结果是 [1, NaN, NaN]

map方法在调用callback函数时,会给它传递三个参数:当前正在遍历的元素, 元素索引, 原数组本身.

index被当做是进制传入parseInt的第二个参数 ，对于1来说parseInt("1",0)当parseInt的第二个参数为0或没有时表示以十进制输出，所以输出1

对于2来说 parseInt("2“，1)；输出NaN 3同理

**24、javascript 代码中的"use strict";是什么意思 ? 使用它区别是什么？**

* use strict是一种ECMAscript 5 添加的（严格）运行模式,这种模式使得 Javascript 在更严格的条件下运行,使JS编码更加规范化的模式,消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为。
* 默认支持的糟糕特性都会被禁用，比如不能用with，也不能在意外的情况下给全局变量赋值;
* 全局变量的显示声明,函数必须声明在顶层，不允许在非函数代码块内声明函数,arguments.callee也不允许使用；
* 消除代码运行的一些不安全之处，保证代码运行的安全,限制函数中的arguments修改，严格模式下的eval函数的行为和非严格模式的也不相同;
* 提高编译器效率，增加运行速度；
* 严格模式下的一些限制：

严格模式下都会存在哪些问题？

1、不能用前缀0表示八进制

2、无法使用隐式声明的变量（不允许使用未声明的变量）

3、不允许在对象中一个属性多次定义

4、不允许形参参数名重复

5、不允许使用转义字符

6、不允许使用arguments.callee

7、函数内的 arguments,无法更改arguments 对象的成员的值

**25、请写出以下输出结果：**

function Foo() {

getName = function () { alert (1); };

return this;

}

Foo.getName = function () { alert (2);};

Foo.prototype.getName = function () { alert (3);};

var getName = function () { alert (4);};

function getName() { alert (5);}

//请写出以下输出结果：

Foo.getName();

getName();

Foo().getName();

getName();

new Foo.getName();

new Foo().getName();

new new Foo().getName();//new ((new Foo()).getName)();

答案：（2,4,1,1,2,3,3）

拓展：

var getName=function(){console.log("1");}

function getName(){console.log("2");}

console.log(getName);//1 function name的写法总是被提升到作用域的最顶端

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

非目标元素捕获 -> 目标元素代码顺序(先绑定哪种就先发生那种) -> 非目标元素冒泡。

**27、定时器setInterval有一个有名函数fn1，setInterval（fn1,500）与setInterval（fn1(),500）有什么区别？**

第一个是重复执行每500毫秒执行一次，后面一个只执行一次。

**28、window.onload与$(document).ready() 区别**

* 执行时间

window.onload必须等到页面内包括图片的所有元素加载完毕后才能执行。

$(document).ready()是DOM结构加载完后就执行，不必等到图片加载完。

* 编写个数不同

window.onload不能同时编写多个，如果有多个window.onload方法，只会执行一个

$(document).ready()可以同时编写多个，并且都可以得到执行

* 简化写法

window.onload没有简化写法

$(document).ready(function(){})可以简写成$(function(){});

* $(document).ready 相当于原生js 的DOMContentLoaded

**29、jQuery 的属性拷贝(extend)的实现原理是什么，如何实现深拷贝？**

递归赋值 引用类型和基本类型

**30、滚轮事件（处理兼容代码）**

var testNode = document.getElementById("test");

testNode.onmousewheel=fn;//IE/Opera/Chrome

test.addEventListener && test.addEventListener('DOMMouseScroll' ,fun)//FF

function fn(ev){

var e = ev||event;

//方向

var flag="";

if(e.wheelDelta){//IE/Opera/Chrome/Opera

//正:向上,负:向下

flag = e.wheelDelta>0?"up":"down";

}else if(e.detail){//FF

//正:向下,负:向上

flag = e.detail>0?"down":"up";

}

switch (flag){

case "up":

testNode.style.height = testNode.offsetHeight -10+"px";

break;

case "down":

testNode.style.height = testNode.offsetHeight +10+"px";

break;

