# JavaScript概述

## 学习要点：

一、什么是JavaScript

二、JavaScript特点

三、JavaScript历史

四、JavaScript组成

五、JavaScript在网页中引入方式

六、如何调试JavaScript代码

JavaScript诞生于1995年。它当时的目的是为了验证表单输入的验证。因为在JavaScript问世之前，表单的验证都是通过服务器端验证的。而当时都是电话拨号上网的年代，服务器验证数据是一件非常痛苦的事情。

经过许多年的发展，JavaScript从一个简单的输入验证成为一门强大的编程语言。所以，学会使用它是非常简单的，而真正掌握它则需要很漫长的时间。

## 一、什么是JavaScript

JavaScript是一种具有面向对象能力的、解释型的程序设计语言。更具体一点，它是基于对象和事件驱动并具有相对安全性的客户端脚本语言。因为他不需要在一个语言环境下运行，而只需要支持它的浏览器即可。它的主要目的是，验证发往服务器端的数据、增加Web互动、加强用户体验度等。

## 二、JavaScript特点

**1、解释型语言，动态类型、弱类型的语言**

在运行的时候将程序翻译成机器语言，在运行期间才去做数据类型检测；解释型语言在运行时需要解释器，JS的解释器被称为JavaScript引擎，是浏览器的一部分。

**2、基于对象的语言**

**3、基于事件驱动**

**4、跨平台性**

JavaScript是浏览器的编程语言，只要有浏览器存在就能够解释JavaScript，与操作系统无关，因此具有跨平台性。

## 三．JavaScript历史

**引子**

大概在1992年，有一家公司Nombas开发一种叫做C--(C-minus-minus,简称Cmm)的嵌入式脚本语言。后应觉得名字比较晦气，最终改名为ScripEase。而这种可以嵌入网页中的脚本的理念将成为因特网的一块重要基石。

**诞生**

1995年，当时工作在Netscape(网景)公司的布兰登(Brendan Eich)为解决类似于“向服务器提交数据之前验证”的问题。在Netscape Navigator 2.0与Sun公司联手开发一个称之为LiveScript的脚本语言。为了营销便利，之后更名为JavaScript(目的是在Java这课大树下好乘凉)。

**邪恶的后来者**

因为JavaScript 1.0如此成功，所以微软也决定进军浏览器，发布了IE 3.0 并搭载了一个JavaScript的克隆版，叫做JScript（这样命名是为了避免与Netscape潜在的许可纠纷），并且也提供了自己的VBScript。

**标准的重要**

在微软进入后，有3种不同的JavaScript版本同时存在：Netscape Navigator 3.0中的JavaScript、IE中的JScript以及CEnvi中的ScriptEase。与C和其他编程语言不同的是，JavaScript并没有一个标准来统一其语法或特性，而这3种不同的版本恰恰突出了这个问题。随着业界担心的增加，这个语言标准化显然已经势在必行。

**ECMA**

1997年，JavaScript 1.1作为一个草案提交给欧洲计算机制造商协会（ECMA）。第39技术委员会（TC39）被委派来“标准化一个通用、跨平台、中立于厂商的脚本语言的语法和语义”（http://www.ecma-international.org/memento/TC39.htm）。由来自Netscape、Sun、微软、Borland和其他一些对脚本编程感兴趣的公司的程序员组成的TC39锤炼出了ECMA-262，该标准定义了叫做ECMAScript的全新脚本语言。

**灵敏的微软、迟钝的网景**

虽然网景开发了JavaScript并首先提交给ECMA标准化，但因计划改写整个浏览器引擎的缘故，网景晚了整整一年才推出“完全遵循ECMA规范”的JavaScript1.3。而微软早在一年前就推出了“完全遵循ECMA规范”的IE4.0。这导致一个直接恶果：JScript成为JavaScript语言的事实标准。

**标准的发展**

在接下来的几年里，国际标准化组织及国际电工委员会（ISO/IEC）也采纳ECMAScript作为标准（ISO/IEC-16262）。从此，Web浏览器就开始努力（虽然有着不同程度的成功和失败）将ECMAScript作为JavaScript实现的基础。

**山寨打败原创**

JScript成为JavaScript语言的事实标准，加上Windows绑定着IE浏览器，几乎占据全部市场份额，因此，1999年之后，所有的网页都是基于JScript来开发的。而JavaScript1.x变成可怜的兼容者。

**网景的没落与火狐的崛起**

网景在微软强大的攻势下，1998年全面溃败。但，星星之火可以燎原。同年成立Mozilla项目中Firefox(火狐浏览器)在支持JavaScript方面无可比拟，在后来的时间里一步步蚕食IE的市场，成为全球第二大浏览器。

**谷歌的野心**

Google Chrome，又称Google浏览器，是一个由Google（谷歌）公司开发的开放原始码网页浏览器。他以简洁的页面，极速的浏览，一举成为全球第三大浏览器。随着移动互联网的普及，嵌有Android系统的平板电脑和智能手机，在浏览器这块将大有作为。

**苹果的战略**

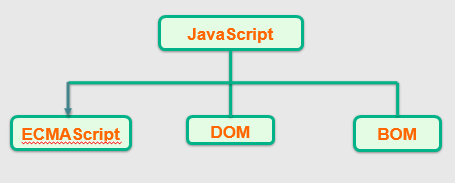
Safari浏览器是苹果公司各种产品的默认浏览器，在苹果的一体机(iMac)、笔记本(Mac)、MP4(ipod)、iphone(智能手机)、ipad(平板电脑)，并且在windows和Linux平台都有相应版本。目前市场份额全球第四，但随着苹果的产品不断的深入人心，具有称霸之势。

**幸存者**

Opera的全球市场份额第五，2%左右。它的背后没有财力雄厚的大公司，但它从“浏览器大战”存活下来的，有着非常大的潜力。

## 四、JavaScript组成

虽然JavaScript和ECMAScript通常被人们用来表达相同的含义，但JavaScript的含义却比ECMA-262中规定的要多得多。一个完整的JavaScript应该由下列三个不同的部分组成。



**1、ECMAScript介绍**

由ECMAScript-262定义的ECMAScript与Web浏览器没有依赖关系。ECMAScript定义的只是这门语言的基础，而在此基础之上可以构建更完善的脚本语言。我们常见的Web浏览器只是ECMAScript实现可能的宿主环境之一。

既然他不依赖于Web浏览器，那么他还在哪些环境中寄宿呢？比如：ActionScript、ScriptEase等。而他的组成部分有：语法、类型、语句、关键字、保留字、操作符、对象等。ECMAScript目前有五个版本，1、2、3、4、5版本，这里不再进行详细探讨。有兴趣的同学，可以搜索查阅。

