### ECMAScript简介

ECMAScript是一种由[Ecma国际](http://baike.baidu.com/item/Ecma%E5%9B%BD%E9%99%85" \t "http://baike.baidu.com/item/_blank)通过ECMA-262标准化的脚本[程序设计语言](http://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F%E8%AE%BE%E8%AE%A1%E8%AF%AD%E8%A8%80" \t "http://baike.baidu.com/item/_blank)。或者说ECMAScript实际上是一种脚本在语法和语义上的标准。

Ecma国际（Ecma International）是一家国际性会员制度的信息和电信标准组织。1994年之前，名为[欧洲计算机制造商协会](http://baike.baidu.com/item/%E6%AC%A7%E6%B4%B2%E8%AE%A1%E7%AE%97%E6%9C%BA%E5%88%B6%E9%80%A0%E5%95%86%E5%8D%8F%E4%BC%9A" \t "http://baike.baidu.com/item/_blank)（European Computer Manufacturers Association）。

js是运行在浏览器端的，是面向对象的脚本语言。

|  |  |
| --- | --- |
| **js是一门编程语言，编程的三要素** | |
| 数据 | 变量 → 数据类型 → 运算符 |
| 流程控制 | 流程分支if（单分支、双分支、多分支）/switch、循环for/while  continue跳出当前循环，break 结束整个循环 |
| 函数 | 声明、调用 |

面向对象：构造函数（类）、对象（实例的属性和方法）

面向对象的特性：抽象性（①只抽出需要的核心信息 / ②数学对象就是抽象出来的）、封装性、继承性、多态性

### 在sublime中创建javascript模版

1.点击工具->新代码段，

2.将CDATA[内容] 中括号里面的内容删除掉，换成需要的模版代码（模版网页的所有代码）

3.将<tabtrigger>注释打开，将标签中的内容更改为需要的命名，如scriptModel。最后保存文件到默认目录

4.新建html文件，输入scriptModel+tab，就会出现默认的模版网页了。

WebStorm 的颜色：

|  |
| --- |
| 关键字：**var function if true return this** |
| 字符串：**“Hello, world!”** |
| 属性：**style value className length** |
| 方法：**getElementById onclick alert log** |

### JavaScript关键字和保留字

|  |  |
| --- | --- |
| 关键字 | 保留字 |
| if else for in do while  switch case break  var new return void  try catch throw delete  finally default  typeof instanceof  with | 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 |

### JavaScript出现的位置

* 行间

<input type=”button” onclick=”alert(1);” value=”Click”/>

* 内嵌

<script>

//这里是js代码

</script> 可以放在body里，也可以放在title之下

* 外链js文件

<script src=”demo.js”></script>

异步：浏览器开启了新的线程执行代码，因为Javascript是单线程语言，而浏览器是多线程的。

1. async = “async” body里的代码执行时随机出现
2. defer = “derfer” body里的代码执行完最后出现

驼峰命名法：newName，第一个单词首字母小写，后面的首字母大写

var a=b=c=3; 同时声明并赋值

var num1, num2, num3; num1是局部变量，后两个是全局变量。

var a;

consloe.log(a); undefined

console.log(typeof a); undefined

把两个变量的值交换的方法：

1. 定义一个空的变量做容器；
2. a = a + b; b = a - b; a = a - b; 该方法只能用于数字类型的变量交换。

|  |  |  |
| --- | --- | --- |
| **数据类型** | | |
| 简单数据类型：存储在栈中 | | 复杂数据类型object：存储在堆中 |
| string | 字符串 | 对象类型，包括：array / function / Date  正则表达式 |
| number | 数字 |
| NaN\* |
| boolean | 布尔 |
| 空类型：null / undefined | | |

NaN是一种特殊的number。

null是一种特殊的object，表示“没有对象”，即该处不应该有值。典型用法是：

1. 作为函数的参数，表示该函数的参数不是对象；
2. 将事件的处理函数赋值为null，该事件将不会产生任何后果；
3. 作为原型链的终点。

undefined表示“缺少值”，就是此处应该有一个值，但是还没有定义。典型用法是：

1. 声明变量但是未赋值；
2. 声明了length，但是未给数组的项赋值；
3. 函数如果没有返回值，默认返回undefined；所以console.log(fn());打印结果是undefined，console.log(fn); 打印的是函数体。
4. 调用函数时，应该提供的参数没有提供，也就是形式参数没有接收到实际参数的值，那么它的值是undefined；如果传入的实参个数多于形参，那么多余的实参不起任何作用。
5. 对象属性如果没有赋值，系统自动赋给它undefined；

