

# Javascript

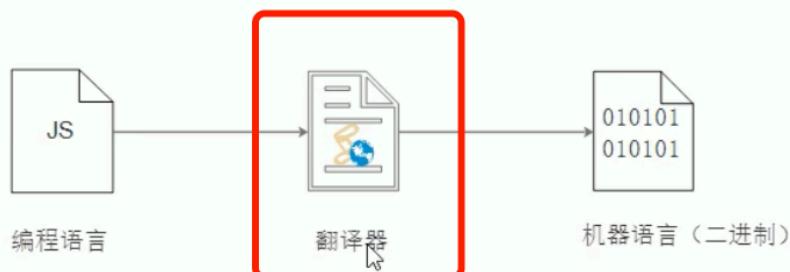
弄清楚变量转换如何实现的？？？？

## 1、计算机原理

### 1.4 翻译器

高级语言所编制的程序不能直接被计算机识别，必须经过转换才能被执行，为此，我们需要一个翻译器。

翻译器可以将我们所编写的源代码转换为机器语言，这也被称为二进制化。记住1和0。



## 2.2 数据存储

1. 计算机内部使用二进制0和1来表示数据。
2. 所有数据，包括文件、图片等最终都是以二进制数据(0和1)的形式存放在硬盘中的。
3. 所有程序，包括操作系统，本质都是各种数据，也以二进制数据的形式存放在硬盘中。平时我们所说的安装软件，其实就是把程序文件复制到硬盘中。
4. 硬盘、内存都是保存的二进制数据。



## 2.3 数据存储单位

bit < byte < kb < GB < TB <.....

- 位(bit) : 1bit 可以保存一个 0 或者 1 ( 最小的存储单位 )
- 字节(Byte) :  $1B = 8b$
- 千字节(KB) :  $1KB = 1024B$
- 兆字节(MB) :  $1MB = 1024KB$
- 吉字节(GB):  $1GB = 1024MB$
- 太字节(TB):  $1TB = 1024GB$

## 2.4 程序运行



1. 打开某个程序时，先从硬盘中把程序的代码加载到内存中
2. CPU执行内存中的代码

**注意：**之所以要内存的一个重要原因，是因为 CPU 运行太快了，如果只从硬盘中读数据，会浪费 CPU 性能，所以，才使用存取速度更快的内存来保存运行时的数据。（内存是电，硬盘是机械）

## 1.2 JavaScript 是什么

- JavaScript 是世界上最流行的语言之一，是一种运行在客户端的脚本语言 ( Script 是脚本的意思 )
- 脚本语言：不需要编译，运行过程中由 js 解释器 (js 引擎) 逐行来进行解释并执行
- 现在也可以基于 Node.js 技术进行服务器端编程

## 2、初识Javascript

## 1.3 JavaScript 的作用

- 表单动态校验 ( 密码强度检测 ) ( JS 产生最初的目的 )
- 网页特效
- 服务端开发 (Node.js)
- 桌面程序 (Electron)
- App (Cordova)
- 控制硬件 - 物联网 (Ruff)
- 游戏开发 (cocos2d-js)

