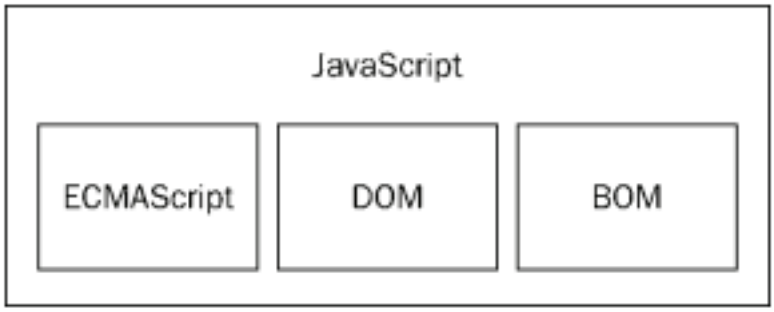
# 基础+内存管理+API

2016年6月18日 星期六　　上午10:58

JavaScript 基于事件驱动和原型的具有动态类型的弱类型语言




DOM：提供访问和操作网页内容的（HTML）API，把整个页面映射为一个多层节点结构。

BOM：提供与浏览器交互的API。

XHTML语法比HTML严格

<![CDATA[CData片段是文档中的一个特殊区域，可以包含不需要解析的任意格式的文本内容]

```
eg:<script  type="text/javascript"  src="example.js"></script>
```

文档模式：quirks mode、standards mode

<!-- HTML 4.01 严格型 -->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.d

<!-- HTML 5 -->

<!DOCTYPE html>

E5引入：  "use strict";

数据类型（具有动态性，故不支持自定义数据类型）

Undefined（undefined）在使用var声明变量但未对其初始化时其值是undefined

Null（null）表示一个空对象指针  alert(null==undefined);//true  比较时会转换操作数

Boolean（true、false）

Number

String（基本类型）表示由零或多个 16 位 Unicode 字符组成的字符序列，字符串是不可变的

eg：var lang = "Java"; langf = lang + "Script";

]>

dtd">

首先创建一个能容纳 10 个字符的 新字符串，然后在这个字符串中填充"Java"和"Script" ，最后
字符串"Java"和字 符串"Script"。

num.toString(2) 该方法返回字符串的一个副本，null 和 undefined 值没有这个方法。

String(null) == “null”
String(undefined) == “undefined”

Object 是一组数据和功能的集合（引用类型）

constructor：保存着用于创建当前对象的函数

hasOwnProperty(name)：检查给定的属性在当前对象实例中（而不是在实例的原型中）是否

isPrototypeOf(object)：检查当前的对象是否是传入对象的原型

propertyIsEnumerable(name)：检查给定的属性是否能够使用for-in来枚举

valueOf()：返回对象的字符串、数值或布尔值表示 toString()

ECMAScript中内置函数和操作符执行流程：

在基于对象调用isNaN()、Number()等函数时，会首先调用对象的valueOf()方法，然后确定该
可以转。如果不能，则基于这个返回值再调用toString()方法，再测试返回值。

位操作符 ~、&、|、 ！、^、<<、 >>、 >>>、 &&、 ||、 ==、 ===

| 表 达 式 | 值 | 表 达 式 | 值 |
|---|---|---|---|
| null == undefined | true | true == 1 | true |
| "NaN" == NaN | false | true == 2 | false |
| 5 == NaN | false | undefined == 0 | false |
| NaN == NaN | false | null == 0 | false |
| NaN != NaN | true | "5"==5 | true |
| false == 0 | true | | |

2. 全等和不全等

除了在比较之前不转换操作数之外，全等和不全等操作符与相等和不相等操作符没有什么区别。

obj1 && obj2 -> obj2 如果第一个对象能决定结果不会再往后执行，否则返回第二个对象

ECMAScript 中的所有数值都以 IEEE-754 64 位格式存储，但位操作符并不直接操作64位的值
的值转换成 32 位的整数，然后执行操作，最后再将结果转换回 64 位

