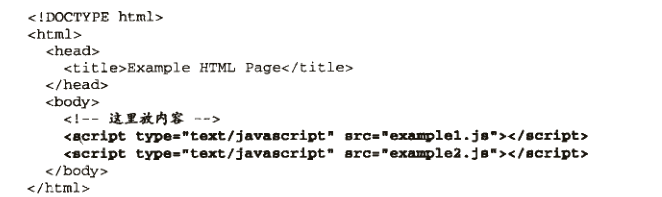
# 1 <script>

## 1.1 标签位置

body中放内容，内容后面放<script>，但<script>还是被包含在body标签中，如图1。

之所以不放在head中，是因为浏览器先解释head标签再解释body标签，如果放在head中，会导致script全部加载完才渲染内容。

图1



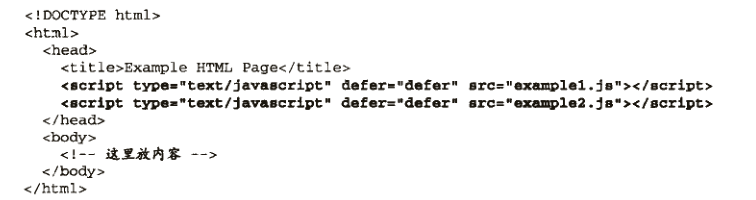
## 1.2 延迟执行脚本defer

script标签加上defer后，可以在</html>解释完再执行脚本，如图1。

现实中，因为第一个延迟脚本不一定会先于第二个延迟脚本执行，所以一个页面最好只包含一个延迟脚本。

defer只适用于外部脚本，内部脚本的defer会被忽略。

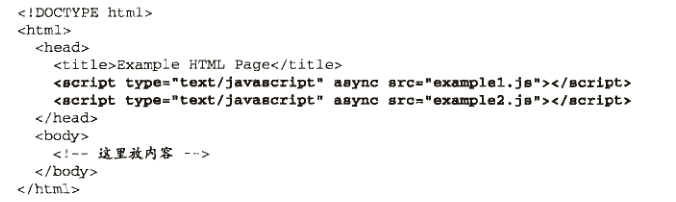
图1



## 1.3 异步脚本 async

异步脚本一定会在页面的load事件前执行，使用如图1。

图1



# 2 变量

## 2.1 范围

2.1.1 函数中用var定义的变量在函数外就无法用了，如图1。但如果省略var，就相当于定义了一个全局变量，如图2。有种例外，如果在if或for中用var定义的变量，会在if和for所处的作用域中也能被使用，如图3和图4。

图1

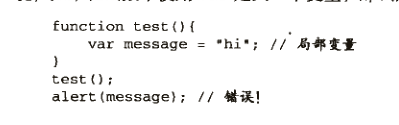


图2

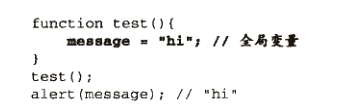


图3

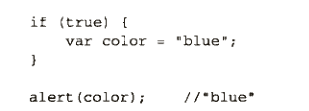
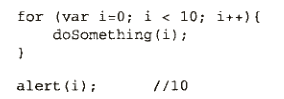


图4



## 2.2 变量初始值

2.2.1 用var声明的变量如果在定义时未被初始化，值是undefined，如图1，等价于图2。

图1

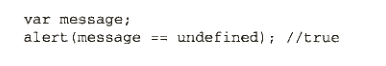
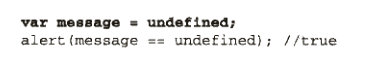


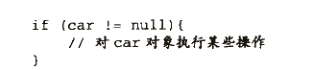
图2



## 2.3 null

如果一个变量准备在后面要保存其他的值，则最好初始化成null，便于后面进行空判断。

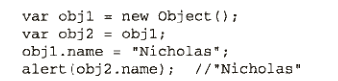
图1



## 2.4 变量之间的赋值

2.4.1 引用之间赋值，改变一个变量会影响另一个变量，如图1，因为2个引用变量指向的是同一个对象。

图1



## 2.5 变量为什么可以调用方法？

变量本身不该有方法，方法是属于对象才有的，因此变量在调用方法时，进行了一系列的后台操作，如图1的操作实际上是创建了对象String的实例，调用实例的substring方法，然后销毁这个实例，如图2。

有后台自动为基本类型创建的基本包装类型，包括String，Boolean和Number，这个对象实例的生存周期只有一行代码的执行瞬间，然后立即销毁，如图3。

图1



图2

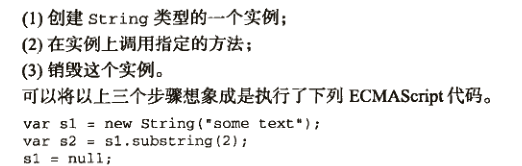
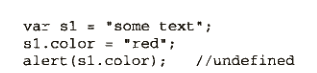


