**《JavaScript高级程序设计(第3版)》读书笔记**

## 第1章 JavaScript 简介

一个完整的JavaScript 实现应该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），提供访问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。

我们常见的Web 浏览器只是ECMAScript 实现可能的宿主环境之一。宿主环境不仅提供基本的ECMAScript 实现，同时也会提供该语言的扩展，以便语言与环境之间对接交互。而这些扩展——如DOM，则利用ECMAScript 的核心类型和语法提供更多更具体的功能，以便实现针对环境的操作。其他宿主环境包括Node（一种服务端JavaScript 平台）和Adobe Flash。

ECMAScript 就是对实现该标准规定的各个方面内容的语言的描述。

JavaScript 实现了ECMAScript，Adobe ActionScript 同样也实现了ECMAScript。

文档对象模型（DOM，[Document Object Model](http://www.w3.org/DOM/)）是针对XML 但经过扩展用于HTML 的应用程序编程接口（API，Application Programming Interface）。DOM把整个页面映射为一个多层节点结构。HTML或XML 页面中的每个组成部分都是某种类型的节点，这些节点又包含着不同类型的数据。DOM 被设计为与语言无关。

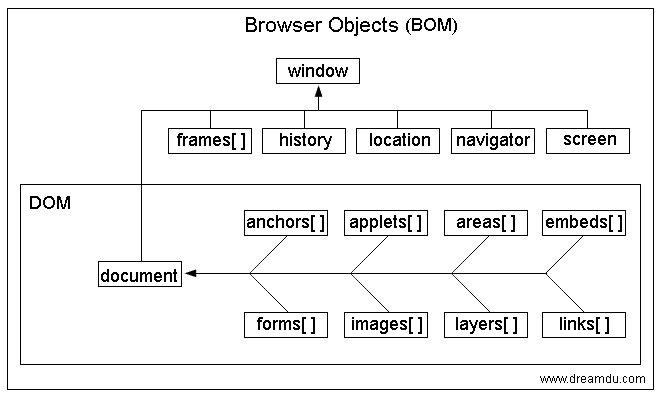
DOM 的发展经历了不同级别，每一级别在前一级别基础上添加了新功能。

级别 1 涵盖了基本功能；

级别 2 添加了名称空间支持、UI 事件模型、迭代器等功能；

级别 3 添加了可从 XML 文档文件进行加载和保存的 API，并集成了 XPath，添加了验证支持等。

BOM(Browser Object Model) 是指浏览器对象模型，是用于描述这种对象与对象之间层次关系的模型，浏览器对象模型提供了独立于内容的、可以与浏览器窗口进行互动的对象结构。BOM由多个对象组成，其中代表浏览器窗口的Window对象是BOM的顶层对象，其他对象都是该对象的子对象。



## 第2章 在HTML中使用JavaScript

所有<script>元素都会按照它们在页面中出现的先后顺序依次被解析。在不使用defer 和

async 属性的情况下，只有在解析完前面<script>元素中的代码之后，才会开始解析后面

<script>元素中的代码。

由于浏览器会先解析完不使用defer 属性的<script>元素中的代码，然后再解析后面的内容，

所以一般应该把<script>元素放在页面最后，即主要内容后面，</body>标签前面。(过时了)

使用defer 属性可以让脚本在文档完全呈现之后再执行。延迟脚本总是按照指定它们的顺序执行。

使用async 属性可以表示当前脚本不必等待其他脚本，也不必阻塞文档呈现。不能保证异步脚本按照它们在页面中出现的顺序执行。

对于标准模式，可以通过使用下面任何一种文档类型来开启：

<!-- HTML 4.01 严格型 -->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"

"http://www.w3.org/TR/html4/strict.dtd">

<!-- XHTML 1.0 严格型 -->

<!DOCTYPE html PUBLIC

"-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- HTML 5 -->

<!DOCTYPE html>

## 第3章 基本概念

let 和yield 是第5 版新增的保留字；其他保留字都是第3 版定义的。为了最大程度地保

证兼容性，建议读者将第3 版定义的保留字外加let 和yield 作为编程时的参考。

给未经声明的变量赋值在严格模式下会导致抛出ReferenceError 错误。

在严格模式下，不能定义名为eval 或arguments 的变量，否则会导致语法错误。

包含undefined 值的变量与尚未定义的变量还是不一样的。看看下面这个例子：

var message; // 这个变量声明之后默认取得了undefined 值

// 下面这个变量并没有声明

// var age;

alert(message); // "undefined"

alert(age); // 产生错误

对于那些极大或极小的数值，可以用e 表示法（即科学计数法）表示的浮点数值表示。用e 表示法表示的数值等于e 前面的数值乘以10 的指数次幂。ECMAScript 中e 表示法的格式也是如此，即前面是一个数值（可以是整数也可以是浮点数），中间是一个大写或小写的字母E，后面是10 的幂中的指数，该幂值将用来与前面的数相乘。下面是一个使用e 表示法表示数值的例子：

var floatNum = 3.125e7; // 等于31250000

在这个例子中，使用e 表示法表示的变量floatNum 的形式虽然简洁，但它的实际值则是31250000。在此，e 表示法的实际含义就是“3.125 乘以107”。

