### 文章目录

* 一、JavaScript 简介
  + 1. 为什么学习 JavaScript?
  + 2. JavaScript 组成
  + 3. 引入方式
    - 3.1 内部方式
    - 3.2 外部方式
    - 3.3 行内方式
  + 4. 注释
    - 4.1 单行注释
    - 4.2 多行注释
  + 5. 结束符
  + 6. 输入和输出
    - 6.1 输出
    - 6.2 输入
* 二、变量
  + 1.声明
  + 2.赋值
  + 3.关键字
  + 4.变量名命名规则
* 三、常量
* 四、JavaScript 数据类型
  + 基本数据类型
    - 1. 数字类型
    - 2.字符串类型
      * 2.1 字符串拼接
    - 3.布尔类型
    - 4.undefined
    - 5.null
* 五、类型转换
  + 1.隐式转换
  + 2.显式转换
    - 2.1 Number
    - 2.2 parseInt
    - 2.3 parseFloat
* 六、模版字符串
* 七、运算符
  + 1. 算术运算符
  + 2. 赋值运算符
  + 3. 自增/自减运算符
  + 4. 比较运算符
  + 5. 逻辑运算符
  + 6. 运算符优先级
* 八、语句
  + 1. 表达式和语句
  + 2. 分支语句
    - 2.1 if 单分支语句
    - 2.2 if双分支语句
    - 2.3 if 多分支语句
    - 2.4 三元运算符（三元表达式）
    - 2.5 switch语句
  + 3. 断点调试
  + 4. 循环语句
    - 4.1 while循环
    - 4.2 中止循环
    - 4.3 无限循环
  + 5. for 语句
    - 5.1 for语句的基本使用
    - 5.2 循环嵌套
* 九、数组
  + 1. 定义数组和数组单元
  + 2. 访问数组和数组索引
  + 3. 数据单元值类型
  + 4. 数组长度属性
  + 5. 遍历数组
  + 6. 操作数组
    - 6.1 数组操作：修改
    - 6.2 数组操作：新增
    - 6.3 数组操作：删除
  + 7. 数组排序
* 十、函数
  + 1. 声明和调用
    - 1.1 声明（定义）
    - 1.2 调用
  + 2. 函数传参
  + 3. 函数返回值
  + 4. 作用域
  + 5. 匿名函数
    - 5.1 函数表达式
    - 5.2 立即执行函数
  + 6. 逻辑中断
  + 7. 转换为布尔型
* 十一、对象
  + 1. 对象声明语法
  + 2. 属性和访问
  + 3. 对象的操作：增 删 改 查
  + 4. 方法和调用
  + 5. null
  + 6. 遍历对象
  + 7. 遍历数组对象
  + 8. 内置对象
    - 8.1 Math
      * 8.1.1 属性
      * 8.1.2 方法
    - 8.2 随机数函数
  + 9. 基本数据类型和引用数据类型
    - 9.1 堆、栈空间分配区别：

# 一、JavaScript 简介

JavaScript 是互联网上最流行的脚本语言，这门语言可用于 HTML 和 web，更可广泛用于服务器、PC、笔记本电脑、平板电脑和智能手机等设备。

JavaScript 是脚本语言

JavaScript 是一种轻量级的编程语言。

JavaScript 是可插入 HTML 页面的编程代码。

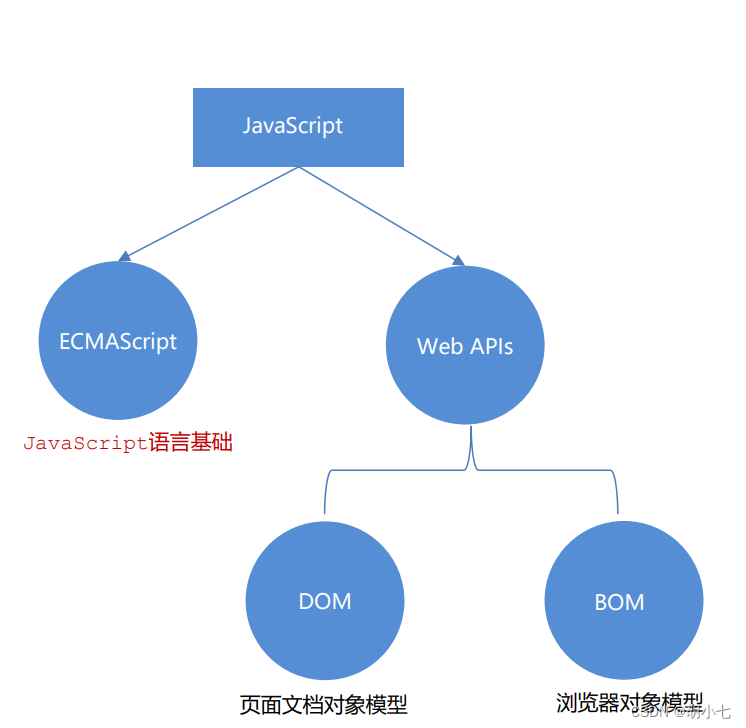
JavaScript 插入 HTML 页面后，可由所有的现代浏览器执行。

## 1. 为什么学习 JavaScript?

JavaScript 是 web 开发人员必须学习的 3 门语言中的一门：

1. ****HTML**** 定义了网页的内容
2. ****CSS**** 描述了网页的布局
3. ****JavaScript**** 控制了网页的行为

## 2. JavaScript 组成



**ECMAScript**：规定了js基础语法核心知识。

比如：变量、分支语句、循环语句、对象等等

**Web APIs**：

* + DOM 操作文档，比如对页面元素进行移动、大小、添加删除等操作
  + BOM 操作浏览器，比如页面弹窗，检测窗口宽度、存储数据到浏览器等等

## 3. 引入方式

JavaScript 程序不能独立运行，它需要被嵌入 HTML 中，然后浏览器才能执行

JavaScript 代码。通过 script 标签将 JavaScript 代码引入到 HTML 中，有三种方式：

### 3.1 内部方式

通过 script 标签包裹 JavaScript 代码

<body><!-- 内部形式：通过 script 标签包裹 JavaScript 代码 --><script>alert('欢迎学习JavaScript技术！')</script></body>

### 3.2 外部方式

一般将 JavaScript 代码写在独立的以 .js 结尾的文件中，然后通过 script 标签的 src 属性引入

// demo.js

document.write('一起来学习JavaScript技术吧！')

<body><!-- 外部方式：通过 script 的 src 属性引入独立的 .js 文件 --><script src="demo.js"></script></body>

如果 script 标签使用 src 属性引入了某 .js 文件，那么 标签的代码会被忽略！！！如下代码所示：

<body><!-- 外部方式：通过 script 的 src 属性引入独立的 .js 文件 --><script src="demo.js">// 此处的代码会被忽略掉！！！！alert(666)</script></body>

### 3.3 行内方式

<body><button onclick="alert('哈哈~逗你玩')">点我月薪过万~</button></body>

## 4. 注释

通过注释可以屏蔽代码被执行或者添加备注信息，JavaScript 支持两种形式注释语法：

### 4.1 单行注释

使用 // 注释单行代码

<body> <script>// 这种是单行注释的语法// 一次只能注释一行// 可以重复注释document.write('欢迎学习JavaScript技术！')</script></body>

### 4.2 多行注释

使用 /\* \*/ 注释多行代码

<body> <script>/\* 这种的是多行注释的语法 \*//\*更常见的多行注释是这种写法这些可以任意换行多少行都可以\*/document.write('欢迎学习JavaScript技术！')</script></body>