## 1.4 浏览器执行 JS 简介

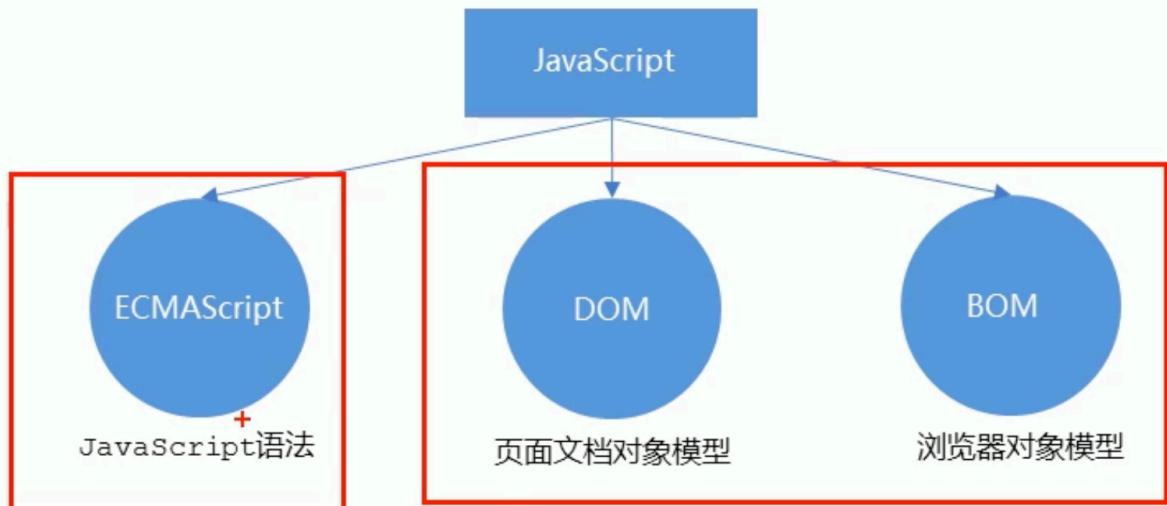
浏览器分成两部分：渲染引擎和JS引擎

- **渲染引擎**：用来解析HTML与CSS，俗称内核，比如chrome浏览器的blink，老版本的webkit
- **JS引擎**：也称为JS解释器。用来读取网页中的JavaScript代码，对其处理后运行，比如chrome浏览器的V8

浏览器本身并不会执行JS代码，而是通过内置JavaScript引擎(解释器)来执行JS代码。JS引擎执行代码时逐行解释每一句源码（转换为机器语言），然后由计算机去执行，所以JavaScript语言归为脚本语言，会逐行解释执行。



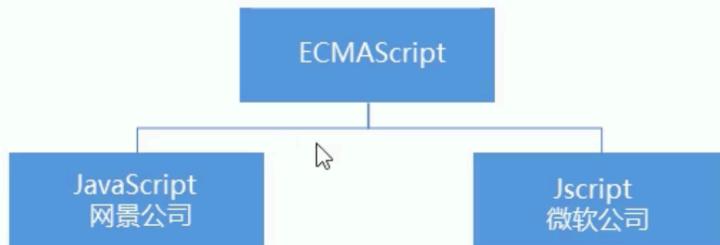
## 1.5 JS 的组成



## 1.5 JS 的组成

### 1. ECMAScript

**ECMAScript** 是由ECMA 国际（原欧洲计算机制造商协会）进行标准化的一门编程语言，这种语言在万维网上应用广泛，它往往被称为 JavaScript 或 JScript，但实际上后两者是 ECMAScript 语言的实现和扩展。



**ECMAScript** : ECMAScript 规定了JS的编程语法和基础核心知识，是所有浏览器厂商共同遵守的一套JS语法工业标准。

更多参看MDN: [https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/JavaScript\\_technologies\\_overview](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/JavaScript_technologies_overview)

## 1.5 JS 的组成

### 2. DOM —— 文档对象模型

**文档对象模型** ( Document Object Model , 简称DOM ) , 是W3C组织推荐的处理可扩展标记语言的**标准编程接口**。通过 DOM 提供的接口可以对页面上的各种元素进行操作 ( 大小、位置、颜色等 ) 。

### 3. BOM —— 浏览器对象模型

**BOM** ( Browser Object Model , 简称BOM) 是指浏览器对象模型，它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。通过BOM可以操作浏览器窗口，比如弹出框、控制浏览器跳转、获取分辨率等。

02-JS初体验.html

```

3
4 <head>
5     <meta charset="UTF-8">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <meta http-equiv="X-UA-Compatible" content="ie=edge">
8     <title>Document</title>
9     <style></style>
10    <!-- 2. 内嵌式的js -->
11    <script>
12        alert('沙漠骆驼');
13    </script>
14 </head>
15
16 <body>
17    <!-- 1. 行内式的js 直接写到元素的内部 -->
18    <!-- <input type="button" value="唐伯虎" onclick="alert('秋香姐')"> -->
19 </body>
20
21 </html>

```

文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 调试(D) 终端(T) 帮助(H) 02-JS初体验.html - code - Visual Studio Code [管理员] 黑马程序员pink老师

