目录

[第一节：初识JavaScript 1](#_Toc6923)

[1.浏览器执行JS简介 1](#_Toc27148)

[2. JS组成 1](#_Toc2267)

[3.JS书写位置 2](#_Toc3364)

[3.1行内式JS 2](#_Toc4504)

[3.2内嵌式 3](#_Toc30212)

[3.3外部JS文件 3](#_Toc16985)

[4.注释 3](#_Toc17337)

[5.JS输入输出语句 3](#_Toc21229)

[第二节：变量 4](#_Toc22536)

[1.变量 4](#_Toc10884)

[2.变量在内存中的存储 4](#_Toc24532)

[3.变量的使用 4](#_Toc777)

[3.1声明变量 4](#_Toc13228)

[3.2赋值 4](#_Toc26608)

[3.3变量的初始化 5](#_Toc31178)

[4.变量语法扩展 5](#_Toc29885)

[4.1更新变量 5](#_Toc32011)

[4.2同时声明多个变量 5](#_Toc6301)

[4.3声明变量特殊情况 5](#_Toc2216)

[5.变量命名规范 5](#_Toc4020)

[第三节 数据类型 6](#_Toc22200)

[1.为什么需要数据类型 6](#_Toc2078)

[2.变量的数据类型 6](#_Toc20422)

[3.数据类型的分类 7](#_Toc22450)

[3.1简单数据类型(基本数据类型) 7](#_Toc3064)

[第四节 获取变量数据类型 11](#_Toc4347)

[1.获取检测变量的数据类型 11](#_Toc19103)

[2.字面量 11](#_Toc31001)

[第五节 数据类型转换 11](#_Toc31534)

[1.什么是数据类型转换 11](#_Toc11605)

[2.转换为字符串 11](#_Toc2783)

[3.转换为数字型 12](#_Toc32433)

[4.转换为布尔型 12](#_Toc1174)

[5.标识符、关键字、保留字 13](#_Toc31432)

[第六节：运算符 14](#_Toc15550)

[1.运算符 14](#_Toc5288)

[2.算数运算符 14](#_Toc20790)

[3.递增运算符 15](#_Toc13718)

[4.比较运算符 16](#_Toc17805)

[5.逻辑运算符 16](#_Toc31960)

[6.短路运算（逻辑中断） 17](#_Toc1554)

[7.赋值运算符 17](#_Toc28417)

[8. 运算符优先级 18](#_Toc12430)

[第七节 流程控制 18](#_Toc6217)

[1.流程控制 18](#_Toc14240)

[2. 顺序流程控制 19](#_Toc760)

[3. 分支流程控制if语句 19](#_Toc24073)

[4. 三元表达式 22](#_Toc10743)

[5. 分支流程控制switch语句 22](#_Toc12667)

[第八节 循环 24](#_Toc4342)

[1.循环 24](#_Toc10396)

[2. JS中的循环 24](#_Toc14037)

[3. for循环 24](#_Toc26203)

[4. 双重for循环 25](#_Toc31784)

[5. for循环小结 25](#_Toc29907)

[6. while循环 26](#_Toc10001)

[7. do while 循环 26](#_Toc7237)

[8. 循环小结 27](#_Toc14030)

[9. continue关键字 27](#_Toc24313)

[10.break关键字 27](#_Toc3226)

[11.JS命名规范以及语法格式 28](#_Toc10495)

[第九节 数组 29](#_Toc23779)

[1. 数组的创建方式 29](#_Toc31533)

[2. 利用new创建数组 29](#_Toc13024)

[3.利用数组字面量创建数组 29](#_Toc21191)

[4.数组元素的类型 29](#_Toc25088)

[5.获取数组中的元素 29](#_Toc23480)

[6.遍历数组 30](#_Toc23515)

[7.数组长度 30](#_Toc10564)

[8.数组中新增元素 30](#_Toc22142)

[9.冒泡排序 31](#_Toc20893)

[第十节 函数 32](#_Toc20046)

[1.函数的概念 32](#_Toc27325)

[2.函数的使用 33](#_Toc2011)

[3.函数的封装 33](#_Toc7026)

[4.函数的参数 34](#_Toc6477)

[5.函数的返回值 34](#_Toc18825)

[6.arguments的使用 35](#_Toc94)

[7.函数的两种声明方式 36](#_Toc30319)