}

test.addEventListener && event.preventDefault()

return false;

}

**31、区分onmouseover,onmouseout,onmouseenter,onmouseleave**

onmouseover,onmouseout会发生冒泡事件，会传递到上一级

onmouseenter,onmouseleave单纯的划入划出，里边的元素不会受到影响

**32、PC端事件在移动端会有300ms延迟的问题**

click:点击事件，300ms后遮罩消失，只触发click事件，但是，click在移动端会默认触发touchstart事件，300ms后，click事件让遮罩消失

touchstart：点击事件，首先立即响应，遮罩消失，然后300ms后，会默认触发click事件，页面的a标签发生跳转，此时click作用在a标签上

所以，基于上述问题：需要取消系统的事件默认行为

document.addEventListener("touchstart",function(ev){

ev.preventDefault();

})

1. **js访问对象的两种方式的区别？**

.（点）和[]（括号）

* 点表示法的属性名是标识符，括号表示法的属性名是字符串
* 标识符是静态的，在程序中必须对其硬编码，字符串是可以通过动态构建
* 如果属性名是动态的，只能使用括号的方式
* 在对象中只能使用点标示法访问对象的方法

1. **对象的可枚举属性、几种遍历方式的差异？**

* js中基本包装类型的原型属性是不可枚举的
* Object对象的propertyIsEnumerable()方法可以判断此对象是否包含某个属性，并且这个属性是否可枚举。
* 需要注意的是：如果判断的属性存在于Object对象的原型内，不管它是否可枚举都会返回false。
* for ...in 、Object.keys()、Object.getOwnProperty()、for ...of的遍历对象的返回值的区别？
* for...in:能遍历自身和继承于原型的可枚举属性，遍历的顺序不确定，当设置数组的length值大于数组中真正数据的长度时，不会输出数组中undefined的值。
* Object.keys():用于遍历自身的可枚举属性，返回属性名构成的数组。
* Object.getOwnProperty():返回对象的自有属性，包括可枚举和不可枚举的

1. **for循环、for..in、foreach、for...of遍历数组的区别？**

**枚举属性：**我们可以通过各个对象所提供的propertyIsEnumerable()判断某个属性是否可枚举，内建对象和方法如constructor、length等不可枚举，**来自原型链上的属性**也是不可枚举的，但是会被for...in遍历出来。

举个例子：

Array.prototype.name=10;

var a=[1,2,3];

for(var i in a){console.log(i);} // i=0 1 2 name

a.propertyIsEnumerable("name") //false

但是propertyIsEnumerable()调用的是来自原型链的某个对象，那么该对象中的属性是可枚举的。

a.constructor.prototype.propertyIsEnumerable("name") //true

**判断是自身属性还是原型属性：**hasOwnProperty

**for...in:**不建议用来遍历数组，在遍历数组的时候有许多弊端，主要如下几点

* 数组的索引值是string类型而不是number类型
* 在遍历元素时，顺序可能是不确定的
* 会将扩展出来的属性也遍历出来

Array.prototype.name=10;

var a=[1,2,3];

for(var i in a){console.log(i);}

// i=0 1 2 name

// a[i]=1 2 3 10

**foreach：**在遍历时不能使用break跳出循环，在回调中不能使用return 语句

**for...of:**es6新增的的方法，遍历数组时避免了上述两种方式的缺点，还可以用来遍历其他类数组对象、set、map集合，但不支持对普通对象的迭代（可以使用for...in）。

**Symbol 作为属性名：**该属性不会出现在for...in、for...of循环中，也不会被Object.keys()、Object.getOwnPropertyNames()、JSON.stringify()返回。但是，它也不是私有属性，有一个Object.getOwnPropertySymbols方法，可以获取指定对象的所有 Symbol 属性名。

