

前端好帖学习整理

2.你应该知道的25道Javascript面试题

<http://web.jobbole.com/84723/>

1)用typeof是否能判断一个对象变量？如何规避这个问题？

答案：不能，null是object,因为null是空对象指针；除function类型外的其他引用类型，如Array,Date,RegExp,都会返回object；单体内置对象的Math对象也会返回object

规避方法：

```
...
var obj = {};

// 1不够准确，没有排除obj是Date等类型
console.log((obj !== null) & (typeof obj === "object") && (toString.call(obj) !== "[object Array]"));

// 2 没懂，待再研究
console.log(Object.prototype.toString.call(obj) === "[object Object]");//toString转化为字符串

// 3用instanceof
...
```

2)以下代码会输出什么？

```
...
(function(){
  var a = b = 3;
})();

console.log("a defined? " + (typeof a !== 'undefined'));//false
console.log("b defined? " + (typeof b !== 'undefined'));//true
...
```

答案：a为undefined，因为a是局部变量，只能在上述函数中访问，不能在函数外访问；b是全局变量，故b为undefined。以上函数赋值过程（从右到左赋值）等价于：

```
...
(function(){
  b=3;
  var a=b;
})();
...
```

3) 以下代码会输出什么？

```
...
var myObject = {
  foo: "bar",
  func: function() {
    var self = this;//指向myObject
    console.log("outer func:  this.foo = " + this.foo);//bar
    console.log("outer func:  self.foo = " + self.foo);//bar
    (function() {
      console.log("inner func:  this.foo = " + this.foo);//undefined
      console.log("inner func:  self.foo = " + self.foo);//bar
    })();
  }
};
myObject.func();
...
```

答案：只有第三个输出undefined,因为第三个的this是匿名函数中的，指向的是全局对象windows。

每个函数在被调用时都会自动取得两个特殊变量:this和arguments。内部函数在搜索这两个变量时，只会搜索到其活动对象为止，因此永远不可能直接访问外部函数中的这两个变量。不过，把外部作用域中的this对象保存到一个闭包能够访问到的变量里，就可以让闭包访问这个变量了。

（见《JavaScript高级》P182 匿名函数的执行环境具有全局性，this通常指向window）

4)为什么要用IIFE？

答案：为了模仿块级作用域，创建私有变量。

（参见《JavaScript高级程序设计》P184）

(参见 详解javascript立即执行函数表达式（IIFE）)

整理的答案：IIFE的好处

- （1）立即执行函数能配合闭包保存状态。
- （2）立即执行函数在模块化中也大有用处。用立即执行函数处理模块化可以减少全局变量造成的空间污染，构造更多的私有变量。

5)在严格模式下进行JavaScript开发有啥好处？

答案：好处主要有：- 消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为；- 消除代码运行的一些不安全之处，保证代码运行的安全；- 提高编译器效率，增加运行速度；- 为未来新版本的Javascript做好铺垫。（详细参见JavaScript严格模式详解）

举例：

- (1) 不加var创建变量。严格模式下报错，非严格模式下创建全局变量
- (2) 一个对象里面有重名属性。严格模式下报错，非严格模式下以第二个属性值为准。

6) 执行以下两个函数，返回相同吗

```
...
function foo1()
{
    return {
        bar: "hello"
    };
}

function foo2()
{
    return
    {
        bar: "hello"
    };
}

...
```

答案： 尝试

```
console.log(foo1()+foo2());//返回[object Object]undefined
```

不会返回相同的東西，第二个函数会返回 undefined。这是由于 Javascript 的分号插入机制决定的，如果某行代码，return 关键词后没有任何东西了，将会自动插入一个分号，显然 foo2 函数中，当 return 后被插入一个分号后，尽管后面的语句不符合规定，但是因为没有执行到，所以也不会报错了。没有 return 任何东西的函数，默认返回 undefined。

所以很多 **Javascript** 规范建议把 { 写在一行中，而不是另起一行。

7) NaN是什么？typeof的结果是？怎么确定一个变量的值是不是NaN

答案： NaN，即非数值（not a number），用于表示一个本来要返回数值的操作数未返回数值的情况（这样就不会出错了）。如JavaScript中任何数值除以非数值都会返回NaN。

typeof NaN的结果是number。

通过isNaN(value)的方法可以判断value是不是NaN。

（详见《JavaScript高级教程》P29）

8) 以下代码输出结果是什么？并解释。

```
...
console.log(0.1 + 0.2);//0.30000000000000004
console.log(0.1 + 0.2 == 0.3);//false
...
```

答案： 详见<http://www.cnblogs.com/zichi/p/5034201.html>

JS数字存储原理复习： JavaScript中的数字以IEEE754双精度64位浮点数标准。表示格式为

```
...
(s)*(m)*(2^e)
...
```

s:符号位，正负1。

m:尾数(小数点后的部分)，52bits（52为二进制位）

e:指数，有11bits。范围是[-1074,971]。

故JavaScript能表示的最大值为:

```
...
1* (Math.pow(2,53)-1)*Math.pow(2,971)=1.7976931348623157e+308
...
```

最小值（正的最小值）为:

```
...
1* Math.pow(2,-1074)=5e-324
...
```

回到0.1+0.2的问题

1.十进制转二进制

十进制转二进制，整数部分“除二取余，倒叙排列”，小数部分“乘二取整，顺序排列”。（也可用JS的toString(2)转换）

```
...
// 0.1 转化为二进制
0.0 0011 0011 0011 0011...(0011循环)

// 0.2 转化为二进制
```

```
0.0011 0011 0011 0011 0011... (0011循环)
...

```

2. 转化为IEEE754双精度64位浮点数（使用IEEE754 Rounding modes）

计算机并不能表示无限小数，毕竟只有有限的资源，于是我们得把它们用 IEEE 754 双精度 64 位浮点数来表示：

```
...
e = -4; m = 1.1001100110011001100110011001100110011001100110011010 (52位)
e = -3; m = 1.100110011001100110011001100110011001100110011010 (52位)
...

```

虽然我们已经明确 m 只能有 52 位（小数点后），但是如果第 53 位是 1，是该进位还是不进位？这里需要考虑 IEEE 754 Rounding modes

1.101和1.10,1.11一样近，取末位是偶数的,即1.10。所以，52为和53位都为1，就要进位（得到上述结果）

3.做加法

相加时如果指数不一致，需要对齐，一般情况下是向右移，因为最右边的即使溢出了，损失的精度远远小于左边溢出。

```
...
e = -4; m = 1.100110011001100110011001100110011001100110011010 (52位)
+ e = -3; m = 1.100110011001100110011001100110011001100110011010 (52位)
-----
e = -3; m = 0.110011001100110011001100110011001100110011001101
+ e = -3; m = 1.100110011001100110011001100110011001100110011010
-----
e = -3; m = 10.011001100110011001100110011001100110011001100111
-----
e = -2; m = 1.001100110011001100110011001100110011001100110100(52位)
-----
= 0.010011001100110011001100110011001100110011001100110100
= 0.30000000000000004(十进制)
...

```

4.扩展

9007199254740992 + 1 = 9007199254740992 的推理过程大同小异。

9007199254740992 其实就是 2 ^ 53。

```
...
e = 0; m = 10000000000000000000000000000000000000000000000000000000 (53个0)
+ e = 0; m = 1
-----
e = 0; m = 10000000000000000000000000000000000000000000000000000001
...

