# JavaScript技术栈

正如巴基斯坦和卡巴斯基，JavaScript和Java也没有半毛钱的关系。本来以来只要学点皮毛就可以了，后来看到网上鹅厂和网易他们的校招面试题，感觉这点皮毛的知识远远不够用，所以潜下心来系统地学习JavaScript。至于为什么选这本书《JavaScript权威指南》（简称‘犀牛书’）这本书，一方面是看到网上的推荐，另一方面是这本书够厚重^\_ ^

## 词法结构

### 字符集

* 使用Unicode字符集，它是ASCII和Latin-1的超集
* 区分大小写，不过html不区分，这个需要注意，虽然xhtml区分大小写
* 空格：普通空格符，水平制表符，垂直制表符，换页符，不中断空白，字序列标记，以及在Unicode中所有Zs类别的字符（Zs类别表示在Unicode中没有标志符号，但是不属于控制和格式字符的空格字符一类）；
* 行结束符：换行符，回车符，行分隔符，段分隔符。回车符+换行符被解析为一个单行结束符。

### 转义序列

为什么要用转义序列，这是因为一些老旧的电脑不支持Unicode字符集，所以JavaScript就定义了一种特殊的序列，“\u”+ 4个16进制数，表示任意一个Unicode字符。

### 标准化

Unicode允许以不同的方式对同一个字符进行编码，但会默认一个首选编码方式。而且还给出一个标准化处理不同编码方式的方法。但是JavaScript在解析的过程中会默认已经是标准化的编码格式，而不会去做进一步的标准化处理。

### 直接量

就是在程序中直接使用的数据值，包括数字，布尔值，正则表达式，字符串和空值。

### 标识符和保留字

1. 标识符必须以字母或者下划线或者美元符号开头，不允许使用数字开头
2. 下列关键字在普通的JavaScript代码中是合法的，但是在严格模式下是关键字：

implements, let, private, public, protected, yield, interface, package, static

### 可选的分号

1. 一般来讲，如果当前的语句与随后的非空字符不能当成一个整体进行解析的话，他就会在当前语句和随后的语句行结束符处填补分号；
2. 第一个例外，在return，break和continue后面换行后，不管后面能否连成一个可以解析的整体，都会被自动填补分号：

如：return

true

本意是 return true；但是实际上会被解析为return；true；

1. 第二个例外是当++或者--后面加上换行再接其他非空字符，那么JavaScript会把++或者--当成是后面一行的前缀；

如x

++

y

javascript就会解释为x; ++y;，而不会解析为x++；y;

1. 配置映射关系，注意属性name的值要和实体类中的属性名称要一致

<hibernate-mapping>

<class name="com.lvhongbin.bean.User" table="t\_User">

<id name="id" column="id">

<generator class="native"></generator>

</id>

<property name="name" column="name"></property>

<property name="password" column="password"></property>

<property name="email" column="email"></property>

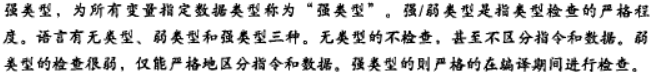
<property name="sex" column="sex"></property>

<property name="date" column="date"></property>

</class>

</hibernate-mapping>

### 强弱类型

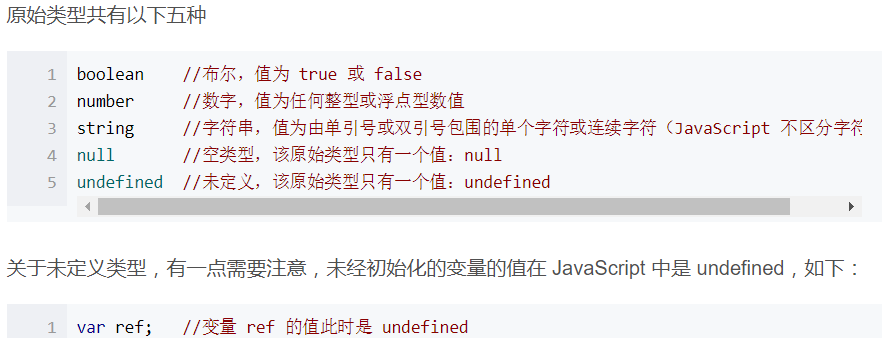


## 类型，值和变量

### 类型

1. 类型分为原始类型primitive type和对象类型object type。
2. 原始类型包括数字，字符串，布尔值，null和undefined；

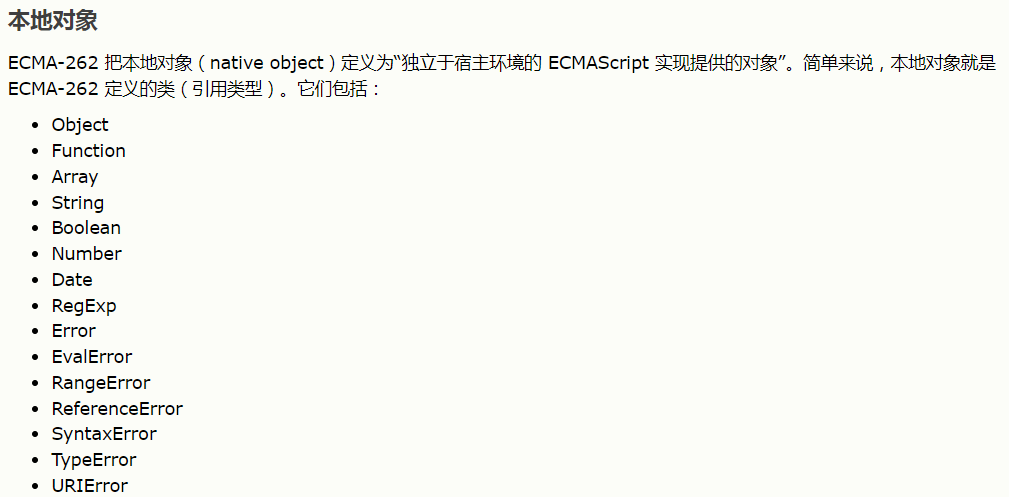
http://blog.csdn.net/qq\_20974467/article/details/49357269





1. 对象类型

首字母都是大写



1. 类型按照不同的划分方法还可以分为可变类型和不可变类型；

### 表达式

所谓的表达式就是—— 一个JavaScript的“短语”，JavaScript的解释器可以计算它，从而生成一个值。表达式可以分为以下三种：

1. 直接量，如1.7是数字直接量、"JavaScript权威指南"是一个字符串直接量等。
2. 变量直接量表达式的值就是这个直接量本身，变量表达式的值则是该变量所存放或引用的值。
3. 可以合并上面提到的“简单”表达式来创建为比较复杂的表达式。例如1.7是表达式，i是表达式，下面的代码所示的同样也是（也可以叫）表达式：

i+ 1.7

上面这个表达式的值是两个简单表达式（一个变量表达式和一个简单表达式）的和。在此例中，“+”是一个运算符，用于将两个简单表达式合并起来以组成一个复杂的表达式。“”

### 数字

1. JavaScript是不区分整数值和浮点数类型的，统一用64位浮点数类型表示。然而，对于数组索引和位操作符，却用的是32位整数。
2. 负号

负号不直接参与数字直接量的一部分，但它作为一元求反运算的符号使用

1. 可以使用十进制和十六进制

十六进制前缀为“0x”或者“0X”，八进制有的JavaScript可以实现，用0作为前缀，有的不可以实现，在ECMAScript6的严格模式下，八进制是严令禁止的