****注：编辑器中单行注释的快捷键为**ctrl + /**

## 5. 结束符

在 JavaScript 中 ; 代表一段代码的结束，多数情况下可以省略 ; 使用回车（enter）替代。

<body><script> alert(1);alert(2);alert(1)alert(2)</script></body>

实际开发中有许多人主张书写 JavaScript 代码时省略结束符 ;

## 6. 输入和输出

输出和输入也可理解为人和计算机的交互，用户通过键盘、鼠标等向计算机输入信息，计算机处理后再展示结果给用户，这便是一次输入和输出的过程。

举例说明：如按键盘上的方向键，向上/下键可以滚动页面，按向上/下键这个动作叫作输入，页面发生了滚动了这便叫输出。

### 6.1 输出

JavaScript 可以接收用户的输入，然后再将输入的结果输出：

alert()、document.wirte()

以数字为例，向 alert() 或 document.write()输入任意数字，他都会以弹窗形式或网页形式展示（输出）给用户。

### 6.2 输入

向 prompt() 输入任意内容会以弹窗形式出现在浏览器中，一般提示用户输入一些内容。

<body><script> // 1. 输入的任意数字，都会以弹窗形式展示document.write('要输出的内容')alert('要输出的内容')// 2. 以弹窗形式提示用户输入姓名，注意这里的文字使用英文的引号prompt('请输入您的姓名:')</script></body>

# 二、变量

变量是计算机中用来存储数据的“容器”，它可以让计算机变得有记忆，通俗的理解变量就是使用【某个符号】来代表【某个具体的数值】（数据）

<script>// x 符号代表了 5 这个数值x = 5// y 符号代表了 6 这个数值y = 6//举例： 在 JavaScript 中使用变量可以将某个数据（数值）记录下来！// 将用户输入的内容保存在 num 这个变量（容器）中num = prompt('请输入一数字!')// 通过 num 变量（容器）将用户输入的内容输出出来alert(num)document.write(num)</script>

## 1.声明

声明(定义)变量有两部分构成：声明关键字、变量名（标识）

<body><script> // let 变量名// 声明(定义)变量有两部分构成：声明关键字、变量名（标识）// let 即关键字，所谓关键字是系统提供的专门用来声明（定义）变量的词语// age 即变量的名称，也叫标识符let age</script></body>

关键字是 JavaScript 中内置的一些英文词汇（单词或缩写），它们代表某些特定的含义，如 let 的含义是声明变量的，看到 let 后就可想到这行代码的意思是在声明变量，如 let age;

let 和 var 都是 JavaScript 中的声明变量的关键字，推荐使用 let 声明变量！！！

## 2.赋值

声明（定义）变量相当于创造了一个空的“容器”，通过赋值向这个容器中添加数据。

<body><script> // 声明(定义)变量有两部分构成：声明关键字、变量名（标识）// let 即关键字，所谓关键字是系统提供的专门用来声明（定义）变量的词语// age 即变量的名称，也叫标识符let age// 赋值，将 18 这个数据存入了 age 这个“容器”中age = 18// 这样 age 的值就成了 18document.write(age)// 也可以声明和赋值同时进行let str = 'hello world!'alert(str)</script></body>

## 3.关键字

JavaScript 使用专门的关键字 let 和 var 来声明（定义）变量，在使用时需要注意一些细节：

以下是使用 let 时的注意事项：

1. 允许声明和赋值同时进行
2. 不允许重复声明
3. 允许同时声明多个变量并赋值
4. JavaScript 中内置的一些关键字不能被当做变量名

以下是使用 var 时的注意事项：

1. 允许声明和赋值同时进行
2. 允许重复声明
3. 允许同时声明多个变量并赋值

大部分情况使用 let 和 var 区别不大，但是 let 相较 var 更严谨，因此推荐使用 let。

## 4.变量名命名规则

关于变量的名称（标识符）有一系列的规则需要遵守：

1. 只能是字母、数字、下划线、$，且不能能数字开头
2. 字母区分大小写，如 Age 和 age 是不同的变量
3. JavaScript 内部已占用于单词（关键字或保留字）不允许使用
4. 尽量保证变量具有一定的语义，见字知义

注：所谓关键字是指 JavaScript 内部使用的词语，如 let 和var，保留字是指 JavaScript 内部目前没有使用的词语，但是将来可能会使用词语。

<body><script> let age = 18 // 正确let age1 = 18 // 正确let \_age = 18 // 正确// let 1age = 18 // 错误，不可以数字开头let $age = 18 // 正确let Age = 24 // 正确，它与小写的 age 是不同的变量// let let = 18 // 错误，let 是关键字let int = 123 // 不推荐，int 是保留字</script></body>

# 三、常量

概念：使用 const 声明的变量称为“常量”。

使用场景：当某个变量永远不会改变的时候，就可以使用 const 来声明，而不是let。

命名规范：和变量一致

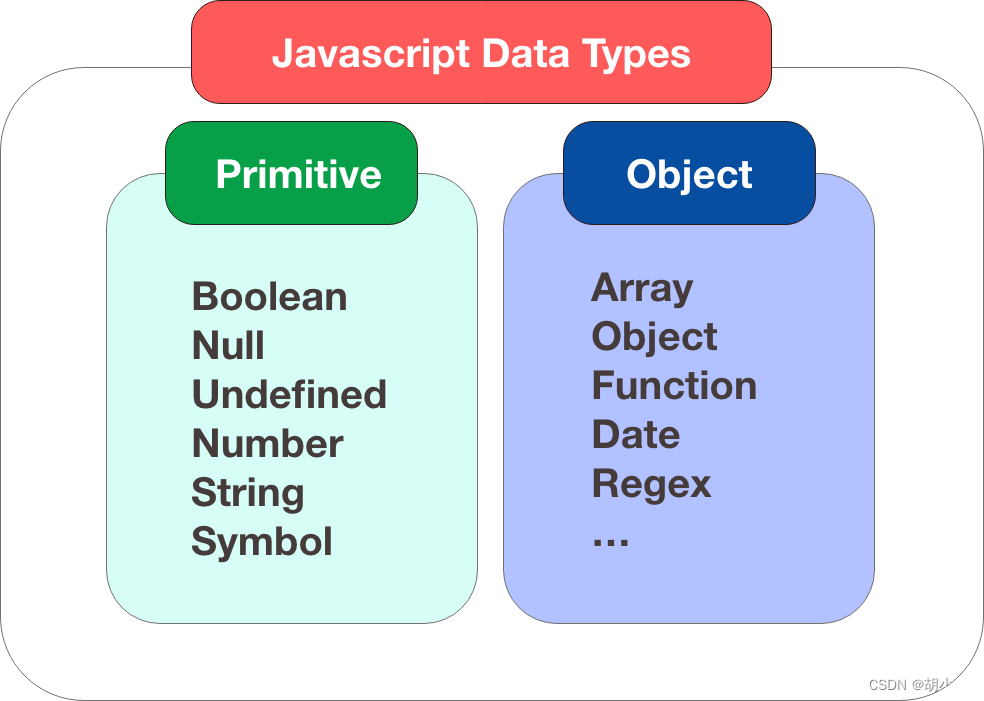
const PI = 3.14

注意： 常量不允许重新赋值,声明的时候必须赋值（初始化）

# 四、JavaScript 数据类型

****值类型(基本类型)：**** 字符串（String）、数字(Number)、布尔(Boolean)、空（Null）、未定义（Undefined）、Symbol。

****引用数据类型（对象类型）****：对象(Object)、数组(Array)、函数(Function)，还有两个特殊的对象：正则（RegExp）和日期（Date）。



注：Symbol 是 ES6 引入了一种新的原始数据类型，表示独一无二的值。

## 基本数据类型

计算机世界中的万事万物都是数据。

计算机程序可以处理大量的数据，为了方便数据的管理，将数据分成了不同的类型：

注：通过 typeof 关键字检测数据类型

<body><script> // 检测 1 是什么类型数据，结果为 numberdocument.write(typeof 1)</script></body>

