《JavaScript高级程序设计（第3版）》

阅读记录：

2017-05-09：开始~~9

2017-05-10：10~~11

2017-05-15：12

2017-05-15：68~~71

# 第1章 JavaScript简介

本章内容

* JavaScript历史回顾
* JavaScript是什么
* JavaScript与ECMAScript的关系
* JavaScript的不同版本

小结

JavaScript是一种专为与网页交互而设计的脚步语言，由下列三个不同的部分组成：

* ECMAScript，由ECMA-262定义，提供核心语言功能；
* 文档对象模型（DOM），提供方位和操作网页内容的方法和接口；
* 浏览器对象模型（BOM），提供与浏览器交互的方法和接口。

# 第2章 在HTML中使用JavaScript

本章内容

* 使用<script>元素
* 嵌入脚步与外部脚本
* 文档模式对JavaScript的影响
* 考虑禁用JavaScript的场景

## 2.1 <script>元素

HTML4.01为<script>定义的6个属性：

* async：可选。表示应该立即下载脚本，但不应妨碍页面中的其他操作，比如下载其他资源或等待加载其他脚本。只对外部脚本文件有效。
* charset：可选。表示通过src属性指定的代码的字符集。由于大多数浏览器会忽略它的值，因此这个属性很少有人用。
* defer：可选。表示脚本可以延迟到文档完全被解析和显示之后再执行。只对外部脚本文件有效。IE7及更早版本对嵌入脚本也支持这个属性。
* language：已废弃。
* src：可选。表示包含要执行代码的外部文件。
* type：可选。表示编写代码使用的脚本语言的内容类型（也称为MIME类型）。实际上，服务器在传送JavaScript文件时使用的MIME类型通常是application/x-javascript，但在type中设置这个值却可能导致脚本被忽略。在非IE浏览器中还可以使用以下值：application/javascript和application/ecmascript。考虑到约定俗成和最大限度的浏览器兼容性，目前type属性的值依旧还是text/javascript。默认值为text/javascript。

使用方式：

1. 直接在页面中嵌入JavaScript代码。这时，只须为<script>指定type属性。
2. 包含外部JavaScript文件。

在解释器对<script>元素内部的所有代码求值完毕以前，页面中的其余内容都不会被浏览器加载或显示。

**在使用<script>嵌入JavaScript代码时，不要在代码中的任何地方出现”</script>”字符串。**因为按照解析嵌入式代码的规则，当浏览器遇到字符串”</script>”时，就会认为那是结束的</script>标签。

解决方法：

<script type=”text/javascript”>

funciton sayScript(){

**alert(“<\/script>”);**

}

</script>

在XHTML文档中，可以省略结束的</script>标签。

|  |
| --- |
| 按照惯例，外部JavaScript文档带有.js扩展名。但这个扩展名不是必需的，因为浏览器不会检查包含JavaScript的文件的扩展名。这样一来，使用JSP、PHP或者其他服务端语言动态生成JavaScript代码也就成为了可能。但是服务器通常还是需要看扩展名决定为响应引用哪种MIME类型。如果不使用.js扩展名，请确保服务器能返回正确的MIME类型。 |

带有src属性的<script>元素不应该在其<script>和</script>标签之间再包含额外的JavaScript代码。如果包含了嵌入的代码，则只会下载并执行外部脚本文件，嵌入的代码会被忽略。

src属性还可以包含来自外部域的JavaScript文件。

### 2.1.1 标签的位置

按照惯例，所有<script>元素都应该放在页面的<head>元素中。这样做的弊端是JavaScript加载完成前浏览器窗口中将是一片空白，为了避免这个问题，**现代Web应用程序一般都会把全部JavaScript引用放在<body>元素中页面的内容后面**。

# 第3章 基本概念

# 第4章 变量、作用域和内存问题

本章内容

* 理解基本类型和引用类型的值
* 理解执行环境
* 理解垃圾收集

## 4.1 基本类型和引用类型的值

ECMAScript变量可能包含两种不同数据类型的值：基本类型值和引用类型值。基本类型值指的是简单的数据段，引用类型值指那些可能由多个值构成的对象。

基本数据类型按值访问，**可操作保存在变量中的实际的值**。

**引用类型的值是保存在内存中的对象**，由于JavaScript不允许直接访问内存中的位置，即不能直接操作对象的内存空间，**在操作对象时，实际上是在操作对象的引用而不是实际的对象**，因此引用类型的值是按引用访问的。

### 4.1.1 动态的属性

对于引用类型的值，我们可以为其添加属性和方法，也可以改变和删除其属性和方法。

基本类型无法添加属性，尽管这样做不会导致任何错误。

### 4.1.2复制变量值

如果从一个变量向另外一个变量复制**基本类型**的值，会在变量对象上创建一个新值，然后把该值复制到为新的变量分配的位置上。这两个变量可以参与任何操作而不会相互影响。

当一个变量向另外一个变量复制**引用类型**的值时，同样也会将存储在变量对象中的值复制一份放到为新变量分配的空间中。不同的是，这个值的副本实际上是一个指针，而这个指针指向存储在堆中的一个对象。

### 4.1.3 传递参数

ECMAScript中所有函数的参数都是按值传递的，即把函数外部的值复制给函数内部的参数，如同变量的复制一样。

在向参数传递基本类型的值时，被传递的值会被复制给一个局部变量（即命名参数，或者用ECMAScript的概念来说，就是arguments对象中的一个元素）。**在向参数传递引用类型的值时，会把这个值在内存中的地址复制给一个局部变量，因此这个局部变量的变化会反映在函数的外部。**

基本类型参数传递例子：

Function addTen(num){

num += 10;

return num;

}

var count = 20;

var result = addTen(count);

alert(count); //20，没有变化

alert(result); //30

引用类型参数传递例子1：

Function setName(obj){

obj.name = “Nicholas”;

}

var person = new Obejct();

setName(person);

alert(person.name); //”Nicholas”

在这个函数内部，obj和person引用的是同一个对象，而且是全局对象。有很多开发人员错误地认为：在局部作用域中修改的对象会在全局作用域中反映出来，就说明参数是按引用传递的。为了证明对象是按值传递的，我们再看一看下面这个经过修改的例子：

引用类型参数传递例子2：

function setName(obj){

obj.name = “Nicholas”;

**obj = new Object();**

**obj.name = “Greg”;**

}

var person = new Object();

setName(person);

alert(person.name); //”Nicholas”

这个例子与前一个例子的唯一区别，就是在setName()函数中添加了两行代码：一行代码为obj重新定义了一个对象，另一行代码为该对象定义了一个带有不同值的name属性。在把person传递给setName()后，其name属性被设置为”Nicholas”。然后，又将一个新对象赋给变量obj，同时将其name属性设置为”Greg”。如果person是按引用传递的，那么person就会自动被修改为指向其name属性值为”Greg”的新对象。但是，当接下来再访问person.name时，显示的值仍然是”Nicholas”。这说明即使在函数内部修改了参数的值，但原始的引用仍然保持未变。实际上，当在函数内部重写obj时，这个变量引用的就是一个局部对象了。而这个局部对象会在函数执行完毕后立即被销毁。

|  |
| --- |
| 可以把ECMAScript函数的参数想象成局部变量。 |