# 1.

1995 js诞生，目的是为了处理输入验证，后面发展成具有几乎所有交互功能；

1996.8 微软 发布JScript1.0

1997 提交JavaScript1.1为蓝本制作国际标准；

ECMA-262版本：

1：等同与JavaScript1.1,

2：使适用于ISO/IEC16022,

3：使成为一门语言，是浏览器实现最好的版本

4：本来大改，后被取消，

5：替代4，发布；在3.0版本上小改相当于3.1；

DOM分为3个等级；

BOM浏览器对象模型：把针对浏览器的javascript扩展作为BOM的一部分，cookies/xmlhttprequest;

# 2.

Script 4个属性：

Async:外部文件立即下载，异步执行，不能有DOM操作；

Defrc:外部文件立即下载，延迟执行；

Src/type;

Script至上而下执行；

中间不允许出现/script

Js扩展名不是必须的，但必须保证能返回正确的Mini类型；

一般把SCript放在最后；

对于XHTML中一些特殊符号：

1.使用HTML实体

2.使用<![CDATA[

]]>

3.对于一些不支持XHTML的兼容，并且能平稳退化：

//<![CDATA[

加//

//]]>

对于禁用script的可以加noscript标签提示；

# 3.1语法

区分大小写

标示符：开头：字母、下划线、美元符号$；

其他字符可以是字母、下划线、美元符号、数字；

采用驼峰大小写格式；

注释：

1.//

2./\*

\*/

严格模式：

顶部添加:user strict;

IE10+才支持；

以分号结尾；

# 3.3变量

Js的变量是松散的，用var+变量名来定义变量；

在function中 var为局部变量，外部是调用不到的；

变量前不加var 为全局变量，但不推荐；

定义多个变量只用一个var 就行，变量之间用逗号隔开；

例如;var x=1,y=2,z=3;

# 3.4数据类型

只有6个数据类型：

5个基本数据类型：boolean/number/string/null/underfind

复杂数据类型：object（本质为一组无序名值对的集合）;

typeof 检测数据类型:

|  |  |
| --- | --- |
| 字符串 | string |
| 数字 | number |
| 布尔 | boolean |
| null | object |
| underfind | underfind |
| object | object |
| function | function |

数据类型的介绍：

1. underfind:声明了但未初始化的变量的值；（推荐初始化变量）；
2. null：指向空对象指针；对于要作为对象的变量初始化为Null;
3. boolean:true/false;

所有类型都可以转换为boolean;

Bollean():

|  |  |  |
| --- | --- | --- |
| 类型 | true | false |
| string | 非空 | 空 |
| number | 非0的数字 | 0/NaN |
| object | 任何 | null |
| underfind | 不支持 | underfind |

number:IEEE754格式；

4.1浮点数值：内存空间是整数的2倍；不允许对浮点计算的值作为boolean的判断条件；

4.2数值范围：

最小=Number.MIN\_VALUE,

最大=Number.MAX\_VALUE;

isFinite()检测数值是否在范围内；返回true/false;

4.3 NaN:非数值；用来替代本应返回数值却没有返回的错误；

isNaN（）检测是否为‘非数值’；

4.4数值转换：Number()/parseInt()/parseFloat();

4.4.1 Number();

任何类型都可以使用

|  |  |  |
| --- | --- | --- |
| 类型 | 值 | 转换 |
| boolean | True/false | 1/0 |
| null |  | 0 |
| underfind |  | NaN |
| string | number/空/其他 | number/0/NaN |
| object | valueof()-toString |  |

4.4.2 parseInt(x,10) 只针对字符串；基数必须有；

1. 数字开头-转换为数值；
2. 其他情况为NaN;

# String 类型；

无单、双引号的规定；

一旦声明无法改变，再声明就是销毁，然后覆盖；

5.1 转换为String;

使用x.toString(）或者后面使用 +‘’加空字符串；

null和underfind无法使用，

但可以使用String(x)方法来判断；null会返回null,underfind同理；其他会调用toString();

# object类型

是一组数据和功能的集合；new来创建，可以加属性，方法，自定义对象；

6.1每个对象都具有的属性和方法

constructor:当前对象函数；

hasOwnProperty:属性是否在对象中；属性必须为字符串形式作为参数；

isProtypeof:传入对象是否为对象原型；

propertyIsEnumberable:是否可以枚举；

toLoacleString():返回字符串；

toString()

valueOf():返回对象字符串、数值。布尔表示；

valueOf偏向于运算，toString偏向于显示。  
1、 在进行对象转换时（例如:alert(a)）,将优先调用toString方法，如若没有重写toString将调用valueOf方法，如果两方法都不没有重写，但按Object的toString输出。  
2、 在进行强转字符串类型时将优先调用toString方法，强转为数字时优先调用valueOf。  
3、 在有运算操作符的情况下，valueOf的优先级高于toString。

# 3.5 操作符；

# 3.5.1 一元操作符：

# 递增与递减

1. 前置：

var a = 1;

var b = ++a +2; //4

a先加1再和2相加

1. 后置：

var a = 1;

var b = a++ +2; //3

alert(a); //2

a不参加var b 的计算；

c. 其他数据类型；先进行Number（）再操作；

# 一元加和减

var a = +a ;

对于number不会任何操作，对于其他数据类型则会number()；

# 相加操作符 var x = a+b;

除了2个变量都为number，其他情况数据类型会转换为string然后拼接起来；

# 相减操作符

都会转换为number进行运算；

# 相等操作符==

转换为Number类型；

null与underfind相等；

NaN与NaN不相等；

对象看是否指向同一个对象

# 全等操作符 ===

不转换类型就相等；

函数

function xx(value){}

调用函数xx(value);