### 1. 数字类型

即我们数学中学习到的数字，可以是整数、小数、正数、负数

<body><script> let score = 100 // 正整数let price = 12.345 // 小数let temperature = -40 // 负数document.write(typeof score) // 结果为 numberdocument.write(typeof price) // 结果为 numberdocument.write(typeof temperature) // 结果为 number</script></body>

JavaScript 中的数字类型与数学中的数字是一样的，分为正数、负数、小数等。

### 2.字符串类型

通过单引号（ ''） 、双引号（ ""）或反引号包裹的数据都叫字符串，单引号和双引号没有本质上的区别，推荐使用单引号。

注意事项：

1. 无论单引号或是双引号必须成对使用
2. 单引号/双引号可以互相嵌套，但是不以自已嵌套自已
3. 必要时可以使用转义符 \，输出单引号或双引号

<body><script> let user\_name = '小明' // 使用单引号let gender = "男" // 使用双引号let str = '123' // 看上去是数字，但是用引号包裹了就成了字符串了let str1 = '' // 这种情况叫空字符串documeent.write(typeof user\_name) // 结果为 stringdocumeent.write(typeof gender) // 结果为 stringdocumeent.write(typeof str) // 结果为 string</script></body>

#### 2.1 字符串拼接

字符串拼接：数字相加，字符相连

<body><script>let number = 1 + 1document.write(number) // 2let age = '刘德华' + '40岁了'document.write(age) // 刘德华40岁了let age1 = 50document.write('成龙今年' + age1 + '岁了') // 成龙今年50岁了</script></body>

### 3.布尔类型

表示肯定或否定时在计算机中对应的是布尔类型数据，它有两个固定的值 true 和 false，表示肯定的数据用 true，表示否定的数据用 false。

<body><script> let isCool = true // 是isCool = false // 否document.write(typeof isCool) // 结果为 boolean</script></body>

### 4.undefined

未定义是比较特殊的类型，只有一个值 undefined，只声明变量，不赋值的情况下，变量的默认值为 undefined，一般很少【直接】为某个变量赋值为 undefined。

<body><script> // 只声明了变量，并末赋值let tmp;document.write(typeof tmp) // 结果为 undefined</script></body>

### 5.null

null：空类型 表示内容为空

null 和 undefined 区别：

undefined 表示没有赋值

null 表示赋值了，但是内容为空

****注：JavaScript 中变量的值决定了变量的数据类型。****

# 五、类型转换

1. 使用表单、prompt 获取过来的数据默认都是字符串类型的，此时就不能直接简单的进行加法运算。
2. 此时需要转换变量的数据类型。
3. 通俗来说，就是把一种数据类型的变量转换成我们需要的数据类型。

## 1.隐式转换

某些运算符被执行时，系统内部自动将数据类型进行转换，这种转换称为隐式转换。

<body><script> console.log(11 + 11) // 22// + 号两边只要有一个是字符串，都会把另外一个转换成字符串console.log('11' + 11) // 1111console.log(5 - 5) // 0// 除了 + 号以外的算术运算符，比如：- 、 \* 、 / 等都会把数据转换为数字类型console.log('5' - 5) //0// + 号作为正号解析可以转换为数字类型console.log(typeof +'123') // numberconsole.log(+'22' + 22) // 44// 任何数据和字符串相加结果都是字符串</script></body>

## 2.显式转换

编写程序时过度依靠系统内部的隐式转换是不严禁的，因为隐式转换规律并不清晰，大多是靠经验总结的规律。为了避免因隐式转换带来的问题，通常根逻辑需要对数据进行显示转换。

### 2.1 Number

通过 Number 显示转换成数值类型，当转换失败时结果为 NaN（Not a Number）即不是一个数字。

<body><script>let t = '12'let f = 8// 显式将字符串 12 转换成数值 12t = Number(t)// 检测转换后的类型// console.log(typeof t);console.log(t + f) // 结果为 20// 并不是所有的值都可以被转成数值类型let str = 'hello'// 将 hello 转成数值是不现实的，当无法转换成// 数值时，得到的结果为 NaN （Not a Number）console.log(Number(str))</script></body>

### 2.2 parseInt

只保留整数 parseInt(数据)

<body><script>console.log(parseInt(12.33)) // 12</script></body>

### 2.3 parseFloat

可以保留小数 parseFloat(数据)

<body><script>console.log(parseFloat(12.66)) //12.66</script></body>

# 六、模版字符串

语法：${变量}

<body><script>// 模版字符串let age = 18document.write(`我今年${age}岁了`)</script></body>

# 七、运算符

## 1. 算术运算符

数字是用来计算的，比如：乘法 \* 、除法 / 、加法 + 、减法 - 等等，所以经常和算术运算符一起。

算术运算符：也叫数学运算符，主要包括加、减、乘、除、取余（求模）等

| **运算符** | **作用** |
| --- | --- |
| + | 求和 |
| - | 求差 |
| \* | 求积 |
| / | 求商 |
| ****%**** | 取模（取余数），开发中经常用于作为某个数字是否被整除 |

注意：在计算失败时，显示的结果是 NaN （not a number）

// 算术运算符console.log(1 + 2 \* 3 / 2) // 4 let num = 10console.log(num + 10) // 20console.log(num + num) // 20// 1. 取模(取余数) 使用场景： 用来判断某个数是否能够被整除console.log(4 % 2) // 0 console.log(6 % 3) // 0console.log(5 % 3) // 2console.log(3 % 5) // 3// 2. 注意事项 : 如果我们计算失败，则返回的结果是 NaN (not a number)console.log('刘德华' - 2) // NaN console.log('刘德华' \* 2) // NaN console.log('刘德华' + 2) // 刘德华2

## 2. 赋值运算符

赋值运算符：对变量进行赋值的运算符

= 将等号右边的值赋予给左边, 要求左边必须是一个容器

| **运算符** | **作用** |
| --- | --- |
| += | 加法赋值 |
| -+ | 减法赋值 |
| \*= | 乘法赋值 |
| /= | 除法赋值 |
| %= | 取余赋值 |

let num = 1// num = num + 1// 采取赋值运算符// num += 1num += 3console.log(num) // 4

## 3. 自增/自减运算符

| **符号** | **作用** | **说明** |
| --- | --- | --- |
| ++ | 自增 | 变量自身的值加1，例如: x++ |
| – | 自减 | 变量自身的值减1，例如: x– |

自增和自减经常用于计数来使用

// 自增：-- (让变量的值 +1)let i = 1i++console.log(i) // 2// 自减：-- (让变量的值 -1)let num = 4num--console.log(num) // 3