图3



# 3 对象 Object

## 3.1 自带方法

3.1.1 hasOwnProperty：检查某属性在对象实例（不是原型）中是否存在。

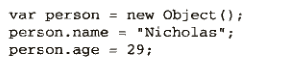
图1



## 3.2 构造方法

### 3.2.1 new

图1



### 3.2.2 {}，推荐，当参数个数很多，且不一定每一个都有值时，可以用图3的写法。

优点：参数很多时，不需要去记住参数的顺序。

图2

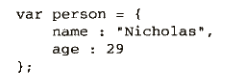
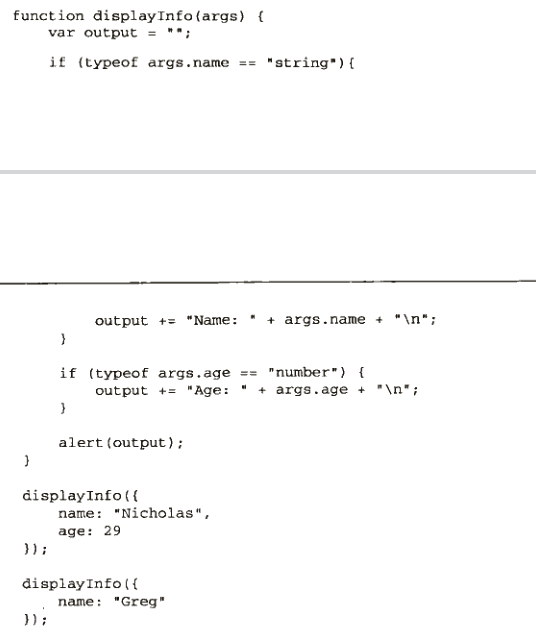


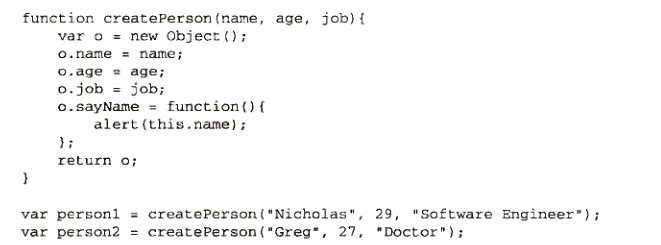
图3



### 3.2.3 工厂模式

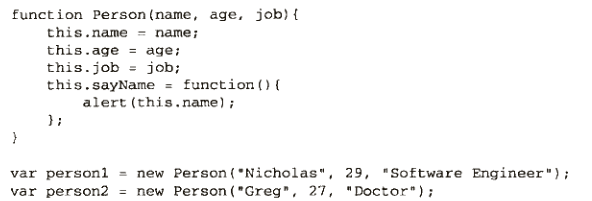
工厂提供一个方法来构造对象，这样的好处是构造对象时不用重写很多代码，缺点是还是不能区分多个对象的区别。

图1



### 3.2.4 自定义构造函数

图1



## 3.3 defineProperty

defineProperty可以修改对象默认的属性，如图1，第一个参数是对象，第二个参数是属性的名字，第三个参数是描述符，这个描述可以修改属性的一些特性。

第三个参数也可以用get或set方法的集合，如图2。

图1

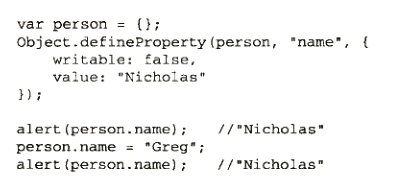
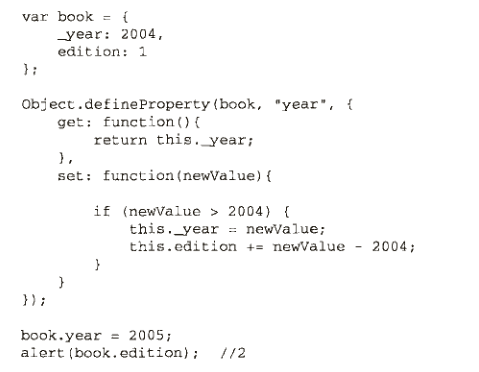


图2



## 3.2.5 组合使用构造函数和原型对象

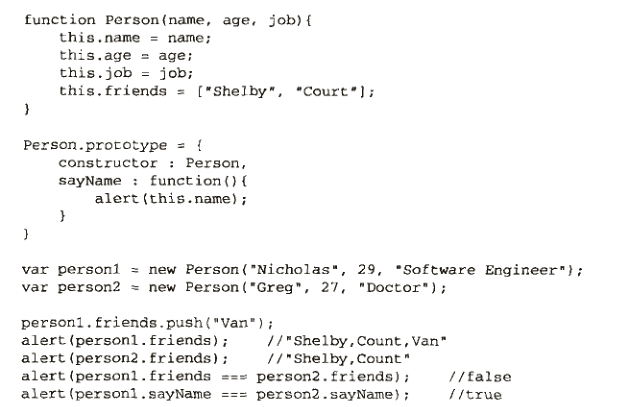
构造函数用来定义实例属性，原型对象用来定义方法和共享的属性。

优点：1.因为无论多少个实例p1、p2、p3都是共用一个原型对象Person prototype，当其中一个实例p1修改了原型中的count值（p1.prototype.count++），p2和p3也会读到修改后的值（p2.count、p3.count），所以原型对象建议用来定义共享的属性。