1. 指数计数法也是允许的；
2. 数字计算的溢出overflow，下溢underflow（所谓下溢就是所得的结果的绝对值比0大，但是在JavaScript中没有更小的数可以表达）或被0整除都不会有问题，overflow对应的是Infinity或者-Infinity，如一个非0值被0整除的时候，underflow对应的是0或者-0（这取决于是正下溢还是负下溢），0/0的时候返回NaN，not-a-number。
3. 在JavaScript中，其实预定了两个全局变量Infinity和NaN，分别表示正无穷大和非数字值。在过去ECMAScript3的时代，这两个变量是可读可写的，但是在ECMAScript5的时代，这两个变量就变成只读了。不过ECMAScript3的时代Number对象定义的属性值也是只读的。

如Infinity； //在ECMAScript3时可读可写，但是ECMAScript5时只可读；

1/0；//它是原始类型，数字直接量，可以通过new Number隐式转换为Number对象，该对象的属性值在ECMAScript3和ECMAScript5中都是只读的；

1. 比较奇怪的是，NaN的值跟谁都不相等，包括他自己，所以不能通过

来判断这个数是否是NaN，只能通过

来判断，当且仅当x=NaN的时候，上式返回true。其实isNaN()也是利用了同样的原理去判断。

1. 0和-0这对孪生兄弟有点特别，因为即使在严格模式下，他们两个都是相等的，有区别的是当他们当除数的时候，一个返回Infinity，另一个返回-Infinity。
2. 二进制浮点数和四舍五入错误

最后会返回false，原因是JavaScript采用的是二进制表示法来表示浮点数，可以精确表示1/2，1/8等分数，但是对于十进制分数比如1/10，1/100等就无法精确表示。

虽然情况是这样，但是对于计算来讲精度是足够的，差就差在比较和判断上。

1. 日期

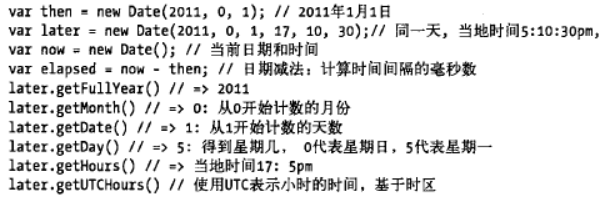


图2-1 日期的用法

1. Math对象和它的属性

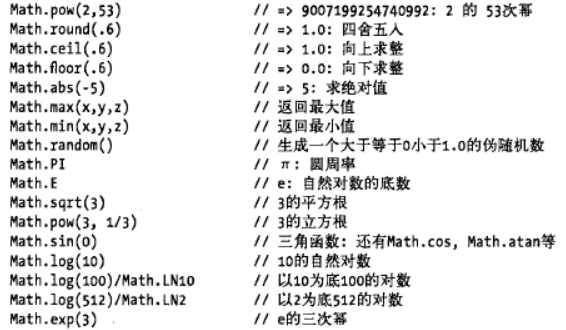


图2-2 Math对象和它的属性

### 文本

1. 字符串是由一组无符号两字节组成的不可变有序序列；
2. JavaScript采用的是UTF-16编码的Unicode字符集，每个字符都通过16位内码表示，并表示字符串中的单个字符。那些不能用16位内码表示的字符可以用两个16位内码组成的“代理项对”来表示。也就是说，有些Unicode字符可能占4个字节
3. 在字符串中当在一行写不下的时候，可以在行末尾添加反斜杠“\”,这样的话多行可以当作一行处理。
4. 字符串支持索引，如s[0]，这跟C有点像，字符串常量其实首字符的地址，然后通过索引来寻址。
5. 模式匹配RegExp()

RegExp()构造函数可以用来创建文本匹配对象，采用的是Perl的写法，第一种方式是直接通过/正则表达式/写出来，第二种方式是通过new RegExp('正则表达式')创建一个RegExp对象。

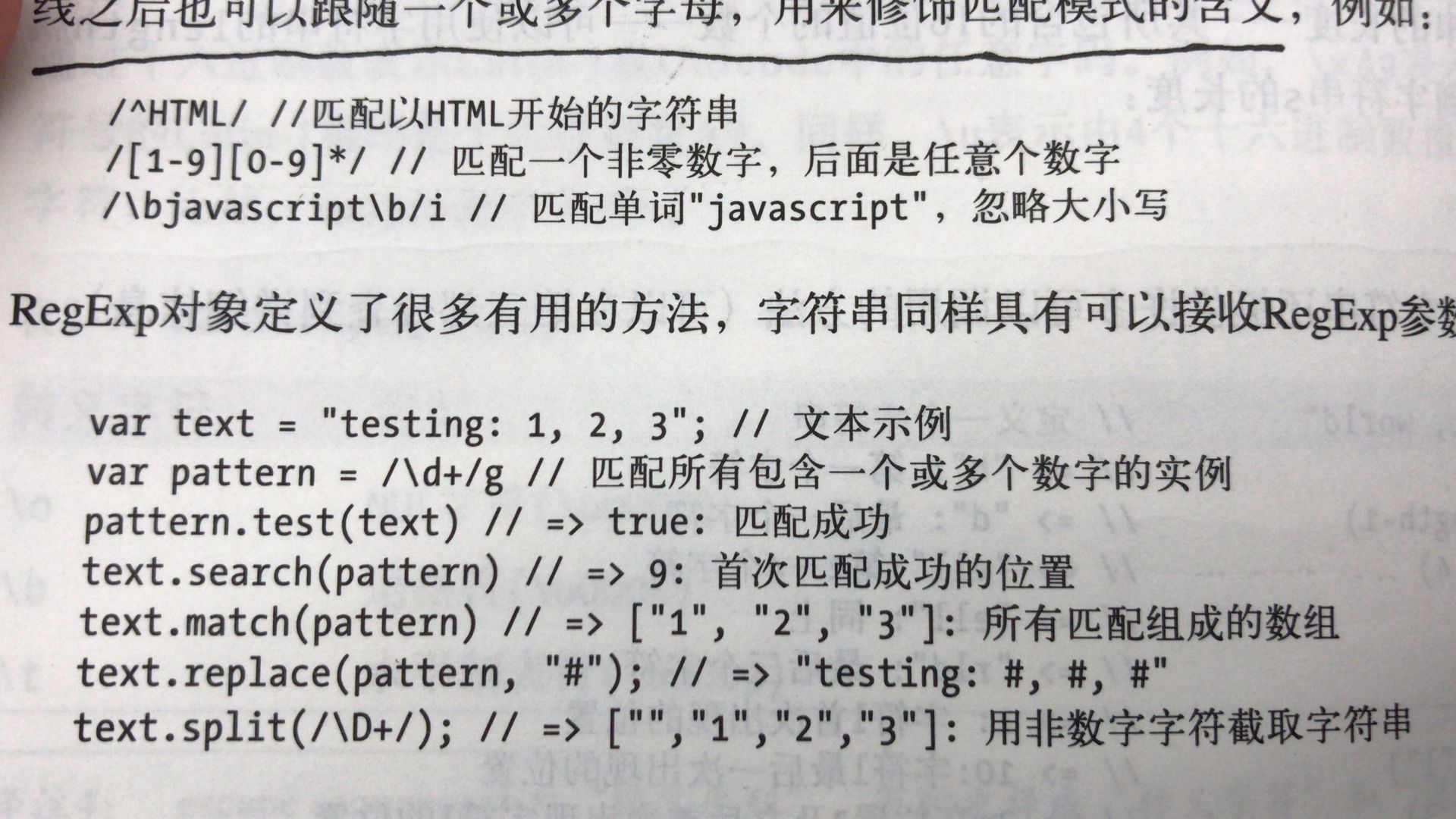


图2-3 RegExp()匹配

### 布尔量

1. 遇到undefined，NaN，0，-0，null和“”都会表示成false，其他的值，包括所有的对象，都表示成true。
2. undefined，全局变量，用typeof运算得出undefined，系统级的出乎意料的值的空缺，不包含任何属性和方法，它不是关键字，无法使用“.”或者“[ ]”;
3. null，对象，表示“非对象，”用typeof运算得出object，程序级的，正常的或者意料之中的值的空缺，如果想把它赋值给某个属性或者变量，最佳的是null，而不是undefined。null是关键字。