**number**是算盘，**string**是账本，**object**是房子，**栈**中放的是算盘、账本、地契（object在**堆**中的地址），一个朋友把地契复制了一份，然后通过地址去把房子里的花瓶摔了，那么原地址上的地址所对应的房子里的花瓶也摔了。

栈：先进后出FILO

堆：先进先出FIFO，类似事件队列

**直接量**（字面量）：从字面上能看出含义，可以直接用来计算打印。

八进制的直接量是在前面加一个0，十六进制的直接量是在前面加0x，但在控制台打印出来的是十进制。

小数的计算不是很精确，千万不要判断两个小数是否相等，不过判断范围没问题。由于存储空间的限制，ECMAScript并不能保存所有的数值。

比MAX\_VALUE还大的数为Infinity，MIN\_VALUE为5e-324，比它还小的是零。

|  |  |
| --- | --- |
| alert(a);  alert(“text”); | 弹出对话框，并提供信息 |
| confirm(“Are you sure?”); |  |
| typeof “ ”;  typeof(“ ”); | alert(typeof (123)); typeof不是函数 |
| document.write(“<h4>title</h4>”);  document.write(“<br>”); | 打印内容或换行  打印出title |

var a;

console.log(a);

console.log(typeof a);

// typeof打印出来都是字符串类型

|  |  |  |
| --- | --- | --- |
| **数学运算符** | | |
| % 求余 / 取模 | a += 3; 自增 | a++ |

a=a+3可以简写成a+=3。a-=3, a\*=3, a/=3

console.log(5/0); Infinity

console.log(0/0); NaN

console.log(5%0); NaN

console.log(5%6); 5

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Meaning | Code | Meaning |
| var f = 100;  var g = f++; alert(g); | //先赋值再自增  g = 100 | var f = 100;  var g = ++f; alert(g); | //先自增再赋值  g = 101 |

var a = 1; var b = ++a + ++a; console.log(b);

var a = 1; var b = a++ + ++a; console.log(b);

var a = 1; var b = a++ + a++; console.log(b);

var a = 1; var b = ++a + a++; console.log(b);

|  |  |  |  |
| --- | --- | --- | --- |
| **比较运算符** | | | |
| = 赋值 | == 仅值相等，id类型不同 | | === 值相等且类型相同，（id相同） |
| ！= 值不相等，类型可以相同 | | ！== 值或类型只要有一个不同 | |
| 比较的结果必然是boolean | | | |

==的运用：

var num = prompt(“Please enter your password:”);

alert (num == 123456); 密码对不对只需要判断数据的值，不需要关注类型。

|  |  |  |  |
| --- | --- | --- | --- |
| **逻辑运算符** | | | |
| t && t并且 → t | t && f → f | f && f → f | 一flase皆flase |
| t || t或者 → t | t || f → t | f || f → f | 一true皆true |
| 优先级：先算！，再算&&，最后算|| | | | |
| 用于多个条件的判断 | | | |

短路操作会返回对当前表达式的值起决定作用的值。

逻辑与： （1）对于布尔值，只要有一个false，就返回false；

（2）对于不是布尔值的情况则：

1. 如果第一个操作数是对象，则返回第二个数
2. 如果第二个操作数是对象，则只有在第一个操作数的求值结果为true的情况下才会返回该对象；
3. 如果第两个操作数都是对象，则返回第二个数操作数
4. 如果有一个操作数是null / NaN / undefined，则返回null / NaN / undefined
5. 如果两个操作数都是**数字**：1&&1=1，1&&2=2，1&&0=0

### 优先级

圆括号、++/--、！、乘除加减、比较、&&、||、赋值

var a = 1;

var b = "1";

var c = a!=b;

console.log(c);

### 显式转换

|  |  |
| --- | --- |
| **var a = Number(a);** | |
| 转换前 | 转换后 |
| null | 0 |
| undefined | **NaN** |
| false | 0 |
| true | **1** |
| 空的字符串 | 0 |
| 数值类型的字符串“-123”  “12**.**34” | -123  12.34 |
| 112.255.255 | NaN |
| 非数值类型的字符串：123abc | NaN |
| 一言不和就NaN | |