2.原型对象中定义方法不会造成上面的问题，唯一的好处就是节省空间，因为既然作用都是一样的，如果用构造函数定义方法，p1.sayName()和p2.sayName()不一样且存了2份，浪费空间，但作用是一样的。放在原型对象中只存了一份。所以方法建议在原型对象中定义。

3.实例属性建议放在构造函数中，因为如果放在原型对象中，可能会被其他实例修改。

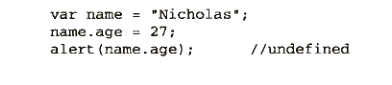
图1



# 4 不能给基本类型的变量动态添加属性

## 4.1 不能给基本类型的变量动态添加属性，只能给引用类型的变量动态添加属性，如图1。

图1

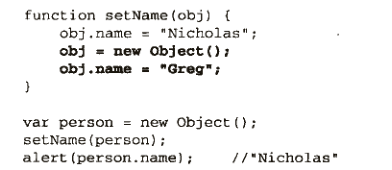


# 5 函数

## 5.1 入参是对象

如图1，当入参是对象时，传给obj的是一个地址，这个地址指向一个对象，当obj被new时，赋予了另一个地址，这个地址指向了另一个对象。所以obj被new后修改，不会影响原来的对象。

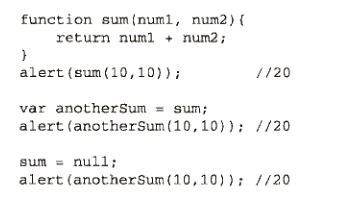
图1



## 5.2 函数是对象，函数名是指针

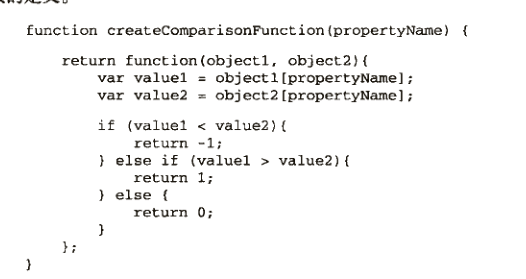
根据这个道理，多个函数名可以指向同一个函数，如图1。

图1



## 5.3 在一个函数中返回另一个函数（函数嵌套）

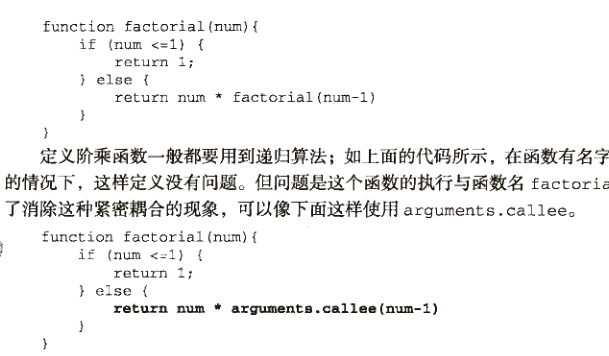
图1



## 5.4 arguments callee caller

arguments表示函数所有的入参，callee指向这个函数。

图1



## 5.5 this

函数内部的this值指向函数执行时的环境对象。

注意：是函数执行时的环境对象，不是调用函数的那个人，如图2，person.growOld调用者是person，环境是window，console出来的this值是window而不是person。

注意：构造函数的this是实例本身，其他函数的this是执行的环境。

图1

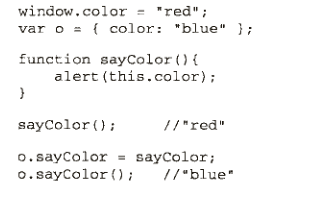
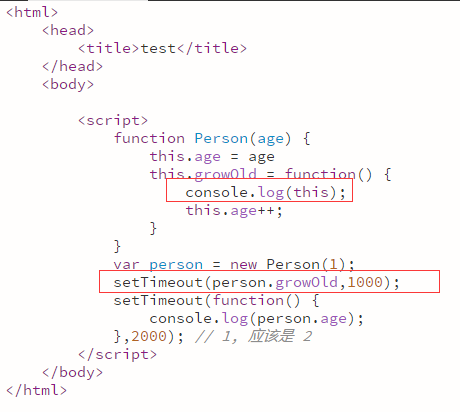


