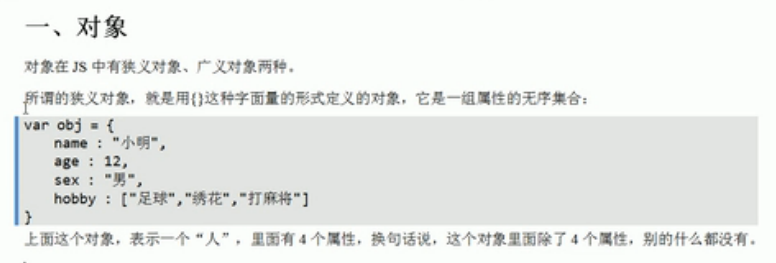
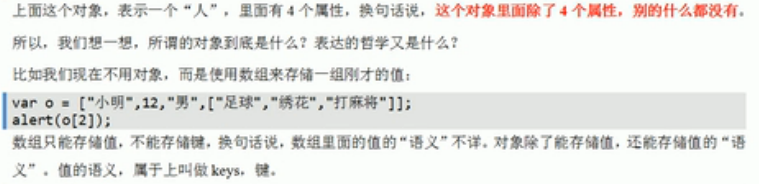
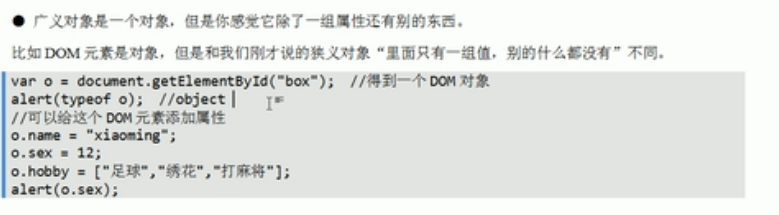
# 01认识对象



字面量创建的对象与new Object定义一样



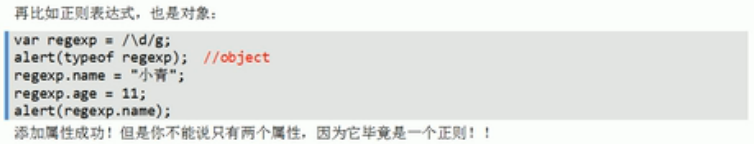


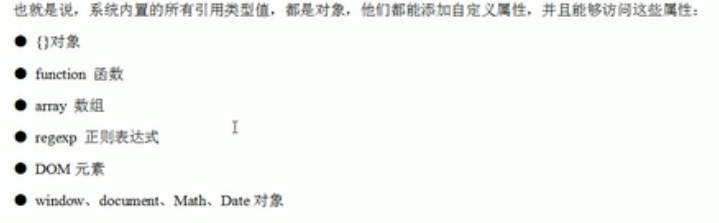




以下代码测试在node

function test(){}  
undefined  
typeof test  
'function'  
test.name='function'  
'function'  
test  
[Function: test]   
test.name  
'test'





对象都能添加自定义属性，除了基本数据类型（number string boolean undefined null）外所有的都是对象。

> reg=/box/ig

/box/gi

> reg

/box/gi

> reg.name='regexp'

'regexp'

> reg

{ /box/gi name: 'regexp' }

> reg.test('this is a box')

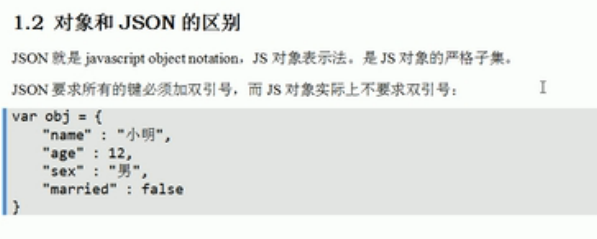
true

基本类型的包装类能够加属性，是对象；但是基本数据类型不能加属性，然而尽管字符串在java里面是对象，但是在js里面它不是对象，不能加属性。

尽管这些基本类型不能加属性，但是如果强行写属性表达式也不会报错。



# 02 json和对象



JSON是javascript对象表示法，是js的严格子集。

在js中，定义json时键名不需要加双引号，但是不符合js标识符命名规范的键（如1a）需要加双引号（不能是单引号）,且访问这些属性需要使用[]访问而不能是.访问。

变量值key也需要用方括号枚举。

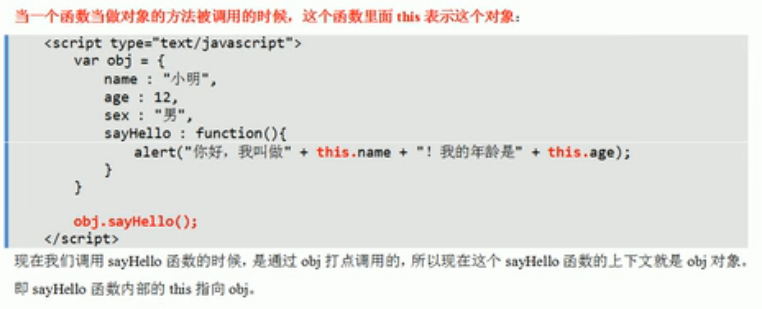


# 03 对象的方法

函数的上下文是什么取决于函数怎么被调用，而不是函数如何定义。



当一个函数当做对象的方法被调用时，上下文在这里指的是函数的this，而它指代的是这个对象



> var obj={

... name:"小明",

... age:12,

... sex:"男",

... sayHello:function(){

..... console.log("name: "+this.name);