```

由Rounding modes 的偶数原则，所以将 53 bit 的 1 舍去，所以大小跟 2 ^ 52 并没有变化，试想下，如果是 + 2，那么结果就不一样了。

9)写一个方法 isInteger(x)，可以用来判断一个变量是否是整数。

1. ES6 中自带Number.isInteger()方法

```
...
var a=-1.12;
console.log(Number.isInteger(a));//false
...

```

2. ES5中可以对变量求整数，然后通过判断求整后是否与原变量相等来判断其是否为整数

```
...
var a=-1.2223;

console.log(a^0);//-1 ---按位异或，直接去掉小数
console.log(a|0);//-1 ---按位或，直接去掉小数
console.log(a>0);//false--- 比较运算符，不能去掉小数
console.log(a>>0);//-1 --- 有符号右移0位，直接去掉小数
console.log(a>>>0);//4294967295---无符号右移0位，不能去掉小数，但对于正数可以去掉小数
console.log(a<<0);//-1---左移0位，直接去掉小数

console.log(Math.ceil(a));//-1--向上求整
console.log(Math.floor(a));//-2--向下求整
console.log(Math.round(a));//-1--四舍五入求整
...

```

参见<http://www.cnblogs.com/kkun/archive/2012/01/30/2332309.html>

10)以下代码输出结果是什么？

```
...
(function() {
  console.log(1);
  setTimeout(function(){console.log(2)}, 1000);
  setTimeout(function(){console.log(3)}, 0);
  console.log(4);
})();
...

```

答案： 1 4 3 2

因为JavaScript 是单线程的语言，一些异步事件是在主体 **js** 执行完之后执行，所以主体1、4先输出，而后是3、2（因为3的定时设置比2早）

具体参见 [从setTimeout谈JavaScript运行机制](#) *待再认真研究*

11）判断一个字符串是不是回文？（回文：字符串从左到右读，和从右到左读是一样的）

```
...
function isPalindrome(str) {
    str = str.replace(/W/g, '').toLowerCase();//字符串的replace()方法用于在字符串中用一些字符替换另一些字符，或替换一个与正则表达式匹配的子串。
                                           //toLowerCase()把字符串转化为小写
                                           //正则表达式/W/g,匹配所有非单词字符，为什么W不写成\\w??
    return (str == str.split('').reverse().join(''));//字符串的split()把字符串分割为字符串数组
                                           //数组的reverse()颠倒数组中元素的顺序。
                                           //数组的join('')把数组的所有元素放入一个字符串。元素通过指定的分隔符进行分隔。
}

var a='aaab$bcCB$baaa';

alert(isPalindrome(a));
...
```

个人觉得，只要 `return (str == str.split('').reverse().join(''))`；一句就可以。答案是回文忽略了字母大小写。

没太懂.replace(/W/g, ")的作用，第一，正则表达式匹配所有非单词字符不是用\\W/g吗？第二，即使没有这一项，又会有什么影响呢？

12）写一个 sum 方法，使得以下代码得到预期结果。

```
...
console.log(sum(2,3)); // Outputs 5
console.log(sum(2)(3)); // Outputs 5
...
```

答案：

```
...
function sum(x) {
    if (arguments.length==2) {
        return arguments[0]+arguments[1];
    }
    else{
        return function(y){//这步不太理解
            return x+y;
        }
    }
}
...
```

或者

```
...
function sum(x,y) {
    if (y!=undefined) {
        return x+y;
    }
    else{
        return function(y){
            return x+y;
        }
    }
}
...
```

有关sum(x)(y)还是不怎么理解，待深入研究

更多好编程题见 [汤姆大叔的6道JavaScript编程题题解](#)

13）思考下面的代码段：

```
...
for(var i=0;i<5;i++){
    var btn=document.createElement('button');//document.createElement(<tag>)方法：创建一个属于指定标签类型的新HTMLElement对象
    btn.appendChild(document.createTextNode('Button'+i));//document.createTextNode(<text>)方法：创建一个带有指定内容的新Text对象（即上述
    btn.addEventListener(//为元素添加事件监听器
        'click',
        function(){
            console.log(i);
        }
    );
    document.body.appendChild(btn);//document.A.appendChild(B)方法：将B元素添加为A的子元素
}
...
```

(以上方法查看《HTML5权威指南》P589)

- 点击“Button4”后输出什么？如何使得输出和预期相同
- 给出一个可以和预期相同的写法。

答案：

- 输出5，因为形成了闭包，循环结束后，i为5，所有按钮点击都是5

b. 1. 我的方法：

```
...
for(var i=0;i<5;i++){
    var btn=document.createElement('button');
    btn.appendChild(document.createTextNode('Button'+i));
    btn.addEventListener(
        'click',
        function(e){
            for(var i=0;i<5;i++){
                if (e.target.innerHTML=="Button'+i) {
                    console.log(i);
                }
            }
        }
    );
    document.body.appendChild(btn);//document.A.appendChild(B)方法：将B元素添加为A的子元素
}
...
```

作者大神的方法：

这10道javascript笔试题你都会么 中的第 8 题。

参考可以得出其他方法： 2. 闭包

这是错误的：

```
...
for(var i=0;i<5;i++){
    var btn=document.createElement('button');
    btn.appendChild(document.createTextNode('Button'+i));
    btn.addEventListener(
        'click',
        (function(i){
            (function(){
                console.log(i);
            })();
        })(i)
    );
    document.body.appendChild(btn);//document.A.appendChild(B)方法：将B元素添加为A的子元素
}
...
```

这也是错误的：

```
...
for(var i=0;i<5;i++){
    var btn=document.createElement('button');
    btn.appendChild(document.createTextNode('Button'+i));
    btn.addEventListener(
        'click',
        (function(i){
            console.log(i);
        })(i)
    );
    document.body.appendChild(btn);//document.A.appendChild(B)方法：将B元素添加为A的子元素
}
...
```

这才是正确的闭包：

```
...
for(var i=0;i<5;i++){
    var btn=document.createElement('button');
    btn.appendChild(document.createTextNode('Button'+i));
    (function(a){
        btn.addEventListener(
            'click',
            function () {
                console.log(a);
            }
        )
    })(i);
    document.body.appendChild(btn);//document.A.appendChild(B)方法：将B元素添加为A的子元素
}
...
```

其实，闭包就是在要引用外部变量i的函数外面加上一个 用作块级作用域的匿名函数

```
...
(function(i){
    //某某内部使用了i的函数
})(i);
...
```

13) 引申 这10道javascript笔试题你都会么 中的第 8 题。

实现一段脚本，使得点击对应链接alert出相应的编号

1. DOM 污染法 通过给document元素对象添加了属性值，故污染了DOM

```
...
var lis = document.links;// 属于DOM Document对象，非Dom Element对象，返回文档里具备href属性的a和area元素的对象。参见《HTML5权威指南》P54
for(var i = 0, length = lis.length; i < length; i++) {
    lis[i].index = i;//此index为自己设置的任意变量值，可任意替换为myIndex等等，也可使用固有的元素对象属性，如id等
    lis[i].onclick = function() {
        alert(this.index);//也可用function(e),后面this换为e.target
    };
}
```



该法自己的习惯写法为:

```
...
var lis = document.getElementsByTagName("a");// 属于DOM Document对象, 非Dom Element对象
for(var i = 0; i < lis.length; i++) {
    lis[i].id = i;//把number型的i赋值给id后, 直接转换为了string型, 因为id都是string型的。
    lis[i].onclick = function(e) {
        alert(e.target.id);
    };
}
...
```

2. 使用闭包

```
...
var lis=document.links;

for(var i=0,len=lis.length;i<len;i++){
    (function(a){
        lis[a].onclick=function(){
            alert(a);
        };
    })(i);
}
...
```

3.我的惯用方法(事件循环索引法*自己命的名*) `` var lis=document.links;

for(var i=0,len=lis.length;i<len;i++){ lis[i].onclick=function(){ for(var j=0;j<lis.length;j++){ if (this==lis[j]) { alert(j); } } } } `` 其实, 上述j也可就写为i,因为内部循环参数是在局部函数中的, 故循环完成后自动销毁, 对外部没有影响。