对于里面的value 可以用arguments[0]来调用；只能在函数内部使用；

# 变量、作用域、内存问题；

4.1基本类型、引用类型

变量的值分为：基本类型、引用类型；

基本类型：5个基本数据类型；

引用数据类型：对象；

4.1.1动态属性

只能给引用类型值动态添加属性和方法；

4.1.2复制变量

基本：复制值，和被复制的值没有关系；

引用：复制地址，和被复制的指向同一个对象；

4.1.3传递参数：都是按值传递；

和复制变量一个道理；

4.1.4检测类型

检测Object是否属于哪类对象；

对象 instanceof 数据类型

返回Boolean值；

4.2执行环境和作用域；

局部和全局；

查询一个变量从局部开始向上查询；

是没有块级作用域的；

4.3垃圾收集

标记清除；

4.4管理内存

一旦数据不用了，设置为null，解除引用；

5引用类型

5.2.3栈方法

ECMAScript 数组也提供了一种让数组的行为类似与其他数据类型结构的方法。具体说来，数组可以表现的像栈一样，后者是一种可以限制插入和删除的数据结构，一种LIFO的数据结构（最新添加的最先删除），其操作只发生在栈的顶部。ECMAScript为数组提供了push()/pop()方法，以便实现类似栈的方法；

push()可以接收任意数量的参数，把它们逐个添加到数组末尾，并返回数组的长度。

pop()从数组移除最后一项，并返回移除的值；

var a = new Array();

var b = a.push('a','b','c');

console.log(b);

var c = a.pop();

console.log(c);

console.log(a.length);

5.2.4队列方法

栈数据结构的访问规则（后进先出，而队列数据结构的访问规则是FIFO（frist-in-frist-out先进先出））,push()是向数组末端添加项的方法，因此要模拟队列只需数组从前端取得项的方法。实现这一数组操作方法就是shift()和push（）方法，可以像使用队列一样操作数组。

shift()也是返回被删除项；

var a = new Array();

var b = a.push('a','b','c');

console.log(b);

var c = a.pop();

var d = a.shift();

console.log(c);

console.log(d);

ECMAScript还为数组提供了一个unshift()方法，在数组前端添加任意项，并返回数组长度；与pop（）组合可以实现前加后减的效果。

var a = new Array();

a[0] = 'e';

var b = a.unshift('a','b','c','d');

console.log(b);

var c = a.pop();

var d = a.shift();

console.log(c);

console.log(d);

注意：push(),unshift()，添加的顺序和里面的参数顺序是一致的，整体加入数组里面，顺序没有改变。

5.2.3重排序方法

数组中存在2个可以直接拿来重排序的方法，reverse()和sort()。reverse()可以反转数组的排序.

var a = new Array();

a.push('a','b','c');

a.reverse();

console.log(a);

sort()默认为升序排序，会把其他类型转换为String来比较；

5比10小，但转为String后10小于5，因此必须给sort（）添加一个比较函数作为参数，以便我们指定哪个值位于哪个值的前面、

比较函数接收2个参数，如果第一个参数位于第二个参数的前面返回一个负数，如果相等返回0，反之为正。

function xx1(a,b){

if (a > b) {

return 1;}

else if (a < b) {return -1;}

else{return 0;}

}

var a = new Array();

a.push('10','5','40','100');

a.sort(xx1);

console.log(a);

function xx2(a,b){

if (a > b) {

return 1;

}else if (a < b) {

return -1;

}else{

return 0;

}

}

var a = [1,10,40,5];

a.sort(xx2);

console.log(a);

function xx3(a,b){

if (a > b) {

return 1;

}else if (a < b) {

return -1;

}else{

return 0;}}

var a = new Array();

a.push(10,5,40,100);

a.sort(xx);

console.log(a);

比较函数最好转换为数字比较，不然会出现上面第一个排序出错；

要使用sort（）反序，则在比较函数中取反；

对于数值类型或会返回数值类型的数组，有更好的比较函数，第二个函数减去第一个：

function xx(a,b){

return a-b;

}

5.26操作方法

ECMAScript 为已经在数组中的项提供了方法，concat()基于当前数组所有项创建新数组。没有参数的时候就相当于复制，有参数1.参数为就把参数添加到数组后面形成新的数组；2.参数不为数组添加到结果数组的末尾；

var a = [1,2,3];

var b = a.concat('a');

var c = b.concat();

var d = c.concat('b','c');

var e = d.concat(['o','w','q']);

console.log(a);// [1, 2, 3]

console.log(b);// [1, 2, 3, "a"]

console.log(c);// [1, 2, 3, "a"]

console.log(d);// [1, 2, 3, "a", "b", "c"]

console.log(e); // [1, 2, 3, "a", "b", "c", "o", "w", "q"]

slice()基于当前数组一个或多个项创建新的数组。

参数为起始和结束位置2个参数，第二个参数可选；

注：参数和数组序号一致；

var a = [1,2,3];

var e = a.slice(1,2);

console.log(e);// 2；

splice()

1. 删除：2个参数，（起始位置，结束位置）；
2. 添加：3个参数，（起始位置，0，添加项）；
3. 替换：3个参数，（起始位置，删除个数，替换项）；

返回删除的项，没有删除则为空；

var a = [1,2,3];

var e = a.splice(0,2);

console.log(e);

console.log(a);

var b = a.splice(1,0,2,3,4);

console.log(b);

console.log(a);

var c = a.splice(0,1,2);

console.log(c);

console.log(a);