**第 12 章 变量、作用域及内存**

学习要点：

1.变量及作用域

2.内存问题

**主讲教师：李炎恢 合作网站：**[**http://www.ibeifeng.com**](http://www.ibeifeng.com/) **讲师博客：**[**http://hi.baidu.com/李炎恢**](http://hi.baidu.com/%E6%9D%8E%E7%82%8E%E6%81%A2)

JavaScript 的变量与其他语言的变量有很大区别。JavaScript 变量是松散型的(不强制类 型)本质，决定了它只是在特定时间用于保存特定值的一个名字而已。由于不存在定义某个 变量必须要保存何种数据类型值的规则，变量的值及其数据类型可以在脚本的生命周期内改 变。

## 一．变量及作用域

1.基本类型和引用类型的值

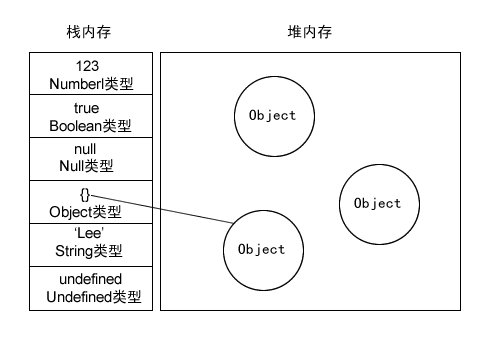
ECMAScript 变量可能包含两种不同的数据类型的值：基本类型值和引用类型值。基本 类型值指的是那些保存在栈内存中的简单数据段，即这种值完全保存在内存中的一个位置。 而引用类型值则是指那些保存在堆内存中的对象，意思是变量中保存的实际上只是一个指 针，这个指针指向内存中的另一个位置，该位置保存对象。

将一个值赋给变量时，解析器必须确定这个值是基本类型值，还是引用类型值。基本类 型值有以下几种：Undefined、Null、Boolean、Number 和 String。这些类型在内存中分别占 有固定大小的空间，他们的值保存在栈空间，我们通过按值来访问的。

PS：在某些语言中，字符串以对象的形式来表示，因此被认为是引用类型。ECMAScript

放弃这一传统。 如果赋值的是引用类型的值，则必须在堆内存中为这个值分配空间。由于这种值的大小

不固定，因此不能把它们保存到栈内存中。但内存地址大小的固定的，因此可以将内存地址 保存在栈内存中。这样，当查询引用类型的变量时，先从栈中读取内存地址，然后再通过地 址找到堆中的值。对于这种，我们把它叫做按引用访问。



2.动态属性

定义基本类型值和引用类型值的方式是相似的：创建一个变量并为该变量赋值。但是， 当这个值保存到变量中以后，对不同类型值可以执行的操作则大相径庭。

var box = new Object(); //创建引用类型

box.name = 'Lee'; //新增一个属性

alert(box.name); //输出

如果是基本类型的值添加属性的话，就会出现问题了。

var box = 'Lee'; //创建一个基本类型

box.age = 27; //给基本类型添加属性

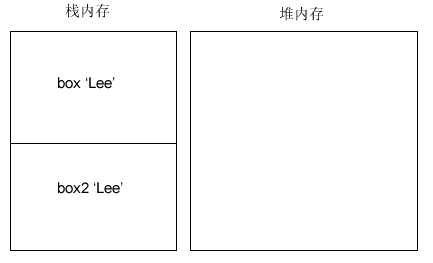
alert(box.age); //undefined

3.复制变量值 在变量复制方面，基本类型和引用类型也有所不同。基本类型复制的是值本身，而引用

类型复制的是地址。

var box = 'Lee'; //在栈内存生成一个 box 'Lee'

var box2 = box; //在栈内存再生成一个 box2 'Lee'