..... }

... }

undefined

> obj

{ name: '小明', age: 12, sex: '男', sayHello: [Function: sayHello] }

> obj.sayHello()

name: 小明

undefined

> obj.name="jack"

'jack'

> obj.sayHello()

name: jack

> fn=obj.sayHello //当将函数赋值给另一个函数fn时则fn()相当于是window.fn()（浏览器中）

[Function: sayHello]

> fn()

name: undefined

undefined

但是即使函数写在对象里面，其上下文也不一定是这个对象。



# 04 函数的上下文-规律1

规律1：函数用圆括号调用，函数的上下文是window对象(因为这相当于window.函数)

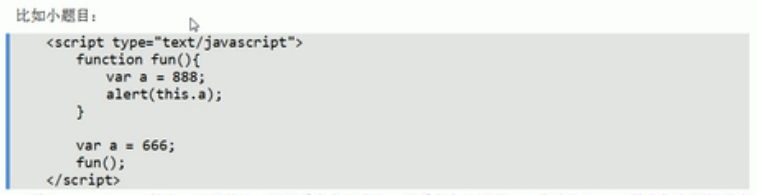
规律2：函数如果作为一个对象的方法，对象打点调用，则函数的上下文是这个对象

规律3：函数是事件处理函数，函数的上下文是触发这个事件的对象

规律4：定时器调用函数，上下文是window

规律5：数组中存放的函数被数组索引调用，this指的就是存放这个函数的数组





所有的全局变量都是window对象的属性，而函数里面的局部变量，不是任何对象的属性只是局部变量，不属于任何对象（所以上面的this.a访问的不是函数内部的局部变量）



function outter(){

var a='outter';

console.log(this.a);

function inner(){

var a='inner';

console.log(this.a);

}

inner();

}

console.log('window.a:',this.a);

outter();//内层和外层的this在这里都是window

05 函数的上下文-规律2



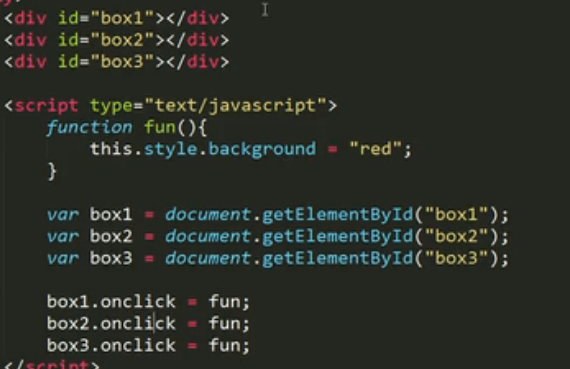
这也印证了上面fn=obj.sayHello后，fn()执行时访问的是window的属性；因为那相当于将函数赋值给window的对象的fn属性。





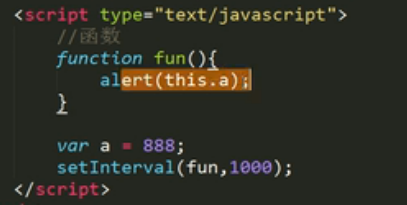
06 函数的上下文-规律3





04 函数的上下文-规律4







<div id="box1">box1</div>

<script>

var obj={

a:1000,

output:function output(){

console.log(this);

}

}

obj.output();//obj

var a=100;

// setInterval(output,1000);

setTimeout(obj.output,1000);//window

var box1=document.getElementById('box1');

box1.onclick=function(){

setTimeout(function(){console.log(this);},1000);//window

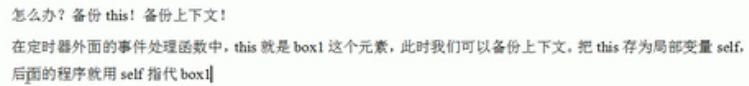
}

</script>



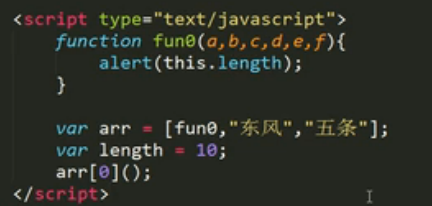
直接调用指定的dom对象或者备份上下文（使用其他非关键词，that=this或者self=this，总之你不要使用this）