更多关于闭包其实闭包并不高深莫测

14)以下代码段输出什么？

```
...
var arr1 = "john".split('');
console.log(arr1);// ["j", "o", "h", "n"]

var arr2 = arr1.reverse();
console.log(arr1);// ["n", "h", "o", "j"]
console.log(arr2);// ["n", "h", "o", "j"]

var arr3 = "jones".split('');
console.log(arr3);// ["j", "o", "n", "e", "s"]

arr2.push(arr3);
console.log(arr1);// ["n", "h", "o", "j", Array[5]]
console.log(arr2);// ["n", "h", "o", "j", Array[5]]

console.log("array 1: length=" + arr1.length + " last=" + arr1.slice(-1)); //array 1: length=5 last=j,o,n,e,s
console.log("array 2: length=" + arr2.length + " last=" + arr2.slice(-1)); //array 2: length=5 last=j,o,n,e,s

...
```

答案： 每一步输出结果如每行注释所示

坑： 1. 字符串的split()方法、数组的reverse()方法都是就地执行, 即改变了原变量, 而非创建一个副本

1. 数组是引用类型, 故arr2和arr1其实是引用了同一个对象, 故在arr2上的操作也会反映到arr1上。**好厉害！！待举一反三至其他数据类型**

15) 以下代码段输出什么？

```
...
console.log(1 + "2"+"2");//122

console.log(1 + "2"+"2");//32
console.log("1" + +2+"2");//122
console.log("1" + +"2"+"2");//122

console.log(1 + -"1"+"2");//02
console.log("1" + -"1"+"2");//1-12

console.log(+ "1" + "1" + "2");//112
console.log( "A"-"B"+"2");//NaN2
console.log( "A"-"B"+2);//NaN

...
```

答案： 如上述注释

坑 1. 正常情况下, 两个操作数之间都只有一个运算符 (含第一个操作数前面有符号)：

- 对于“+”：如果两个操作数都是字符串, 则做拼接
- 对于“+”：如果两个操作数都是数值, 则做加法
- 对于“+”：如果两个操作数一个数值, 一个字符串, 则把操作数转化为字符串, 再做拼接
- 对于“-”：如果两个操作数都是数值, 则做减法
- 对于“-”：如果至少有一个操作数是字符串、布尔值、null或undefined, 则先把该操作数转换为数值, 再

做减法，转换结果为NaN，则结果为NaN

- 非正常情况下，两个操作数之间有两个符号（+ - 或 + +）
- 如果一前一后两个操作数前一个数值，后一个字符串，则先把后面的字符串转换为数值再做加法
- 若+ - 号前后都为字符串，则输出 前字符串-后字符串

更多参见《JavaScript高级程序设计》P48-49

16) 以上代码可能会由于递归调用导致栈溢出，如何规避这个问题？先放着

```
...
var list = readHugeList();

var nextListItem = function() {
    var item = list.pop();

    if (item) {
        // process the list item...
        nextListItem();
    }
};
...
```

答案：

方法一：利用setTimeout的异步性质避免调用栈的形成

```
`` var list = readHugeList();
```

```
var nextListItem = function() { var item = list.pop();
```

```
    if (item) {
        // process the list item...
        setTimeout( nextListItem, 0);
    }
};
```

}; `` 解释： 原文代码形成了一个调用栈，而新方法利用setTimeout的异步性质，完美地除去了这个调用栈 举例：

```
...
var list=[0,1];

var nextListItem=function(){
    var item=list.pop();
    if (item) {
        nextListItem();
    }
    console.log(item);
};

nextListItem();
...
```

因为递归，程序中形成了一个调用栈，1被压入了调用栈栈底，0后进先出，故上面的代码会依次输出0和1

改写程序为： `` var list=[0,1];

```
var nextListItem=function(){ var item=list.pop(); if (item) { setTimeout(nextListItem,0); } console.log(item);
};
```

nextListItem(); `` 这回输出的就是 1 和 0 了，因为 setTimeout 的回调只有当主体的 js 执行完后才会去执行，所以先输出了 1，自然也就没有调用栈这一说法了。

事实上，并不是所有递归都能这样改写，如果下一次递归调用依赖于前一次递归调用返回的值，就不能这么改了。

方法二：迭代

任何递归都可以用迭代来代替

- 递归和迭代都是循环的一种。简单地说，递归是重复调用函数自身实现循环。迭代是函数内某段代码实现循环，而迭代与普通循环的区别是：循环代码中参与运算的变量同时是保存结果的变量，当前保存的结果作为下一次循环计算的初始值。迭代经典例子：实现1-100所有实数的和

int v=1; for(i=2;i<=100;i++) { v=v+i; } 此例迭代写法： `` var list=[0,1];

```
var item=list.pop();
```

```
while (item!=undefined) { console.log(item); item=list.pop(); } ``
```

17)谈谈JavaScript的闭包

待整理，参见 闭包初窥 及闭包拾遗

18)以下代码输出什么？如何输出期望值？

```
...
for (var i = 0; i < 5; i++) {
```

```
    setTimeout(function() { console.log(i); }, i * 1000 );
  }
  ...
```

答案：输出5个5，和十三题类似，又是循环+闭包的问题。

修改方法一：

修改代码如下：

```
...
for (var i = 0; i < 5; i++) {
  (function(i){
    setTimeout(function() { console.log(i); }, i * 100);
  })(i);
}
...
```

规律总结：遇上循环+闭包最后输出一排同样的值的问题时，采用立即执行函数+闭包

修改方法二：

```
...
for(var i = 0; i < 5; i++) {
  setTimeout(console.log.bind(console, i), i * 100);
}
...
```

这里优雅地使用bind函数。***bind函数待研究***

19)以下代码在控制台输出什么？

```
...
console.log("0 || 1 = "+(0 || 1));//1
console.log("1 || 2 = "+(1 || 2));//1
console.log("0 & 1 = "+(0 && 1));//0
console.log("1 & 2 = "+(1 && 2));//2
...
```

答案： 参见代码后注释。

||和&&都为短路运算符。

||：如果前面变量值为false（包括0、null、undefined、false、空字符串等等），则返回后面变量值，否则返回前面变量值；

&&：与||恰恰相反。若前面变量为false则返回前面变量值，否则返回后面变量值。

20）以下代码在控制台输出什么？

```
...
console.log(false == '0');//true
console.log(false === '0');//false
...
```

答案： 参见代码后注释。

==比较时值相等即可，类型可以不等，其在比较时会进行隐式的转换。

===为全等，只有两个值完全相同，包括其类型也相同或者两个对象引用相同时，才会返回true。

21）以下代码在控制台输出什么？先放着

```
...
var a={},
    b={key:'b'},
    c={key:'c'};

a[b]=123;
a[c]=456;

console.log(a[b]);//456
...
```

Javascript 中对象的 key 值，一定会是一个 string 值，如果不是，则会隐式地进行转换。当执行到 a[b]=123] 时，b 并不是一个 string 值，将 b 执行 toString() 方法转换（得到 "[object Object]"），a[c] 也是相同道理。所以代码其实可以看做这样执行：

```
...
var a={},
    b={key:'b'},
    c={key:'c'};

// a[b]=123;
a["[object Object]"]=123;

// a[c]=456;
a["[object Object]"]=456;

console.log(a["[object Object]"]);
...
```


好神奇

22) 以下代码在控制台输出什么？先放着

```
...
var ans=(function f(n){
    return ((n>1)?n*f(n-1):n);
})(10);

console.log(ans);//3628800
...
```