函数参数：不关乎参数个数和参数类型，没有传递值的命名参数将自动被赋予undefined值

arguments对象中的值会自动反映到对应的命名参数，它们的内存空间是独立的，但值会同步

ECMAScript 中函数没有重载，相同命名函数会被后者覆盖。

古一步是销毁原来的

存在

方法返回的值是否

。而是先将 64 位

内存管理

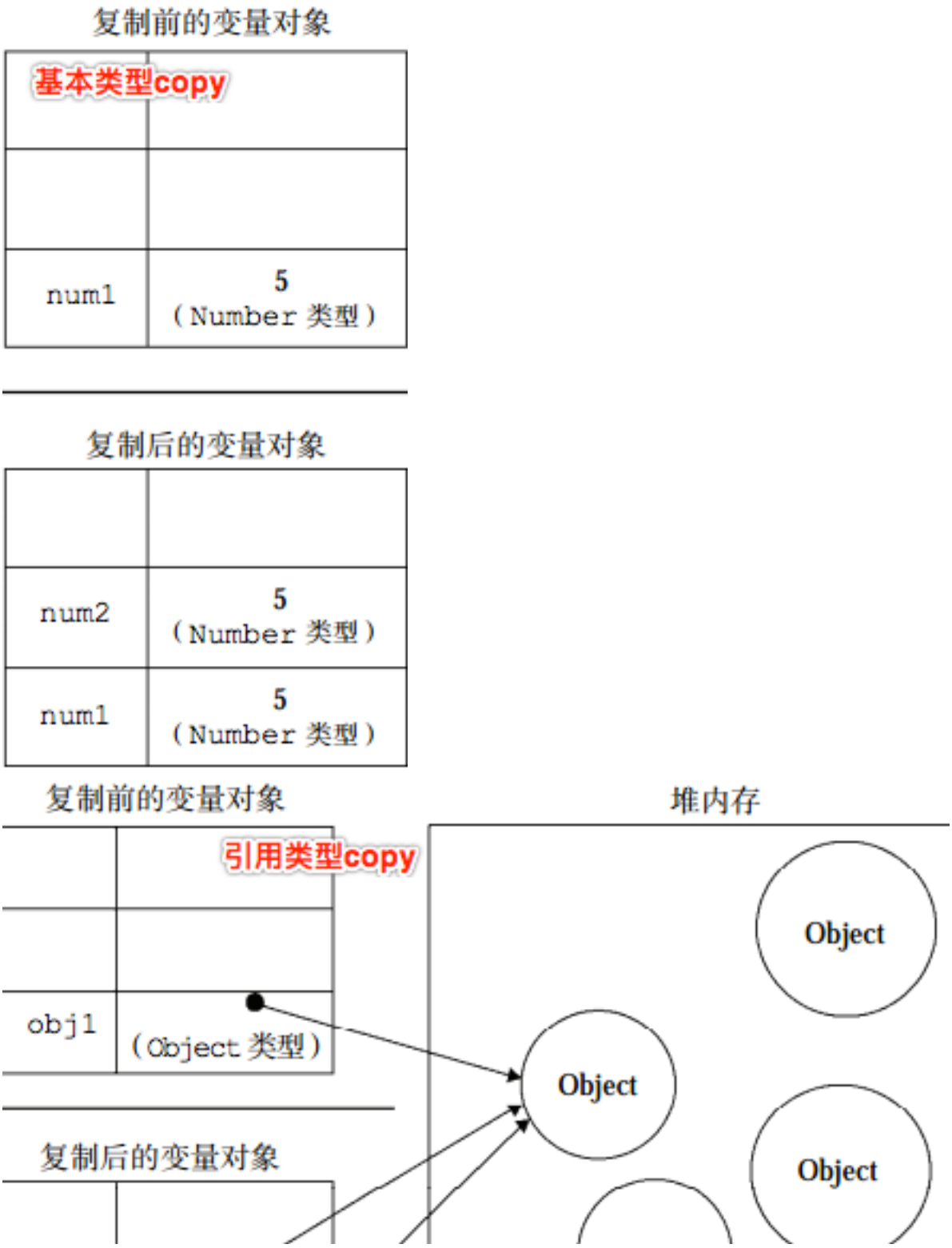
基本类型：指简单的数据段，可以操作保存在变量中的实际的值（5中基本类型）

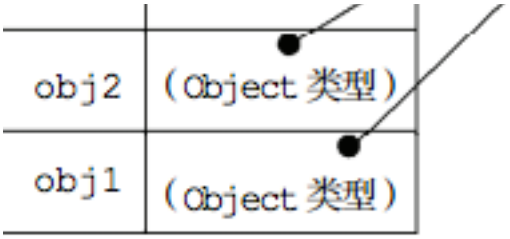
引用类型：由多个值构成的对象，不能直接操作对象的内存空间，通过引用访问（Object）

只能给引用类型值动态地添加属性，以便将来使用。

值传递：实参赋值给形参后，形参在函数中发生变化不会影响实参(不会传回给实参)

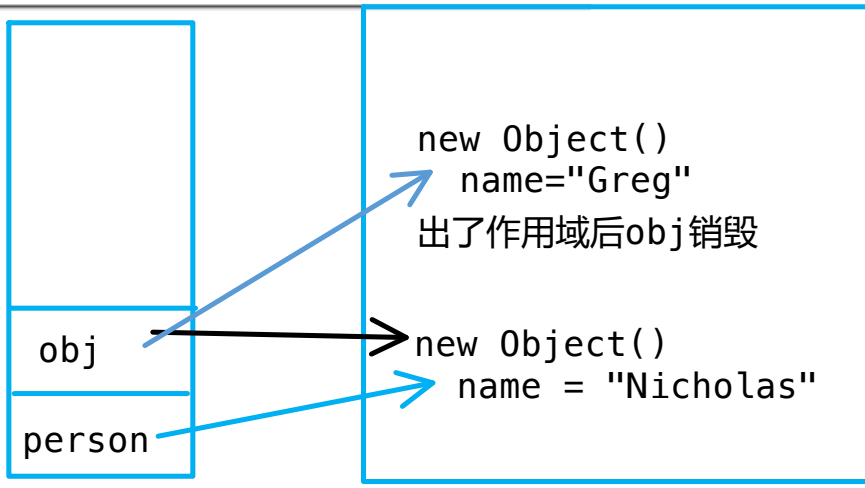