-JS三种书写位置

资源管理器

- 打开的编辑器
  - 02-JS初体验.html
  - JS my.js
- CODE
  - 01-编程语言.html
  - 02-JS初体验.html
  - JS my.js

```

5     <meta charset="UTF-8">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <meta http-equiv="X-UA-Compatible" content="ie=edge">
8     <title>Document</title>
9     <style></style>
10    <!-- 2. 内嵌式的js -->
11    <script>
12        // alert('沙漠骆驼');
13    </script>
14    <!-- 3. 外部js script 双标签 -->
15    <script src="my.js"></script>
16 </head>
17
18 <body>
19    <!-- 1. 行内式的js 直接写到元素的内部 -->
20    <!-- <input type="button" value="唐伯虎" onclick="alert('秋香姐')"> -->
21 </body>
22

```

JS 有3种书写位置，分别为行内、内嵌和外部。

## 1. 行内式 JS

```
<input type="button" value="点我试试" onclick="alert('Hello World')"/>
```



- 可以将单行或少量 JS 代码写在HTML标签的事件属性中（以 on 开头的属性），如：onclick
- 注意单双引号的使用：在HTML中我们推荐使用双引号，JS 中我们推荐使用单引号
- 可读性差，在html中编写JS大量代码时，不方便阅读；
- 引号易错，引号多层嵌套匹配时，非常容易弄混；
- 特殊情况下使用

## 2. 内嵌 JS

```
<script>  
    alert('Hello World~!');  
</script>
```

- 可以将多行JS代码写到 <script> 标签中
- 内嵌 JS 是学习时常用的方式



## 3. 外部 JS文件

```
<script src="my.js"></script>
```



- 利于HTML页面代码结构化，把大段 JS代码独立到 HTML 页面之外，既美观，也方便文件级别的复用
- 引用外部 JS文件的 script 标签中间不可以写代码  
    +  
● 适合于JS 代码量比较大的情况

```
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<title>Document</title>
<script>
    // 1. 单行注释 ctrl + /
    /* 2. 多行注释 默认的快捷键 shift + alt + a
     | 2. 多行注释 vscode 中修改多行注释的快捷键: ctrl + shift + /
    */
</script>
式 head>
段 dy>
```

JS输入输出语句

方法	说明	归属
alert(msg)	浏览器弹出警示框	浏览器
console.log(msg)	浏览器控制台打印输出信息 →	浏览器
prompt(info)	浏览器弹出输入框，用户可以输入	浏览器

js可以不申明就赋值，但是会变成全局变量

js申明变量可以只用一个var，然后用逗号分隔

## 1.5 变量命名规范

- 由字母(A-Za-z)、数字(0-9)、下划线(\_)、美元符号(\$)组成，如：usrAge, num01, \_name  
+ \_\_\_\_\_
- 严格区分大小写。var app; 和 var App; 是两个变量
- 不能以数字开头。18age 是错误的
- 不能是关键字、保留字。例如：var、for、while
- 变量名必须有意义。MMD BBD nl → age
- 遵守驼峰命名法。首字母小写，后面单词的首字母需要大写。myFirstName
- 推荐翻译网站：有道 爱词霸

区分大小写，不能数字开头，驼峰：首字母小写，后面大写，你要用name做变量名，name不申明可以打印不会报错，不过不要用

## 变量转换

### 4.2 转换为字符串

方式	说明	案例
toString()	转成字符串	var num= 1; alert(num.toString());
String() 强制转换	转成字符串	var num = 1; alert(String(num));
加号拼接字符串	和字符串拼接的结果都是字符串	var num = 1; alert(num+ "我是字符串");

- `toString()` 和 `String()` 使用方式不一样。
- 三种转换方式，我们更喜欢用第三种加号拼接字符串转换方式，这一种方式也称之为隐式转换。

## 标识符、关键字、保留字

### 2. 标识符、关键字、保留字

#### 1. 标识符

标识(zhi)符：就是指开发人员为变量、属性、函数、参数取的名字。

**标识符不能是关键字或保留字。**



#### 2. 关键字

关键字：是指 JS 本身已经使用了的字，不能再用它们充当变量名、方法名。

包括：`break`、`case`、`catch`、`continue`、`default`、`delete`、`do`、`else`、`finally`、`for`、`function`、`if`、`in`、`instanceof`、`new`、`return`、`switch`、`this`、`throw`、`try`、`typeof`、`var`、`void`、`while`、`with` 等。

```
console.log(num);
// 1. 前置自增和后置自增如果单独使用 效果是一样的
// 2. 后置自增 口诀：先返回原值 后自加1
var age = 10;
console.log(age++ + 10); 20
script>
```

```
++a; // ++a 11 a = 11
var b = ++a + 2; // a = 12 ++a = 12
console.log(b); // 14

var c = 10;
c++; // c++ 11 c = 11
var d = c++ + 2; // c++ = 11 c = 12
console.log(d); // 13

var e = 10;
var f = e++ + ++e; // 1. e++ = 10 e = 11 2. e = 12 ++e = 12
console.log(f); // 22
script>
```