图2

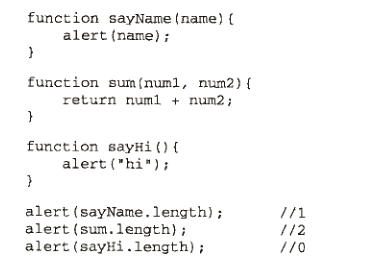


## 5.6 length

因为函数是一个对象，length就是这个对象中的其中一个属性，另一个是prototype。

length表示函数的入参个数。

图1



## 5.7 apply call

apply函数接收2个参数，第一个是函数运行的作用域，第二个是参数数组。

每一个函数都有apply函数。

此时的this是window对象，因为callSum1和callSum2的调用环境是window。

call方法和apply差不多，区别是参数要一个一个拆出来，如图2。

apply和call最大的好处不是传递参数，而是修改作用域，如图3。

图1

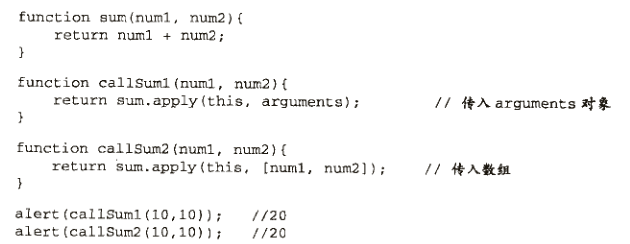


图2

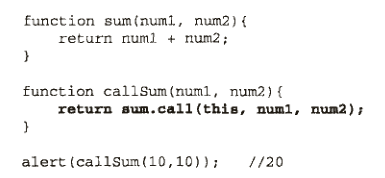
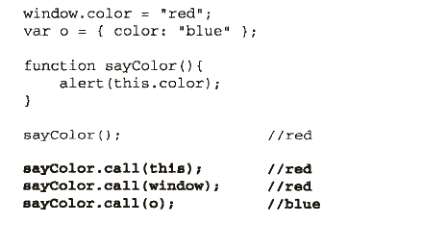


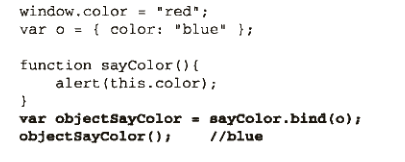
图3



## 5.8 bind

bind是在apply和call的修改作用域的基础上，再返回一个函数的实例。

图1

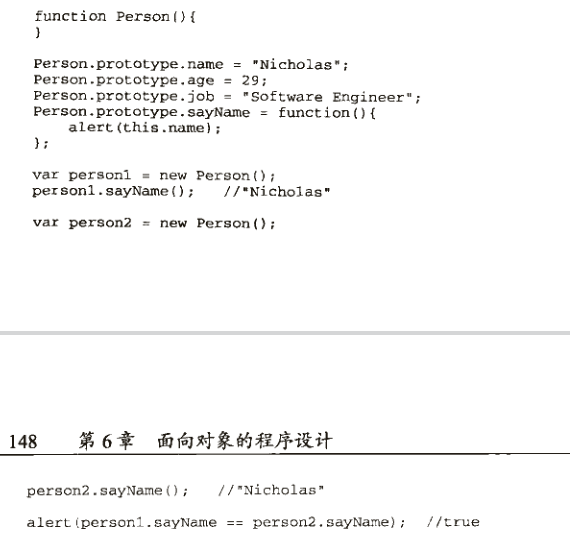


## 5.9 prototype

prototype是函数的另一个属性，其中一个属性是length。

因为函数是对象，当把函数看成对象时，prototype包含了所有实例共享的属性和方法。

图1



## 5.10 对象和对象的实例们和对象原型的关系

如图1，函数是一个对象Person，可知函数Person有一个属性prototype，这个属性指向原型对象Person Prototype。

原型对象Person Prototype默认有一个construtor属性指向对象Person，还有一些原型后面添加的属性。

对象的实例们person1和person2不包括属性和方法，只有一个属性指向原型对象。

当调用实例的属性和方法时，比如person1.sayName()时，先问“实例person1有这个方法吗？”，显然没有；然后会问“实例person1的原型有这个方法吗？”，有，然后返回这个方法。

注意：如图2，当实例p1读取属性时，的确像上面所说的，如果在实例p1中找不到就会返回原型中的值。但如果是实例p1是写属性的时候，如：p1.count = p1.count + 1，”=”的左边是写属性，这时候就会在实例p1中直接创建一个属性count，而不是将原型中的属性count + 1。