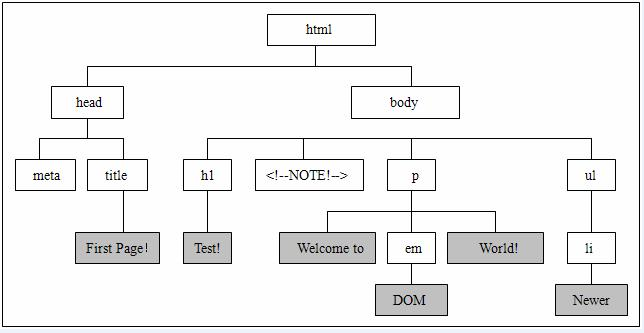
**Web浏览器对ECMAScript的支持**

到了2008年，五大主流浏览器(IE、Firefox、Safari、Chrome、Opera)全部做到了与ECMA-262兼容。其中，只有Firefox力求做到与该标准的第4版兼容。以下是支持表。

|  |  |
| --- | --- |
| 浏 览 器 | ECMAScript兼容性 |
| Netscape Navigator 2 | ---- |
| Netscape Navigator 3 | ---- |
| Netscape Navigator 4 -- 4.05 | ---- |
| Netscape Navigator 4.06 -- 4.79 | 第1版 |
| Netscape 6+ (Mozilla 0.6.0+) | 第3版 |
| Internet Explorer 3 | ---- |
| Internet Explorer 4 | ---- |
| Internet Explorer 5 | 第1版 |
| Internet Explorer 5.5 -- 7 | 第3版 |
| Internet Explorer 8 | 第3.1版(不完全兼容) |
| Internet Explorer 9 | 第5版 |
| Opera 6 - 7.1 | 第2版 |
| Opera 7.2+ | 第3版 |
| Opera 11+ | 第5版 |
| Safari 3+ | 第3版 |
| Firefox 1--2 | 第3版 |
| Firefox 3/4/5/6/7/8/9 | 第3/5版 |

**2、文档对象模型(DOM)**

文档对象模型(DOM，Document Object Model)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)。描述了处理网页内容的方法和接口 ；DOM 将把整个页面规划成由节点层级构成的文档。



DOM有三个级别（分别是DOM-1,DOM-2和DOM-3级别），每个级别都会新增很多内容模块和标准(有兴趣可以搜索查询)。以下是主流浏览器对DOM支持的情况：

|  |  |
| --- | --- |
| 浏 览 器 | DOM兼容性 |
| Netscape Navigator 1 -- 4.x | ---- |
| Netscape Navigator 6+(Mozilla 0.6.0+) | 1级、2级(几乎全部)、3级(部分) |
| Internet Explorer 2 -- 4.x | ---- |
| Internet Explorer 5 | 1级(最小限度) |
| Internet Explorer 5.5 -- 7 | 1级(几乎全部) |
| Opera 1 -- 6 | ---- |
| Opera 7 -- 8.x | 1级(几乎全部)、2级(部分) |
| Opera 9+ | 1级、2级(几乎全部)、3级(部分) |
| Safari 1.0x | 1级 |
| Safari 2+ | 1级、2级(部分) |
| Chrome 0.2+ | 1级、2级(部分) |
| Firefox 1+ | 1级、2级(几乎全部)、3级(部分) |

**3、浏览器对象模型(BOM)**

访问和操作浏览器窗口的浏览器对象模型(BOM，Browser Object Model)。开发人员使用BOM可以控制浏览器显示页面以外的部分。描述了与浏览器进行交互的方法和接口，主要处理浏览器窗口和框架，不过通常浏览器特定的 JavaScript 扩展都被看做 BOM 的一部分。这些扩展包括：弹出新的浏览器窗口、移动、关闭浏览器窗口以及调整窗口大小、提供 Web 浏览器详细信息的定位对象 、提供用户屏幕分辨率详细信息的屏幕对象 、对 cookie 的支持等。

4、**JavaScript版本**

身为Netscape“继承人”的Mozilla公司，是目前唯一沿用最初的JavaScript版本编号的浏览器开发商。在网景把JavaScript转手给Mozilla项目的时候，JavaScript在浏览器中最后的版本号是1.3。后来，随着Mozilla继续开发，JavaScript版本号逐步递增。

|  |  |
| --- | --- |
| 浏 览 器 | JavaScript版本 |
| Netscape Navigator 2 | 1.0 |
| Netscape Navigator 3 | 1.1 |
| Netscape Navigator 4 | 1.2 |
| Netscape Navigator 4.06 | 1.3 |
| Netscape 6+ (Mozilla 0.6.0+) | 1.5 |
| Firefox 1 | 1.5 |
| Firefox 1.5 | 1.6 |
| Firefox 2 | 1.7 |
| Firefox 3 | 1.8 |
| Firefox 3.1+ | 1.9 |

## 五、JavaScript在网页中的插入方法

JavaScript在网页中的插入方法有两种：1、在网页上直接插入JS代码2、引入JS外部文件。

**1、<script>标记**

<script>xxx</script>标签，是用于在html页面中插入js的主要方法。它主要有以下几个属性：

|  |  |  |
| --- | --- | --- |
| 属性名 | 属性值 | 说明 |
| type | type="text/javascript" | 必需。表示代码使用的脚本语言的内容类型。 |
| src |  | 可选。表示包含要执行代码的外部文件。 |
| charset |  | 可选。表示通过src属性指定的字符集。由于大多数浏览器忽略它，所以很少有人用它。 |
| 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。由于大多数浏览器不支持，故很少用。 |
| ~~language~~ |  | 已废弃。原来用于代码使用的脚本语言。由于大多数浏览器忽略它，所以不要用了。 |

代码示例：

<script type="text/javascript">

alert('欢迎来到JavaScript世界！');

</script>

<script type="text/javascript">

document.write('欢迎来到JavaScript世界！');

</script>

2、JS代码嵌入的一些问题

如果你想弹出一个</script>标签的字符串，那么浏览器会误解成JS代码已经结束了。解决的方法，就是把字符串分成两个部分，通过连接符‘+’来连接。

<script type="text/javascript">

alert('</scr'+'ipt>');

</script>

一般来说，JS代码越来越庞大的时候，我们最好把他另存为一个.js文件，通过src引入即可。它还具有维护性高、可缓存(加载一次，无需加载)、方便未来扩展的特点。

<script type="text/javascript" src="demo1.js"></script>

这样标签内就没有任何JS代码了。但要注意的是，虽然没有任何代码，也不能用单标签：

<script type="text/javascript" src="demo1.js" />

也不能在里面添加任何代码：

<script type="text/javascript" src="demo1.js">alert('我很可怜，执行不到！')</script>

**3、标签的位置**

按照常规，我们会把<script>标签存放到<head>...</head>之间。但有时也会放在body之间。如果把所有JS文件放在<head>元素中，意味着必须等到全部JS代码都被下载、解析、执行完成后，才开始呈现页面内容（浏览器在遇到<body>标签时才开始呈现内容）；对于那些需要很多JS代码的页面，会导致延迟且等待期间浏览器一片空白。

