JavaScript包括三个部分 分别是核心(ECMAScript), 文档对象模型(DOM) ,浏览器对象模型(BOM)

Web浏览器只是ECMAScript实现可能存在的一种宿主环境（host environment）

## **ECMAScript简介**

ECMAScript，即ECMA-262定义的语言，并不局限于Web浏览器。

ECMA-262描述语法 类型 语句 关键字 保留字 操作符 全局对象

ECMAScript只是对实现这个规范描述的所有方面的一门语言的称 呼。JavaScript实现了ECMAScript，而Adobe ActionScript同样也实现了 ECMAScript。

### . ECMAScript版本

ECMAScript不同的版本以“edition”表示（也就是描述特定实现的 ECMA-262的版本）。ECMA-262最近的版本是第10版，发布于 2019年6月。

ECMAScript版本的历程

ECMA-262的第1版本质上跟网景的JavaScript 1.1相 同，只不过删除了所有浏览器特定的代码，外加少量细微的修 改。

ECMA-262第2版只是做了一些编校工作，主要是为了更新之后严 格符合ISO/IEC-16262的要求，并没有增减或改变任何特性。

ECMA-262第3版第一次真正对这个标准进行更新，更新了字符串 处理、错误定义和数值输出。

ECMA-262第4版是对这门语言的一次彻底修订。

第5版致力于厘清第3版存在的歧义，也增加了新功能。

ECMA-262第6版，俗称ES6、ES2015或ES Harmony（和谐版）， 于2015年6月发布。

ECMA-262第7版，也称为ES7或ES2016，于2016年6月发布。这 次修订只包含少量语法层面的增强，如 Array.prototype.includes 和指数操作符。

ECMA-262第8版，也称为ES8、ES2017，完成于2017年6月。

ECMA-262第9版，也称为ES9、ES2018，发布于2018年6月。

ECMA-262第10版，也称为ES10、ES2019，发布于2019年6月。

### ECMAScript符合型

要成为ECMAScript的实现就要满足两个条件

1. 支持ECMA-262中描述的所有”类型、值、对象、属性、函数、以及程序语法与语义”
2. 支持Unicode字符标准

符合性实现还可以满足下列要求。

增加ECMA-262中未提及的“额外的类型、值、对象、属性和 函数”。ECMA-262所说的这些额外内容主要指规范中未给出 的新对象或对象的新属性。 支持ECMA-262中没有定义的“程序和正则表达式语法”（意 思是允许修改和扩展内置的正则表达式特性）。

## DOM

概念：文档对象模型（DOM，Document Object Model）是一个应用编程 接口（API），用于在HTML中使用扩展的XML

DOM级别：

1. 1998年10月，DOM Level 1成为W3C的推荐标准。这个规范由两 个模块组成：DOM Core和DOM HTML。
2. DOM Level 2新增了以下模块，以支持新的接口。

DOM视图：描述追踪文档不同视图（如应用CSS样式前后的 文档）的接口。 DOM事件：描述事件及事件处理的接口。 DOM样式：描述处理元素CSS样式的接口。 DOM遍历和范围：描述遍历和操作DOM树的接口。

1. W3C不再按照Level来维护DOM了，而是作为DOM Living Standard来维护，其快照称为DOM4。

## BOM

概念：

IE3和Netscape Navigator 3提供了浏览器对象模型（BOM） API， 用于支持访问和操作浏览器的窗口

人们通常会把任何特定于浏览器的扩展都归在BOM的范畴内 例如：

1. 弹出新浏览器窗口的能力；
2. 移动、缩放和关闭浏览器窗口的能力；

3． navigator 对象，提供关于浏览器的详尽信息；

4. location 对象，提供浏览器加载页面的详尽信息；

5. screen 对象，提供关于用户屏幕分辨率的详尽信息；

6. performance 对象，提供浏览器内存占用、导航行为和时间统 计的详尽信息；

7. 对cookie的支持；

8. 其他自定义对象，如 XMLHttpRequest 和IE的 ActiveXObject 。

## <script>元素

<script> 元素有下列8个属性。1. async ：可选。

2. charset ：可选。

3. crossorigin ：可选。

4. defer ：可选。

5. integrity ：可选。

6. language ：废弃。

7. src ：可选。

8. type ：可选。

使用方法：

使用 <script> 的方式有两种：通过它直接在网页中嵌入

JavaScript代码，以及通过它在网页中包含外部JavaScript文件。

要嵌入行内JavaScript代码，直接把代码放在 <script> 元素中

就行：例如：<script>

function sayHi() {

console.log("Hi!");