**注意**：空字符串用Number来转的话变成0，用parseInt转的话变成NaN。

parseInt和parseFloat专门用来处理字符串，它们脾气比Number好，只要是数字开头的就会截取，后面不能识别的部分则全部忽略。处理其他类型还是用Number

|  |  |  |  |
| --- | --- | --- | --- |
| 转换前 | **var a = parseInt(a);** | **var a = parseFloat(a);** | |
| “”空字符串 | NaN | | |
| 第一个字符不是数字或负号：abc123 | NaN  字符串前面有空格的话忽略，直至找到第一个非空字符 | | |
| 数字开头：120px  120.255.255 | 120 | | 120  120.255 |
| 小数：12.34 | 截尾取整12 | | 小数12.34 |

|  |  |
| --- | --- |
|  | 作用 |
| a = String(a); | 将a转成字符串 |
| a = a**.**toString( ); | 将a转成字符串：100**.** **.** toString( );必须有两个**.** |
| a = Boolean(a); | 除了false、“”空字符串、0、NaN、undefined、null，其他全是true |

|  |  |  |  |
| --- | --- | --- | --- |
| !“” T | ![] F | !{} F | !3 F |

var num = 123;

console.log(num);

console.log(num.toString());

var bool = true;

console.log(bool);

console.log(bool.toString());

console.log(typeof bool);

console.log(typeof bool.toString());

console.log(typeof (typeof bool));

console.log(undefined);

console.log(typeof undefined);

//null和undefined 没有toString 方法,String()所有类型都能转

//console.log(null.toString());

console.log(String(undefined));

|  |  |  |  |
| --- | --- | --- | --- |
| **隐式转换** | | | |
|  | var a = “abc”;  a = +a; | var b =！！a; | var b = a + “” |
| Equivalent | b = Number(a); | b = Boolean(a);  一次嚼两粒 | String(a); |

**＊**将string转成number：加减乘除以及取余都可以使数据进行隐式转换。

区别：a = - a; a = - - a; a = - (- a) 负负得正;

练：var a = “123”

a = a + 123 123123

a = +a + 123 246

a = a - 123 0

### 两个不同类型数据比较时隐式转换规则

|  |  |
| --- | --- |
| undefined === null //false | undefined == null //true |

1. **数字**和字符串、布尔类型、数组进行比较时，字符串或布尔类型先转换为数字（Number），数组是把第一个元素取出来再进行比较。

|  |  |  |
| --- | --- | --- |
| 数字与字符串比较 | 数字与布尔值比较 | 数字与数组比较 |
| 123 == “123abc” F | 0 == false T | 0 == [] T |
| 123 == “123” T | 1 == true T | 0 == !![] F |
|  | 2 == true F | 0 == [0] T |
|  |  | 1 == [1] T |
|  |  | 1 === [1] F |

1. **字符串**和数组（或对象）进行比较时，数组（或对象）会转换成字符串再进行比较。

|  |  |  |
| --- | --- | --- |
| [ ] == "" T | [0] == "" F | [0] == "0" T |

1. 特殊的是null和undefined不论和谁比较都不转。

|  |  |  |  |
| --- | --- | --- | --- |
| null==undefined T | null== 0 F | undefined == 0 F | null== false F |

1. 不同类型比较都转成数字类型

|  |  |  |
| --- | --- | --- |
| [] == false; T | [] == ![] T | [0] == false T |

### 关于NaN

\* NaN是计算发生错误时的状态。isNaN(is not a number)是非数字。

|  |  |
| --- | --- |
| 变量 | isNaN ( ) |
| NaN | true |
| number | false |
| string | true |
| undefined | true |
| { } | true |

var b = 1 + “a”; alert(b == NaN); 结果是false，因为NaN和任何值都不相等，和任何一个NaN也不相等。

转义字符 \ 可以把有特殊含义的字符转换成原义，也可以把普通字符转换成特殊字符：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| \n | \t | \ |  |  |  |

在一个字符串中敲回车，编辑工具会自动用 + 把前后拼接起来。

字符串的不可变性会导致大量拼接字符串的话很消耗性能。申明一个变量需要在内存中开辟空间，在拼接时，之前的值依旧存在，新的值会保存在新的空间中。开辟空间的过程需要消耗时间，开辟的空间会消耗内存。

快速替换变量：逗逗加加

|  |  |  |  |
| --- | --- | --- | --- |
| var a = "cat";  alert(isNaN (a)); | | | var a = "123";  alert(isNaN (a)); |
| 结果是true | | | 结果是false |
| 过程分两步：1. 先计算Number(a)；2. 再判断isNaN (a) | | | |
| 代码 | 结果 | 解释 | |
| var a = "cat";  alert(typeof (+a)); | number | 第一步：+a是把数据转换成number，但是a不是number，所以转换发生错误，返回NaN，而NaN是特殊的number类型 | |
| alert (typeof (1+undefined)); | number |
| var a = "a"; alert(typeof(!a)); | boolean |  | |
| var a = 1 + "1";  alert(typeof a); | string | 这里的+是拼接的作用 | |
| var a = "a";  alert(!!a); | true | ！a是false，！！a相当于Boolean(a) | |

变量名 + .log，按tab，能直接出来console.log(变量名);