答案： 就是求（10！）。其实就是立即执行函数+递归。。给立即执行函数加了个名字 f，方便在递归里调用，其实完全可以用 arguments.callee 代替：

```
...
var ans=(function f(n){
    return ((n>1)?n*arguments.callee(n-1):n);
})(10);

console.log(ans);//3628800
...
```

23)以下代码片段在控制台输出什么？先放着

```
...
(function(x) {
    return (function(y) {
        console.log(x);
    })(2)
})(1);//1
...
```

答案： 为1， 因为这是闭包，可以引用外层函数的变量。

引申：改成这样呢？ (function(y) { return (function(y) { console.log(y); })(2) })(1);//2 答案：为2，因为闭包其作用域链从最内层函数开始，搜索到了相应变量就不再继续向外搜索。

24) 以下代码段在控制台输出什么？如何规避这个问题？先放着

```
...
var hero = {
    _name: 'John Doe',
    getSecretIdentity: function (){
        return this._name;
    }
};

var stoleSecretIdentity = hero.getSecretIdentity;

console.log(stoleSecretIdentity()); //undefined
console.log(hero.getSecretIdentity()); //John Doe
...
```

答案：第一次输出时，this指向window 该知识点待查书研究 第二次输出时，this指向hero

规避办法，使用bind())

```
...
var hero = {
    _name: 'John Doe',
    getSecretIdentity: function (){
        return this._name;
    }
};

var stoleSecretIdentity = hero.getSecretIdentity.bind(hero);

console.log(stoleSecretIdentity()); //John Doe
console.log(hero.getSecretIdentity()); //John Doe
...
```

更多关于bind()的讲解参见 ECMAScript5(ES5)中bind方法简介备忘

也可以使用call()或apply():

```
...
var hero = {
    _name: 'John Doe',
    getSecretIdentity: function (){
        return this._name;
    }
};

var stoleSecretIdentity = hero.getSecretIdentity;

console.log(stoleSecretIdentity.call(hero)); //John Doe
console.log(stoleSecretIdentity.apply(hero)); //John Doe
console.log(hero.getSecretIdentity()); //John Doe
...
```

关于bind()、apply()、call()小结：

apply: 应用场景为func.apply(obj,[3,4]),接收两个参数，分别为运行函数作用域和参数数组

call: 应用场景为func.call(obj,3,4),接收若干参数，第一个为运行函数作用域，后面的依次为各参数。作用

和apply完全一样。

bind: bind的核心是返回一个未执行的方法.bind是和apply、call一样，是Function的扩展方法,所以应用场景是func.bind(), 而传的参数形式和call一样，第一个参数是this指向，之后的参数是func的实参，fun.bind(thisArg[, arg1[, arg2[, ...]]])。([]表示内容可以没有)

25)写一段代码，遍历 DOM 树。二面重点看

答案： 使用深度优先搜索，如下所示：

```
...
function Traverse(p_element,p_callback) {
    p_callback(p_element);
    var list = p_element.children;//children 属性：取得元素所有的子元素形成的数组
    for (var i = 0; i < list.length; i++) {
        Traverse(list[i],p_callback); // recursive call
    }
}

(function () {
    var root=document.getElementsByTagName("body")[0];

    var callback=function(element){
        console.log(element);
    };

    Traverse(root,callback);
})();
...
```

直接这样写即可：

```
function traverse(root) {
    console.log(root.tagName);
    var elems=root.children;
    for (var i=0,len=elems.length;i<len;i++) {
        traverse(elems[i]);
    }
}

traverse(document.body);
```

知识点补充1： 二叉树的三种遍历<http://zhidao.baidu.com/question/1509136365112556140.html>

- 先序遍历：先遍历根，再遍历左子树，再遍历右子根
- 中序遍历：先遍历左子树，再遍历根，再遍历右子根
- 后序遍历：先遍历左子树，再遍历右子数，最后遍历根

知识点补充2: 深度优先搜索

深度优先搜索是一种在开发爬虫早期使用较多的方法。它的目的是要达到被搜索结构的叶结点(即那些不包含任何超链的HTML文件)。在一个HTML文件中，当一个超链被选择后，被链接的HTML文件将执行深度优先搜索，即在搜索其余的超链结果之前必须先完整地搜索单独的一条链。深度优先搜索沿着HTML文件上的超链走到不能再深入为止，然后返回到某一个HTML文件，再继续选择该HTML文件中的其他超链。当不再有其他超链可选择时，说明搜索已经结束。

25) 引申： 二叉树三种遍历的递归和迭代算法待写为具体到DOM的代码

4.其实闭包并不高深莫测

<http://web.jobbole.com/84456/>

1)引入第一个闭包

```
...
function createCounter() {
    var counter = 0;

    function increment() {
        counter = counter + 1;

        console.log("Number of events: " + counter);
    }

    return increment;
}
...
```

使用上述函数创建两个计时器：

```
...
var counter1=createCounter();//创建新计数器实际是生成新函数
var counter2=createCounter();

counter1();//1
counter1();//2
counter2();//1
counter1();//3
...
```

注意： 在 createCounter() 的最后一步返回了局部函数 increment。请注意，这并不是返回调用函数的运行结果，而是函数本身。这就意味着，当我们创建新的计数器时，实际上是生成新函数。

注意：这就是闭包生命周期的力量所在。 每个生成的函数，都会保持在 createCounter() 所创建的 counter 变量的引用。在某种意义上，被返回的函数记住了它所被创建时的环境。

注意： 内部变量 counter 都是独立存在于每个作用域！例如，如果我们创建两个计数器，那么它们都会在闭包体内会分配一个新的 counter 变量。

2) 为闭包传递参数以命名计数器

```
...
function createCounter(counterName) {
  var counter = 0;

  function increment() {
    counter = counter + 1;

    console.log("Number of "+counterName+":" + counter);
  }

  return increment;
}

var catCounter=createCounter("cats");
var dogCounter=createCounter("dogs");

catCounter();
catCounter();
dogCounter();
catCounter();
...
```

记住： 返回函数不仅记住了局部变量 counter，而且记住了传递进来的参数。

3)改善公用接口

可以接口返回一个包含该闭包所有功能的对象。

```
...
function createCounter(counterName) {
  var counter = 0;

  function increment() {
    counter = counter + 1;

    console.log("Number of "+counterName+":" + counter);
  }

  return{
    incre:increment
  };
}

var dogCounter=createCounter("dogs");

dogCounter.incre();//Number of dogs:1
...
```

4) 增加一个减量计数器

```
...
function createCounter(counterName) {
  var counter = 0;

  function increment() {
    counter = counter + 1;
    console.log("Number of "+counterName+":" + counter);
  }
  function decrement() {
    counter=counter-1;
    console.log("Number of "+counterName+":"+counter);
  }
  return{
    incre:increment,//这里方法名和方法的名字可以不同，调用时调用的是方法名
    decre:decrement
  };
}

var dogCounter=createCounter("dogs");

dogCounter.incre();//Number of dogs:1
dogCounter.incre();//Number of dogs:1
dogCounter.decre();//Number of dogs:1
...
```

5) 去掉冗余代码

```
...
function createCounter(counterName) {
  var counter = 0;

  function display() {
    console.log("Number of "+counterName+":" + counter);
  }

  function increment() {
    counter = counter + 1;
    display();
  }
  function decrement() {
    counter=counter-1;
    display();
  }
}
```

```
    }
    return{
      incre:increment,
      decre:decrement
    };
  }
}

var dogCounter=createCounter("dogs");

dogCounter.incre();//Number of dogs:1
dogCounter.incre();//Number of dogs:1
dogCounter.decre();//Number of dogs:1
...
```