}

</script>

注：1. 在使用行内JavaScript代码时，要注意代码中不能出现字符串 。

2浏览器解析行内脚本的方式决定了它在看到字符串 时，会将其当成结束的 标签。想避免这个问题，只需要 转义字符“\”1即可

### 标签占位符

所有 <script> 元素都被放在页面的 <head> 标签内

### 推迟执行脚本

HTML 4.01为 <script> 元素定义了一个叫 defer 的属性。

### 异步执行脚本

HTML5为 <script> 元素定义了 async 属性。

### 动态加载脚本

除了 <script> 标签，还有其他方式可以加载脚本。因为

JavaScript可以使用DOM API，所以通过向DOM中动态添加 script

元素同样可以加载指定的脚本。只要创建一个 script 元素并将其添

加到DOM即可。XHTML中的变化

可扩展超文本标记语言（XHTML，Extensible HyperText Markup Language）是将HTML作为XML的应用重新包装的结果。

## 语法

### 标识符

概念：所谓标识符，就是变量、函数、属性或函数参数的名称。标识符 可以由一或多个下列字符组成

格式:

1. 第一个字符必须是一个字母、下划线（ \_ ）或美元符号 （ $ ）；
2. 剩下的其他字符可以是字母、下划线、美元符号或数字。

注：

关键字、保留字、 true 、 false 和 null 不能作为标识符

### 注释

包括单行注释和块注释

单行注释：

块注释以一个斜杠和一个星号（ /\* ）开头，以它们的反向组合 （ \*/ ）结尾，

如： /\* 这是多行 注释 \*

多行注释：

/\* 这是多行 注释 \*/

### 语句

ECMAScript中的语句以分号结尾。

### 保留字和关键字

保留的关键字不能用作标识符或属性名。

### 变量

有3 个关键字可以声明变量： var 、 const 和 let 。其中， var 在 ECMAScript的所有版本中都可以使用，而 const 和 let 只能在 ECMAScript 6及更晚的版本中使用。

### Var关键字

要定义变量，可以使用 var 操作符（注意 var 是一个关键 字），后跟变量名（即标识符，如前所述）： var message;

作用：

1. var 声明作用域
2. var 声明提升

### let声明

let 跟 var 的作用差不多，但有着非常重要的区别。最明显的 区别是， let 声明的范围是块作用域，而 var 声明的范围是函数作 用域。

区别：

1. let 与 var 的另一个重要的区别，就是 let 声明的变量不会 在作用域中被提升。
2. 与 var 关键字不同，使用 let 在全局作用域中声明的变量不会 成为 window 对象的属性（ var 声明的变量则会）。
3. 在使用 var 声明变量时，由于声明会被提升，JavaScript引擎会 自动将多余的声明在作用域顶部合并为一个声明。因为 let 的 作用域是块，所以不可能检查前面是否已经使用 let 声明过同 名变量，同时也就不可能在没有声明的情况下声明它。
4. 在 let 出现之前， for 循环定义的迭代变量会渗透到循环体外部，改成使用 let 之后，这个问题就消失了，因为迭代变量的作用 域仅限于 for 循环块内部

### Const声明

const 的行为与 let 基本相同，唯一一个重要的区别是用它声 明变量时必须同时初始化变量，且尝试修改 const 声明的变量会导 致运行时错误。

### 数据类型

ECMAScript有6种简单数据类型（也称为原始类型）： Undefined 、 Null 、 Boolean 、 Number 、 String 和 Symbol 。 Symbol （符号）是ECMAScript 6新增的。还有一种复 杂数据类型叫 Object （对象）。 Object 是一种无序名值对的集 合。

### Typeof操作符

对一个值使 用 typeof 操作符会返回下列字符串之一：

1. "undefined" 表示值未定义；
2. "boolean" 表示值为布尔值；
3. "string" 表示值为字符串；
4. "number" 表示值为数值；
5. "object" 表示值为对象（而不是函数）或 null ；
6. "function" 表示值为函数；
7. "symbol" 表示值为符号。

### Undefined类型

Undefined 类型只有一个值，就是特殊值 undefined

例子：

let message; // 这个变量被声明了，只是值为 undefined

// 确保没有声明过这个变量

// let age

console.log(message); // "undefined"

console.log(age); // 报错

Null类型

Null 类型同样只有一个值，即特殊值 null

let car = null;

console.log(typeof car); // "object"

### Boolean类型

Boolean （布尔值）类型是ECMAScript中使用最频繁的类型之 一，有两个字面值： true 和 false 。

例子：

let found = true;

let lost = false;