### 包装对象

1. JavaScript中的对象是一种复合值，他是属性和以命名值的集合；
2. 包装对象就是数字，字符串和布尔值的属性创建的临时对象，但她指示临时的，用完即销毁。



图2-4 包装对象的临时创建

结果t的值是undefined。

1. 可以通过new String()，new Number()和new Boolean()等构造函数的new方法显示创建包装对象。
2. 用“= =”可以判断得出原始值与其包装对象是相等的，而采用“= = =”可以判断得出原始值与其包装对象是不相等的

### 不可变的原始值和可变的对象引用

1. 原始值是不可改变的，但有时候我们可以看到一些字符串变量可以使用方法来返回一个新的字符串。比如返回一个全是小写的字符串，但这在JavaScript中是明令禁止的，之所有可以这样做，是因为它首先隐性地变成了一个包装对象，然后使用属性方法，返回了一个新的字符串。前后两个字符串的引用地址是不一致的。
2. 即使两个单独的对象一模一样，但是由于他们两个是单独的，地址不一样，采用“= = =”判断的时候他们就是不相等的，如果他们是通过指针传递而相等的，即使他们的值不一样，那比较的时候也是相等的。总的来说，对象的比较是应用的比较。
3. 类型转换

类型转换主要是一些隐式类型转换，转换的方向是往有可能得出有意义的值的方向。无意义的字符串转为NaN。

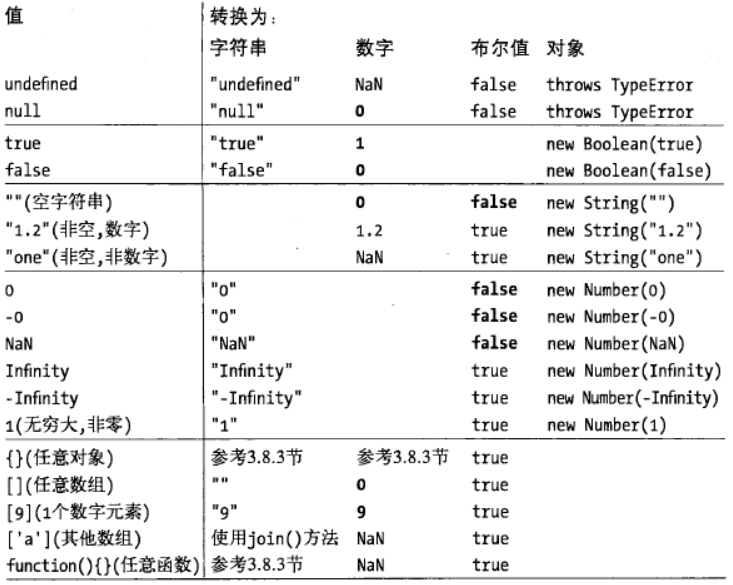
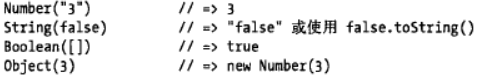


图2-5 隐式类型转换表

显式类型转换



“+”作用于一个字符串，则会把另外一个字符串也变成字符串，一元运算符“+”可以将操作数转换为数字，数字前加感叹号直接变成一个取反的布尔量，

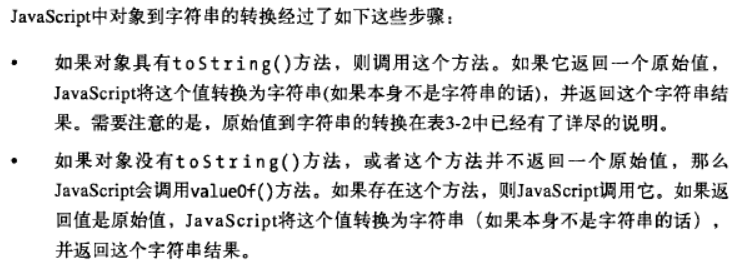


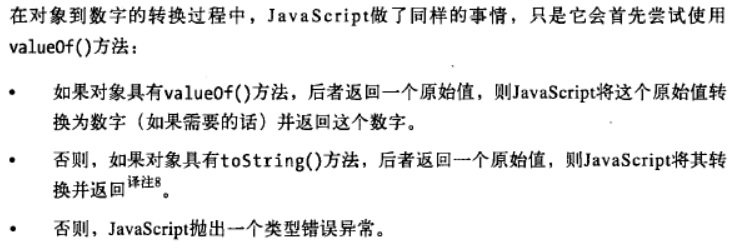


图2-6 显式类型转换

### 对象转换为原始值

1. 所有的对象类型（对象，数组和函数）继承了两种不同的方法：toString()和valueOf()，前者用来转字符串的，后者用来转数字的。
2. 细节的转换规则如下：



其实就是对象转字符串的时候，如果用toString()转不成功，就用valueOf()去转，变成数字直接量后再转成字符串。对象转数字的时候同理。

1. 对象转换数字的细节解释了为什么空数组会被转换为数字0以及为什么具有单个元素的数组会被转换为一个数字。数组继承了默认的valueOf()方法，这个方法返回一个对象而不是一个原始值,因此数组到数组的转换调用toString()方法。空数组转换成空字符串,空字符串转换为数字0。含有一个元素的数组转换为字符串的结果和这个元素转换字符串的结果一样。如果数组只包含一个数字元素,这个数字转换为字符串,再转换为数字。
2. valueOf()方法的任务并未详细定义:如果存在任意的原始值,它就默认将对象转换为它的原始值。对象时复合值而且大多数对象无法真正表示一个原始值,因此默认valueOf()方法简单地返回对象本身,而不是返回一个原始值。

var n=new Number(1);

console.log(n.valueOf());//输出1

var b=new Boolean(false);

console.log(b.valueOf());//输出false

var str=new String("str");

console.log(str.valueOf());//输出str

var n=new Number("a");

console.log(n.valueOf());//NaN

var b=new Boolean("str");

console.log(b.valueOf());//输出true

1. 和”= =”一样，”<”运算符以及其它关系算术运算符也会做到对象到原始值得转换，但要除去日期对象的特殊情形:任何对象都会先尝试调用valueOf，然后调用toString。不管得到的原始值是否直接使用，它都不会进一步被转换为数字或字符串。所以非日期类对象是无选择性的转换。
2. 日期类是JavaScript语言核心中唯一的预先定义类型,它定义了有意义的向字符串和数字类型的转换。所以日期类对象是有选择性的转换。

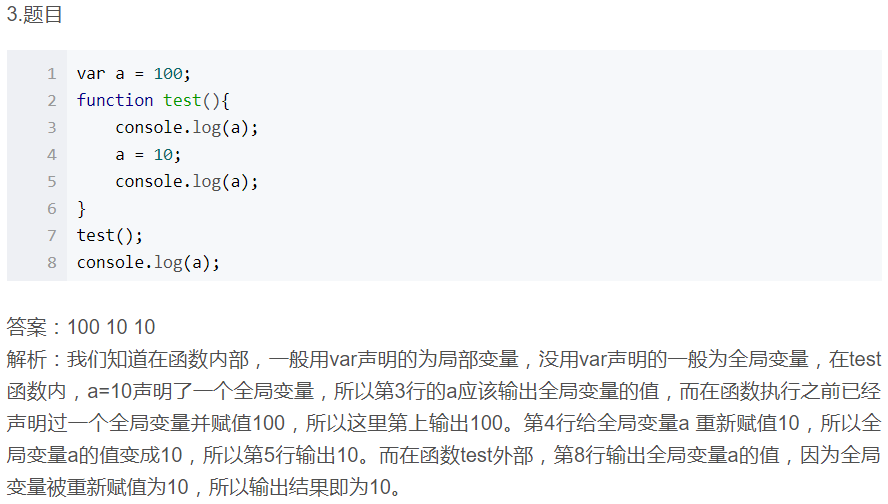
### 变量声明

1. 编程语言分成动态（类型）语言和静态（类型）语言，区别在于是否在编译期间检查数据类型。
2. 在ECMAScript5的严格模式下，如果不声明变量就直接复制就会报错，如果在不严格的模式下，则不会报错，它会自动给全局对象创建一个同名属性，而且它工作起来的时候像一个正确的全局变量。把变量变成全局变量
3. 变量作用域