可知： 闭包的内层函数不仅可以引用外层函数的局部变量，还可以引用外层函数的局部函数

注意： createCounter返回的字面量中没有display()，故不能这样调用dogCounter.display()。即我们让display() 函数对外部来说是不可见的，它仅在 createCounter() 内可用。

6) 延伸：通过闭包引入抽象数据类型

通过闭包可以非常简单地引入抽象数据类型。如，通过闭包实现一个数组堆栈

```
...
function createStack() {
  var elements=[];// 数组字面量方法创建一个空数组

  return{
    myUnshift:function(a){
      elements.unshift(a);//在数组前端添加一项
    },
    myShift:function(){
      elements.shift();//在数组前端移除一项
    },
    ele:elements
  }
}

var stack=createStack();

stack.myUnshift(3);
stack.myUnshift(4);
stack.myUnshift(5);
stack.myShift();

console.log(stack.ele);
...
```

5.关于Chrome控制台的使用

<http://web.jobbole.com/81918/>

获取元素上绑定的事件的方法有待总结研究

6.JS 一定要放在 Body 的最底部么？聊聊浏览器的渲染机制

<http://web.jobbole.com/84843/>

- **首屏时间：** 网站用户体验的一个重要指标。指一个网站被浏览器如IE窗口上部800*600的区域（我的电脑屏幕是1366768）被充满所需时间。 页面打开时，总加载时间要比首屏时间要长很多，但是对于用户体验来说，首屏时间是用户对一个网站的重要体验因素。当页面充满800600的区域时，对用户来说就可以看到内容并可以点击了。 目前可以使用基调网络(networkbench)的webwatch工具来测试首屏时间（webwatch集成在基调网络提供的工具包里面）

1) 从一个面试题说起

为什么大家普遍把这样的代码放在body最底部？（为了沟通效率，我会提前和对方约定所有的讨论都以chrome为例）

应聘者一般会回答：因为浏览器生成Dom树的时候是一行一行读HTML代码的，script标签放在最后面就不会影响前面的页面的渲染。

我很鸡贼地接着问：既然Dom树完全生成后页面才能渲染出来，浏览器又必须读完全部HTML才能生成完整的Dom树，script标签不放在body底部是不是也一样？

提问：就算不影响页面渲染，那也影响script代码的运行吧？如果运行script的时候前面找不到dom元素，那script要出错吧？——已验证，这不是问题，放在前面一样不会影响script代码的运行

什么叫页面渲染出来了？

指的是“首屏显示出来了”还是“页面完整地加载好了”（后面统称StepC）？ 如果指的是首屏显示出来了，那么问题又来了：假设网页首屏有图片，这里的“首屏”指的是“显示了全部图片的首屏”（后面统称StepB）还是“没有图片的首屏”（后面统称StepA）。

- StepA:没有图片的首屏
- StepB:显示了全部图片的首屏
- StepC:页面完整地加载好了——如果是指这个，则确实script放在哪都不会影响整个页面完全加载好的时间。

我们往往更关心“没有图片的首屏”。而“没有图片的首屏”并不以“完整的DOM树”为必要条件，即在生成Dom树的过程中只要某些条件具备了，“没有图片的首屏”就能显示出来。

故，问题其实是：**script标签的位置会影响首屏时间吗？**

2) 浏览器的渲染机制

几个概念

- DOM: Document Object Model, 浏览器将HTML解析成树形的数据结构。
- CSSOM: CSS Object Model, 浏览器将CSS代码解析成树形的数据结构。
- DOM 和 CSSOM 都是以 Bytes → characters → tokens → nodes → object model. 这样的方式生成最终的数据。
- DOM树的构建过程是一个深度遍历过程：即当前节点的所有子节点都构建好后，才会去构建当前节点的下一个兄弟节点。
- Render Tree(渲染树)：DOM和CSSOM合并后生成Render Tree。Render Tree 和DOM一样，以多叉树的形式保存了每个节点的css属性、节点本身属性、以及节点的孩子节点。

插播display:none和visibility:hidden的区别

- 区别是：display:none的节点不会被加入RenderTree，而 visibility: hidden 则会，所以，如果某个节点最开始是不显示的，设为 display:none 是更优的。

浏览器的渲染过程

1. Create/Update DOM And request css/image/js: 浏览器请求到HTML代码后，在生成DOM的最开始阶段（应该是 Bytes → characters 后），并行发起css、图片、js的请求，无论他们是否在HEAD里。
注意：发起 js 文件的下载 request 并不需要 DOM 处理到那个 script 节点，比如：简单的正则匹配就能做到这一点，虽然实际上并不一定是通过正则(这句话是什么意思？)。这是很多人在理解渲染机制的时候存在的误区。
2. Create/Update Render CSSOM: CSS文件下载完成，开始构建CSSOM
3. Create/Update Render Tree: 所有CSS文件下载完成，CSSOM构建结束后，和 DOM 一起生成 Render Tree。
4. Layout: 有了Render Tree，浏览器已经能知道网页中有哪些节点、各个节点的CSS定义以及他们的从属关系。这一步就是计算出每个节点在屏幕中的位置。
5. painting: 按照计算出的规则，通过显卡把内容画到屏幕上。

前3个步骤使用“Create/Update”是因为DOM、CSSOM、Render Tree都可能在第一次Painting后又被更新多次，比如JS修改了DOM或者CSS属性。

Layout 和 Painting 也会被重复执行，除了DOM、CSSOM更新的原因外，图片下载完成后也需要调用 Layout 和 Painting来更新网页。

通过TimeLine得出的结论

1. 首屏时间和DomContentLoaded事件没有必然的先后关系
2. 所有CSS尽早加载是减少首屏时间的最关键
3. js的下载和执行会阻塞DOM树的构建（严谨地说是中断了Dom树的更新），索引script标签放在首屏范围内的HTML代码会截断首屏的内容。
4. 普通script标签放在body底部，做与不做async或者defer处理，都不会影响首屏时间，但影响 DomContentLoaded和load的时间，进而影响依赖他们的代码的执行的开始时间。

3) 问题的答案

问题：script标签的位置会影响首屏时间吗？**答案：**不影响（如果这里里的首屏指的是页面从白板变成网页画面——也就是第一次Painting），但有可能截断首屏的内容，使其只显示上面的一部分。

另外，使用开源项目Tiny-loader可以使JS都不用放在body最底部。

6. 页面守护者：Service Worker

<http://imweb.io/topic/56592b8a823633e31839fc01>

(1)Service Worker的身份

是独立于页面的一个运行环境，在页面关闭后仍可以运行。同时，也能对它负责的页面的网络请求进行截取和返回请求。

它的绩效目标如下：- 入职（install）后永不下班，而能更新。- 能处理Boss需要的资源（HTTPS请求），以便离线时也能让BOSS取到数据（从cache中）。**cache:**高速缓冲存储器 - 能向客户推送消息 (push notifications) - 不允许越权管理Boss的事（DOM ACCESS）

7.setTimeout的那些事先放着

<http://imweb.io/topic/56ac67f3e39ca21162ae6c78>

(2) 理解setTimeout

- JS主线程 => BOSS
- 同步任务 => BOSS手头上正在做的任务
- 异步任务（队列） => BOSS的小本本上的任务

实际上，**setTimeout做的事情是：在指定delay时间后，将指定方法作为异步任务添加到异步任务队列中。** 所以就比较惨：- 如果setTimeout的定时到了执行时间，JS主线程仍然还在执行同步任务，setTimeout所指定的方法并不会立刻执行。- 更惨的是，即使JS主线程执行完了同步任务，也不一定会执行setTimeout指定的方法，因为异步任务队列中可能有更早加入的异步任务。- 最惨的是，即使天时地利人和，到了定时的时间时，JS主线程空闲，异步任务队列中只有setTimeout执行的方法，这个方法的执行时间也并不是精确的delay时间（精确到毫秒），因为浏览器上的计时器精确度有限。