注意：布尔值字面量 true 和 false 是区分大小写的，因此 True 和 False （及其他大小混写形式）是有效的标识符，但不是 布尔值。

### Number类型

最基本的数值字面量格式是十进制整数

例子：

let intNum = 55; // 整数

### String类型

String （字符串）数据类型表示零或多个16位Unicode字符序 列。字符串可以使用双引号（"）、单引号（'）或反引号（`）标示

字符串的特点

ECMAScript中的字符串是不可变的（immutable），意思是一旦 创建，它们的值就不能变了。要修改某个变量中的字符串值，必 须先销毁原始的字符串，然后将包含新值的另一个字符串保存到 该变量

转换为字符串

首先是使用几乎所有值都有 的 toString() 方法

### Symbol类型

Symbol （符号）是ECMAScript 6新增的数据类型。符号是原始 值，且符号实例是唯一、不可变的。

1. 符号的基本用法

符号需要使用 Symbol() 函数初始化。因为符号本身是原始类 型，所以 typeof 操作符对符号返回 symbol 。

1. 使用全局符号注册表

如果运行时的不同部分需要共享和重用符号实例，那么可以用一 个字符串作为键，在全局符号注册表中创建并重用符号。 为此，需要使用 Symbol.for() 方法

1. 使用符号作为属性

凡是可以使用字符串或数值作为属性的地方，都可以使用符号。

1. 常用内置符号

ECMAScript 6也引入了一批常用内置符号（well-known symbol），用于暴露语言内部行为，开发者可以直接访问、重写 或模拟这些行为。

1. Symbol.asyncIterator
2. Symbol.hasInstance
3. Symbol.isConcatSpreadable
4. Symbol.iterator
5. Symbol.match
6. Symbol.replace
7. Symbol.search
8. Symbol.species
9. Symbol.split
10. Symbol.toPrimitive
11. Symbol.toStringTag
12. Symbol.unscopables

### 操作符

一元操作符：

只操作一个值的操作符叫一元操作符（unary operator）。一元操 作符是ECMAScript中最简单的操作符。

递增/递减操作符：

递增和递减操作符直接照搬自C语言，但有两个版本：前缀版和 后缀版。

1. ++
2. –

一元加和减

加：一元加由一个加号 （ + ）表示，放在变量前头，对数值没有任何影响

减：一元减由一个减号（ - ）表示，放在变量前头，主要用于把数值 变成负值

### 布尔操作符

逻辑非：!

逻辑与：&&

逻辑或：||

除法操作符：/

取模操作符: %

减法操作符: -

关系操作符: ，包括小于（ < ）、大于 （ > ）、小于等于（ <= ）和大于等于（ >= ），

相等操作符: 1. 等于和不等于 2. 全等和不全等

条件操作符

赋值操作符:

乘后赋值（ \*= ）

除后赋值（ /= ）

取模后赋值（ %= ）

加后赋值（ += ）

减后赋值（ -= ）

左移后赋值（ <<= ）

右移后赋值（ >>= ）

无符号右移后赋值（ >>>= ）

## 语句

if 语句:

这里的条件（ condition ）可以是任何表达式，并且求值结果 不一定是布尔值。ECMAScript会自动调用 Boolean() 函数将这个表 达式的值转换为布尔值。如果条件求值为 true ，则执行语句 statement1 ；如果条件求值为 false ，则执行语句 statement2 。这里的语句可能是一行代码，也可能是一个代码块 （即包含在一对花括号中的多行代码）。

do-while 语句:

do-while 语句是一种后测试循环语句，即循环体中的代码执 行后才会对退出条件进行求值。

while 语句:

while 语句是一种先测试循环语句，即先检测退出条件，再执 行循环体内的代码。

for 语句:

for 语句也是先测试语句，只不过增加了进入循环之前的初始 化代码，以及循环执行后要执行的表达式，

for-in 语句:

for-in 语句是一种严格的迭代语句，用于枚举对象中的非符号 键属性，

for-of 语句:

for-of 语句是一种严格的迭代语句，用于遍历可迭代对象的元 素，

with 语句:

with 语句的用途是将代码作用域设置为特定的对象

switch 语句:

switch 语句是与 if 语句紧密相关的一种流控制语句，从其他 语言借鉴而来。

## 函数

function 关键字声明，后跟一组参数，然后是函数体。

例如：

function functionName(arg0, arg1,...,argN) {

statements

}

注：除了 return 语句之外没有任何特殊声明表明该函数有返回值

严格模式对函数也有一些限制：

函数不能以 eval 或 arguments 作为名称；

函数的参数不能叫 eval 或 arguments ；

两个函数的参数不能叫同一个名称。

如果违反上述规则，则会导致语法错误，代码也不会执行。、

## 变量，作用域与内存

原始值与引用值

概念：ECMAScript变量可以包含两种不同类型的数据：原始值和引用 值。原始值（primitive value）就是最简单的数据，引用值（reference value）则是由多个值构成的对象。