但又不能将所有的script放在body之后，因为有一些页面的效果的实现，是需要预先动态的加载一些js脚本。所以这些脚本应该放在<body>之前。

所以应该遵循：需要调用时才执行的js脚本放在<body>中，当页面被加载时需立即执行来完成页面效果的js代码应放在<head>中。

## 六、如何调试JavaScript代码

（1）主流浏览器现在都带有一个运行JavaScript的控制台（IE、Chrome、Firefox），在控制台中输出日志console.log ——在控制台显示

（2）alert——弹出对话框

（3）document.write——在浏览器页面内显示

# JavaScript语法基础

## 学习要点：

一、JavaScript语法构成

二、JavaScript关键词、保留字

三、JavaScript变量

四、JavaScript的6种数据类型

五、JavaScript运算符

六、JavaScript流程控制语句

任何语言的核心都必然会描述这门语言最基本的工作原理。而JavaScript的语言核心就是ECMAScript，而目前用的最普遍的是第3版，我们就主要以这个版本来讲解。

## 语法构成

1、区分大小写

ECMAScript中的一切，包括变量、函数名和操作符都是区分大小写的。例如：text和Text表示两种不同的变量。

2、标识符

所谓标识符，就是指变量、函数、属性的名字，或者函数的参数。标识符可以是下列格式规则组合起来的一或多个字符：

(1)第一字符必须是一个字母、下划线\_或一个美元符号$。

(2)其他字符可以是字母、下划线、美元符号或数字。

(3)不能把关键字、保留字、true、false和null作为标识符。

例如：myName、book123等

3、注释

ECMAScript使用C风格的注释，包括单行注释和块级注释。

// 单行注释

/\*

\* 这是一个多行

\* 注释

\*/

4、语句块

就是用 {} 包裹的一些js代码

{

var yourName;

var \_price=250;

}

5、直接量(字面量literal)

所有直接量(字面量)，就是程序中直接显示出来的数据值。

100 //数字字面量

'张三' //字符串字面量

false //布尔字面量

/js/gi //正则表达式字面量

null //对象字面量

在ECMAScript第3版中，像数组字面量和对象字面量的表达式也是支持的，如下：

{x:1, y:2} //对象字面量表达式

[1,2,3,4,5]或

[‘a’,123,3,’abc’,false] //数组字面量表达式

## 二、关键字和保留字

ECMAScript-262描述了一组具有特定用途的关键字，一般用于控制语句的开始或结束，或者用于执行特定的操作等。关键字也是语言保留的，不能用作标识符。

ECMAScript全部关键字

|  |  |  |  |
| --- | --- | --- | --- |
| break | else | new | var |
| case | finally | return | void |
| catch | for | switch | while |
| continue | function | this | with |
| default | if | throw |  |
| delete | in | try |  |
| do | instanceof | typeof |  |

ECMAScript-262还描述了另一组不能用作标识符的保留字。尽管保留字在JavaScript中还没有特定的用途，但它们很有可能在将来被用作关键字。

ECMAScript-262第3版定义的全部保留字

|  |  |  |  |
| --- | --- | --- | --- |
| abstract | enum | int | short |
| boolean | export | interface | static |
| byte | extends | long | super |
| char | final | native | synchronized |
| class | float | package | throws |
| const | goto | private | transient |
| debugger | implements | protected | volatile |
| double | import | public |  |

## 三．变量

1、变量声明

所谓变量，就是可以初始化后可以再次改变的量。声明变量时要使用var操作符（var是关键），后面跟一个变量名（变量名是标识符）。JavaScript采用弱类型的变量形式，因此声明时可以不需要变量类型。

var可以省略，但是不建议省略

var box;

alert(box);

这句话定义了box变量，但没有对它进行初始化（也就是没有给变量赋值）。这时，系统会给它一个特殊的值 -- **undefined（表示未定义）。**

var box= '张三';

alert(box);

2、命名规则

(1)第一字符必须是一个字母、下划线\_或一个美元符号$。

(2)其他字符可以是字母、下划线、美元符号或数字。

(3)不能把关键字、保留字、true、false和null作为标识符。

3、变量命名的一些注意事项：

（1）所谓变量，就是可以初始化后可以再次改变的量。ECMAScript属于弱类型的语言，可以同时改变不同类型的量。(PS：虽然可以改变不同类型的量，但这样做对于后期维护带来困难，而且性能也不高，导致成本很高！)

var boxString = '张三';

boxString = 100;

alert(boxString);

（2）重复的使用var声明一个变量，只不过是一个赋值操作，并不会报错。但这样的操作是比较二的，没有任何必要。

var box= '张三';

var box= 'Lee';

（3）还有一种变量不需要前面var关键字即可创建变量。这种变量和var的变量有一定的区别和作用范围，变量没有声明而直接赋值得话，默认会作为一个新的全局变量，要尽量避免使用全局变量，我们会在作用域那一节详细探讨。

box= '张三';

（4）当你想声明多个变量的时候，可以在一行或者多行操作。

var box= '张三';var age= 100;

（5）而当你每条语句都在不同行的时候，你可以省略分号。(PS：这是ECMAScript支持的，但绝对是一个非常不好的编程习惯，切记不要)。

var box= '张三'

var age= 100

alert(box)

（6）可以使用一条语句定义多个变量，只要把每个变量(初始化或者不初始化均可)用逗号分隔开即可，为了可读性，每个变量，最好另起一行，并且第二变量和第一变量对齐(PS：这些都不是必须的)。

var box= '张三',

age = 28,

height;

## 四、JavaScript的6中数据类型

ECMAScript中有5种简单数据类型：String、number、boolean 、null和Undefined。还有一种复杂数据类型——Object。ECMAScript不支持任何创建自定义类型的机制，所有值都成为以上6中数据类型之一。

在学习数据类型之前先来看一个typeof**操作符**是用来检测变量的数据类型。对于值或变量使用typeof操作符会返回如下字符串。

|  |  |
| --- | --- |
| 字符串 | 描述 |
| undefined | 未定义 |
| boolean | 布尔值 |
| string | 字符串 |
| number | 数值 |
| object | 对象或null |
| function | 函数 |

var box = '张三';

alert(typeof box);

alert(typeof '张三');

typeof操作符可以操作变量，也可以操作字面量。虽然也可以这样使用：typeof(box)，但，typeof是操作符而非内置函数。(PS：函数在ECMAScript中是对象，不是一种数据类型。所以，使用typeof来区分function和object是非常有必要的。)

#### 1、字符串类型（String）

String类型用于表示由于零或多个字符组成的字符序列，即字符串。字符串可以由双引号(")或单引号(')表示。

var box = 'Lee';

var box = "Lee";