注意：

1. 只有变量能够使用自增和自减运算符
2. ++、-- 可以在变量前面也可以在变量后面，比如: x++ 或者 ++x

自增运算符分为前置自增和后置自增

1. 前置自增

let age1 = 1++age1console.log(age1) // 2

1. 后置自增

let age2 = 1age2++console.log(age2) // 2

前置自增和后置自增单独使用时二者并没有差别，都是每执行1次，当前变量数值加1

但参与运算时，两者就有区别

// 前置自增：先自加再使用（记忆口诀：++在前 先加）let num1 = 1console.log(++num1 + 2) // 4 (此时num1 是2, num1 先自加1变成2之后, 再跟后面的2相加)// 后置自增：先使用再自加（记忆口诀：++在后 后加）let num2 = 1console.log(num2++ + 2) // 3 (此时num2 是1, num2 先和后面的2相加, 先运算步骤完成之后, num2再自加1, 变成2)// 面试题：let a = 1/\* 1. a++还是1，但a完成第一步后，就会自加1，此时a就变为2了2. ++a就是a再自加1，就是2 + 1 = 3，此时a就变为3了3. 所以就是 1 + 3 + 3 = 7\*/console.log(a++ + ++a + a) // 7

## 4. 比较运算符

使用场景：比较两个数据大小、是否相等，根据比较结果返回一个布尔值（true / false）

| **运算符** | **作用** |
| --- | --- |
| > | 左边是否大于右边 |
| < | 左边是否小于右边 |
| >= | 左边是否大于或等于右边 |
| <= | 左边是否小于或等于右边 |
| === | 左右两边是否类型和值都相等（重点） |
| == | 左右两边值是否相等 |
| != | 左右值不相等 |
| !== | 左右两边是否不全等 |

console.log(3 > 5) // falseconsole.log(3 >= 3) // true console.log(2 == 2) // true// 比较运算符有隐式转换 把'2' 转换为 2 双等号 只判断值console.log(2 == '2') // trueconsole.log(undefined === null) // false// === 全等 判断 值 和 数据类型都一样才行console.log(2 === '2') // falseconsole.log(NaN === NaN) // NaN 不等于任何人，包括他自己console.log(2 !== '2') // true console.log(2 != '2') // false console.log('a' < 'b') // trueconsole.log('aa' < 'ab') // trueconsole.log('aa' < 'aac') // true

1.字符串比较，是比较的字符对应的ASCII码。从左往右依次比较，如果第一位一样再比较第二位，以此类推

2.NaN不等于任何值，包括它本身。涉及到 “NaN” 都是false

3.尽量不要比较小数，因为小数有精度问题

4.不同类型之间比较会发生隐式转换。最终把数据隐式转换转成number类型再比较

## 5. 逻辑运算符

使用场景：可以把多个布尔值放到一起运算，最终返回一个布尔值

| **符号** | **名称** | **日常读法** | **特点** | **口诀** |
| --- | --- | --- | --- | --- |
| && | 逻辑与 | 并且 | 符号两边有一个假的结果为假 | 一假则假 |
| || | 逻辑或 | 或者 | 符号两边有一个真的结果为真 | 一真则真 |
| ! | 逻辑非 | 取反 | true变false false变true | 真变假，假变真 |

// 逻辑与 && (并且) 符号两边都为true，结果才为true。 口诀：一假则假console.log(true && true) // trueconsole.log(true && false) // false// 逻辑或 || (或者) 符号两边有一个true就为true。 口诀：一真则真console.log(true || false) // true// 逻辑非 ! (取反) true变false，false变true。 口诀：真变假，假变真console.log(!true) // false

## 6. 运算符优先级

运算符优先级：

1. 小括号 （）
2. 一元运算符 ++ – ！
3. 算术运算符 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 ,
9. 一元运算符里面的逻辑非（！）优先级很高
10. 逻辑与（&&）比逻辑或（||）优先级高

// false true falselet a = 3 > 5 && 2 < 7 && 3 == 4console.log(a); // false// true true truelet b = 3 <= 4 || 3 > 1 || 3 != 2console.log(b); // true// falselet c = 2 === "2"console.log(c); // false// true falselet d = !c || b && aconsole.log(d); // true

逻辑运算符优先级： ！> && > ||

# 八、语句

## 1. 表达式和语句

表达式：表达式是可以被求值的代码，JavaScript 引擎会将其计算出一个结果。

let num = 3 + 4

语句：语句是一段可以执行的代码。

比如： prompt() 可以弹出一个输入框，还有 if语句 for 循环语句等等

prompt()

## 2. 分支语句

程序三大流程控制语句：

1. 以前我们写的代码，写几句就从上往下执行几句，这种叫顺序结构
2. 有的时候要根据条件选择执行代码，这种就叫分支结构
3. 某段代码被重复执行，就叫循环结构

分支语句可以根据条件判定真假，来选择性的执行想要的代码

分支语句包含：

1. if分支语句
2. 三元运算符
3. switch语句

### 2.1 if 单分支语句

语法：

if(条件表达式) {// 满足条件要执行的语句}

小括号内的条件结果是布尔值，为 true 时，进入大括号里执行代码；为false，则不执行大括号里面代码

小括号内的结果若不是布尔类型时，会发生类型转换为布尔值，类似Boolean()

如果大括号只有一个语句，大括号可以省略，但是，不提倡这么做

// 当小括号内的条件为true时，进入大括号里执行代码// 小括号内的结果若不是布尔类型时，会发生隐式转换转为布尔类型if (true) {console.log('执行代码')}// 0 为假，其他的所有数字都是真，包括负数、小数等。假就不会执行代码if (0) {console.log('执行代码')}// 空字符串 '' 也是假if ('') {console.log('执行代码')}// 输入成绩，判断成绩是否优秀let num = +prompt('请输入成绩：')if (num > 90) {alert('您的成绩为优秀！')}

### 2.2 if双分支语句

如果有两个条件的时候，可以使用 if else 双分支语句

if (条件表达式){// 满足条件要执行的语句} else {// 不满足条件要执行的语句}

例如：

<script>// 1. 用户输入let uname = prompt('请输入用户名:')let pwd = prompt('请输入密码:')// 2. 判断输出if (uname === 'admin' && pwd === '123456') {alert('恭喜登录成功')} else {alert('用户名或者密码错误')}</script>

### 2.3 if 多分支语句

使用场景： 适合于有多个条件的时候

/\* f多分支语句：if (条件1) { // 先判断条件1，若满足条件1就执行代码1，其他不执行 代码1} else if (条件2) { // 若不满足则向下判断条件2，满足条件2执行代码2，其他不执行 代码2} else if (条件3) { // 若依然不满足继续往下判断，依次类推 代码3} else { // ......代码n // 若以上条件都不满足，执行else里的代码n}

\*/// 判断成绩let result = +prompt('请输入成绩：')if (result > 90) {alert('您的成绩为优秀！')} else if (result >= 70 && result <= 90) {alert('您的成绩为良好！')} else if (result >= 60 && result < 70) {alert('您的成绩为及格！')} else {alert('您的成绩为不及格！')}

### 2.4 三元运算符（三元表达式）

****使用场景****： 一些简单的双分支，可以使用 三元运算符（三元表达式），写起来比 if else双分支 更简单

****符号****：? 与 : 配合使用

语法：

条件 ？ 满足条件要执行的代码 : 不满足条件执行的代码

例如：

// 三元运算符（三元表达式）// 1. 语法格式// 条件 ? 表达式1 : 表达式2 // 2. 执行过程 // 2.1 如果条件为真，则执行表达式1// 2.2 如果条件为假，则执行表达式2// 3. 验证// 5 > 3 ? '真的' : '假的'

console.log(5 < 3 ? '真的' : '假的')// 判断最大数let num1 = +prompt('请输入第一个数：')let num2 = +prompt('请输入第二个数：')

console.log(num1 > num2 ? num1 : num2)// 1. 用户输入 let num = prompt('请您输入一个数字:')// 2. 判断输出- 小于10才补0// num = num < 10 ? 0 + num : num

num = num >= 10 ? num : 0 + numalert(num)