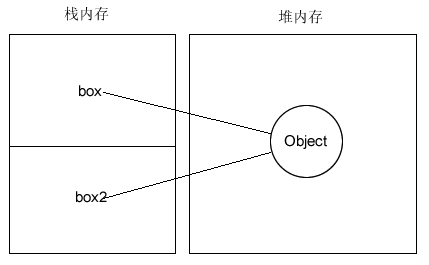


box2 是虽然是 box1 的一个副本，但从图示可以看出，它是完全独立的。也就是说，两 个变量分别操作时互不影响。

var box = new Object(); //创建一个引用类型

box.name = 'Lee'; //新增一个属性

var box2 = box; //把引用地址赋值给 box2



在引用类型中，box2 其实就是 box，因为他们指向的是同一个对象。如果这个对象中的

name 属性被修改了，box2.name 和 box.name 输出的值都会被相应修改掉了。

4.传递参数

ECMAScript 中所有函数的参数都是按值传递的，言下之意就是说，参数不会按引用传 递，虽然变量有基本类型和引用类型之分。

function box(num) { //按值传递，传递的参数是基本类型 num += 10; //这里的 num 是局部变量，全局无效 return num;

}

var num = 50;

var result = box(num);

alert(result); //60

alert(num); //50

PS：以上的代码中，传递的参数是一个基本类型的值。而函数里的 num 是一个局部变 量，和外面的 num 没有任何联系。

下面给出一个参数作为引用类型的例子。

function box(obj) { //按值传递，传递的参数是引用类型 obj.name = 'Lee';

}

var p = new Object(); box(p);

alert(p.name);

PS：如果存在按引用传递的话，那么函数里的那个变量将会是全局变量，在外部也可 以访问。比如 PHP 中，必须在参数前面加上&符号表示按引用传递。而 ECMAScript 没有这 些，只能是局部变量。可以在 PHP 中了解一下。

PS：所以按引用传递和传递引用类型是两个不同的概念。

function box(obj) { obj.name = 'Lee';

var obj = new Object(); //函数内部又创建了一个对象

obj.name = 'Mr.'; //并没有替换掉原来的 obj

}

最后得出结论，ECMAScript 函数的参数都将是局部变量，也就是说，没有按引用传递。

5.检测类型

要检测一个变量的类型，我们可以通过 typeof 运算符来判别。诸如：

var box = 'Lee';

alert(typeof box); //string

虽然 typeof 运算符在检查基本数据类型的时候非常好用，但检测引用类型的时候，它就 不是那么好用了。通常，我们并不想知道它是不是对象，而是想知道它到底是什么类型的对 象。因为数组也是 object，null 也是 Object 等等。

这时我们应该采用 instanceof 运算符来查看。

var box = [1,2,3];

alert(box instanceof Array); //是否是数组 var box2 = {};

alert(box2 instanceof Object); //是否是对象

var box3 = /g/;

alert(box3 instanceof RegExp); //是否是正则表达式 var box4 = new String('Lee');

alert(box4 instanceof String); //是否是字符串对象

PS：当使用 instanceof 检查基本类型的值时，它会返回 false。 5.执行环境及作用域

执行环境是 JavaScript 中最为重要的一个概念。执行环境定义了变量或函数有权访问的 其他数据，决定了它们各自的行为。

全局执行环境是最外围的执行环境。在 Web 浏览器中，全局执行环境被认为是 window

对象。因此所有的全局变量和函数都是作为 window 对象的属性和方法创建的。

var box = 'blue'; //声明一个全局变量 function setBox() {

alert(box); //全局变量可以在函数里访问

}

setBox(); //执行函数

全局的变量和函数，都是 window 对象的属性和方法。

var box = 'blue'; function setBox() {

alert(window.box); //全局变量即 window 的属性

}

window.setBox(); //全局函数即 window 的方法

PS：当执行环境中的所有代码执行完毕后，该环境被销毁，保存在其中的所有变量和 函数定义也随之销毁。如果是全局环境下，需要程序执行完毕，或者网页被关闭才会销毁。

PS：每个执行环境都有一个与之关联的变量对象，就好比全局的 window 可以调用变量 和属性一样。局部的环境也有一个类似 window 的变量对象，环境中定义的所有变量和函数 都保存在这个对象中。(我们无法访问这个变量对象，但解析器会处理数据时后台使用它)

函数里的局部作用域里的变量替换全局变量，但作用域仅限在函数体内这个局部环境。

var box = 'blue'; function setBox() {

var box = 'red'; //这里是局部变量，出来就不认识了

alert(box);