(3)setTimeout应用例子

eg1 替换setInterval来实现重复定时

```
....
setTimeout(function(){
  console.log("I love you."); //do what you want to do
  setTimeout(arguments.callee,2000);
},2000);
....
```

若首次不想等待直接显示，则可以以下代码：

```
....
(function(){
  console.log("I love you."); //do what you want to do
  setTimeout(arguments.callee,2000);
})();
....
```

更好的写法：

```
var num=0;
var max=10;
function increateNumber() {
  num++;
  console.log(num);
  if (num<max) {
    setTimeout(increateNumber,200)
  }
  else{
    alert("Done");
  }
}

setTimeout(increateNumber,200);
```

使用以上setTimeout链式调用的方式，可以保证在下一一次定时器代码执行之前，至少要等待指定的时间间隔，避免连续的运行

eg2 防止事件疯狂触发

浏览器上会有一些疯狂触发的事件，如onresize。如果给该事件绑定了处理函数，浏览器窗口大小改变时会很高频地触发处理函数。如果处理函数中有DOM操作的话，对页面性能影响会很大，尤其是在IE浏览器中，甚至可能让浏览器崩溃。

如果你实在需要在这类事件上绑定操作DOM的函数，那么可以考虑一下限制一下事件执行的时间间隔，至少不要那么频繁。利用setTimeout可以实现事件执行频率控制： ``/** * 限制method执行频次，当方法100ms之内没有 * 再次被调用时，才执行method方法 * @param {function} method 被限制的方法 * @param {Object} context method执行的上下文 /* * 限制method执行频次，当方法100ms之内没有 * 再次被调用时，才执行method方法 * @param {function} method 被限制的方法 * @param {Object} context method执行的上下文 */ function throttle(method, context) { clearTimeout(method.tid);

```
method.tid = setTimeout(function() { //定义定时器method.tid
  method.call(context);
}, 2000);
```

}

function fnResize() { console.log(111); }

window.onresize = function() { throttle(fnResize>window); } ``

eg3 延迟事件生效

经常有这种场景：监控input或者textarea中文本的变化，然后触发某个事件处理程序。考虑到除了键盘输入，还有鼠标的粘贴和剪切操作，比较完整的监控输入内容改变的方法是：

// 响应键盘输入，粘贴和剪切事件 \$('#input').on('keyup paste cut', function() { // 使鼠标粘贴和剪切时，输入框内内容为最新 console.log(\$(this).val()); //jQuery方法，获取\$this元素的值 });

以上代码在键盘输入场景下，能够在控制台输入最新的输入框内文本。但是当使用鼠标右键操作进行粘贴或剪切时，控制台输入的文本内容是操作前的旧内容。**试了下，还真是这样！！** 为了获取操作后的新文本内容，可以将对文本的获取和处理放在setTimeout中延时执行：

// 响应键盘输入，粘贴和剪切事件 \$('#input').on('keyup paste cut', function() { var \$this = \$(this); setTimeout(function() { // 使鼠标粘贴和剪切时，输入框内内容为最新 console.log(\$this.val()); //jQuery方法，获取\$this元素的值 }, 0) });

8.frameset与iframe的区别

http://blog.csdn.net/lyr1985/article/details/6067026

9.2016十家公司前端面试小记

http://web.jobbole.com/85156/

10.Flex布局

关于Flexbox， 参见教程：

Flex布局教程：语法篇

Flex布局教程：实例篇(内含骰子布局、网格布局、圣杯布局)

看《HTML5权威指南》Chapter21

网格布局实例：

```
<head>
...
<style type="text/css">
  .Grid{
    display: -webkit-flex;
  }
  .Grid-cell{
    flex:1; /*flex是flex-grow,flex-shrink,flex-basis的简写*/

    background-color: yellow;
    border:thin solid black;
  }
  .Grid-cell.u-1of3{ /*中间不能有空格*/
    flex:0 0 33.3333%;
  }
  .Grid-cell.u-1of4{
    flex:0 0 25%;
  }
}

</style>
</head>
<body>
  <div class="Grid">
    <div class="Grid-cell u-1of4">abc</div>
    <div class="Grid-cell">abc</div>
    <div class="Grid-cell u-1of3">abc</div>
  </div>

</body>
```

圣杯布局实例：

```
<head>
...
<style type="text/css">
  .HolyGrail {
    display: -webkit-flex;
    min-height: 300px;
    flex-direction: column; /*容器的flex-direction设置其内部项目排列方式，此处为竖着排列*/
  }

  header, footer {
    flex: 1; /*flex是<flex-grow>|<flex-shrink>|<flex-basis>的简写，后两个可以省略。flex-grow为1表示有剩余空间就自动放大填满剩余空间*/
    height: 50px;
    background-color: yellow;
  }

  .HolyGrail-body {
    display: -webkit-flex;
    flex: 1;
    min-height: 200px;
  }

  .HolyGrail-content {
    flex: 1;
    background-color: green;
  }

  .HolyGrail-nav, .HolyGrail-ads {
    /* 两个边栏的宽度设为12em */
    flex: 0 0 12em; /*即flex-basis为12em*/
  }

  .HolyGrail-nav {
    /* 导航放到最左边 */
    order: -1;
    background-color: red;
  }

  @media (max-width: 768px) { /*小屏幕，塞干的三栏自动变为垂直叠加*/
    .HolyGrail-body {
      flex-direction: column;
      flex: 1;
    }
    .HolyGrail-nav,
    .HolyGrail-ads,
    .HolyGrail-content {
      flex: auto; /*flex:auto是flex:1 1 auto,flex-basis为auto即项目本来的大小*/
    }
  }

</style>
```

```
</head>
<body class="HolyGrail">
  <header>...</header>
  <div class="HolyGrail-body">
    <div class="HolyGrail-content">...</div>
    <nav class="HolyGrail-nav">...</nav>
    <aside class="HolyGrail-ads">...</aside>
  </div>
  <footer>...</footer>
</body>
```

输入框的布局

输入框往往前面加提示，后面加按钮。其弹性盒布局的思想就是中间输入框的flex-grow为1,即存在剩余空间时，中间框放大占满剩余空间。

HTML:

```
<div class="InputAddOn">
  <span class="InputAddOn-item">...</span>
  <input class="InputAddOn-field">
  <button class="InputAddOn-item">...</button>
</div>
```

CSS:

```
.InputAddOn {
  display: flex;
}

.InputAddOn-field {
  flex: 1;
}
```

悬挂式布局

主栏的左侧或右侧需添加一个图片栏。此时布局思想是aline-items属性定义为flex-start(即在交叉轴上的上方对齐)，然后主栏的flex-grow为1

HTML:

```
<div class="Media">
  <img class="Media-figure" src="" alt="">
  <p class="Media-body">...</p>
</div>
```

CSS:

```
.Media {
  display: flex;
  align-items: flex-start;
}

.Media-figure {
  margin-right: 1em;
}

.Media-body {
  flex: 1;
}
```

固定高的底栏

页面内容太少，无法占满一屏的话，底栏就会抬高到页面的中间。可以让中间栏的flex-grow为1,自动填满交叉轴的剩余高度。

HTML:

```
<body class="Site">
  <header>...</header>
  <main class="Site-content">...</main>
  <footer>...</footer>
</body>
```

CSS:

```
.Site {
  display: flex;
  min-height: 100vh;
  flex-direction: column;
}

.Site-content {
  flex: 1;
}
```

流式布局

就是每行的项目固定，且可以自动分行。实现方法是：
容器flex-direction为row横向布局，flex-wrap为wrap自动换行，align-content多行对齐方式为flex-start。
项目的flex为0 0 xx%，即有剩余空间时项目不放大不缩小，每个都在主轴上占一定比例。