08 函数上下文-规律5

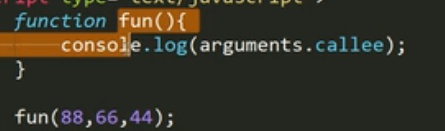




因为数组是object,索引访问调用类似对象.方法



09 函数的上下文小考题



在函数内部，如果想得到函数自身，用this是不可能的（原因见上面5个规律），我们必须使用arguments.callee。



function cc(a,b,c){

}

console.log(cc.length);//函数**形参**的个数等价于arguments.callee.length

console.log(arguments.length);//指的是**实参**的个数

function fun(){

console.log(arguments);

console.log(arguments.callee);

console.log(arguments.length);//实参

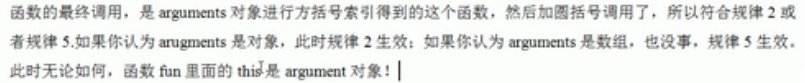
console.log(arguments.callee.length);//形参

console.log(fun.length);//形参

}

fun(1,2,3,4);





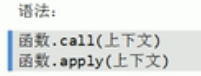
10 call()和apply()



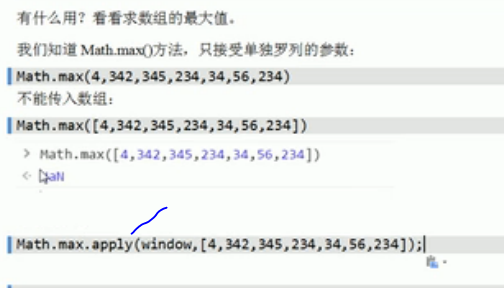
对象冒充，改变函数执行时的上下文。只有函数才能调用call和apply方法（函数的方法）

console.log('call' in cc);//true

console.log('apply' in cc);//true

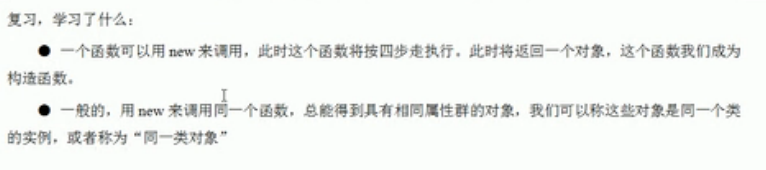






Apply接受数组作为参数并将数组参数的元素作为各个单独的参数解包



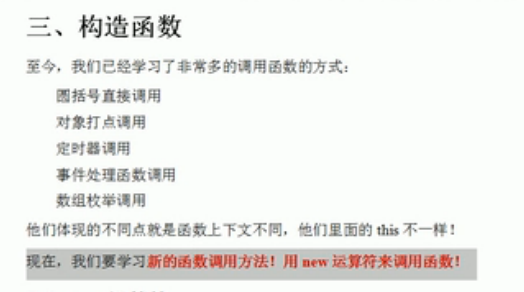


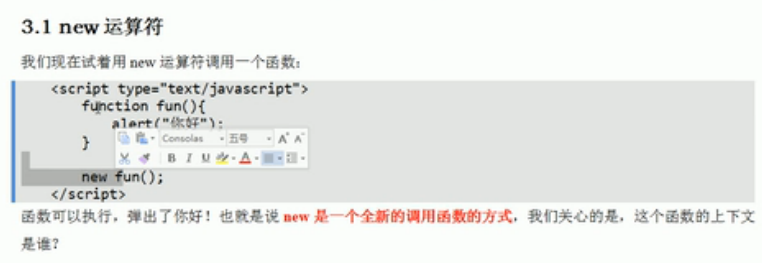
New关键字调用函数：

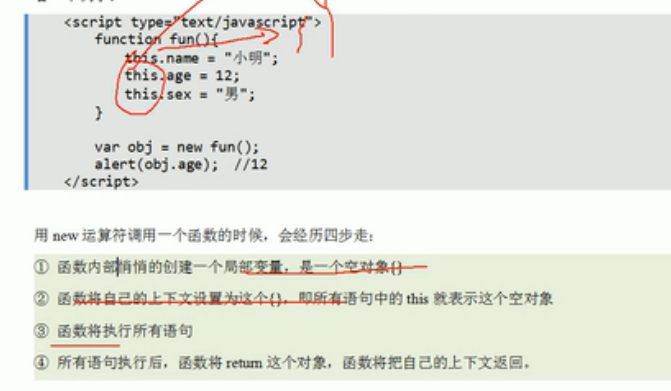
1. 首先创建一个新的对象，
2. 然后将当前函数执行的上下文设置为这个对象
3. 并在最后返回这个对象。

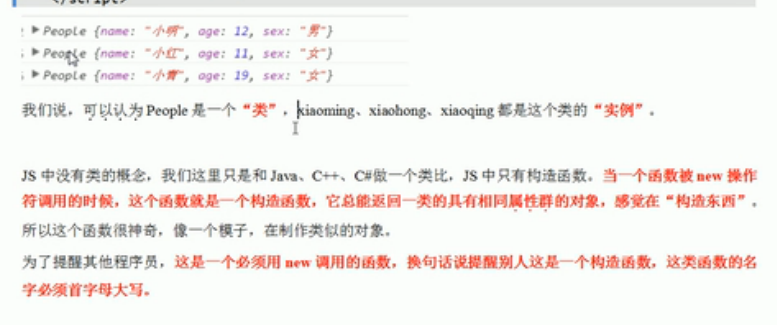
New 就是运算符，不要太有类的概念。













//===========================20170606