“声明提前”在我看来就是在函数中局部变量的声明和赋值可以分开，甚至赋值可以放在声明之前。这叫“声明提前”，但是这么做有一个缺点，就是万一前面忘了赋值，后面只有声明，然后声明前面又有该变量的调用，那么该变量就会变成undefined。







### 全局变量

1. 全局对象用this表示，全局变量是全局对象的一个属性，创建一个全局变量的时候可以使用直接赋值法，或者用“this.变量名”赋值法
2. 只有全局变量才能使用delete 变量名来删除变量；
3. 作用域链；

## 表达式和运算符

### 数组表达式

1. 其初始化表达式又称为“数组直接量”；
2. 初始化的时候，如果没有值只用逗号隔开，则初始化为undefined



### 对象表达式

1. 其初始化表达式又称为“对象直接量”；
2. 属性名可以是字符串





### 函数定义表达式

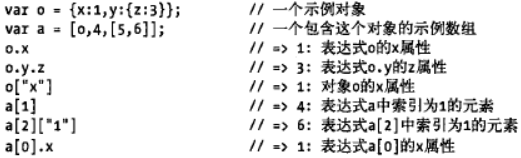
1. 其初始化表达式又称为“函数直接量”；



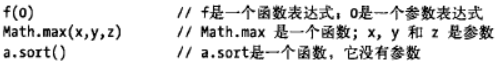
### 属性访问表达式

1. 可以用来取数组元素或者对象的属性；
2. 方括号法适用于对象访问和数组索引，但是·号法只能用于对象属性访问；
3. 方括号法和·号法在“[”或者“.”前面的值都会首先被计算，假如其值为undefined，则会抛出异常，因为undefined和null都没有属性值。





### 调用表达式



### 对象创建表达式

1. 使用new关键字，可以后面不加参数



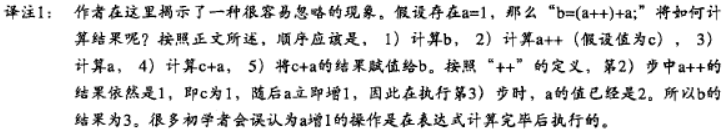
1. 其实创建的过程：首先创建一个空的对象，然后调用相应的构造函数，然后初始化新对象的属性

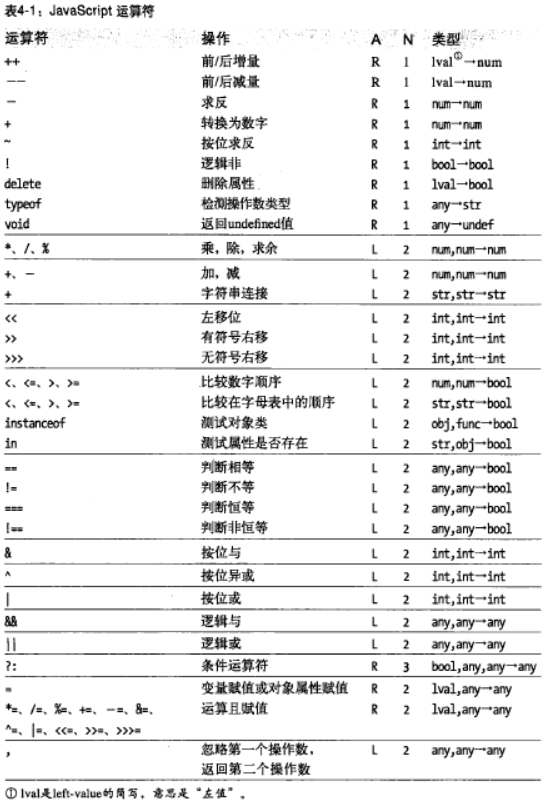
### 运算符概述

1. 左值

lval，这是一个比较古老的术语，只能出现在赋值运算符的左侧；变量，数组元素和对象属性都属于左值。

1. ECMAScript规范允许内置的函数返回一个左值，但是不允许自定义的函数返回一个左值。
2. 表达式先会计算等号左边的左值，然后再计算“右值”
3. 这说明了自加符号如果在后面，则第一次使用大的时候不自加，而在第二次使用前完成自加操作，为了避免出错，最后在同一条表达式中自加符号的值只使用一次。