### 2.5 switch语句

使用场景： 适合于有多个条件的时候，也属于分支语句，大部分情况下和 if多分支语句 功能相同

注意：

1. switch case语句一般用于等值判断, if适合于区间判断
2. switch case一般需要配合break关键字使用 没有break会造成case穿透
3. if 多分支语句开发要比switch更重要，使用也更多

例如：

// switch分支语句// 1. 语法// switch (表达式) {// case 值1:// 代码1// break// case 值2:// 代码2// break// ...// default:// 代码n// }// 简单计算器let num1 = +prompt('请输入第一个数：')let num2 = +prompt('请输入第二个数：')let operator = prompt('请输入运算符：')switch (operator) {case '+':alert(`结果为：${num1 + num2}`)breakcase '-':alert(`结果为：${num1 - num2}`)breakcase '\*':alert(`结果为：${num1 \* num2}`)breakcase '/':alert(`结果为：${num1 / num2}`)breakdefault:alert(`请输入+ - \* /`)}

****if 多分支语句和 switch的区别：****

共同点

* + 都能实现多分支选择， 多选1
  + 大部分情况下可以互换

区别：

* + switch…case语句通常处理case为比较****确定值****的情况，而if…else…语句更加灵活，通常用于****范围判断****(大于，等于某个范围)。
  + switch 语句进行判断后直接执行到程序的语句，效率更高，而if…else语句有几种判断条件，就得判断多少次
  + switch 一定要注意 必须是 === 全等，一定注意 数据类型，同时注意break否则会有穿透效果
  + 结论：
    - 当分支比较少时，if…else语句执行效率高。
    - 当分支比较多时，switch语句执行效率高，而且结构更清晰。

## 3. 断点调试

****作用：**** 学习时可以帮助更好的理解代码运行，工作时可以更快找到bug

浏览器打开调试界面

1. 按F12打开开发者工具
2. 点到源代码一栏 （ sources ）
3. 选择代码文件

****断点：**** 在某句代码上加的标记就叫断点，当程序执行到这句有标记的代码时会暂停下来

## 4. 循环语句

使用场景：重复执行 指定的一段代码。

### 4.1 while循环

while : 在…. 期间， 所以 while循环 就是在满足条件期间，重复执行某些代码。

****语法：****

while (条件表达式) {// 循环体 }/\*

释义：

1. 跟if语句很像，都要满足小括号里的条件为true才会进入 循环体 执行代码

2. while大括号里代码执行完毕后不会跳出，而是继续回到小括号里判断条件是否满足。若满足又执行大括号里的代码，然后再回到小括号判断条件，直到小括号内条件不满足，即跳出

\*/

例如：

// while循环: 重复执行代码// 页面中打印输出10句“月薪过万”let i = 1while (i <= 10) {document.write('月薪过万<br />')i++ // 这里千万不要忘了变量自增否则造成死循环}

循环三要素：

1.初始值 （经常用变量）

2.终止条件（没有终止条件，循环会一直执行，造成死循环）

3.变量的变化量（用自增或者自减）

例如：

<script>// 1. 变量的起始值// let i = 12. 终止条件// while (i <= 3) {// document.write('我要循环三次 <br>')// 3. 变量的变化量// i++// }// 1. 变量的起始值let end = +prompt('请输入次数:')let i = 1// 2. 终止条件while (i <= end) {document.write('我要循环三次 <br>')// 3. 变量的变化量i++}</script>

### 4.2 中止循环

break 退出整个循环，一般用于结果已经得到, 后续的循环不需要的时候可以使用

continue 退出本次循环，一般用于排除或者跳过某一个选项的时候, 可以使用continue

<script>// break 退出循环let a = 1while (a <= 5) {if (a === 3) {break}document.write(`${a}<br />`)a++}// continue 结束本次循环，继续下次循环let b = 1while (b <= 5) {if (b === 3) {b++continue}document.write(`${b}<br />`)b++}</script>

### 4.3 无限循环

1.while(true) 来构造“无限”循环，需要使用break退出循环。（常用）

2.for(; ;) 也可以来构造“无限”循环，同样需要使用break退出循环。

// 无限循环 // 需求： 页面会一直弹窗询问你爱我吗？// (1). 如果用户输入的是 '爱'，则退出弹窗// (2). 否则一直弹窗询问// 1. while(true) 无限循环// while (true) {// let love = prompt('你爱我吗?')// if (love === '爱') {// break// }// }// 2. for(;;) 无限循环for (; ;) {let love = prompt('你爱我吗?')if (love === '爱') {break}}

## 5. for 语句

for 是 JavaScript 提供的另一种循环控制的话句，它和 while 只是语法上存在差异。

1. 作用：重复执行代码
2. 好处：把声明起始值、循环条件、变化值写到一起，让人一目了然，它是最常使用的循环形式

### 5.1 for语句的基本使用

1. 实现循环的 3 要素

<script>// 1. 语法格式// for(起始值; 终止条件; 变化量) {// // 要重复执行的代码// }// 2. 示例：在网页中输入标题标签// 起始值为 1// 变化量 i++// 终止条件 i <= 6for(let i = 1; i <= 6; i++) {document.write(`<h${i}>循环控制，即重复执行<h${i}>`)}</script>

变化量和死循环，for 循环和 while 一样，如果不合理设置增量和终止条件，便会产生死循环。

跳出和终止循环

<script>// 1. continue for (let i = 1; i <= 5; i++) {if (i === 3) {continue // 结束本次循环，继续下一次循环}console.log(i)}// 2. breakfor (let i = 1; i <= 5; i++) {if (i === 3) {break // 退出结束整个循环}console.log(i)}</script>

结论：

* JavaScript 提供了多种语句来实现循环控制，但无论使用哪种语句都离不开循环的3个特征，即起始值、变化量、终止条件，做为初学者应着重体会这3个特征，不必过多纠结三种语句的区别。
* 起始值、变化量、终止条件，由开发者根据逻辑需要进行设计，规避死循环的发生。
* 当如果明确了循环的次数的时候推荐使用for循环,当不明确循环的次数的时候推荐使用while循环

注意：for 的语法结构更简洁，故 for 循环的使用频次会更多。

### 5.2 循环嵌套

for 循环嵌套: 一个循环里再套一个循环，一般用在for循环里

// for (外部声明记录循环次数的变量; 循环条件; 变化量) {// for (内部声明记录循环次数的变量; 循环条件; 变化量) {// 循环体// }// }// 外面执行一次，里面执行五次for (let i = 1; i <= 3; i++) {document.write(`第${i}天 <br />`)for (let j = 1; j <= 5; j++) {document.write(`今天学习了${j}个单词 <br />`)}}

记住，外层循环循环一次，里层循环循环全部

# 九、数组

****数组：****(Array)是一种可以按顺序保存数据的数据类型

****使用场景：**** 如果有多个数据可以用数组保存起来，然后放到一个变量中，管理非常方便

// 1.数组：(Array)是一种可以按顺序保存数据的数据类型，数组可以存储任意类型的数据let arr = [10,20,30,40]// 2.声明数组:// let 数组名 = [数据1, 数据2, ..., 数据n]// let 数组名 = new Array(数据1, 数据2, ..., 数据n)// 3.使用数组：数组里面的值都有索引号，索引号是从 0 开始的

console.log(arr[0])// 4.数组长度：arr.length ,数组中值的个数

console.log(arr.length)