地址传递：在函数中修改指针指向的变量可以改变主函数中的变量，且函数可返回多个值



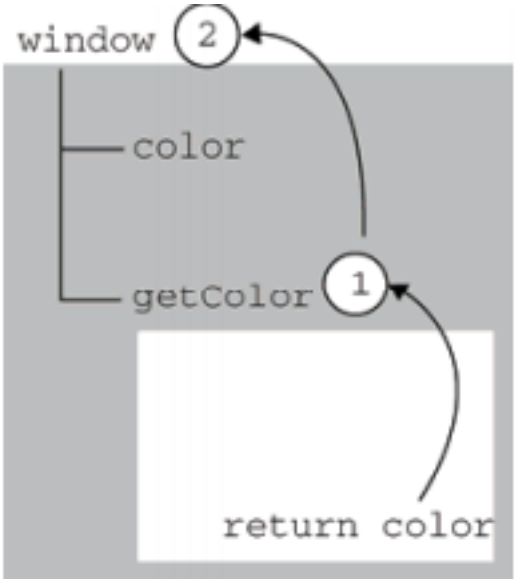




```
function setName(obj) {
        obj.name = "Nicholas";
        obj = new Object();
        obj.name = "Greg";
}
var person = new Object();
setName(person);
alert(person.name); //"Nicholas"
```




每个函数都有自己的执行环境，每个执行环境都有一个与之关联的变量对象，环境中定义的所有
存在这个对象中。当执行流进入一个函数时，函数的环境就会被推入一个环境栈中。而在函数执
环境弹出，把控制权返回给之前的执行环境。
作用域链：保证对执行环境有权访问的所有变量和函数的有序访问
标识符解析是沿着作用域链一级一级地搜索，始终从作用域链的前端开始，然后逐级地向后回溯
符为止，否则报错。