位运算符会把NaN，Infinity和-Infinity都转化为0；

“>>>”无符号右移；

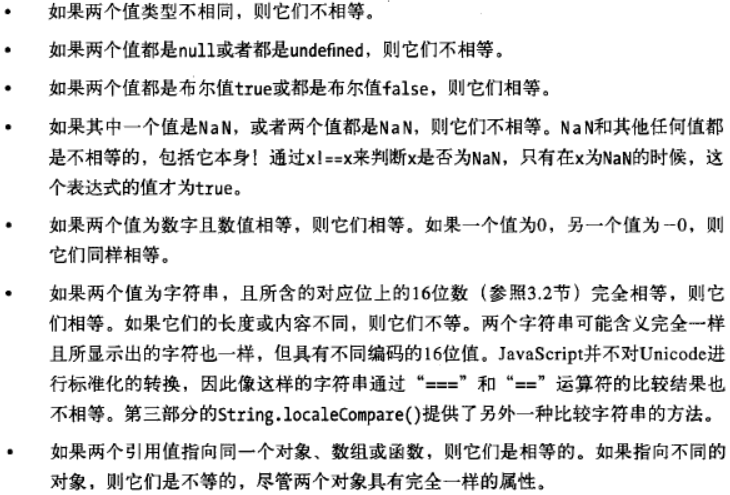
“>>”有符号右移，不超过本身的最大整数（负数同样适用）；

### 关系表达式

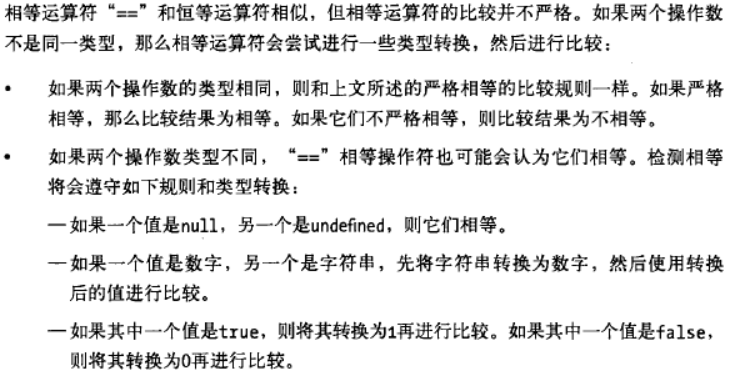
1. “= = =”严格相等

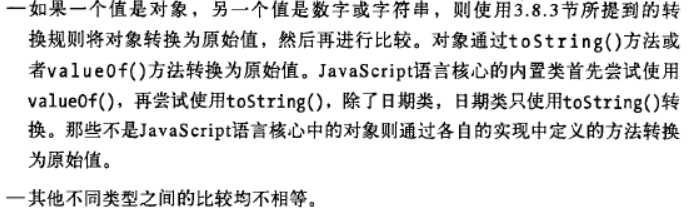
不允许进行类型转换，裸体进行相等比较；

字符串的比较看的是16位编码，而不是字符串的含义。



1. “= =”不严格相等

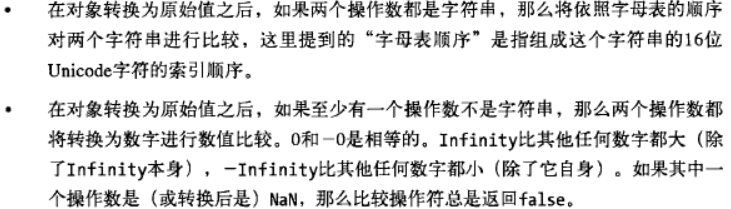




1. 比较运算符

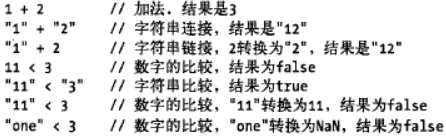
总的原则是转化为数字进行比较；

对象要转化为原始值才能进行比较；



字符串的比较是按照Unicode字符的索引顺序进行比较，，而且是区分大小写的，大写比小写的小。

任何与NaN的比较都返回false；

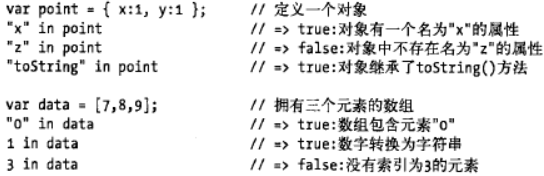


1. in运算符

这个运算符是用来判断某些指定的方法或者属性或者索引被指定的对象或者数组拥有：

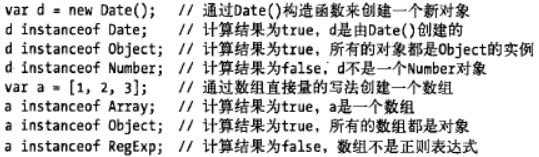
左边的操作数会被隐式转化为字符串；

数组，对象和函数（对象类型）都天然拥有toString和valueOf的方法



1. instanceof运算符

理解“原型链”prototype chain，“原型链”作为JavaScript的继承机制。



### 带操作的赋值运算

这两行代码的结果是不一致的，前者把数组的操作定义成是一种原子性的操作，“一荣俱荣”，可以等价成

data[i]= data[i]\*2；

i=i+1；

而后者可以等价成

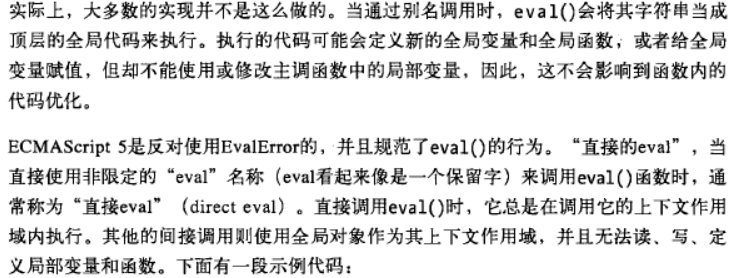
data[i]= data[i+1]\*2；

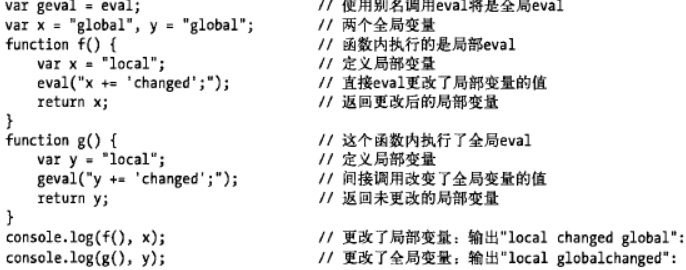
i=i+2；



### eval()

1. 如果传入的参数不是字符串，则返回该参数，如果是字符串，则把他当成代码来编译执行，如果编译成功则运行，否则返回语法错误（SyntaxError）异常，而且还可以调用它的变量作用域环境
2. 全局eval()

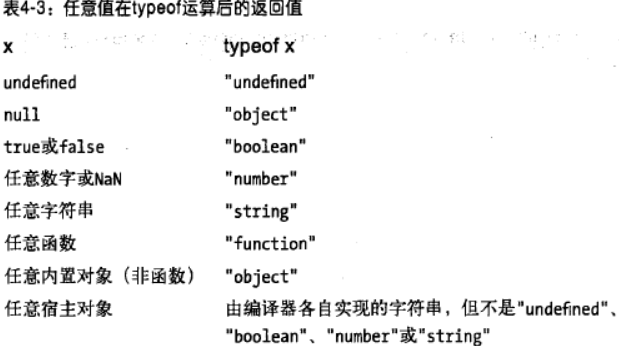




1. 实际上全局eval()是一个很好的特性，允许我们在局部作用域修改全局的变量，我们用的最多的也是这个特性。
2. 严格模式下的eval()

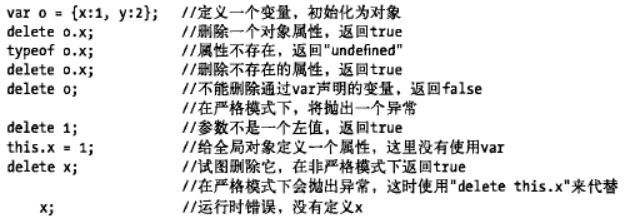
不允许使用别名，这时候的eval()更像是运算符，而且里面只能对局部变量进行查询和修改，但是不允许新建变量或者函数。

### typeof

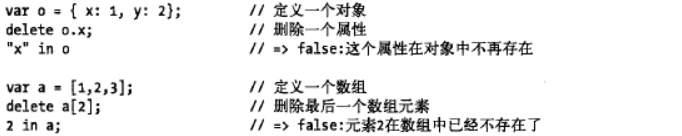


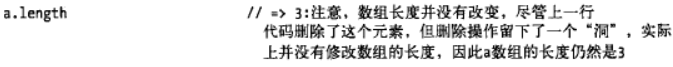
### delete

1. 右值必须是一个属性访问表达式，否则返回false，在严格模式下会报错；

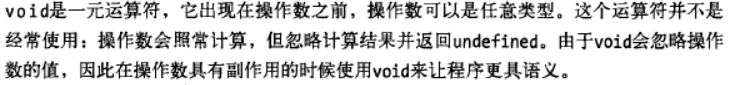


1. 数组某个元素使用了delete后，该索引上的值会变成undefined，但是数组的长度没有变化；

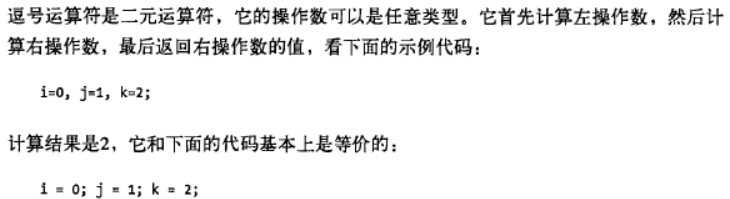




### void



### 逗号运算符



比如：http://blog.csdn.net/wl110231/article/details/8162732

var a = 10, b = 20;

function CommaTest(){

return a++, b++, 10;

}

var c = CommaTest();

console.log(a);

console.log(b);