## 5. 逻辑运算符

### 5.4 短路运算（逻辑中断）

**短路运算的原理：**当有多个表达式（值）时，左边的表达式值可以确定结果时，就不再继续运算右边的表达式的值；

#### 1. 逻辑与

- 语法：[表达式1 && 表达式2](#)
- 如果第一个表达式的值为真，则返回表达式2
- 如果第一个表达式的值为假，则返回表达式1

```
<script>
    // 1. 用我们的布尔值参与的逻辑运算 true && false == false
    // 2. 123 && 456 是值 或者是 表达式 参与逻辑运算?
    // 3. 逻辑与短路运算 如果表达式1 结果为真 则返回表达式2
    console.log(123 && 456); // 456
</script>
</head>
```

T

↓

不短路

```
<title>Document</title>
<script>
    // 1. 用我们的布尔值参与的逻辑运算 true && false == false
    // 2. 123 && 456 是值 或者是 表达式 参与逻辑运算?
    // 3. 逻辑与短路运算 如果表达式1 结果为真 则返回表达式2 如果表达式1为假 那么返回表达式1
    console.log(123 && 456); // 456
    console.log(0 && 456); // 0
    console.log(0 && 1 + 2 && 456 * 56789); // 0
</script>
</head>
```

不短路

```
console.log(0 && 1 + 2 && 456 * 56789); // 0
// 如果有空的或者否定的为假 其余是真的 0 '' null undefined NaN
```

## 5.4 逻辑中断（短路操作）

### 2. 逻辑或

- 语法：**表达式1 || 表达式2**
- 如果第一个表达式的值为真，则返回表达式1
- 如果第一个表达式的值为假，则返回表达式2

```
console.log( 123 || 456 );           // 123
console.log( 0 || 456 );             // 456
console.log( 123 || 456 || 789 );   // 123
```

# 5. 逻辑运算符

来复习

毫无压力

## 5.4 逻辑中断（短路操作）

```
var num = 0;
```

```
console.log(123 || num++);
```

```
console.log(num);
```

0

123

## 7. 运算符优先级

黑马

优先级	运算符	顺序
1	小括号	()
2	一元运算符	++ -- !
3	算数运算符	先 * / % 后 + -
4	关系运算符	> >= < <=
5	相等运算符	== != === !==
6	逻辑运算符	先 && 后
7	赋值运算符	=
8	逗号运算符	,

- 一元运算符里面的逻辑非优先级很高
- 逻辑与比逻辑或优先级高

不要直接给数组里的元素赋值

```
// 2. 新增数组元素 修改索引号 追加数组元素
var arr1 = ['red', 'green', 'blue'];
arr1[3] = 'pink';
console.log(arr1);
arr1[4] = 'hotpink';
console.log(arr1);
arr1[0] = 'yellow'; // 这里是替换原来的数组元素
console.log(arr1);
arr1 = '有点意思';
console.log(arr1); // 不要直接给 数组名赋值 否则里面的数组元素都没有了
</script>
</head>
<body>
</body>
```

妙呀 妙啊 这个厉害了  
黑马程序员pink老师 bilibili

```
console.log(newArr);    牛批    妙啊    秒啊
// 方法2    wc妙呀    妙啊    神奇    妙到家了
var arr = [2, 0, 6, 1, 77, 0, 52, 0, 25, 7];
var newArr = [];
// 刚开始 newArr.length 就是 0
for (var i = 0; i < arr.length; i++) {
    if (arr[i] >= 10) {
        // 新数组索引号应该从0开始 依次递增
        newArr[newArr.length] = arr[i];
    }
}
console.log(newArr);
```

1人正在看