PS：在某些其他语言(PHP)中，单引号和双引号表示的字符串解析方式不同，而ECMAScript中，这两种表示方法没有任何区别。但要记住的是，必须成对出现，不能穿插使用，否则会出错。

var box = '张三"; //出错

String类型包含了一些特殊的字符字面量，也叫转义序列。

|  |  |
| --- | --- |
| 字面量 | 含义 |
| \n | 换行 |
| \t | 制表 |
| \b | 空格 |
| \r | 回车 |
| \f | 进纸 |
| \\ | 斜杠 |
| \' | 单引号 |
| \" | 双引号 |
| \xnn | 以十六进制代码nn表示的一个字符(0~F)。例：\x41 |
| [\](file:///\\\\unnn)unnn | 以十六进制代码nnn表示的一个Unicode字符(0~F)。例：\u03a3 |

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。

var box = 'Mr.';

box = box + ' Lee';

**补充：**

（1）toString()方法可以把值转换成字符串。

var box = 11;

var box = true;

alert(typeof box.toString());

（2）toString()方法一般是不需要传参的，但在数值转成字符串的时候，可以传递进制参数。

var box = 10;

alert(box.toString()); //10，默认输出

alert(box.toString(2)); //1010，二进制输出

alert(box.toString(8)); //12，八进制输出

alert(box.toString(10)); //10，十进制输出

alert(box.toString(16)); //a，十六进制输出

（3）如果在转型之前不知道变量是否是null或者undefined的情况下，我们还可以使用转型函数String()，这个函数能够将任何类型的值转换为字符串。

var box = null;

alert(String(box));

PS：如果值有toString()方法，则调用该方法并返回相同的结果；如果是null或者undefined，则返回"null"或者"undeinfed"。

#### 2、数值类型（number）

Number类型包含两种数值：整型和浮点型。为了支持各种数值类型，ECMA-262定义了不同的数值字面量格式。

最基本的数值字面量是十进制整数。

var box = 100; //十进制整数

八进制数值字面量，(以8为基数)，前导必须是0，八进制序列(0~7)。

var box = 070; //八进制，56

var box = 079; //无效的八进制，自动解析为79

var box = 08; //无效的八进制，自动解析为8

十六进制字面量前面两位必须是0x，后面是(0~9及A~F)。

var box = 0xA; //十六进制，10

var box = 0x1f; //十六进制，31

浮点类型，就是该数值中必须包含一个小数点，并且小数点后面必须至少有一位数字。

var box = 3.8;

var box = 0.8;

var box = .8; //有效，但不推荐此写法

**浮点类型注意事项（补充）：**

（1）内存空间

由于保存浮点数值需要的内存空间比整型数值大两倍，因此ECMAScript会自动将可以转换为整型的浮点数值转成为整型。

var box = 8.; //小数点后面没有值，转换为8

var box = 12.0; //小数点后面是0，转成为12

（2）科学计数法

对于那些过大或过小的数值，可以用科学技术法来表示(e表示法)。用e表示该数值的前面10的指数次幂。

var box = 4.12e9; //即4120000000

var box = 0.00000000412; //即4.12e-9

（3）精度

虽然浮点数值的最高精度是17位小数，但算术运算中可能会不精确。由于这个因素，做判断的时候一定要考虑到这个问题(比如使用整型判断)。

alert(0.1+0.2); //0.30000000000000004

（4）范围

浮点数值的范围在：Number.MIN\_VALUE ~ Number.MAX\_VALUE之间。

alert(Number.MIN\_VALUE); //最小值

alert(Number.MAX\_VALUE); //最大值

如果超过了浮点数值范围的最大值或最小值，那么就先出现Infinity(正无穷)或者-Infinity(负无穷)。

var box = 100e1000; //超出范围，Infinity

var box = -100e1000; //超出范围，-Infinity

也可能通过Number.POSITIVE\_INFINITY和Number.NEGATIVE\_INFINITY得到Infinity(正无穷)及-Infinity(负无穷)的值。

alert(Number.POSITIVE\_INFINITY); //Infinity(正无穷)

alert(Number.NEGATIVE\_INFINITY); //-Infinity(负无穷)

要想确定一个数值到底是否超过了规定范围，可以使用isFinite()函数。如果没有超过，返回true，超过了返回false。

var box = 100e1000;

alert(isFinite(box)); //返回false或者true

（5）NaN

NaN，即非数值(Not a Number)是一个特殊的值，这个数值用于表示一个本来要返回数值的操作数未返回数值的情况(这样就不会抛出错误了)。比如，在其他语言中，任何数值除以0都会导致错误而终止程序执行。但在ECMAScript中，会返回出特殊的值，因此不会影响程序执行。

var box = 0 / 0; //NaN

var box = 12 / 0; //Infinity

var box = 12 / 0 \* 0; //NaN

可以通过Number.NaN得到NaN值，任何与NaN进行运算的结果均为NaN，NaN与自身不相等(NaN不与任何值相等)。

alert(Number.NaN); //NaN

alert(NaN+1); //NaN

alert(NaN == NaN) //false

ECMAScript提供了isNaN()函数，用来判断这个值到底是不是NaN。isNaN()函数在接收到一个值之后，会尝试将这个值转换为数值，然后再进行判断。

alert(isNaN(NaN)); //true

alert(isNaN(25)); //false，25是一个数值

alert(isNaN('25')); //false，'25'是一个字符串数值，可以转成数值

alert(isNaN('Lee')); //true，'Lee'不能转换为数值

alert(isNaN(true)); //false true可以转成成1

isNaN()函数也适用于对象。在调用isNaN()函数过程中，首先会调用valueOf()方法，然后确定返回值是否能够转换成数值。如果不能，则基于这个返回值再调用toString()方法，再测试返回值。

var box = {

toString : function () {

return '123'; //可以改成return 'Lee'查看效果

}

};

alert(isNaN(box)); //false

(6)有3个函数可以把非数值转换为数值：Number()、parseInt()和parseFloat()。Number()函数是转型函数，可以用于任何数据类型，而另外两个则专门用于把字符串转成数值。

alert(Number(true)); //1，Boolean类型的true和false分别转换成1和0

alert(Number(25)); //25，数值型直接返回

alert(Number(null)); //0，空对象返回0

alert(Number(undefined)); //NaN，undefined返回NaN

如果是字符串，应该遵循一下规则：

只包含数值的字符串，会直接转成成十进制数值，如果包含前导0，即自动去掉。

alert(Number('456')); //456

alert(Number('070')); //70

只包含浮点数值的字符串，会直接转成浮点数值，如果包含前导和后导0，即自动去掉。

alert(Number('08.90')); //8.9

如果字符串是空，那么直接转成成0。

alert(Number('')); //0

如果不是以上三种字符串类型，则返回NaN。

alert('Lee123'); //NaN

如果是对象，首先会调用valueOf()方法，然后确定返回值是否能够转换成数值。如果转换的结果是NaN，则基于这个返回值再调用toString()方法，再测试返回值。

var box = {

toString : function () {

return '123'; //可以改成return 'Lee'查看效果

}

};