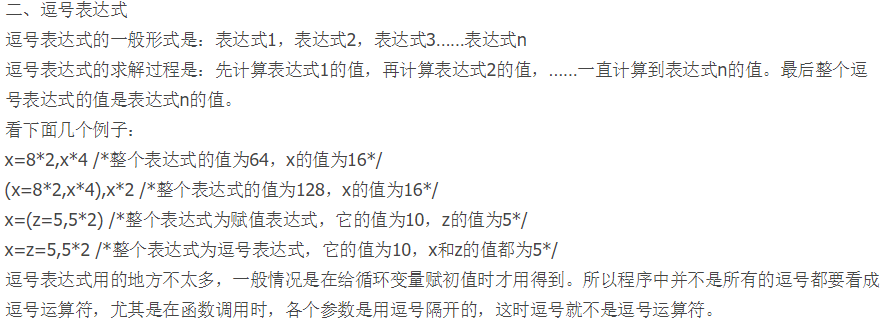
onsole.log(c);

console：

11

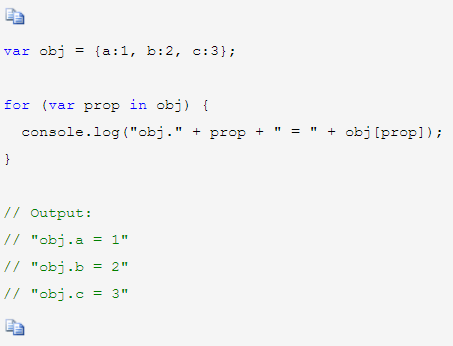
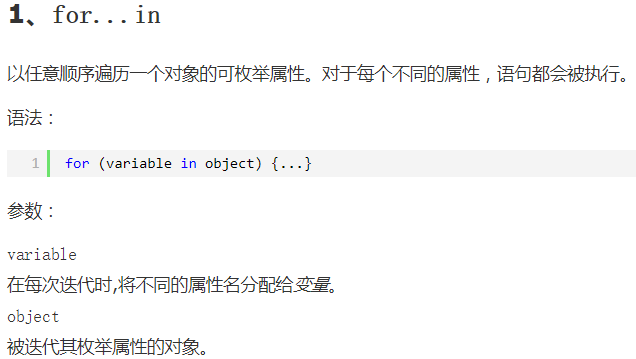
21

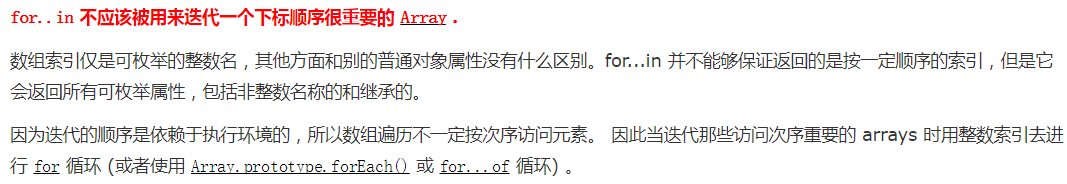
10



## 控制语句

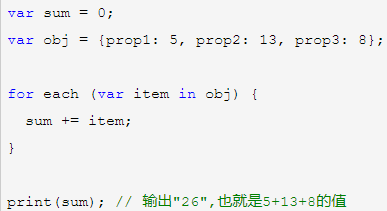
### for/in

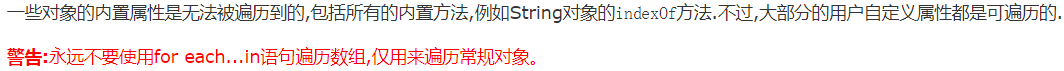




### for each in





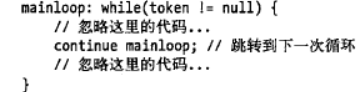


### 标签语句

http://blog.csdn.net/yummy\_go/article/details/50467910

1. 标签+“：“

continue和break是唯一可以使用语句标签的语句。



outerloop:

for (var i = 0; i < 10; i++)

{

innerloop:

for (var j = 0; j < 10; j++)

{

if (j > 3)

{

break;

}

if (i == 2)

{

break innerloop;

}

if (i == 4)

{

break outerloop;

}

document.write("i=" + i + " j=" + j + "<br>");

}

}

1. continue 语句（带有或不带标签引用）只能用在循环中。
2. break 语句（不带标签引用），只能用在循环或 switch 中。
3. 通过标签引用，break 语句可用于跳出任何 JavaScript 代码块：

### throw语句

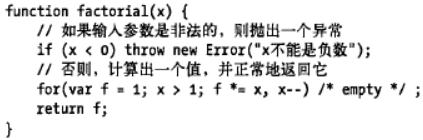
1. 显式抛出错误，

抛出异常的时候通常以Error及其子类型，它有两个属性，name属性表示错误类型，message属性用来存放传递给构造函数的字符串

1. Error对象



1. 我们如果不想使用系统设置的错误信息(例如前面提到的6种)，可以自定义错误，例如让用户输入一个数字，但是用户强制输入了一个字母，系统就会根据自定义错误抛出信息



1. 当抛出异常的时候，需要用户自己定义异常处理程序，然后程序跳到异常处理程序执行。一般异常处理程序都是有catch(e)开头，如果附近没有的话，就会一直往上找，直到找到为止，如果找不到则会把异常当成是错误进行处理，并报告给用户。
2. 一般跟try/catch/finally相结合，放在try代码里面修改错误类型和信息

1数据才会显示

### with语句

1. 模式

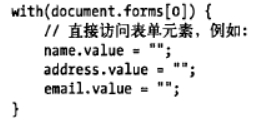


这条语句将object添加到作用域头部，然后执行statement，最后把作用域链恢复到原始状态。

1. 例子



像这样的例子多次出现写起来就很麻烦，用了with语句就省事多了



1. 注意

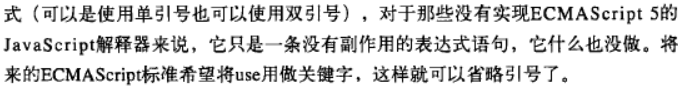
在严格模式下，with语句是禁止的，在不严格模式下，with语句也是不推荐使用的，因为使用了with语句会使得代码很难去优化，而且会使得代码运行得很慢。

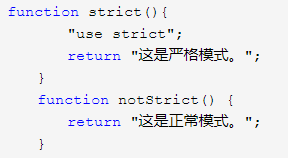
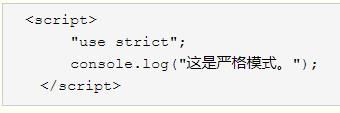
### debugger