调用本例中的函数getColor()时会引用变量color。为了确定变量color的值，将开始一个两步的搜索过程

没有块级作用域

```
if (true) {
        var color = "blue";
```

有变量和函数都保
执行之后，栈将其

溯，直至找到标识

```
        var color = "blue";
}
alert(color);  //"blue"
if 语句中的变量声明会将变量添加到当前的执行环境（在这里是全局环境）中
for (var i=0; i < 10; i++){
        doSomething(i);
}
alert(i);  //10
for 语句创建的变量 i 即使在循环结束后，也依旧会存在于循环外部的执行环境中
```

自动垃圾收集机制

执行环境负责管理内存，分全局执行环境和函数执行环境。

原理：GC会周期性地找出那些不再继续使用的变量，然后释放其占用的内存。

标记清除：当变量进入环境就将这个变量标记为“进入环境”。当变量离开环境时，则将其标记

（用变量列表或标志位实现）。GC在运行的时候会给存储在内存中的所有变量都加上标记，然

变量以及被环境中的变量引用的变量的标记，而在此之后再被加上标记的变量将被视为准备删除

环境中的变量已经无法访问到这些变量了。

IE<9中BOM、DOM是使用C++以COM对象（引用计数策略）的形式实现的，当涉及COM时存

解除引用：一旦数据不再有用，最好通过将其值设置为 null 来释放其引用，适用于大多数全局

基本类型值在内存中占据固定大小的空间，因此被保存在栈内存中，copy时会创建这个值的副

引用类型的值是对象，保存在堆内存中，copy时复制的是指向该对象的指针。（instanceof）

对象创建：构造函数创建、JSON创建

数组：Array.isArray()

concat()：连接两个或更多的数组，并返回结果

join()：元素通过指定的分隔符进行分隔

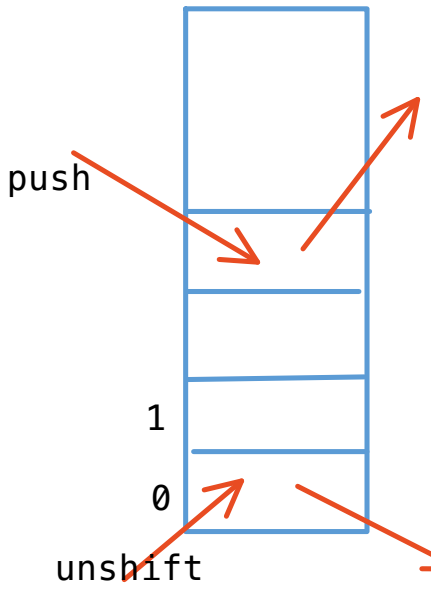
pop()：删除并返回数组的最后一个元素

push()：向数组的末尾添加一个或更多元素，并返回新的长度

shift()：删除并返回数组的第一个元素

unshift()：向数组的开头添加一个或更多元素，并返回新的长度

slice()：从某个已有的数组返回选定的元素



记为“离开环境”
后会去掉环境中的
余的变量，原因是

序在循环引用。
变量和属性。

本。（typeof）

pop

shift

splice()：删除元素，并向数组添加新元素

删除 splice(0,2)　插入 splice(2,0,"red","green")　替换 splice (2,1,"red","green")

map()：返回每次函数调用的结果组成的数组

forEach(function(item, index, array){……});

every()->&&

some()->||

filter()->过滤true

归并：reduce() 、reduceRight()

```
var values = [1,2,3,4,5];
var sum = values.reduce(function(prev, cur, index, array) {
        return prev + cur;
});
alert(sum); //15
```

Date

new Date(毫秒数)

```
var allFives = new Date(2005, 4, 5, 17, 55, 55);
```

Date.parse()

```
var start = +new Date() / Date.now() //取得开始时间
```

RegExp

g：表示全局（global）模式

i：表示不区分大小写（case-insensitive）模式

m：表示多行（multiline）模式

```
var matches = pattern2.exec(text); //exec捕获匹配项
alert(matches.index); //0
alert(matches[0]); //cat
alert(pattern2.lastIndex); //3
```