alert(Number(box)); //123

由于Number()函数在转换字符串时比较复杂且不够合理，因此在处理整数的时候更常用的是parseInt()。

alert(parsetInt('456Lee')); //456，会返回整数部分

alert(parsetInt('Lee456Lee')); //NaN，如果第一个不是数值，就返回NaN

alert(parseInt('12Lee56Lee')); //12，从第一数值开始取，到最后一个连续数值结束

alert(parseInt('56.12')); //56，小数点不是数值，会被去掉

alert(parseInt('')); //NaN，空返回NaN

parseInt()除了能够识别十进制数值，也可以识别八进制和十六进制。

alert(parseInt('0xA')); //10，十六进制

alert(parseInt('070')); //56，八进制

alert(parseInt('0xALee')); //10，十六进制，Lee被自动过滤掉

ECMAScript为parseInt()提供了第二个参数，用于解决各种进制的转换。

alert(parseInt('0xAF')); //175，十六进制

alert(parseInt('AF',16)); //175，第二参数指定十六进制，可以去掉0x前导

alert(parseInt('AF')); //NaN，理所当然

alert(parseInt('101010101',2)); //314，二进制转换

alert(parseInt('70',8)) //56，八进制转换

parseFloat()是用于浮点数值转换的，和parseInt()一样，从第一位解析到非浮点数值的位置。

alert(parseFloat('123Lee')); //123，去掉不是别的部分

alert(parseFloat('0xA')); //0，**不认十六进制**

alert(parseFloat('123.4.5')); //123.4，**只认一个小数点**

alert(parseFloat('0123.400')); //123.4，去掉前后导

alert(parseFloat('1.234e7')); //12340000，**把科学技术法转成普通数值**

#### 3、布尔类型（boolean）

boolean类型有两个值(字面量)：true和false。而true不一定等于1，false不一定等于0。JavaScript是区分大小写的，True和False或者其他都不是Boolean类型的值。

var box = true;

alert(typeof box);

虽然boolean类型的字面量只有true和false两种，但ECMAScript中所有类型的值都有与这两个boolean值等价的值。要将一个值转换为其对应的boolean值，可以使用转型函数Boolean()。

var hello = 'Hello World!';

var hello2 = Boolean(hello);

alert(typeof hello);

上面是一种显示转换，属于强制性转换。而实际应用中，还有一种隐式转换。比如，在if条件语句里面的条件判断，就存在隐式转换。

var hello = 'Hello World!';

if (hello) {

alert('如果条件为true，就执行我这条！');

} else {

alert('如果条件为false，就执行我这条！');

}

以下是其他类型转换成Boolean类型规则

|  |  |  |
| --- | --- | --- |
| 数据类型 | 转换为true的值 | 转换为false的值 |
| String | 任何非空字符串 | 空字符串 |
| Number | 任何非零数字值(包括无穷大) | 0和NaN |
| Object | 任何对象 | null |
| Undefined |  | undefined |

#### 4、Undefined

Undefined类型只有一个值，即特殊的undefined。在使用var声明变量，但没有对其初始化时，这个变量的值就是undefined。

var box;

alert(box);

PS：我们没有必要显式的给一个变量赋值为undefined，因为没有赋值的变量会隐式的(自动的)赋值为undefined；而undefined主要的目的是为了用于比较，ECMAScript第3版之前并没有引入这个值，引入之后为了正式区分空对象与未经初始化的变量。

未初始化的变量与根本不存在的变量(未声明的变量)也是不一样的。

var box;

alert(age); //age is not defined

PS：如果typeof box，typeof age都返回的undefined。从逻辑上思考，他们的值，一个是undefined，一个报错；他们的类型，却都是undefined。所以，我们在定义变量的时候，尽可能的不要只声明，不赋值。

#### 5、NULL

Null类型是一个只有一个值的数据类型，即特殊的值null。它表示一个空对象引用，而typeof操作符检测null会返回object。

var box = null;

alert(typeof box);

如果定义的变量准备在将来用于保存对象，那么最好将该变量初始化为null。这样，当检查null值就知道变量是否已经分配了对象引用了。

var box = null;

if (box != null) {

alert('box对象已存在！');

}

有个要说明的是：undefined是派生自null的，因此ECMA-262规定对它们的相等性测试返回true。

alert(undefined == null);

由于undefined和null两个值的比较是相等的，所以，未初始化的变量和赋值为null的变量会相等。这时，可以采用typeof变量的类型进行比较。但，建议还是养成编码的规范，不要忘记初始化变量。

var box;

var car = null;

alert(typeof box == typeof car)

#### 6、对象类型

ECMAScript中的对象其实就是一组数据和功能（函数）的集合。对象可以通过执行new操作符后跟要创建的对象类型的名称来创建。

var box = new Object();

Object()是对象构造，如果对象初始化时不需要传递参数，可以不用写括号，但这种方式我们是不推荐的。

var box = new Object;

Object()里可以任意传参，可以传数值、字符串、布尔值等。而且，还可以进行相应的计算。

var box = new Object(2); //Object类型，值是2

var age = box + 2; //可以和普通变量运算

alert(age); //输出结果，转型成Number类型了

由于在实际使用中对string,number,boolean的操作比较多，JavaScript进行了进一步的封装，提供了内置的String(),Number(),Boolean()对象，可以使用new操作符来创建相应类型的对象。

var box = new Number(5); //new String('Lee')、new Boolean(true)

alert(typeof box); //Object类型

PS：面向对象是JavaScript课程的重点，这里我们只是简单做个介绍。详细的课程将在以后的章节继续学习。

## 五、运算符

ECMA-262描述了一组用于操作数据值的运算符，包括一元运算符、布尔运算符、算术运算符、关系运算符、三元运算符、位运算符及赋值运算符。ECMAScript中的运算符适用于很多值，包括字符串、数值、布尔值、对象等。