图1

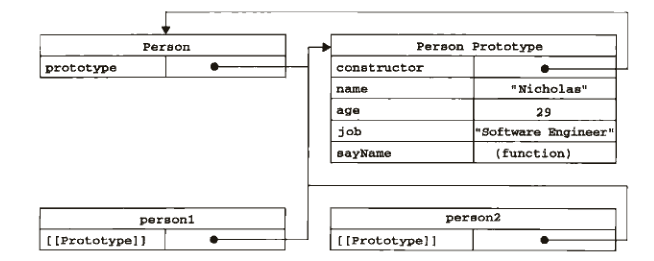
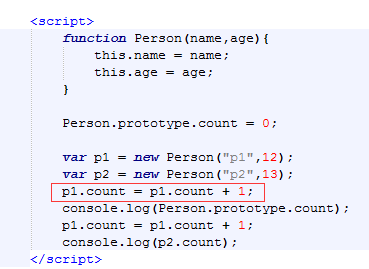


图2



## 5.11 hasOwnProperty in

hasOwnProperty可以用来检查某个属性是否存在对象实例中。

in操作符是无论属性在实例中，还是在原型中都返回true，如图2。

图1

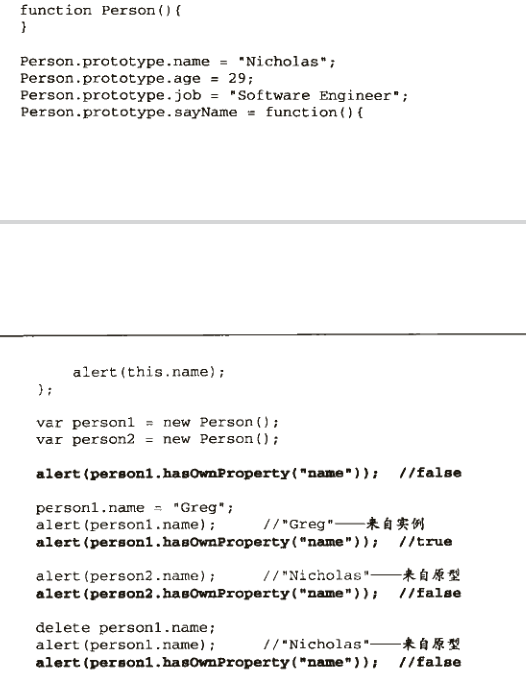


图2



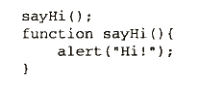
## 5.12 函数声明

### 5.12.1 function functionName(){xxx}

这种声明js会默认提升，并为这个函数的名字提供一个name属性。

因为这种声明方式会先读取函数声明，再读取函数调用，所以不会报错。

图1



### 5.12.2 匿名函数赋值给变量

这种是匿名函数（function关键字后面没有functionName），所以functionName.name属性是空字符串，并且不会提升。

图1

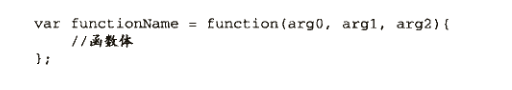
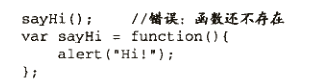


图2



## 5.13 函数递归

### 5.13.1 使用自己的函数名称递归

有缺陷，当如图2调用时，会报错，因为此时函数的引用就剩anotherFactorial。

图1

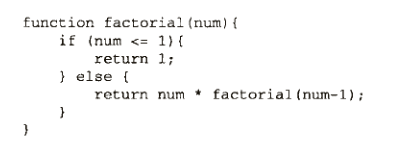
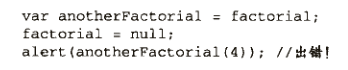


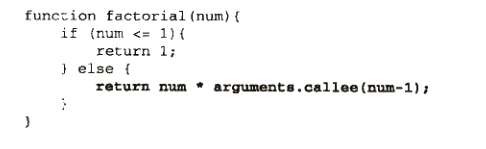
图2



### 5.13.2 使用arguments.callee递归

这样解决了使用固定的函数名导致的问题，因为callee指向本函数的函数名。

图1



## 5.14 原型属性 实例属性 区别

原型属性是在prototype中定义的，是所有实例共享的属性，如果实例f1修改了它的值，f2也能看到修改。

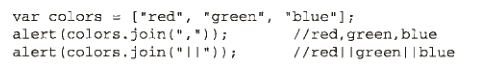
实例属性一般在构造函数中定义，是非共享的，实例f1不可能修改f2的实例属性的值。