也可以使用e 表示法表示极小的数值，如0.00000000000000003，这个数值可以使用更简洁的3e17表示。在默认情况下，ECMASctipt 会将那些小数点后面带有6 个零以上的浮点数值转换为以e 表示法表示的数值（例如，0.0000003 会被转换成3e7）。

浮点数值的最高精度是17 位小数，但在进行算术计算时其精确度远远不如整数。例如，0.1 加0.2的结果不是0.3，而是0.30000000000000004。这个小小的舍入误差会导致无法测试特定的浮点数值。

例如：

if (a + b == 0.3){ // 不要做这样的测试！

alert("You got 0.3.");

}

在这个例子中，我们测试的是两个数的和是不是等于0.3。如果这两个数是0.05 和0.25，或者是0.15和0.15 都不会有问题。而如前所述，如果这两个数是0.1 和0.2，那么测试将无法通过。因此，永远不要测试某个特定的浮点数值。

关于浮点数值计算会产生舍入误差的问题，有一点需要明确：这是使用基于IEEE754 数值的浮点计算的通病，ECMAScript 并非独此一家；其他使用相同数值格式的语言也存在这个问题。

alert(isNaN("blue")); //true（不能转换成数值）

字符串"blue"不能被转换成数值，因此函数返回了true。

在使用parseInt()解析像八进制字面量的字符串时，ECMAScript 3 和5 存在分歧。例如：

//ECMAScript 3 认为是56（八进制），ECMAScript 5 认为是70（十进制）

var num = parseInt("070");

在ECMAScript 3 JavaScript 引擎中，"070"被当成八进制字面量，因此转换后的值是十进制的56。

而在ECMAScript 5 JavaScript 引擎中，parseInt()已经不具有解析八进制值的能力，因此前导的零会被认为无效，从而将这个值当成"70"，结果就得到十进制的70。在ECMAScript 5 中，即使是在非严格模式下也会如此。

为了消除在使用parseInt()函数时可能导致的上述困惑，可以为这个函数提供第二个参数：转换时使用的基数（即多少进制）。不指定基数意味着让parseInt()决定如何解析输入的字符串，因此为了避免错误的解析，我们建议无论在什么情况下都明确指定基数，例如：

var num = parseInt("0xAF", 16); //175

由于parseFloat()只解析十进制值，因此它没有用第二个参数指定基数的用法。

如果字符串包含的是一个可解析为整数的数（没有小数点，或者小数点后都是零），parseFloat()会返回整数。

var num1 = parseFloat("1234blue"); //1234 （整数，不是浮点数）

**字符串的特点**

ECMAScript 中的字符串是不可变的，也就是说，字符串一旦创建，它们的值就不能改变。要改变某个变量保存的字符串，首先要销毁原来的字符串，然后再用另一个包含新值的字符串填充该变量，例如：

var lang = "Java";

lang = lang + "Script";

以上示例中的变量lang 开始时包含字符串"Java"。而第二行代码把lang 的值重新定义为"Java"与"Script"的组合，即"JavaScript"。实现这个操作的过程如下：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"，最后一步是销毁原来的字符串"Java"和字符串"Script"，因为这两个字符串已经没用了。这个过程是在后台发生的，而这也是在某些旧版本的浏览器（例如版本低于1.0 的Firefox、IE6 等）中拼接字符串时速度很慢的原因所在。但这些浏览器后来的版本已经解决了这个低效率问题。

var value1 = 10;

var value2 = true;

var value3 = null;

var value4;

alert(String(value1)); // "10"

alert(String(value2)); // "true"

alert(String(value3)); // "null"

alert(String(value4)); // "undefined"

要把某个值转换为字符串，可以使用加号操作符（3.5 节讨论）把它与一个字符串（""）加在一起。

在ECMAScript 中，如果不给构造函数传递参数，则可以省略后面的那一对圆括号。

var o = new Object; // 有效，但不推荐省略圆括号