## 1. 定义数组和数组单元

<script>// 1. 语法，使用 [] 来定义一个空数组// 定义一个空数组，然后赋值给变量 classes// let classes = [];// 2. 定义非空数组let classes = ['小明', '小刚', '小红', '小丽', '小米']</script>

通过 [] 定义数组，数据中可以存放真正的数据，如小明、小刚、小红等这些都是数组中的数据，我们这些数据称为数组单元，数组单元之间使用英文逗号分隔。

## 2. 访问数组和数组索引

使用数组存放数据并不是最终目的，关键是能够随时的访问到数组中的数据（单元）。其实 JavaScript 为数组中的每一个数据单元都编了号，通过数据单元在数组中的编号便可以轻松访问到数组中的数据单元了。

我们将数据单元在数组中的编号称为索引值，也有人称其为下标。

索引值实际是按着数据单元在数组中的位置依次排列的，注意是从 0 开始的。

<script>let classes = ['小明', '小刚', '小红', '小丽', '小米']// 1. 访问数组，语法格式为：变量名[索引值]document.write(classes[0]) // 结果为：小明document.write(classes[1]) // 结果为：小刚document.write(classes[4]) // 结果为：小米// 2. 通过索引值还可以为数组单重新赋值document.write(classes[3]) // 结果为：小丽// 重新为索引值为 3 的单元赋值classes[3] = '小小丽'document.wirte(classes[3]); // 结果为： 小小丽</script>

## 3. 数据单元值类型

数组做为数据的集合，它的单元值可以是任意数据类型

<script>// 数组单值类型可以是任意数据类型// a) 数组单元值的类型为字符类型let list = ['HTML', 'CSS', 'JavaScript']// b) 数组单元值的类型为数值类型let scores = [78, 84, 70, 62, 75]// c) 混合多种类型let mixin = [true, 1, false, 'hello']</script>

## 4. 数组长度属性

重申一次，数组在 JavaScript 中并不是新的数据类型，它属于对象类型。

<script>// 定义一个数组let arr = ['html', 'css', 'javascript']// 数组对应着一个 length 属性，它的含义是获取数组的长度console.log(arr.length) // 3</script>

## 5. 遍历数组

用循环把数组中每个元素都访问到,一般会用for循环遍历

/\* 用循环把数组中每个元素都访问到,一般会用for循环遍历for (let i = 0; i < 数组名.length; i++) {数组名[i]}

\*/ let arr = ['马超','赵云', '张飞', '关羽','黄忠']for (i = 0; i < arr.length; i++){console.log(arr[i])}

## 6. 操作数组

数组做为对象数据类型，不但有 length 属性可以使用，还提供了许多方法：

1. push 动态向数组的尾部添加一个单元
2. unshit 动态向数组头部添加一个单元
3. pop 删除最后一个单元
4. shift 删除第一个单元
5. splice 动态删除任意单元

使用以上4个方法时，都是直接在原数组上进行操作，即成功调任何一个方法，原数组都跟着发生相应的改变。并且在添加或删除单元时 length 并不会发生错乱。

### 6.1 数组操作：修改

// 重新赋值 数组[下标] = 新值let arr = []

arr[0] = 20

arr[1] = '刘德华'

console.log(arr) // [20, '刘德华']arr[1] = '成龙'

console.log(arr) // [20, '成龙']

### 6.2 数组操作：新增

// 数组添加新的数据let arr = ['red', 'blue']// 数组.push(新增的内容) 将一个或多个元素添加到数组的末尾，并返回该数组的新长度

console.log(arr.push('green', 'pink')) // 4

console.log(arr) // ['red', 'blue', 'green', 'pink']// arr.unshift(新增的内容) 将一个或多个元素添加到数组的开头，并返回该数组的新长度

console.log(arr.unshift('orange', 'black')) // 6

console.log(arr) // ['orange', 'black', 'red', 'blue', 'green', 'pink']

### 6.3 数组操作：删除

<script>// 删除数组的元素（数据）let arr = ['orange', 'black', 'red', 'blue', 'green', 'pink']// 数组. pop() 从数组中删除最后一个元素，并返回该元素的值

console.log(arr.pop()) // pink

console.log(arr) // ['orange', 'black', 'red', 'blue', 'green']// 数组. shift() 从数组中删除第一个元素，并返回该元素的值

console.log(arr.shift()) // orange

console.log(arr) // ['black', 'red', 'blue', 'green']// 数组. splice(start, deleteCount) 方法 删除指定元素// start：指定删除的开始位置（从0计数）// deleteCount：表示要删除的数组元素的个数。可选的。 如果省略则默认从指定的起始位置删除到最后let newArr = [1, 2, 3, 4, 5, 6]

console.log(newArr.splice(2, 1)) // 3

console.log(newArr) // [1, 2, 4, 5, 6]newArr.splice(3)

console.log(newArr) // [1, 2, 4]</script>

## 7. 数组排序

let arr = [4, 2, 5, 1, 3]// 1.升序排列写法

arr.sort(function (a, b) {return a - b})

console.log(arr) // [1, 2, 3, 4, 5]// 降序排列写法

arr.sort(function (a, b) {return b - a})

console.log(arr) // [5, 4, 3, 2, 1]

# 十、函数

函数: function，是被设计为执行特定任务的代码块

## 1. 声明和调用

函数可以把具有相同或相似逻辑的代码“包裹”起来，通过函数调用执行这些被“包裹”的代码逻辑，这么做的优势是有利于精简代码方便复用。

比如我们前面使用的 alert() 、 prompt() 和 console.log() 都是一些 js 函数，只不过已经封装好了，我们直接使用的。

### 1.1 声明（定义）

声明（定义）一个完整函数包括关键字、函数名、形式参数、函数体、返回值5个部分

/\*

函数的声明语法:

function 函数名() {函数体

}函数名命名规范:

1. 和变量命名基本一致

2. 尽量小驼峰式命名法

3. 前缀应该为动词

4. 命名建议：常用动词约定动词：

can：判断是否可执行某个动作； has：判断是否含义某个值；

is：判断是否为某个值； get：获取某个值

set：设置某个值； load：加载某些数据

\*/

### 1.2 调用

声明（定义）的函数必须调用才会真正被执行，使用 () 调用函数。

<script>// 函数的调用语法：// 函数调用，这些函数体内的代码逻辑会被执行// 函数名()// 注意：声明（定义）的函数必须调用才会真正被执行，使用 () 调用函数// 函数一次声明可以多次调用，每一次函数调用函数体里面的代码都会重新执行一次// 我们曾经使用的 alert() , parseInt() 这种名字后面跟小括号的本质都是函数的调用// 函数体：// 函数体是函数的构成部分，它负责将相同或相似代码“包裹”起来，直到函数调用时函数体内的代码才会被执行。// 函数的功能代码都要写在函数体当中。function sayHi() { // 声明函数document.write('Hi~~') // 函数体}sayHi() // 函数调用</script>

注：函数名的命名规则与变量是一致的，并且尽量保证函数名的语义。

## 2. 函数传参

通过向函数传递参数，可以让函数更加灵活多变，参数可以理解成是一个变量。

<script>/\*

函数传参:function 函数名(参数列表) {函数体

} 参数列表:

1. 传入数据列表

2. 声明这个函数需要传入几个数据

3. 多个数据用逗号隔开调用:

函数名(传递的参数列表)