# 6 数组 array

## 6.1 join

join后面接收一个参数，就是分隔符，如图1。

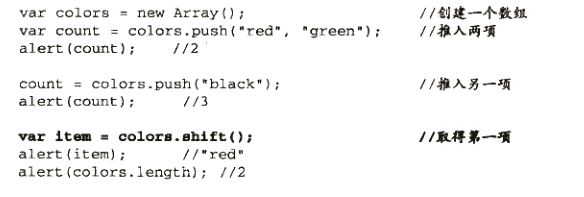
图1



## 6.2 shift

shift方法使数组模拟队列，将队列第一个元素移除，并且长度-1。

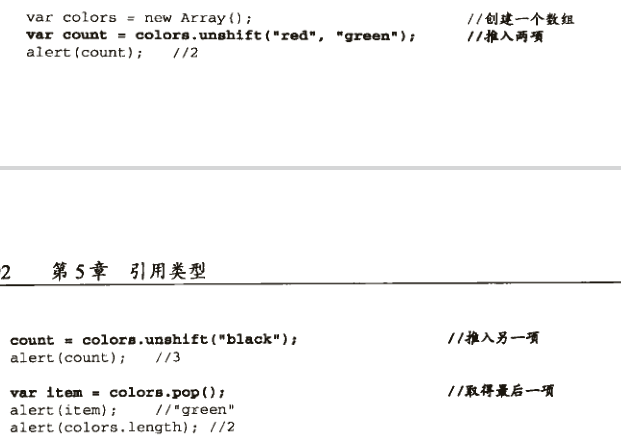
图1



## 6.3 unshift

shift是删除，unshift是添加，即在队列最前面添加任意元素，如图1。

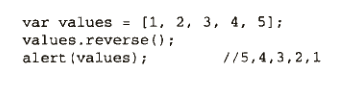
图1



## 6.4 reverse

reverse是使数组倒序，如图1。

图1



## 6.5 sort

sort会根据toString()方法进行顺序排序，所以会出现10排序在5前面的情况，如图1。可以自定义比较函数，纠正这种情况，如图2。

图1

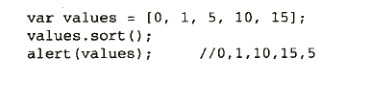
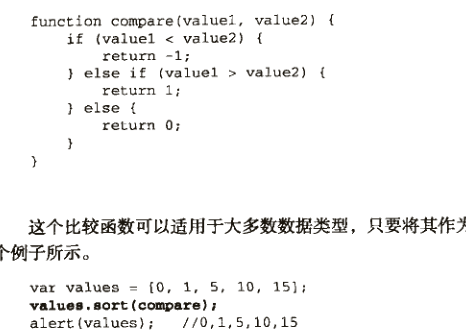