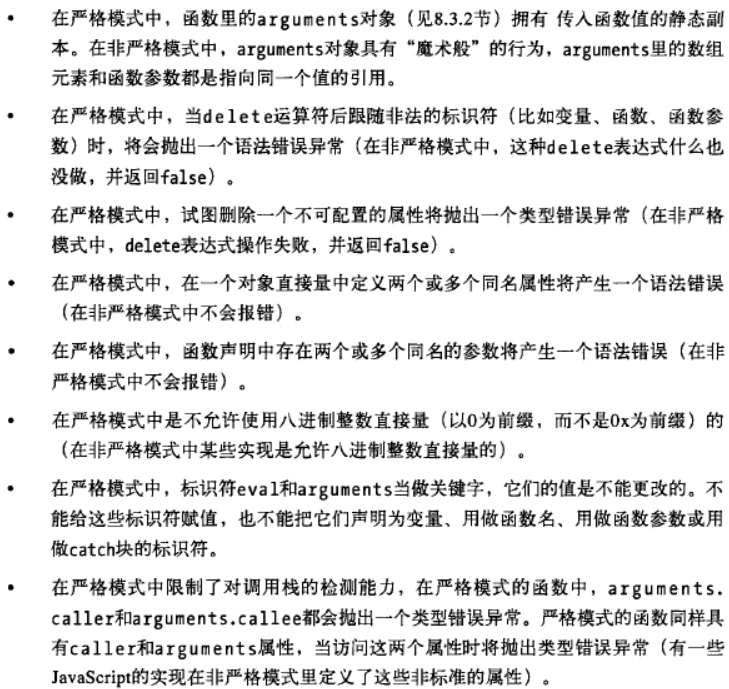
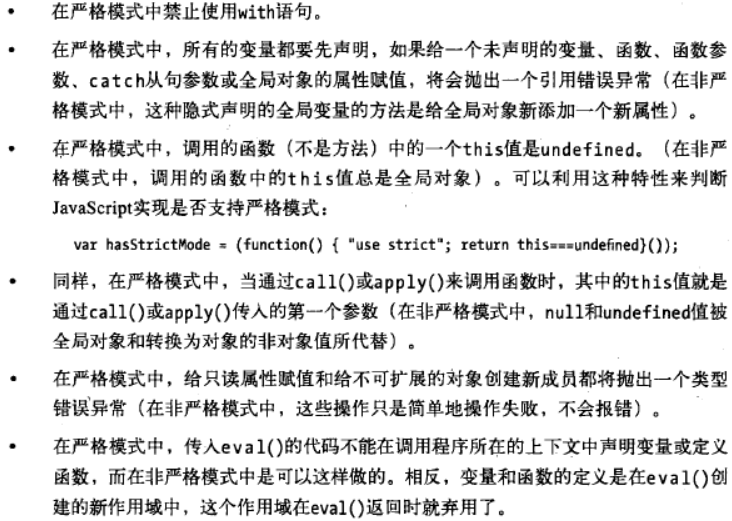
用来调试代码用的，在ECMAScript５的时候才加进去的。

### “use strict”

该语句后的代码将执行严格模式；



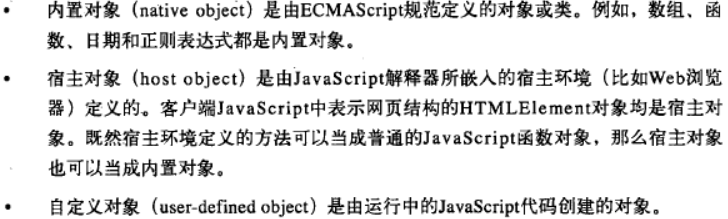




## 对象

### 介绍

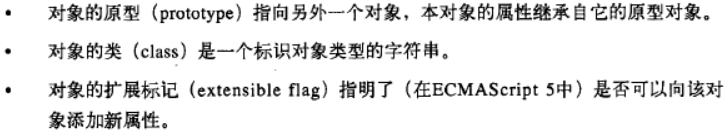
1. 三类对象：内置对象，宿主对象和自定义对象



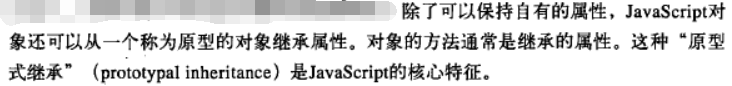
1. 对象的两类属性：自由属性和继承属性



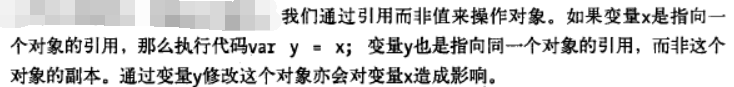
1. 对象的三个特性：对象的原型，对象类和对象的拓展标记



1. 原型式继承prototype inheritance



1. 通过引用来操作对象



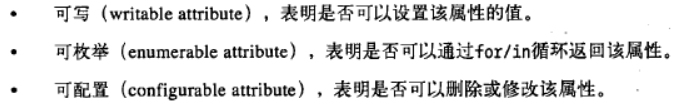
1. 对对象常用的操作

* 创建create
* 设置set
* 查找query
* 删除delete
* 检测test
* 枚举enumerate

1. 属性

属性包括名字和值属性名可以是包括空字符串在内的任意字符串，但同一个对象中不能存在两个同名的属性

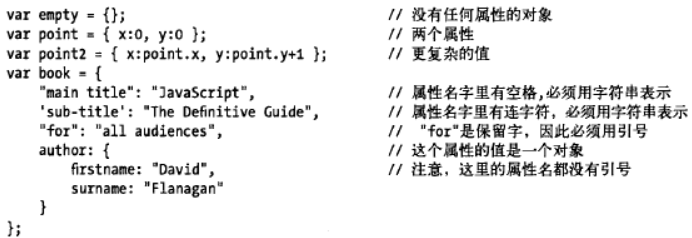
属性特性：可写，可枚举和可配置，ECMAScript5之前，这些特性都是可以的，ECMAScript5则可以对这些特性进行配置。



### 创建对象

1. 对象直接量

可以ECMAScript5中，属性名可以不适用双引号，但是如果属性名中有空格，保留关键字，连字符则必须使用双引号，为了养成良好习惯，以后还是用双引号比较保险。



1. new对象

new后面跟着一个函数调用，该函数叫构造函数constructor



1. Object.creat()

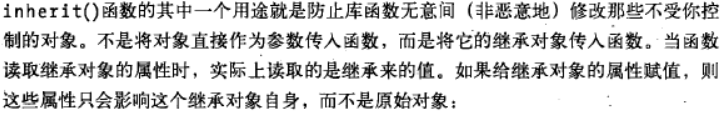
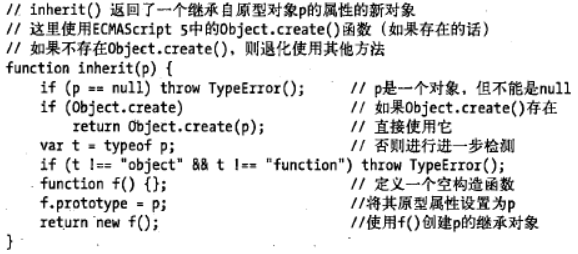
第一个参数就是这个对象的原型，o1是用于创建普通对象；o2用于创建没有原型的新对象；o3用于创建普通空对象；







通过原型继承把一个对象传入一个构造函数，并返回一个由这个构造函数创建的对象。本质就是对象继承另一个对象。



### 对象的原型

1. 所有通过对象直接量创建的对象都具有同一个原型对象，可以通过Object.prototype获得对原型对象的引用
2. 通过关键字new和构造函数创建的对象，其原型就是相应构造函数的prototype原型

如：Array. prototype, Date. prototype

1. 所有的内置构造函数都具有一个继承自Object.prototype的原型，如Date. prototype就是继承自Object.prototype。这一系列链接的原型对象就是所谓的“原型链”

### 属性的查询和设置

1. 两种方式



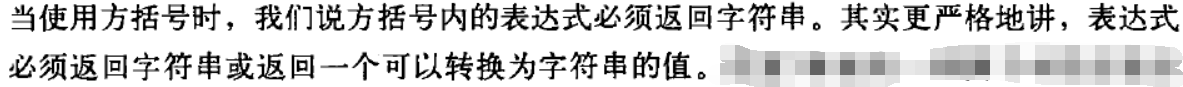
以上两种方式的值是相同的。

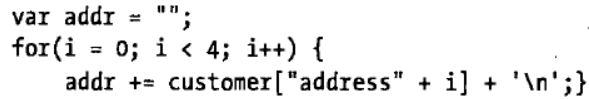
* 点运算符

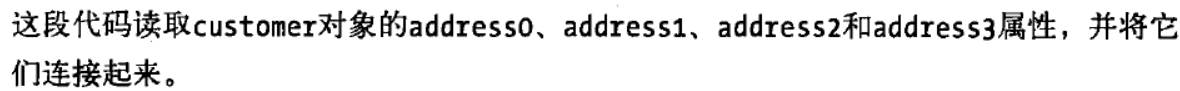
ECMAScript3中规定点运算符后不能加关键字，要加的话务必使用方括号+字符串，ECMAScript5则放宽了这个规定，不过最好还是用方括号+字符串。