调用函数时，需要传入几个数据就写几个，用逗号隔开

\*/// 如果用户不输入实参，可以给 形参默认值，可以默认为 0, 这样程序更严谨// 说明：这个默认值只会在缺少实参参数传递时 才会被执行，所以有参数会优先执行传递过来的实参, 否则默认为undefinedfunction getSum(x = 0, y = 0) { // 形参：声明函数时写在函数名右边小括号里的叫形参（形式上的参数）document.write(x + y)}getSum(10, 20) // 实参：调用函数时写在函数名右边小括号里的叫实参（实际上的参数）</script>

## 3. 函数返回值

函数的本质是封装（包裹），函数体内的逻辑执行完毕后，函数外部如何获得函数内部的执行结果呢？要想获得函数内部逻辑的执行结果，需要通过 return 这个关键字，将内部执行结果传递到函数外部，这个被传递到外部的结果就是返回值。

<script>// 当调用某个函数，这个函数会返回一个结果出来。这个结果就是函数返回的值// 当函数需要返回数据出去时，用return关键字// return 数据function getSum(x = 0, y = 0) {return x + y}let num = getSum(10, 20)document.write(num)/\*

细节：

1. 在函数体中使用 return 关键字能将内部的执行结果交给函数外部使用

2. return后面不接数据或者函数内不写return，函数的返回值是undefined

3. return能立即结束当前函数, 所以 return 后面的数据不要换行写

\*/</script>

总结：

1. 在函数体中使用return 关键字能将内部的执行结果交给函数外部使用
2. 函数内部只能出现1 次 return，并且 return 下一行代码不会再被执行，所以return 后面的数据不要换行写
3. return会立即结束当前函数
4. 函数可以没有return，这种情况默认返回值为 undefined

## 4. 作用域

通常来说，一段程序代码中所用到的名字并不总是有效和可用的，而限定这个名字的可用性的代码范围就是这个名字的作用域。

作用域的使用提高了程序逻辑的局部性，增强了程序的可靠性，减少了名字冲突。

/\*

作用域分为：

1. 全局作用域：全局有效。作用于所有代码执行的环境(整个 script 标签内部)或者一个独立的 js 文件

2. 局部作用域：局部有效。作用于函数内的代码环境，就是局部作用域。 因为跟函数有关系，所以也称为函数作用域。在JavaScript中，根据作用域的不同，变量可以分为：

1. 全局变量：函数外部let 的变量。全局变量在任何区域都可以访问和修改

2. 局部变量：函数内部let的变量。局部变量只能在当前函数内部访问和修改注意：

变量有一个坑， 特殊情况：

如果函数内部，变量没有声明，直接赋值，也当全局变量看，但是强烈不推荐。

但是有一种情况，函数内部的形参可以看做是局部变量。\*/let num = 10function fn() {let num = 20console.log(num) // 20}fn()// 变量的访问原则：就近原则。在能够访问到的情况下 先局部， 局部没有在找全局

## 5. 匿名函数

函数可以分为具名函数和匿名函数

匿名函数：没有名字的函数,无法直接使用。

// 1. 具名函数function fn() {}fn()// 2. 匿名函数function () {} // 匿名函数：没有名字的函数, 无法直接使用。// 使用方式：// 1. 函数表达式// 2. 立即执行函数

### 5.1 函数表达式

// 函数表达式：将匿名函数赋值给一个变量，并且通过变量名称进行调用 我们将这个称为函数表达式// 声明let fn = function() { console.log('函数表达式')}// 调用fn()// 其中函数的形参和实参使用跟具名函数一致。

### 5.2 立即执行函数

// 立即执行函数：避免全局变量之间的污染// 不需要调用，立即执行// 方法1(function () {})();// 方法2(function () {} ());// 注意： 多个立即执行函数要用 ; 隔开，要不然会报错

## 6. 逻辑中断

// 其实类似参数的默认值写法// 只存在于 && 和 || 中，当满足一定条件后右边代码不执行function sum(x, y) {x = x || 0y = y || 0console.log(x + y)}sum(10, 20)// &&：左边为false，后面的代码就不看

console.log(false && 22) // false

console.log(false && 3 + 5) // falselet age = 18

console.log(false && age++) // false 后面的age++ 不执行

console.log(age) // 18 没有执行自加1的操作// ||：左边为true，后面的代码就不看

console.log(true || age++) // true 后面的age++ 也不执行

console.log(age) // 18console.log(11 && 22) // 22 都是真，这返回最后一个真值

console.log(11 || 22) // 11 输出第一个真值

## 7. 转换为布尔型

// ''、0、undefined、null、false、NaN 转换为布尔值后都是false, 其余则为trueconsole.log(false && 20) // false

console.log(5 < 3 && 20) // flase

console.log(undefined && 20) // undefined

console.log(null && 20) // null

console.log(0 && 20) // 0

console.log(10 && 20) // 20console.log(false || 20) // 20

console.log(5 < 3 || 20) // 20

console.log(undefined || 2) // 20

console.log(null || 20) // 20

console.log(0 || 20) // 20

console.log(10 || 20) // 10

# 十一、对象

对象（object）：JavaScript里的一种数据类型

可以理解为是一种无序的数据集合， 注意数组是有序的数据集合

## 1. 对象声明语法

声明一个对象类型的变量与之前声明一个数值或字符串类型的变量没有本质上的区别。

<script>// 1. let 对象名 = {} // 2. let 对象名 = new Object()// 声明对象类型变量，使用一对花括号// user 便是一个对象了，目前它是一个空对象let user = {}/\*

对象由属性和方法组成:

属性：信息或叫特征（名词）。 比如 手机尺寸、颜色、重量等…

方法：功能或叫行为（动词）。 比如 手机打电话、发短信、玩游戏…let 对象名 = {属性名: 属性值,方法名: 函数

}

\*/</script>

## 2. 属性和访问

数据描述性的信息称为属性，如人的姓名、身高、年龄、性别等，一般是名词性的。

1. 属性都是成 对出现的，包括属性名和值，它们之间使用英文 : 分隔
2. 多个属性之间使用英文 , 分隔
3. 属性就是依附在对象上的变量
4. 属性名可以使用 "" 或 ''，一般情况下省略，除非名称遇到特殊符号如空格、中横线等

<script>// 通过对象描述一个人的数据信息// person 是一个对象，它包含了一个属性 name// 属性都是成对出现的，属性名 和 值，它们之间使用英文 : 分隔let person = {name: '小明', // 描述人的姓名age: 18, // 描述人的年龄stature: 185, // 描述人的身高gender: '男', // 描述人的性别}</script>

声明对象，并添加了若干属性后，可以使用 . 或 [] 获得对象中属性对应的值，我称之为属性访问。

<script>// 通过对象描述一个人的数据信息// person 是一个对象，它包含了一个属性 name// 属性都是成对出现的，属性名 和 值，它们之间使用英文 : 分隔let person = {name: '小明', // 描述人的姓名age: 18, // 描述人的年龄stature: 185, // 描述人的身高gender: '男', // 描述人的性别};// 访问人的名字console.log(person.name) // 结果为 小明// 访问人性别console.log(person.gender) // 结果为 男// 访问人的身高console.log(person['stature']) // 结果为 185// 或者console.log(person.stature) // 结果同为 185</script>

扩展：也可以动态为对象添加属性，动态添加与直接定义是一样的，只是语法上更灵活。

<script>// 声明一个空的对象（没有任何属性）let user = {}// 动态追加属性user.name = '小明'user['age'] = 18// 动态添加与直接定义是一样的，只是语法上更灵活</script>