/^\d{11}&/g　以插入符号（^）和美元符号（$）来匹配字符串的开始和结尾

RegExp.$1…RegExp.$9　9 个用于存储捕获组的构造函数属性

调用 exec()或 test()方法时，这些属性会被自动填充。

```
var text = "this has been a short summer";
```

```
    var pattern = /(..)or(.)/g;

    if (pattern.test(text)){
        alert(RegExp.$1);       //sh
        alert(RegExp.$2);       //t
    }
```

```
var text = "cat, bat, sat, fat";
text.replace(/(.at)/g, "word ($1)"); //word (cat), word (bat), word (sat), word (fat)
```

Function

```
var sum = new Function("num1", "num2", "return num1 + num2"); // 不推荐，函数表达
function add(num1, num2) {  // 函数声明
    return num1+num2;
}
```

add 是函数指针，add() 是调用此函数

函数声明：解析器会率先读取函数声明并使其在执行任何代码之前可访问

函数表达式：必须等到解析器执行到它所在的代码行，才会真正被解释执行

arguments.callee：指向拥有这个 arguments 对象的函数（该函数本身）

arguments.callee.caller：保存着调用当前函数的函数的引用，全局作用域中调用当前函数为n

this：引用的是函数据以执行的环境对象

```
window.color = "red";
var o = { color: "blue" };

function sayColor(){
    alert(this.color);
}

sayColor();      //"red"

o.sayColor = sayColor;
o.sayColor();    //"blue"
```

当在全局作用域中调用sayColor()时，this引用的是全局对象window；this.color->window.color求值结果"red"。

当把这个函数赋给对象o并调用o.sayColor()时，this引用的是对象o，this.color->o.color求值，结果"blue"。

length 表示函数希望接收的命名参数的个数，sayColor.length -> 0

Prototype 保存它们所有实例方法的真正所在，不可使用for-in枚举

apply()/call()：能够扩充函数赖以运行的作用域和传递参数，call必须明确地传入每一个参数

在严格模式下，未指定环境对象而调用函数，this值不会转型为window，是undefined。

```
sum.call(this, num1, num2);
sum.apply(this, arguments); //传入arguments对象
sayColor.call(this);  //red
sayColor.call(window);  //red
```

式

ull

sayColor.call(window); //red

sayColor.call(o); //blue

```
var another=sayColor.bind(o);//将函数绑定执行环境o并返回函数指针
```

## 基本包装类

Boolean、Number、String

```
var s1 = "some text";
var s2 = s1.substring(2);
```

基本类型值不是对象，不应该有方法

(1) 创建 String 类型的一个实例：

```
var s1 = new String("some text");
```

当读取一个基本类型值的时候，后台就会创建一个对应的基本包装类型的对象，让我们能够调用一些方法来操作这些数据

(2) 在实例上调用指定的方法：

```
var s2 = s1.substring(2);
```

(3) 销毁这个实例。

```
s1 = null;
```

引用类型与基本包装类型区别：对象的生存期，new创建的实例在离开作用域之前一直保存在内

在于一行代码的执行瞬间，后立即销毁，意味着不能在运行时为基本类型值添加属性和方法

s1.color = "red"; //立即销毁刚创建的包装类

alert(s1.color); //undefined

常用API：

num.toFixed(2)；

charAt(index)　　charCodeAt(index)

slice()　substr(index, len)　substring(start, end)

indexOf()　lastIndexOf()

trim()　toLowerCase()　toUpperCase()　colorText.split(",", 2);

text.replace(/[<>"&]/g, function(match, pos, originalText){ …… }

URI 编码

有效的URI中不能包含某些字符（空格），用的UTF-8编码替换所有无效的字符，让浏览器能够

eval() 完整的 ECMAScript 解析器

另一种取得 Global 对象的方法是使用以下代码

```
var global = function(){
    return this;
}();
```

function(){}匿名函数是指针，指针()表示让此指针指向的函数立即执行

JavaScript 是单线程语言，但它允许通过设置超时值和间歇时间值来调度代码在特定的时刻执行

内存中，包装类存

多识别。

JavaScript 是单线程语言，但它允许通过设置超时值和间歇时间来调度代码在特定的时刻执行