#### 1、什么是表达式

表达式是ECMAScript中的一个“短语”，解释器会通过计算把它转换成一个值。最简单的表达式是字面量或者变量名。例如：

5.96 //数值字面量

'Lee' //字符串字面量

true //布尔值字面量

null //空值字面量

/Java/ //正则表达式字面量  
 {x:1, y:2} //对象字面量、对象表达式

[1,2,3] //数组字面量、数组表达式

function(n) {return x+y;} //函数字面量、函数表达式

box //变量

当然，还可以通过合并简单的表达式来创建复杂的表达式。比如：

box + 5.96 //加法运算的表达式

typeof(box) //查看数据类型的表达式

box > 8 //逻辑运算表达式

通过上面的叙述，我们得知，单一的字面量和组合字面量的运算符都可称为表达式。

#### 2、算术运算符

ECMAScript定义了5个算术运算符，加减乘除求模(取余)。如果在算术运算的值不是数值，那么后台会先使用Number()转型函数将其转换为数值(隐式转换)。

（1）加法

var box = 1 + 2; //等于3

var box = 1 + NaN; //NaN，只要有一个NaN就为NaN

var box = Infinity + Infinity; //Infinity

var box = -Infinity + -Infinity; //-Infinity

var box = Infinity + -Infinity; //NaN，正无穷和负无穷相加等NaN

var box = 100 + '100'; //100100，字符串连接符，有字符串就不是加法

var box = '您的年龄是：' + 10 + 20; //您的年龄是：1020，被转换成字符串

var box = 10 + 20 + '是您的年龄'; //30是您的年龄，没有被转成字符串

var box = '您的年龄是：' + (10 + 20); //您的年龄是：30，没有被转成字符串

（2）减法

var box = 100 - 70; //等于30

var box = -100 - 70 //等于-170

var box = -100 - -70 //-30，一般写成-100 - (-70)比较清晰

var box = 1 - NaN; //NaN，只要有一个NaN就为NaN

var box = Infinity - Infinity; //NaN

var box = -Infinity - -Infinity; //NaN

var box = Infinity - -Infinity; //Infinity

var box = -Infinity - Infinity; //-Infinity

var box = 100 - true; //99，true转成数值为1

var box = 100 - ''; //100，''转成了0

var box = 100 - '70'; //30，'70'转成了数值70

var box = 100 - null; //100，null转成了0

var box = 100 - 'Lee'; //NaN，Lee转成了NaN

（3）乘法

var box = 100 \* 70; //7000

var box = 100 \* NaN; //NaN，只要有一个NaN即为NaN

var box = Infinity \* Infinity; //Infinity

var box = -Infinity \* Infinity ; //-Infinity

var box = -Infinity \* -Infinity ; //Infinity

var box = 100 \* true; //100，true转成数值为1

var box = 100 \* ''; //0，''转成了0

var box = 100 \* null; //0，null转成了0

var box = 100 \* 'Lee'; //NaN，Lee转成了NaN

（4）除法

var box = 100 / 70; //1.42....

var box = 100 / NaN; //NaN

var box = Infinity / Infinity; //NaN

var box = -Infinity / Infinity ; //NaN

var box = -Infinity / -Infinity; //NaN

var box = 100 / true; //100，true转成1

var box = 100 / ''; //Infinity，

var box = 100 / null; //Infinity，

var box = 100 / 'Lee'; //NaN

（5）求模

var box = 10 % 3; //1，余数为1

var box = 100 % NaN; //NaN

var box = Infinity % Infinity; //NaN

var box = -Infinity % Infinity ; //NaN

var box = -Infinity % -Infinity; //NaN

var box = 100 % true; //0

var box = 100 % ''; //NaN

var box = 100 % null; //NaN

var box = 100 % 'Lee'; //NaN

#### 3、一元运算符

只能操作一个值的运算符叫做一元运算符。

（1）递增++和递减--

var box = 100;

++box; //把box累加一个1，相当于box = box+1

--box; //把box累减一个1，相当于box = box-1

box++; //同上

box--; //同上

（2）前置和后置的区别

在没有赋值操作，前置和后置是一样的。但在赋值操作时，如果递增或递减运算符前置，那么前置的运算符会先累加或累减再赋值，如果是后置运算符则先赋值再累加或累减。

var box = 100;

var age = ++box; //age值为101

var height = box++; //height值为101

（3）其他类型应用一元运算符的规则

var box = '89'; box++; //90，数值字符串自动转换成数值

var box = 'ab'; box++; //NaN，字符串包含非数值转成NaN

var box = false; box++; //1，false转成数值是0，累加就是1

var box = 2.3; box++; //3.3，直接加1

var box = { //1，不设置toString或valueOf即为NaN

toString : function() {

return 1;

}

}; box++;

（4）加和减运算符

加运算规则如下：

var box = 100; +box; //100，对于数值，不会产生任何影响

var box = '89'; +box; //89，数值字符串转换成数值

var box = 'ab'; +box; //NaN，字符串包含非数值转成NaN

var box = false; +box; //0，布尔值转换成相应数值

var box = 2.3; +box; //2.3，没有变化

减运算规则如下：

var box = 100; -box; //-100，对于数值，直接变负

var box = '89'; -box; //-89，数值字符串转换成数值

var box = 'ab'; -box; //NaN，字符串包含非数值转成NaN

var box = false; -box; //0，布尔值转换成相应数值

var box = 2.3; -box; //-2.3，没有变化

var box = { //-1，不设置toString或valueOf即为NaN

toString : function() {

return 1;

}

}; -box;

加法和减法运算符一般用于算术运算，也可向上面进行类型转换。

#### 4、关系运算符

用于进行比较的运算符称作为关系运算符：小于(<)、大于(>)、小于等于(<=)、大于等于(>=)、相等(==)、不等(!=)、全等(恒等)(===)、不全等(不恒等)(!==)

和其他运算符一样，当关系运算符操作非数值时要遵循一下规则：

（1）两个操作数都是数值，则数值比较；

（2）两个操作数都是字符串，则比较两个字符串对应的字符编码值；

（3）两个操作数有一个是数值，则将另一个转换为数值，再进行数值比较；

（4）两个操作数有一个是对象，则先调用valueOf()方法或toString()方法，再用结果比较；

var box = 3 > 2; //true

var box = 3 > 22; //false

var box = '3' > 22; //false

var box = '3' > '22'; //true

var box = 'a' > 'b'; //false a=97,b=98

var box = 'a' > 'B'; //true B=66

var box = 1 > 对象; //false，如果有toString()或valueOf()则返回1>返回数的值

在相等和不等的比较上，如果操作数是非数值，则遵循一下规则：

(1)一个操作数是布尔值，则比较之前将其转换为数值，false转成0，true转成1；

(2)一个操作数是字符串，则比较之前将其转成为数值再比较；

(3)一个操作数是对象，则先调用valueOf()或toString()方法后再和返回值比较；

(4)不需要任何转换的情况下，null和undefined是相等的；

(5)一个操作数是NaN，则==返回false，!=返回true；并且NaN和自身不等；

(6)两个操作数都是对象，则比较他们是否是同一个对象，如果都指向同一个对象，则返回true，否则返回false。

(7)在全等和全不等的判断上，比如值和类型都相等，才返回true，否则返回false。

var box = 2 == 2; //true

var box = '2' == 2; //true，'2'会转成成数值2

var box = false == 0; //true，false转成数值就是0

var box = 'a' == 'A'; //false，转换后的编码不一样

var box = 2 == {}; //false，执行toString()或valueOf()会改变

var box = 2 == NaN; //false，只要有NaN，都是false

var box = {} == {}; //false，比较的是地址，每个新创建对象的引用地址都不同

var age = {};

var height = age;

var box = age == height; //true，引用地址一样，所以相等

var box = '2' === 2 //false，值和类型都必须相等

var box = 2 !== 2 //false，值和类型都相等了

**特殊值对比表**

|  |  |
| --- | --- |
| 表 达 式 | 值 |
| null == undefined | true |
| 'NaN' == NaN | false |
| 5 == NaN | false |
| NaN == NaN | false |
| false == 0 | true |
| true == 1 | true |
| true == 2 | false |
| undefined == 0 | false |
| null == 0 | false |
| '100' == 100 | true |
| '100' === 100 | false |