## 3. 对象的操作：增 删 改 查

let obj = {uname: '刘德华',age: '45',weight: '150kg'}// 查：对象名.属性

console.log(obj.uname) // 刘德华// 查的另外一种写法：对象['属性']

console.log(obj['age']) // 45// 改：对象名.属性 = 新值

obj.age = '50'

console.log(obj) // {uname: '刘德华', age: '50', weight: '150kg'}// 增：对象名.新属性 = 新值

obj.gender = '男'

console.log(obj) // {uname: '刘德华', age: '50', weight: '150kg', gender: '男'}// 改和增语法一样，判断标准就是对象有没有这个属性，没有就是新增，有就是改// 删：delete 对象名.属性delete obj.weight

console.log(obj) // {uname: '刘德华', age: '50', gender: '男'}

## 4. 方法和调用

数据行为性的信息称为方法，如跑步、唱歌等，一般是动词性的，其本质是函数。

1. 方法是由方法名和函数两部分构成，它们之间使用 : 分隔
2. 多个属性之间使用英文 , 分隔
3. 方法是依附在对象中的函数
4. 方法名可以使用 "" 或 ''，一般情况下省略，除非名称遇到特殊符号如空格、中横线等

<script>let obj = {uname: '刘德华',song: function() {console.log('冰雨')},num: function(x, y) {console.log(x + y)}}// 方法调用obj.song() // 冰雨// 对象方法也可以添加形参和实参obj.num(1, 2) // 3</script>

扩展：也可以动态为对象添加方法，动态添加与直接定义是一样的，只是语法上更灵活。

<script>// 声明一个空的对象（没有任何属性，也没有任何方法）let user = {}// 动态追加属性user.name = '小明'user.['age'] = 18// 动态添加方法user.move = function () {console.log('移动一点距离...')}</script>

****注：无论是属性或是方法，同一个对象中出现名称一样的，后面的会覆盖前面的。****

## 5. null

null 也是 JavaScript 中数据类型的一种，通常只用它来表示不存在的对象。使用 typeof 检测类型它的类型时，结果为 object。

## 6. 遍历对象

let obj = {uname: '刘德华',age: '45',weight: '150kg'}for (let k in obj) {// k 是获得对象的属性名 k = 'uname' [k] = ['uname'] 查的另外一种写法就是 对象['属性']

console.log(k) // uname age weight// obj[k] 是获得 属性值

console.log(obj[k]) // 刘德华 45 150kg}

for in 不提倡遍历数组 因为 k 是 字符串

## 7. 遍历数组对象

let students = [{name: '小明', age: 18, gender: '男', hometown: '河北省'},{name: '小红', age: 19, gender: '女', hometown: '河南省'},{name: '小刚', age: 17, gender: '男', hometown: '山西省'},{name: '小丽', age: 18, gender: '女', hometown: '山东省'}]for (let i = 0; i < students.length; i++) {// students[i] 每个对象console.log(students[i].name) // 小明 小红 小刚 小丽}

## 8. 内置对象

我们曾经使用过的 console.log，console其实就是 JavaScript 中内置的对象，该对象中存在一个方法叫 log，然后调用 log 这个方法，即 console.log()。

除了 console 对象外，JavaScritp 还有其它的内置的对象

### 8.1 Math

Math 是 JavaScript 中内置的对象，称为数学对象，这个对象下即包含了属性，也包含了许多的方法。

#### 8.1.1 属性

* Math.PI，获取圆周率

// 圆周率

console.log(Math.PI);

#### 8.1.2 方法

* Math.random，生成 0 到 1 间的随机数

// random：生成0-1之间的随机数（包含0不包括1）

console.log(Math.random()) // 0.7275848132638618 随机的小数

* Math.ceil，数字向上取整

// ceil：向上取整(舍弃小数部分，整数部分加1)

console.log(Math.ceil(1.9)) // 2

console.log(Math.ceil(1.1)) // 2

console.log(Math.ceil(-1.5)) // -1

* Math.floor，数字向下取整

// floor：向下取整(舍弃小数部分，整数部分不变)

console.log(Math.floor(1.9)) // 1

console.log(Math.floor(1.1)) // 1

console.log(Math.floor(-1.5)) // -2

* Math.round，四舍五入取整

// round：取整，四舍五入原则

console.log(Math.round(1.5)) // 2

console.log(Math.round(1.4)) // 1

console.log(Math.round(-1.5)) // -1

console.log(Math.round(-1.51)) // -2

* Math.max，在一组数中找出最大的

// max：找最大数

console.log(Math.max(1, 2, 3, 4, 5)) // 5

* Math.min，在一组数中找出最小的

// min：找最小数

console.log(Math.min(1, 2, 3, 4, 5)) // 1

* Math.pow，幂方法

// pow：幂运算 2 的 3 次方

console.log(Math.pow(2, 3)) // 8

* Math.sqrt，平方根

// sqrt：求某数的平方根

Math.sqrt(16) // 4

### 8.2 随机数函数

// Math.random() 随机数函数， 返回一个0 - 1之间，并且包括0不包括1的随机小数[0, 1）

console.log(Math.random())// 生成0-10的随机整数

console.log(Math.floor(Math.random() \* (10 + 1)))// 生成5-10的随机整数

console.log(Math.floor(Math.random() \* (5 + 1)) + 5)// 生成N-M之间的随机整数// Math.floor(Math.random() \* (M - N + 1)) + Nfunction getRandom(N, M) {return Math.floor(Math.random() \* (M - N + 1)) + N}

console.log(getRandom(4, 8))// 生成数组中随机一个元素let arr = ['red', 'blue', 'green']let ele = Math.floor(Math.random() \* arr.length)

console.log(arr[ele])

## 9. 基本数据类型和引用数据类型

简单类型又叫做基本数据类型或者值类型，复杂类型又叫做引用类型。

1. 值类型：简单数据类型/基本数据类型，在存储时变量中存储的是值本身，因此叫做值类型，如：string ，number，boolean，undefined，null
2. 引用类型：复杂数据类型，在存储时变量中存储的仅仅是地址（引用），因此叫做引用数据类型 通过 new 关键字创建的对象（系统对象、自定义对象），如 Object、Array、Date等

### 9.1 堆、栈空间分配区别：

1、栈（操作系统）：由操作系统自动分配释放存放函数的参数值、局部变量的值等。其操作方式类似于数据结构中的栈； （简单数据类型存放到栈里面）

2、堆（操作系统）：存储复杂类型(对象)，一般由程序员分配释放，若程序员不释放，由垃圾回收机制回收。（引用数据类型存放到堆里面）

****简单类型的内存分配：****

值类型（简单数据类型）： string ，number，boolean，undefined，null

值类型变量的数据直接存放在变量（栈空间）中

****复杂类型的内存分配：****

引用类型（复杂数据类型）：通过 new 关键字创建的对象（系统对象、自定义对象），如Object、Array、Date等

引用类型变量（栈空间）里存放的是地址，真正的对象实例存放在堆空间中

// 值类型变量// 是直接把 num1, num2 的值 10 分别存到栈中的两个空间内，修改了 num1 的值，但 num2 的值不会变let num1 = 10let num2 = num1

num1 = 20

console.log(num2) // 10// 引用类型变量// obj1 是在栈中的空间里存了一个地址，age: 18 是存在堆中，地址指向的就是这个堆中存 age: 18 的空间// obj2 = obj1 相当于是把 obj1 的地址给了 obj2，所以 obj2 地址指向的也是这同一个堆中的空间// 而因为obj1 跟 obj2 的地址相同，指向的是同一个变量空间，所以修改了 obj1 的变量，obj2 自然也就跟着改变了let obj1 = {age: 18}let obj2 = obj1

obj1.age = 20

console.log(obj2) // {age: 20}