注： 在很多语言中，字符串是使用对象表示的，因此被认为是引 用类型。ECMAScript打破了这个惯例。

动态属性：

原始值和引用值的定义方式很类似，都是创建一个变量，然后给 它赋一个值。

例：

let person = new Object();

person.name = "Nicholas";

console.log(person.name);

这里，首先创建了一个对象，并把它保存在变量 person 中。然 后，给这个对象添加了一个名为 name 的属性，并给这个属性赋值了 一个字符串 "Nicholas" 。在此之后，就可以访问这个新属性，直 到对象被销毁或属性被显式地删除

原始值不能有属性，尽管尝试给原始值添加属性不会报错

例如：

let name = "Nicholas";

name.age = 27;

console.log(name.age); // undefined

复制值：

在通过变量把一个原始值赋值到另一个变量时，原始值会被复制 到新变量的位置。

例：

let num1 = 5;

let num2 = num1

传递参数：

在按值传递参数时，值会被复制到一个局部变量（即一个命名参 数，或者用ECMAScript的话说，就是 arguments 对象中的一个槽 位）。

确定类型

它是判断一个变量是否为字符串、数值、布尔 值或 undefined 的最好方式。如果值是对象或 null ，那么 typeof 返回 "object"

注：

typeof 操作符在用于检测函数时也会返 回 "function" 。当在Safari（直到Safari 5）和Chrome（直到 Chrome 7）中用于检测正则表达式时，由于实现细节的原因， typeof 也会返回 "function" 。ECMA-262规定，任何实现内 部 [[Call]] 方法的对象都应该在 typeof 检测时返 回 "function" 。因为上述浏览器中的正则表达式实现了这个方 法，所以 typeof 对正则表达式也返回 "function" 。在IE和 Firefox中， typeof 对正则表达式返回 "object" 。

执行上下文与作用域

变量或函数的上下文决定了它们可以访问哪些数据，以及它们 的行为。每个上下文都有一个关联的变量对象（variable object），而 这个上下文中定义的所有变量和函数都存在于这个对象上。虽然无法 通过代码访问变量对象，但后台处理数据会用到它。

因此所有通过 var 定义的全局变量和函数都会成为 window 对象的属性和方法。

上下文中的代码在执行的时候，会创建变量对象的一个作用域链 （scope chain）。

如果上下文是函数，则其活动对象（activation object）用作变量对象。

变量声明

1. 在使用 var 声明变量时，变量会被自动添加到最接近的上下 文。

var 声明会被拿到函数或全局作用域的顶部，位于作用域中所 有代码之前

1. ES6新增的 let 关键字跟 var 很相似，但它的作用域是块级 的，这也是JavaScript中的新概念。块级作用域由最近的一对包含 花括号 {} 界定。
2. 除了 let ，ES6同时还增加了 const 关键字。使用 const 声 明的变量必须同时初始化为某个值。

注：

let 与 var 的另一个不同之处是在同一作用域内不能声明两 次。重复的 var 声明会被忽略，而重复的 let 声明会抛出 SyntaxError 。

原始值和引用值有 以下特点：

原始值大小固定，因此保存在栈内存上。

从一个变量到另一个变量复制原始值会创建该值的第二个副本。

引用值是对象，存储在堆内存上。

包含引用值的变量实际上只包含指向相应对象的一个指针，而不 是对象本身。

从一个变量到另一个变量复制引用值只会复制指针，因此结果是 两个变量都指向同一个对象。

typeof 操作符可以确定值的原始类型，而 instanceof 操作 符用于确保值的引用类型。

执行上下文可以总结如下：

执行上下文分全局上下文、函数上下文和块级上下文。

代码执行流每进入一个新上下文，都会创建一个作用域链，用于 搜索变量和函数。

函数或块的局部上下文不仅可以访问自己作用域内的变量，而且 也可以访问任何包含上下文乃至全局上下文中的变量。

全局上下文只能访问全局上下文中的变量和函数，不能直接访问 局部上下文中的任何数据。 变量的执行上下文用于确定什么时候释放内存。

JavaScript的垃圾回收程序可以总结如下：

离开作用域的值会被自动标记为可回收，然后在垃圾回收期间被 删除。

主流的垃圾回收算法是标记清理，即先给当前不使用的值加上标 记，再回来回收它们的内存。

引用计数是另一种垃圾回收策略，需要记录值被引用了多少次。 JavaScript引擎不再使用这种算法，但某些旧版本的IE仍然会受这 种算法的影响，原因是JavaScript会访问非原生JavaScript对象（如 DOM元素）。

引用计数在代码中存在循环引用时会出现问题。

解除变量的引用不仅可以消除循环引用，而且对垃圾回收也有帮 助。为促进内存回收，全局对象、全局对象的属性和循环引用都 应该在不需要时解除引用。