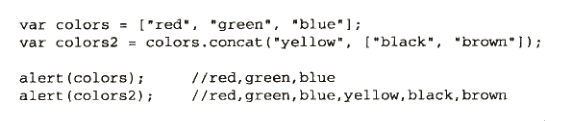
图2



## 6.6 concat

concat首先复制原数组的一个副本，然后将新元素添加到这个副本末尾，再返回添加后的新数组，如图1。

图1

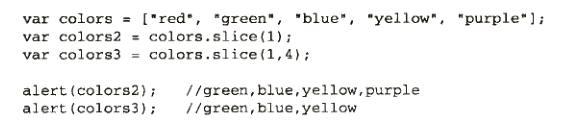


## 6.7 slice

slice当只有1个参数start时，会返回下标从start开始（包括start），一直到数组末尾的数组，也是返回数组的副本。

当slice有2个参数start和end时，会返回从start开始（包括start），一直到end-1（包括end-1，不包括end）的副本，如图1。

图1

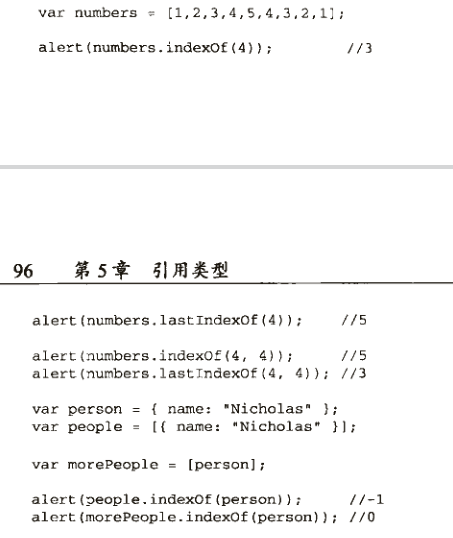


## 6.8 indexOf和lastIndexOf

indexOf从数组的第一个元素往后扫，lastIndexOf从数组的最后一个元素往前扫，如果找不到都返回-1，找到返回下标的值。

都接收2个参数，第一个是要查找的值，第二个是开始查找的起点位置的下标。

图1

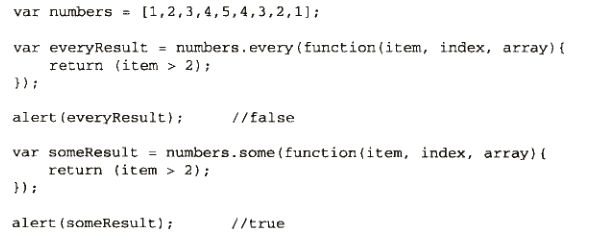


## 6.9 every some

every和some都用来遍历数组，让数组中的每一个元素都执行某个函数。

但every是每一个都返回true，才返回true；some是只要一个返回true，就返回true。

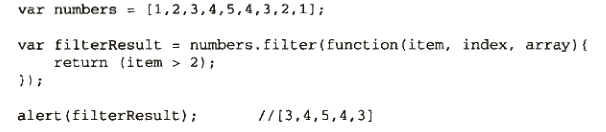
图1



## 6.10 filter

filter从数组中查询满足某个函数的数组，并返回这个数组，如图1。

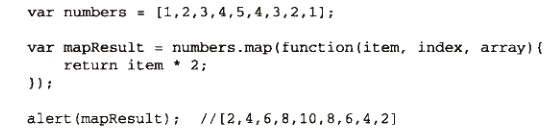
图1



## 6.11 map

map返回数组中的每个元素执行某个函数后的结果，返回这个结果数组，如图1。

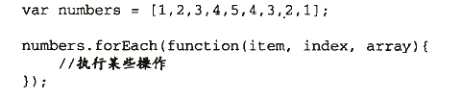
图1



## 6.12 forEach

forEach就是单纯的遍历执行某个函数，没有返回值。

图1

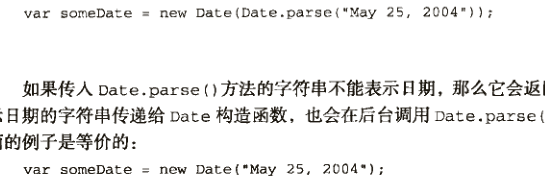


# 7 日期 Date

## 7.1 new Date和Date.parse

new Date可以传入毫秒数，如果传入字符串，自动先带哦用Date.parse将字符串转换成毫秒数。

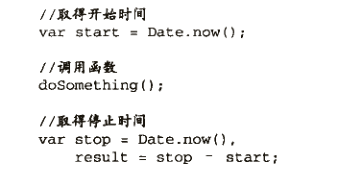
图1



## 7.2 now

now方法返回调用这个函数时的毫秒数，如图1。

图1



# 8 字符串 String

## 8.1 charAt charCodeAt

charAt接收一个下标，返回这个字符串中该下标位置的字符，如图1。

charCodeAt返回字符的编码，如图2。

图1



图2



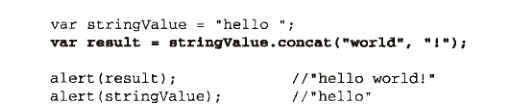
=

## 8.2 concat +

数组中也有concat方法，先复制副本再连接。

字符串中的concat方法也是一样的意思，不过在字符串中用+能达到相同的效果，而且更方便，如图1。

图1



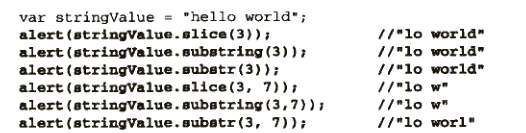
## 8.3 slice substring substr

数组中也有slice方法，接收1或2个参数，截取包含start，不包含end的子串。substring的小姑哦和slice一样。

substr的第二个参数是截取的长度，如图1。

这3个方法都不会影响原来的字符串。

图1



## 8.4 indexOf lastIndexOf

和数组的indexOf和lastIndexOf一样。

## 8.5 fromCharCode

fromCharCode是String的静态方法，接收1或多个字符编码，转换成一个字符串。

静态方法直接用String.调用。

图1



# 9 内置对象

## 9.1 Global

### 9.1.1 所有全局作用域中的属性和函数都是Global对象的属性

比如isNaN()，parseInt()，parseFloat()。

### 9.1.2 enCodeURI和enCodeURIComponent deCodeComponent

enCodeURI对整个URI编码，但不会对特殊字符（:，/，?，#）编码，但对空格等无效字符编码。

enCodeURIComponent对所有非标准字符编码。因此这个方法最好只对URI后面的字符串编码，因为会替换掉http后面的:和/。如图1。

用deCodeComponent可以解码。如图2。

图1

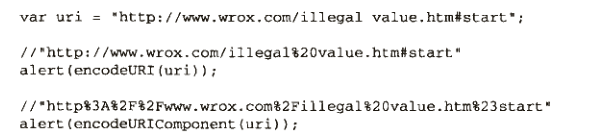
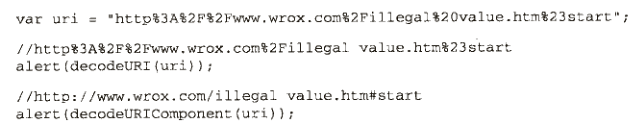