* 方括号+字符串

这也叫“关联数组”对象。

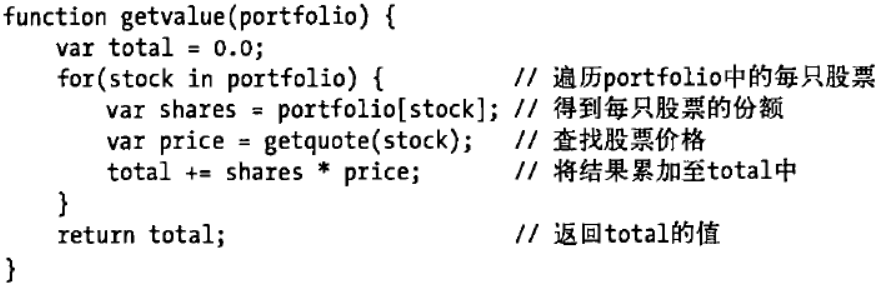






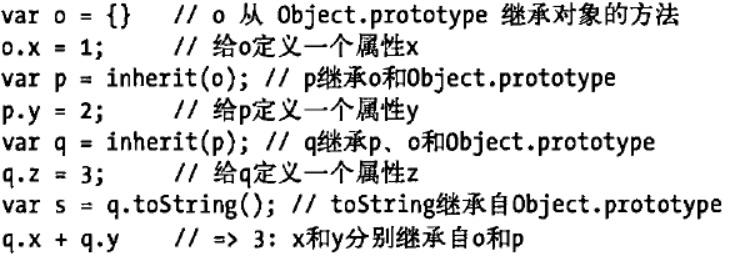
1. 区别

点运算符后面跟着是标识符，是静态的，必须写死在程序中，但是方括号这种是动态的，可以在程序运行中动态获取。

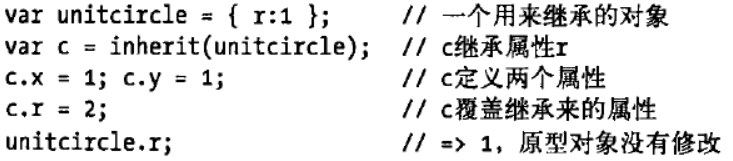


### 继承

1. 原型链

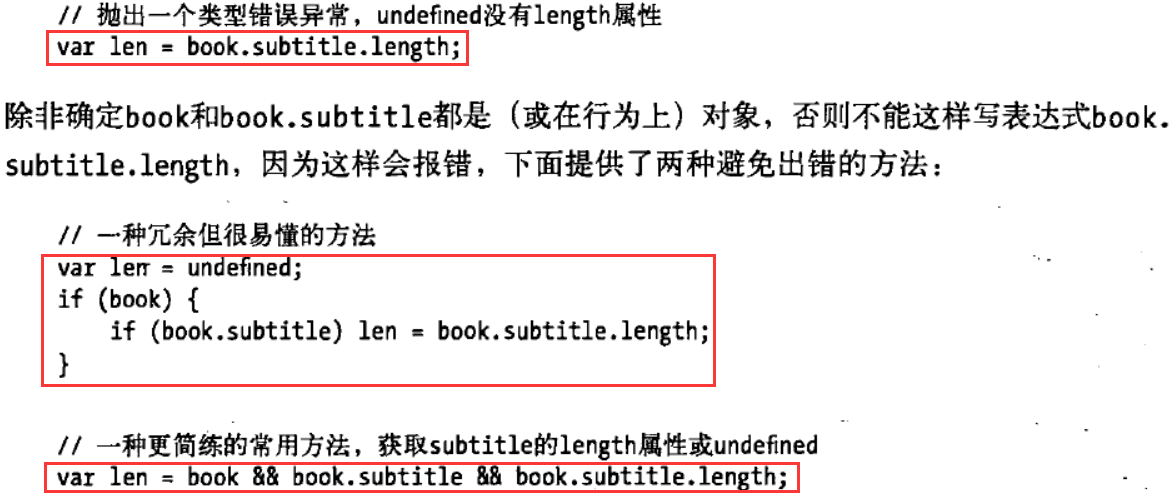


1. 继承不会对原型链进行修改



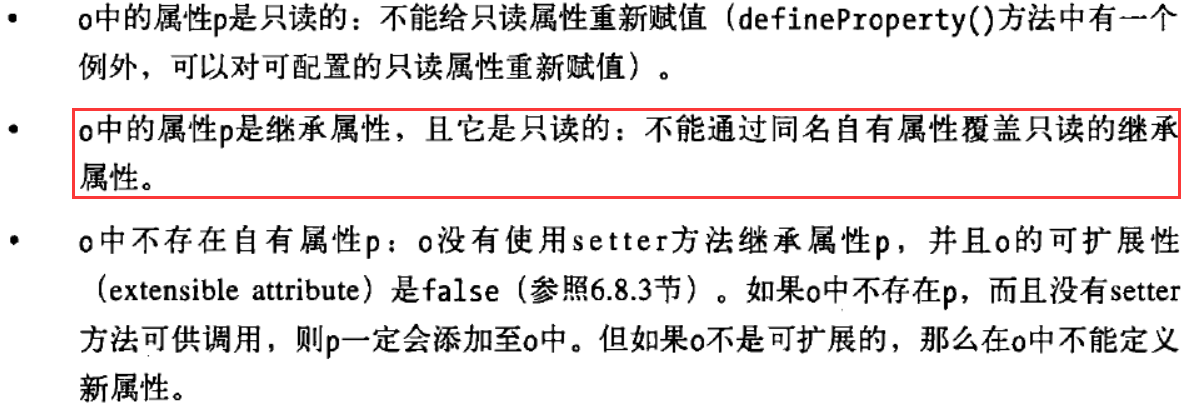
### 属性访问

1. 如果属性中（包括在原型链中）都没有我们想要的，则返回undefined；
2. 在ECMAScript5中，任何关于属性访问失败都会范围类型错误异常
3. 避免的方法



第二种方法可以避免类型错误异常，这是利用了&&的短路行为；

1. 赋值失败的情况

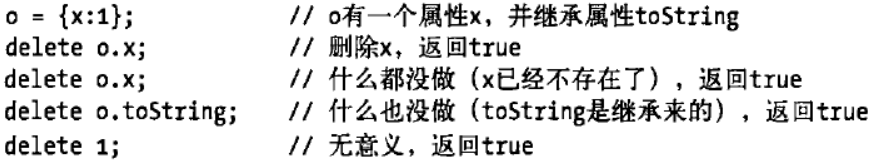


### 删除属性

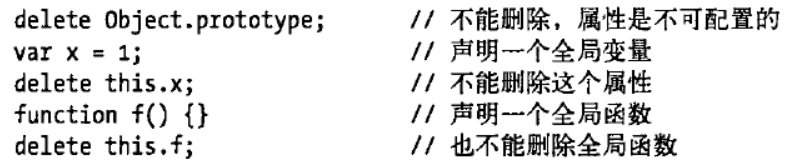
1. 删除属性只能删除自家的属性，不能删除继承过来的属性；

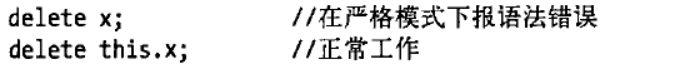


1. delete操作成功后，或者delete一个不是属性表达式的操作数，或者delete一个不存在的属性的时候会返回true，只有删除一些不可配置的属性的时候会返回false，但是在严格模式下，会返回类型异常错误；



1. 那具体哪些是不可配置的属性呢？比如用var声明的全局变量或者函数，严格模式下会报类型错误异常，非严格模式下会返回false





1. ；