CSS:

```
.parent {
```



```
width: 200px;
height: 150px;
background-color: black;
display: flex;
flex-flow: row wrap; /*flex-flow是<flex-direction和flex-wrap的简写，此处row意为横向排列，wrap意为自动换行，第一行在上*/
align-content: flex-start; /*align-content定义了多跟轴线的对齐方式*/
}

.child {
box-sizing: border-box;
background-color: white;
flex: 0 0 25%; /*每个项目有剩余空间时不放大，不缩小，占据占主轴空间的25%*/
height: 50px;
border: 1px solid red;
}
```

11. 2016年Web前端面试题目录汇总

<http://web.jobbole.com/85340/>

HTML/CSS部分

1.什么是盒子模型？

在网页中，一个元素占有空间的大小由几个部分构成，其中包括元素的内容（content），元素的内边距（padding），元素的边框（border），元素的外边距（margin）四个部分。这四个部分占有的空间中，有的部分可以显示相应的内容，而有的部分只用来分隔相邻的区域或区域。4个部分一起构成了css中元素的盒模型。

2、行内元素有哪些？块级元素有哪些？空(void)元素有那些？待记忆

行内元素：a、b、span、img、input、strong、select、label、em、button、textarea 块级元素：div、ul、li、dl、dt、dd、p、h1-h6、blockquote 空元素：即系没有内容的HTML元素，例如：br、meta、hr、link、input、img

3.CSS实现水平垂直居中

（1）图片水平垂直居中

容器position用relative,内容position用absolute,后用top/left,margin-top/margin-left配合使用使其居中。

思考：

- 为什么容器的position用relative?因为子元素想要根据容器定位必须使用position,而**position**是针对第一个**position**不是**static**的父元素定位，若容器还想保持在文档流中故只能选用relative。

```
<head>
  <style type="text/css">
    .wrapper{
      position: relative;
      height: 300px;
      width: 100%;
      border:thin solid red;
    }
    .content{
      position:absolute;
      width: 200px;
      height: 200px;
      top: 50%;
      left: 50%;
      margin-top: -100px;
      margin-left:-100px;
      border:thin solid red;
    }
  </style>
</head>

<body>
  <div class="wrapper">
    
  </div>
```

（2）文字水平垂直居中

容器、内容的position都为默认static。容器使用text-align:center使内容水平居中。内容使用line-height值等于容器的height。

注意： - 内容的line-height属性设置必须在font-size属性设置之后。

```
<style type="text/css">
  .wrapper{
    height: 300px;
    width: 100%;
    text-align: center;
    border:thin solid red;
  }
  .content{
    font:20px "微软雅黑",sans-serif;
    line-height: 300px;
    border:thin solid red;
  }
</style>
</head>

<body>
  <div class="wrapper">
```

```
<div class="content">Hello</div>
</div>
```

4.src和href的区别？

href是指向网络资源所在的位置，建立资源和当前元素或当前文档之间的联系。用于a,link等。

src是指向外部资源的位置，指向的内容将会嵌入到文档中当前标签所在的位置；在请求src资源时，会将资源下载并用到文档内；当浏览器解析到该元素时，会暂停其他资源的下载和处理，直到将该资源加载、编译、执行完毕。用于js脚本、img、iframe、frame等。

5.什么是CSS Hack？

针对不同的浏览器写不同的CSS，就是CSS Hack。

(1)条件Hack

```
<!--[if IE]>//只在IE下生效
<style>
.test{color:red;}
</style>
<![endif]-->

<!--[if IE 6]>

<![endif]-->

<!--[if gte IE 6]>//只在IE6以上版本生效

<![endif]-->

<!--[if ! IE]-->//只在非IE上生效

<![endif]-->
```

(2)属性Hack

用法示例

```
.test{
color:#0909; /* For IE8+ */
*color:#f00; /* For IE7 and earlier */
_color:#ff0; /* For IE6 and earlier */
}
```

规律补充

- “_”是IE6专有的hack
- “\9” IE6/IE7/IE8/IE9/IE10都生效
- “\0” IE8/IE9/IE10都生效，是IE8/9/10的hack
- “\9\0” 只对IE9/IE10生效，是IE9/10的hack

(3)选择符Hack

用法示例

```
* html .test{color:#090;} /* For IE6 and earlier */
* + html .test{color:#ff0;} /* For IE7 */
```

规律补充

```
*html *前缀只对IE6生效
*+html *+前缀只对IE7生效
@media screen\9{...}只对IE6/7生效
@media \0screen {body { background: red; }}只对IE8有效
@media \0screen\,screen\9{body { background: blue; }}只对IE6/7/8有效
@media screen\0 {body { background: green; }} 只对IE8/9/10有效
@media screen and (min-width:0\0) {body { background: gray; }} 只对IE9/10有效
@media screen and (-ms-high-contrast: active), (-ms-high-contrast: none) {body { background: orange; }} 只对IE10有效
```

更多参见 [史上最全CSS hack方式一览](#)

6.同步和异步的区别

同步是阻塞模式，异步是非阻塞模式。

同步就是指一个进程在执行某个请求的时候，若该请求需要一段时间才能返回信息，那么这个进程将会一直等待下去，直到收到返回信息才继续执行下去；

异步是指进程不需要一直等下去，而是继续执行下面的操作，不管其他进程的状态。当有消息返回时系统会通知进程进行处理，这样可以提高执行的效率。

8、什么叫优雅降级和渐进增强？

渐进增强 progressive enhancement： 针对低版本浏览器进行构建页面，保证最基本的功能，然后再针对高级浏览器进行效果、交互等改进和追加功能达到更好的用户体验。

优雅降级 graceful degradation： 一开始就构建完整的功能，然后再针对低版本浏览器进行兼容。

区别：

a. 优雅降级是从复杂的现状开始，并试图减少用户体验的供给

- b. 渐进增强则是从一个非常基础的，能够起作用的版本开始，并不断扩充，以适应未来环境的需要
- c. 降级（功能衰减）意味着往回看；而渐进增强则意味着朝前看，同时保证其根基处于安全地带

9.浏览器的内核分别是什么？

```
IE: trident内核
Firefox: gecko内核
Safari: webkit内核
Opera: 以前是presto内核，Opera现已改用Google Chrome的Blink内核
Chrome: Blink(基于webkit, Google与Opera Software共同开发)
```

JavaScript部分

1.添加、移除、复制、移动、创建、查找节点。

已总结，见《web前端笔试题搜集》

2.实现一个函数clone，可以对JavaScript中的5种主要的数据类型（包括Number、String、Object、Array、Boolean）进行值复制。

3、如何消除一个数组里面重复的元素？并返回删除值。

已做，见《NOWCODE错题好题研究整理（二）》

```
Array.prototype.delRe=function(){
    var i=0;
    var delArr=new Array();
    while (this[i]) {
        if (this.indexOf(this[i])==i) {
            i++;
        }
        else{
            var del=this.splice(i,1);
            delArr=delArr.concat(del);
        }
    }
    return delArr;
}

var myArray=[1,1,1,12,12,12,'a','b','ba','ba','bc','a','e','1','2','2','3'];
var delArr=myArray.delRe();

console.log(myArray);
console.log(delArr);
```