#### 5、逻辑运算符

逻辑运算符通常用于布尔值的操作，一般和关系运算符配合使用，有三个逻辑运算符：逻辑与(AND)、逻辑或(OR)、逻辑非(NOT)。

(1)逻辑与(AND) ：&&

var box = (5 > 4) && (4 > 3) //true，两边都为true，返回true

|  |  |  |
| --- | --- | --- |
| 第一个操作数 | 第二个操作数 | 结果 |
| true | true | true |
| true | false | false |
| false | true | false |
| false | false | false |

如果两边的操作数有一个操作数不是布尔值的情况下，与运算就不一定返回布尔值，此时，遵循已下规则：

(1)第一个操作数是对象，则返回第二个操作数；

(2)第二个操作数是对象，则第一个操作数返回true，才返回第二个操作数，否则返回false;

(3)有一个操作数是null，则返回null；

(4)有一个操作数是undefined，则返回undefined。

var box = 对象 && (5 > 4); //true，返回第二个操作数

var box = (5 > 4) && 对象; //[object Object]

var box = (3 > 4) && 对象; //false

var box = (5 > 4) && null; //null

逻辑与运算符属于短路操作，顾名思义，如果第一个操作数返回是false，第二个数不管是true还是false都返回的false。

var box = true && age; //出错，age未定义

var box = false && age; //false，不执行age了

(2)逻辑或(OR)：||

var box = (9 > 7) || (7 > 8); //true，两边只要有一边是true，返回true

|  |  |  |
| --- | --- | --- |
| 第一个操作数 | 第二个操作数 | 结果 |
| true | true | true |
| true | false | true |
| false | true | true |
| false | false | false |

如果两边的操作数有一个操作数不是布尔值的情况下，逻辑与运算就不一定返回布尔值，此时，遵循已下规则：

1. **第一个操作数是对象，则返回第一个操作数；**
2. **第一个操作数的求值结果为false，则返回第二个操作数；**
3. **两个操作数都是对象，则返回第一个操作数；**
4. **两个操作数都是null，则返回null；**
5. **两个操作数都是NaN，则返回NaN；**
6. **两个操作数都是undefined，则返回undefined；**

var box = 对象 || (5 > 3); //[object Object]

var box = (5 > 3) || 对象; //true

var box = 对象1 || 对象2; //[object Object]

var box = null || null; //null

var box = NaN || NaN; //NaN  
 var box = undefined || undefined; //undefined

和逻辑与运算符相似，逻辑或运算符也是短路操作。当第一操作数的求值结果为true，就不会对第二个操作数求值了。

var box = true || age; //true

var box = false || age; //出错，age未定义

我们可以利用逻辑或运算符这一特性来避免为变量赋null或undefined值。

var box = oneObject || twoObject; //把其中一个有效变量值赋给box

(3)逻辑非(NOT)：!

逻辑非运算符可以用于任何值。无论这个值是什么数据类型，这个运算符都会返回一个布尔值。它的流程是：先将这个值转换成布尔值，然后取反，规则如下：

1. 操作数是一个对象，返回false；
2. 操作数是一个空字符串，返回true；
3. 操作数是一个非空字符串，返回false；
4. 操作数是数值0，返回true；
5. 操作数是任意非0数值(包括Infinity)，false；
6. 操作数是null，返回true；
7. 操作数是NaN，返回true；
8. 操作数是undefined，返回true；

var box = !(5 > 4); //false

var box = !{}; //false

var box = !''; //true

var box = !'Lee'; //false

var box = !0; //true

var box = !8; //false

var box = !null; //true

var box = !NaN; //true

var box = !undefined; //true

使用一次逻辑非运算符，流程是将值转成布尔值然后取反。而使用两次逻辑非运算符就是将值转成成布尔值取反再取反，相当于对值进行Boolean()转型函数处理。

var box = !!0; //false

var box = !!NaN; //false

通常来说，使用一个逻辑非运算符和两个逻辑非运算符可以得到相应的布尔值，而使用三个以上的逻辑非运算符固然没有错误，但也没有意义。

#### 6、位运算符（选学）

位运算符有七种，分别是：位非NOT(~)、位与AND(&)、位或OR（|）、位异或XOR(^)、左移(<<)、有符号右移(>>)、无符号右移(>>>)。

var box = ~25; //-26

var box = 25 & 3; //1

var box = 25 | 3; //27

var box = 25 << 3; //200

var box = 25 >> 2; //6

var box = 25 >>> 2; //6

#### 7、赋值运算符

赋值运算符用等于号(=)表示，就是把右边的值赋给左边的变量。

var box = 100; //把100赋值给box变量

复合赋值运算符通过x=的形式表示，x表示算术运算符及位运算符。

var box = 100;

box = box +100; //200，自己本身再加100

这种情况可以改写为：

var box = 100；

box += 100; //200，+=代替box+100

除了这种+=加/赋运算符，还有其他的几种如下：

1. 乘/赋(\*=)
2. 除/赋(/=)
3. 模/赋(%=)
4. 加/赋(+=)
5. 减/赋(-=)
6. 左移/赋(<<=)
7. 有符号右移/赋(>>=)
8. 无符号有移/赋(>>>=)

#### 8、其他运算符

（1）字符串运算符

字符串运算符只有一个，即："+"。它的作用是将两个字符串相加。

规则：至少一个操作数是字符串即可。

var box = '100' + '100'; //100100

var box = '100' + 100; //100100

var box = 100 + 100; //200

（2）逗号运算符

逗号运算符可以在一条语句中执行多个操作。

var box = 100, age = 20, height = 178; //多个变量声明

var box = (1,2,3,4,5); //5，变量声明，将最后一个值赋给变量，不常用

var box = [1,2,3,4,5]; //[1,2,3,4,5]，数组的字面量声明

var box = { //[object Object]，对象的字面量声明

1 : 2,

3 : 4,

5 : 6

}

（3）三元条件运算符（条件运算符）

三元条件运算符其实就是后面将要学到的if语句的简写形式。

var box = 5 > 4 ? '对' : '错';

//对，5>4返回true则把'对'赋值给box，反之。相当于：

var box = ''; //初始化变量

if (5 > 4) { //判断表达式返回值

box = '对'; //赋值

} else {

box = '错'; //赋值

}

（4） new运算符：用来创建一个对象或生成一个对象的实例。

（5）typeof运算符：返回一个用来返回表示表达式的数据类型的字符串

（6）in运算符：测试一个对象是否存在一种属性

（7）instanceof运算符：返回一个布尔值，该值指示一个对象是否为特定类型的一个实例。

#### 9、运算符的优先级

在一般的运算中，我们不必考虑到运算符的优先级，因为我们可以通过圆括号来解决这种问题。比如：

var box = 5 - 4 \* 8; //-27

var box = (5 - 4) \* 8; //8

但如果没有使用圆括号强制优先级，我们必须遵循以下顺序：

|  |  |
| --- | --- |
| 运算符 | 描述 |
| . [] () | 对象成员存取、数组下标、函数调用等 |
| ++ -- ~ ! delete new typeof void | 一元运算符 |
| \* / % | 乘法、除法、去模 |
| + - + | 加法、减法、字符串连接 |
| << >> >>> | 移位 |
| < <= > >= instanceof | 关系比较、检测类实例 |
| == != === !== | 恒等(全等) |
| & | 位与 |
| ^ | 位异或 |
| | | 位或 |
| && | 逻辑与 |
| || | 逻辑或 |
| ?: | 三元条件 |
| = x= | 赋值、运算赋值 |
| , | 多重赋值、数组元素 |