图2



### 9.1.3 eval

eval接收一个javascript的字符串，并执行这个javascript脚本，如图1。

sayHi虽然在eval内部定义，但因为eval在执行时会把这个拉出来当作javascript，所以能执行。

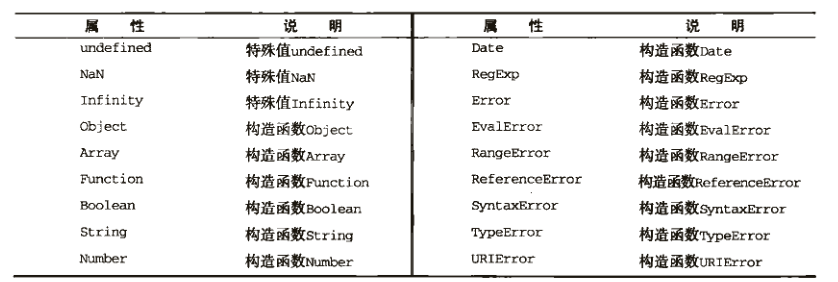
图1



### 9.1.4 Global对象的属性

如图1。

图1



### 9.2.5 获取Global对象

1. 用window对象取，如图1。
2. 用function返回this取，如图2。

图1

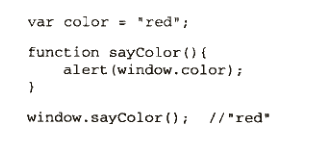
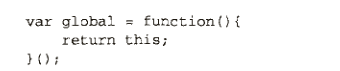


图2

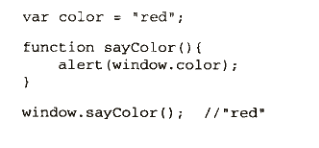


## 9.2 window

### 9.2.1 通过window访问Global

javaScript没有提供方法直接访问Global对象，但把Global对象当作window的一部分，直接用window访问。

图1



## 9.3 Math

# 10 继承

继承的目的是子类获得父类的属性和方法，可以重新实现父类的方法。

## 10.1 原型链

B的原型=A的实例。这样当调用B的实例的方法时，会先去B的实例找，然后去B的原型找，因为B的原型=A的实例，A的实例指向了A的原型，所以最后会去A的原型找。代码如图1，结构图如图2。

图1

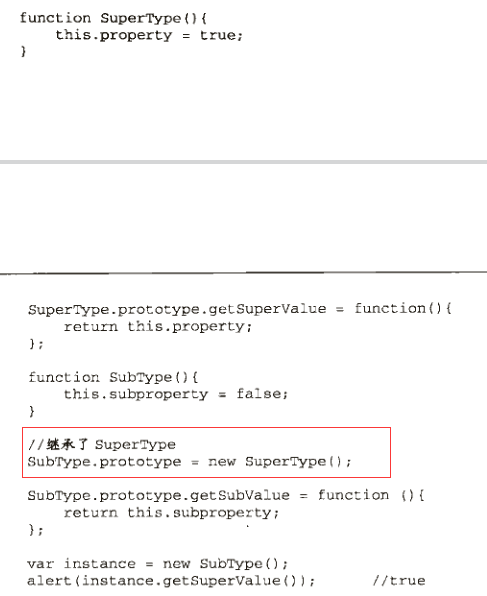
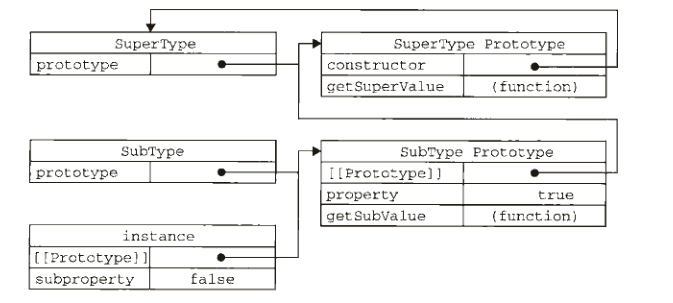


图2



## 10.2 子类构造函数调用父类构造函数

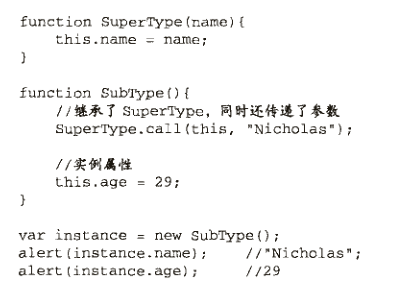
优点：可以向父类的构造函数传递参数，如图1传递了name的值；获取父类所有的实例属性，且可以将入参传给父类的构造函数。

缺点：在父类型中的原型中定义的方法，对子类不可见。

其实SuperType就是一个函数，可以想象成把SuperType.call拆成this.name = name，这个this就是子类的this，其实这样“借用”的效果就是为子类的实例添加了name属性而已。

注意：SuperType.call(this,”aaa”)中，因为父类的构造函数中定义的都是实例属性，不是原型属性，所以当在子类构造函数中调用父类构造函数时，实际上是为子类添加了父类所有的实例属性。如果父类有实例属性name和nickName，这句话就会为子类添加实例属性name和nickName。

图1



## 10.3 组合继承

组合继承就是使用原型链继承父类的原型的属性和方法，用借用构造函数来继承父类的实例属性。

图1