Object.getOwnPropertySymbols方法返回一个数组，成员是当前对象的所有用作属性名的 Symbol 值。

WeakSet不可遍历，因为成员都是弱引用，随时可能消失

详解链接：<http://www.csdn.net/article/1970-01-01/2824965>

1. **深拷贝**

var deepCopy=function(p,c){

var c=c||{};

for (var i in p){

if (typeof p[i]==="Object"){

c[i]=(p[i].constructor==="Array" )? []:{};

deepCopy(p[i],c[i]);

}else{

c[i]=p[i];

}

}

return c;

}

## Ajax

1. **ajax解决浏览器缓存问题？**

* 在ajax发送请求前加上 anyAjaxObj.setRequestHeader("If-Modified-Since","0")。
* 在ajax发送请求前加上 anyAjaxObj.setRequestHeader("Cache-Control","no-cache")。
* 在URL后面加上一个随机数： "fresh=" + Math.random();。
* 在URL后面加上时间戳："nowtime=" + new Date().getTime();。
* 如果是使用jQuery，直接这样就可以了 $.ajaxSetup({cache:false})。

这样页面的所有ajax都会执行这条语句就是不需要保存缓存记录。

**2、页面编码和被请求的资源编码如果不一致如何处理？**

对于ajax请求传递的参数，如果是get请求方式，参数如果传递中文，在有些浏览器会乱码，不同的浏览器对参数编码的处理方式不同，所以对于get请求的参数需要使用 encodeURIComponent函数对参数进行编码处理，后台开发语言都有相应的解码api。对于post请求不需要进行编码。

## **ECMAScript6 相关**

**1、操作符**

Object.is() 与原来的比较操作符“ ===”、“ ==”的区别？

两等号判等，会在比较时进行类型转换；

三等号判等(判断严格)，比较时不进行隐式类型转换,（类型不同则会返回false）；

Object.is 在三等号判等的基础上特别处理了 NaN 、-0 和 +0 ，保证 -0 和 +0 不再相同，但 Object.is(NaN, NaN) 会返回 true.

Object.is 应被认为有其特殊的用途，而不能用它认为它比其它的相等对比更宽松或严格。

**2、比较var, let和const**

\* var : E7S6之前用来定义变量的关键字, 有变量提升, 没有块作用域

\* let : ES6用来定义变量的关键字, 没有变量提升, 有块作用域

\* const : ES6用来定义常量的关键字,没有变量提升, 有块作用域

**3、说说promise的作用, 编码演示promise的基本使用**

解决回调地狱(回调函数的层层嵌套, 编码是不断向右扩展, 阅读性很差)

var promise = new Promise(function(resolve, reject){

//做异步的操作

if(成功) {

resolve(result);

} else {

reject(errorMsg);

}

})

promise.then(function(

result => console.log(result) ,

errorMsg => alert(errorMsg)

)

## 框 架、nodejs

## 1、需求：实现一个页面操作不会整页刷新的网站，并且能在浏览器前进、后退时正确响应。给出你的技术实现方案？

至少给出自己的思路（url-hash,可以使用已有的一些框架history.js等）

**2、Node.js的适用场景？**

1)、实时应用：如在线聊天，实时通知推送等等（如socket.io）

2)、分布式应用：通过高效的并行I/O使用已有的数据

3)、工具类应用：海量的工具，小到前端压缩部署（如grunt），大到桌面图形界面应用程序

4)、游戏类应用：游戏领域对实时和并发有很高的要求（如网易的pomelo框架）

5)、利用稳定接口提升Web渲染能力

6)、前后端编程语言环境统一：前端开发人员可以非常快速地切入到服务器端的开发（如著名的纯Javascript全栈式MEAN架构）

**3、(如果会用node)知道route, middleware, cluster, nodemon, pm2, server-side rendering么?**

Nodejs相关概念的理解程度

**4、什么是“前端路由”?什么时候适合使用“前端路由”? “前端路由”有哪些优点和缺点?**

熟悉前后端通信相关知识

**5、如何设计突发大规模并发架构？**