## 六、JavaScript流程控制语句

ECMA-262规定了一组流程控制语句。语句定义了ECMAScript中的主要语法，语句通常由一个或者多个关键字来完成给定的任务。诸如：判断、循环、退出等。

#### 1、语句的定义

在ECMAScript中，所有的代码都是由语句来构成的。语句表明执行过程中的流程、限定与约定，形式上可以是单行语句，或者由一对大括号“｛｝”括起来的复合语句，在语法描述中，复合语句整体可以作为一个单行语句处理。

**语句的种类**

|  |  |  |
| --- | --- | --- |
| 类型 | 子类型 | 语法 |
| 声明语句 | 变量声明语句 | var box = 100; |
| 标签声明语句 | label : box; |
| 表达式语句 | 变量赋值语句 | box = 100; |
| 函数调用语句 | box(); |
| 属性赋值语句 | box.property = 100; |
| 方法调用语句 | box.method(); |
| 分支语句 | 条件分支语句 | if () {} else {} |
| 多重分支语句 | switch () { case n : ...}; |

|  |  |  |
| --- | --- | --- |
| 类型 | 子类型 | 语法 |
| 循环语句 | for | for (;;;) {} |
| for ... in | for ( x in x) {} |
| while | while () {}; |
| do ... while | do {} while (); |
| 控制结构 | 继续执行子句 | continue ; |
| 终端执行子句 | break ; |
| 函数返回子句 | return ; |
| 异常触发子句 | throw ; |
| 异常捕获与处理 | try {} catch () {} finally {} |
| 其他 | 空语句 | ; |
| with语句 | with () {} |

#### 2、if语句

if 语句即条件判断语句，一共有三种格式：

（1）if (条件表达式) 语句;

var box = 100;

if (box > 50) alert('box大于50'); //一行的if语句，判断后执行一条语句

var box = 100;

if (box > 50)

alert('box大于50'); //两行的if语句，判断后也执行一条语句

alert('不管怎样，我都能被执行到！');

var box = 100;

if (box < 50) {

alert('box大于50');

alert('不管怎样，我都能被执行到！');//用复合语句包含，判断后执行一条复合语句

}

对于if语句括号里的表达式，ECMAScript会自动调用Boolean()转型函数将这个表达式的结果转换成一个布尔值。如果值为true，执行后面的一条语句，否则不执行。

PS：if语句括号里的表达式如果为true，只会执行后面一条语句，如果有多条语句，那么就必须使用复合语句把多条语句包含在内。

PS2：推荐使用第一种或者第三种格式，一行的if语句，或者多行的if复合语句。这样就不会因为多条语句而造成混乱。

PS3：复合语句我们一般喜欢称作为：代码块。

（2）if (条件表达式) {语句;} else {语句;}

var box = 100;

if (box > 50) {

alert('box大于50'); //条件为true，执行这个代码块

} else {

alert('box小于50'); //条件为false，执行这个代码块

}

（3）if (条件表达式) {语句;} else if (条件表达式) {语句;} ... else {语句;}

var box = 100;

if (box >= 100) { //如果满足条件，不会执行下面任何分支

alert('甲');

} else if (box >= 90) {

alert('乙');

} else if (box >= 80) {

alert('丙');

} else if (box >= 70) {

alert('丁');

} else if (box >= 60) {

alert('及格');

} else { //如果以上都不满足，则输出不及格

alert('不及格');

}

#### 3、switch语句

switch语句是多重条件判断，用于多个值相等的比较。

var box = 1;

switch (box) { //用于判断box相等的多个值

case 1 :

alert('one');

break; //break;用于防止语句的穿透

case 2 :

alert('two');

break;

case 3 :

alert('three');

break;

default : //相当于if语句里的else，否则的意思

alert('error');

}

#### 4、do…while语句

do...while语句是一种先运行，后判断的循环语句。也就是说，不管条件是否满足，至少先运行一次循环体。

var box = 1; //如果是1，执行五次，如果是10，执行1次

do {

alert(box);

box++;

} while (box <= 5); //先运行一次，再判断

#### 5、while语句

while语句是一种先判断，后运行的循环语句。也就是说，必须满足条件了之后，方可运行循环体。

var box = 1; //如果是1，执行五次，如果是10，不执行

while (box <= 5) { //先判断，再执行

alert(box);

box++;

}

#### 6、for语句

for语句也是一种先判断，后运行的循环语句。但它具有在执行循环之前初始变量和定义循环后要执行代码的能力。

for (var box = 1; box <= 5 ; box++) { //第一步，声明变量var box = 1;

alert(box); //第二步，判断box <=5

} //第三步，alert(box)

//第四步，box++

//第五步，从第二步再来，直到判断为false

#### 7、for…in语句

for...in语句是一种精准的迭代语句，可以用来枚举对象的属性。

var box = { //创建一个对象

'name' : '张三', //键值对，左边是属性名，右边是值

'age' : 28,

'height' : 178

};

var box=new Array（）;或 var box=[10,2,a,bb,aa,false,true]

box=[10,2,a,bb,aa,false,true]

for (var p in box) { //列举出对象的所有属性

alert(p);

}

#### 8、break和continue语句

break和continue语句用于在循环中精确地控制代码的执行。其中，break语句会立即退出循环，强制继续执行循环体后面的语句。而continue语句退出当前循环，继续后面的循环。

for (var box = 1; box <= 10; box++) {

if (box == 5) break; //如果box是5，就退出循环

document.write(box);

document.write('<br />');

}

for (var box = 1; box <= 10; box++) {

if (box == 5) continue; //如果box是5，就退出当前循环

document.write(box);

document.write('<br />');

}

#### 9、with语句

with语句的作用是将代码的作用域设置到一个特定的对象中。

var box = { //创建一个对象

'name' : '张三', //键值对

'age' : 28,

'height' : 178

};

var n = box.name; //从对象里取值赋给变量

var a = box.age;

var h = box.height;

可以将上面的三段赋值操作改写成：

with (box) { //省略了box对象名

var n = name;

var a = age;

var h = height;

}

或者打印多条信息：

with( document){

write( “Java”);

write(‘Script’);

write(“的世界！”);

}

#### 10、try…catch语句

捕获异常。

try

{

//代码块

}

catch (exception){

//处理错误

}

finally{

//代码块

}

#### 11、block语句

用于组合0到多个语句。语句块用一对{}包含起来

{

语句1；

语句2；

语句3；

}