4、想实现一个对页面某个节点的拖曳？如何做？（使用原生JS）。

方法一：用HTML5的现有事件 待看《JavaScript高级程序设计》P482

```
<style type="text/css">
    #src*{
        float: left;
    }
    #target,#src>img,#target>img{
        border: thin solid black;
        height: 81px;
        width: 81px;
    }
    #target{
        text-align: center;
        display: table;
    }
    #target>p{
        display: table-cell;
        vertical-align: middle;
    }
    img.dragged{
        background-color: lightgrey;
    }
</style>

</head>

<body>
    <div id="src">
        
        
        
        <div id="target">
            <p id="msg">Drop Here</p>
        </div>
    </div>
    <script>
        var src=document.getElementById("src");
        var target=document.getElementById("target");
        var msg=document.getElementById("msg");
        var draggedID;

        target.ondragenter=handleDrag;//针对释放区的事件dragenter:当被拖动元素进入释放区所占据的屏幕空间时触发
        target.ondragover=handleDrag;//针对释放区的事件dragover:当被拖动元素在释放区内移动时触发
        function handleDrag(e) {
            e.preventDefault();//因为dragenter和dragover事件的默认行为是拒绝接受任何被拖放的项目，故要阻止默认行为
        }

        target.ondrop=function(e){//针对释放区的事件drop:当被拖动元素在释放区里放下时触发
            var newElem=document.getElementById(draggedID).cloneNode(false);
            target.innerHTML="";
            target.appendChild(newElem);
        }
    </script>
</body>
</html>
```

```
e.preventDefault();//drop事件默认为有可能会是些出人意料的事情，故要阻止。
}

src.ondragstart=function(e){//针对被拖动元素的事件dragstart:元素开始被拖动时触发
draggedID=e.target.id;
e.target.classList.add("dragged");
}
src.ondragend=function(e){//针对被拖动元素的事件dragend:拖动操作完成时触发
var elems=document.querySelectorAll(".dragged");
for (var i=0;i<elems.length;i++) {
elems[i].classList.remove("dragged");
}
}
src.ondrag=function(e){//针对被拖动元素的事件drag:在元素被拖动时反复触发
msg.innerHTML=e.target.id;
}
}

</script>
```

方法二：使用模块模式

```
var DragDrop=function(){
var dragging=null;
var diffX=0;
var diffY=0;

function handleEvent(event) {
event=EventUtil.getEvent(event);
var target=EventUtil.getTarget(event);

switch (event.type) {
case "mousedown":
if (target.className.indexOf("draggable")>-1) {
dragging=target;
diffX=event.clientX-target.offsetLeft;
diffY=event.clientY-target.offsetTop;
}
break;
case "mousemove":
if (dragging !==null) {
dragging.style.left=(event.clientX-diffX)+"px";
dragging.style.top=(event.clientY-diffY)+"px";
}
break;
case "mouseup":
dragging=null;
break;
}
}

return {
enable:function(){
EventUtil.addHandler(document,"mousedown",handleEvent);
EventUtil.addHandler(document,"mousemove",handleEvent);
EventUtil.addHandler(document,"mouseup",handleEvent);
},
disable:function(){
EventUtil.removeHandler(document,"mousedown",handleEvent);
EventUtil.removeHandler(document,"mousemove",handleEvent);
EventUtil.removeHandler(document,"mouseup",handleEvent);
}
}
}();

DragDrop.enable();//开启拖放
```

5、在Javascript中什么是伪数组？如何将伪数组转化为标准数组？

伪数组：无法直接调用数组方法或期望length属性有什么特殊的行为。但仍可以用遍历真正数组的方法遍历它们。

例如：函数argument参数；再如调用document.getElementsByTagName，Element.childeNodes等，返回NodeList对象都是伪数组。

转化方法：Array.prototype.slice.call(fakearr)

```
var fakearr=document.getElementsByTagName("div");

var arr=Array.prototype.slice.call(fakearr);

arr.push("red");
console.log(arr);
```

6.JavaScript中callee和caller的作用？

caller:函数对象属性。该属性保存在调用当前函数的函数的引用。如果是在全局作用域中调用当前函数，它的值为null。

```
function outer() {
inner();
}

function inner() {
console.log(inner.caller);
//或console.log(arguments.callee.caller)用以实现更松散的耦合
}

outer();//function outer() {inner();}
```

callee:返回正在被执行的function函数。例如计算斐波拉切数列：

```
function fibonacci(num) {
  if (num<=2) {
    return 1;
  }
  else{
    return arguments.callee(num-1)+arguments.callee(num-2);
  }
}

console.log(fibonacci(8));
```

7.请描述一下cookies， sessionStorage和localStorage的区别

已整理过

8.JS手写快速排序

```
function quickSort(arr,low,high) {
  if (low<high) {
    var q=partition(arr,low,high);
    quickSort(arr,low,q-1);
    quickSort(arr,q+1,high);
  }
}

function partition(arr,low,high) {
  var i=low,j=high;
  var t=arr[i];
  while (i<j) {
    while (i<j&&arr[j]>=t) {
      j--;
    }
    if (i<j) {
      arr[i]=arr[j];
      i++;
    }

    while (i<j&&arr[i]<=t) {
      i++;
    }
    if (i<j) {
      arr[j]=arr[i];
      j--;
    }
  }

  arr[i]=t;
  return i;
}

var myarr=[8,7,9,0,5,1,2,6,4,3];
quickSort(myarr,0,9);
console.log(myarr);
```

说明：JS数组名直接是指针，可以直接传递。

9.统计字符串“aaaabbbcccccdddfgh”中不同的字母个数，再统计某字母出现的最大次数。

```
//统计其中不同的字母个数
function countLetter(str) {
  var len=str.length;
  var n=0;
  var arr1=new Array();
  for (var i=0;i<len;i++) {
    if (str.indexOf(str[i])==1) {
      n++;
    }
  }
  return n;
}

//统计字母出现的最大次数
function countMostLetter(str) {
  var len=str.length;
  var arr1=new Array();//arr1存放不同的字母
  var arr2=new Array();//arr2存放对应字母出现的次数
  for (var i=0;i<len;i++) {
    if (str.indexOf(str[i])==1) {
      arr1.push(str[i]);
      arr2.push(1);
    }
    else{
      var j=arr1.indexOf(str[i]);
      arr2[j]++;
    }
  }

  var max=Math.max.apply(Math,arr2);
  return max;
}

var mystr="aaaabbbcccccddfgabchhhhaaaaa";
var re1=countLetter(mystr);
console.log(re1);//7
var re2=countMostLetter(mystr);
console.log(re2);//10
```

10.写一个function，清除字符串前后的空格。（兼容所有浏览器）

方法一：利用indexOf()和lastIndexOf(),写函数

```
function myTrim(str) {
  while (str.indexOf(" ")==0) {
```

```
        str=str.slice(1);
    }
    while (str.lastIndexOf(" ")==str.length-1) {
        str=str.slice(0,-1);
    }
    return str;
}

var myStr=" Hello world ";
console.log(myStr);
newStr=myTrim(myStr);
console.log(newStr);
```

方法二：利用indexOf()和lastIndexOf(),写添加到String原型中的方法

```
String.prototype.myTrim=function() {
    var thisStr=this;
    while (thisStr.indexOf(" ")==0) {
        thisStr=thisStr.slice(1);
    }
    while (thisStr.lastIndexOf(" ")==thisStr.length-1) {
        thisStr=thisStr.slice(0,-1);
    }
    return thisStr;
};

var myStr=" Hello world ";
console.log(myStr);
var newStr=myStr.myTrim();
console.log(newStr);
```

方法三：利用replace方法正则替换

```
function myTrim(str) {
    return str.replace(/(^s+)|(\s+$)/g,"");
}

var myStr=" Hello world ";
console.log(myStr);
var newStr=myTrim(myStr);
console.log(newStr);
```

12.前端需要了解的知识

各种浏览器都要测？用Karma

13.12款

<http://www.cnblogs.com/lhb25/archive/2011/07/18/testing-cross-browser-compatibility-tools.html>