}

setBox(); alert(box);

通过传参，可以替换函数体内的局部变量，但作用域仅限在函数体内这个局部环境。

var box = 'blue';

function setBox(box) { //通过传参，替换了全局变量 alert(box);

}

setBox('red'); alert(box);

函数体内还包含着函数，只有这个函数才可以访问内一层的函数。

var box = 'blue'; function setBox() {

function setColor() { var b = 'orange'; alert(box); alert(b);

}

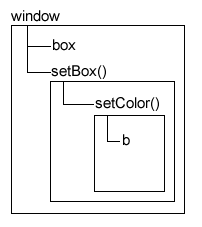
setColor(); //setColor()的执行环境在 setBox()内

}

setBox();

PS：每个函数被调用时都会创建自己的执行环境。当执行到这个函数时，函数的环境 就会被推到环境栈中去执行，而执行后又在环境栈中弹出(退出)，把控制权交给上一级的执 行环境。

PS：当代码在一个环境中执行时，就会形成一种叫做作用域链的东西。它的用途是保 证对执行环境中有访问权限的变量和函数进行有序访问。作用域链的前端，就是执行环境的 变量对象。



6.没有块级作用域

块级作用域表示诸如 if 语句等有花括号封闭的代码块，所以，支持条件判断来定义变 量。

if (true) { //if 语句代码块没有局部作用域

var box = 'Lee';

}

alert(box);

for 循环语句也是如此

for (var i = 0; i < 10; i ++) { //没有局部作用域 var box = 'Lee';

}

alert(i); alert(box);

var 关键字在函数里的区别

function box(num1, num2) {

var sum = num1 + num2; //如果去掉 var 就是全局变量了

return sum;

}

alert(box(10,10));

alert(sum); //报错

PS：非常不建议不使用 var 就初始化变量，因为这种方法会导致各种意外发生。所以初 始化变量的时候一定要加上 var。

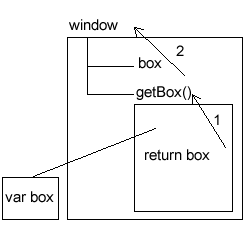
一般确定变量都是通过搜索来确定该标识符实际代表什么。

var box = 'blue'; function getBox() {

return box; //代表全局 box

} //如果加上函数体内加上 var box = 'red'

alert(getBox()); //那么最后返回值就是 red



PS：变量查询中，访问局部变量要比全局变量更快，因为不需要向上搜索作用域链。

## 二．内存问题

JavaScript 具有自动垃圾收集机制，也就是说，执行环境会负责管理代码执行过程中使 用的内存。其他语言比如 C 和 C++，必须手工跟踪内存使用情况，适时的释放，否则会造 成很多问题。而 JavaScript 则不需要这样，它会自行管理内存分配及无用内存的回收。

JavaScript 最常用的垃圾收集方式是标记清除。垃圾收集器会在运行的时候给存储在内 存中的变量加上标记。然后，它会去掉环境中正在使用变量的标记，而没有被去掉标记的变 量将被视为准备删除的变量。最后，垃圾收集器完成内存清理工作，销毁那些带标记的值并 回收他们所占用的内存空间。

垃圾收集器是周期性运行的，这样会导致整个程序的性能问题。比如 IE7 以前的版本， 它的垃圾收集器是根据内存分配量运行的，比如 256 个变量就开始运行垃圾收集器，这样， 就不得不频繁地运行，从而降低的性能。

一般来说，确保占用最少的内存可以让页面获得更好的性能。那么优化内存的最佳方案， 就是一旦数据不再有用，那么将其设置为 null 来释放引用，这个做法叫做解除引用。这一 做法适用于大多数全局变量和全局对象。

var o = {

name : 'Lee'

};

o = null; //解除对象引用，等待垃圾收集器回收

**感谢收看本次教程！**

# 本课程是由北风网(ibeifeng.com) 瓢城 Web 俱乐部(yc60.com)联合提供：

**本次主讲老师：李炎恢 我的博客：hi.baidu.com/李炎恢/ 我的邮件：**[**yc60.com@gmail.com**](mailto:yc60.com@gmail.com)