[8.函数可以调用另外一个函数 36](#_Toc12541)

[第十一节 作用域 37](#_Toc9671)

[1.作用域 37](#_Toc10750)

[2.变量作用域的分类 37](#_Toc17861)

[3.作用域链 38](#_Toc18587)

[第十二节 预解析 38](#_Toc3919)

[第十三节 对象 39](#_Toc6273)

[1.为什么需要对象 39](#_Toc17456)

[2.创建对象的三种方式 39](#_Toc3519)

[3.利用字面量创建对象 39](#_Toc9192)

[4.变量、属性、函数、方法总结 40](#_Toc16097)

[第十四节 内置对象 44](#_Toc29648)

[1. 查文档 44](#_Toc10466)

[2.Math（数学）对象 44](#_Toc7611)

[3.日期对象 45](#_Toc5107)

[4.数组对象 48](#_Toc16437)

[5.字符串对象 53](#_Toc10014)

[第十五节 简单数据类型和复杂数据类型 57](#_Toc9735)

[1. 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。 57](#_Toc27210)

[2.堆栈空间分配区别: 58](#_Toc1721)

[3.简单类型的内存分配 58](#_Toc12598)

[4.复杂类型的内存分配 58](#_Toc16407)

[5.简单类型传参 59](#_Toc32602)

[6.复杂类型传参 59](#_Toc30994)

**第一节：初识JavaScript**

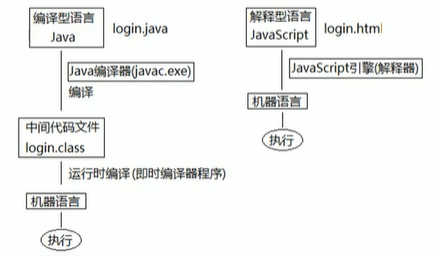
**1.浏览器执行JS简介**

浏览器分成两部分:渲染引擎和JS引擎

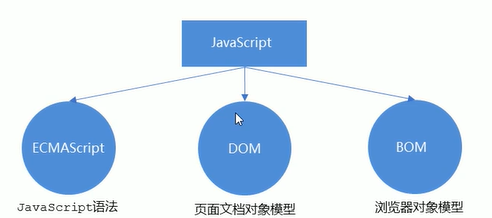
·渲染引擎︰用来解析HTML与CSS，俗称内核，比如chrome浏览器的blink，老版本的webkit

·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。



**2. JS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。

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。

**3.JS书写位置**

JS有3种书写位置，分别为行内、内嵌和外部。

**3.1行内式JS**

N7R7_%7IGLNE(H)QHATNYZU

·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

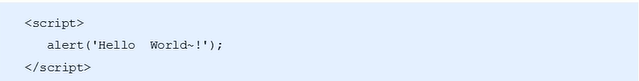
·注意单双引号的使用:在HTML中我们推荐使用双引号,JS中我们推荐使用单引号

·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

**3.2内嵌式**



·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

**3.3外部JS文件**

L6A~7HSQL2TM_[9_[G8@)85

·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S文件的script标签中间不可以写代码

·适合于S代码量比较大的情况

**4.注释**

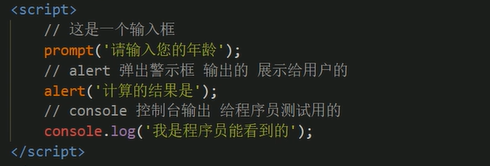
单行注释：// ctrl + /

多行注释：/\* \*/ shift +alt+a

**5.JS输入输出语句**

为了方便信息的输入输出，JS中提供了一些输入输出语句，其常用的语句如下:





**第二节：变量**

**1.变量**

变量是用于存放数据的容器。我们通过变量名获取数据，甚至数据可以修改。

**2.变量在内存中的存储**

本质:变量是程序在内存中申请的一块用来存放数据的空间。

类似我们酒店的房间,一个房间就可以看做是一个变量。

**3.变量的使用**

变量在使用时分为两步:1.声明变量 2.赋值

**3.1声明变量**

RRI{@%A~J3$N@HBHK{QE[[8

·var是一个JS关键字，用来声明变量( variable变量的意思)。使用该关键字声明变量后，计算机会自动为变量分配内存空间，不需要程序员管

·age是程序员定义的变量名，我们要通过变量名来访问内存中分配的空间

**3.2赋值**

NJ(Q(Z$P7[PGVVVE5][6L3H

·=用来把右边的值赋给左边的变量空间中此处代表赋值的意思

·变量值是程序员保存到变量空间里的值

**3.3变量的初始化**

PHR%(~X4ZE@HYM)AM6I4~VE

声明一个变量并赋值,我们称之为变量的初始化。

**4.变量语法扩展**

**4.1更新变量**

一个变量被重新复赋值后，它原有的值就会被覆盖，变量值将以最后一次赋的值为准。

@QKXUR[CG`X6SI(SEQUZ]{A

**4.2同时声明多个变量**

同时声明多个变量时，只需要写一个var，多个变量名之间使用英文逗号隔开。

@@ZWM$(G%S(14@8A8AGF43B

**4.3声明变量特殊情况**



**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

**第三节 数据类型**

**1.为什么需要数据类型**

在计算机中，不同的数据所需占用的存储空间是不同的，为了便于把数据分成所需内存大小不同的数据，充分利用存储空间，于是定义了不同的数据类型。简单来说，数据类型就是数据的类别型号。比如姓名“张三”，年龄18、这些数据的类型是不一样的。

**2.变量的数据类型**

变量是用来存储值的所在处，它们有名字和数据类型。变量的数据类型决定了如何将代表这些值的位存储到计算机的内存中。JavaScript是一种弱类型或者说动态语言。这意味着不用提前声明变量的类型，在程序运行过程中，类型会被自动确定。JS的变量数据类型是只有程序在运行过程中，根据等号右边的值来确定的。

$%HPK_70Q`1Z@7BA@XO9KCS

在代码运行时，变量的数据类型是由JS引擎根据=右边变量值的数据类型来判断的，运行完毕之后，变量就确定了数据类型。

JavaScript拥有动态类型，同时也意味着相同的变量可用作不同的类型:

RT~`LZ8CYL(KGFDP7U0SF$Y

JS是动态语言，变量的数据类型是可以变化的。

**3.数据类型的分类**

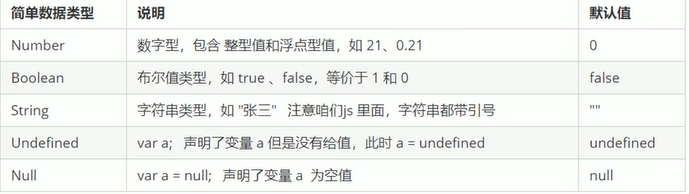
JS把数据类型分为两类:

·简单数据类型( Number, string, Boolean , undefined,Nul1 )

·复杂数据类型( object)

**3.1简单数据类型(基本数据类型)**

JavaScript中的简单数据类型及其说明如下:



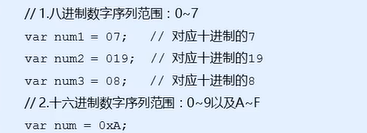
**①数字型Number**

JavaScript数字类型既可以用来保存整数值，也可以保存小数(浮点数)。

GX~2YMVF9{`CIT9}Y9$0(QE

（1）.数字型进制：

最常见的进制有二进制、八进制、十进制、十六进制。

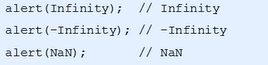


现阶段我们只需要记住,在JS中八进制前面加O，十六进制前面加Ox

（2）.数字型范围：JavaScript中数值的最大和最小值

0PF}[WOS`X5HDRIWIQ1H3KA

（3）.数字型三个特殊值：



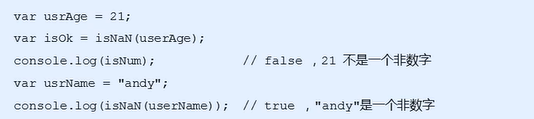
Infinity，代表无穷大，大于任何数值

-Infinity ,代表无穷小，小于任何数值

NaN，Not a number，代表一个非数值

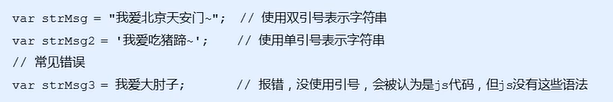
（4）.isNaN（）

用来判断一个变量是否为非数字的类型,返回true或者false



**②字符串型String**

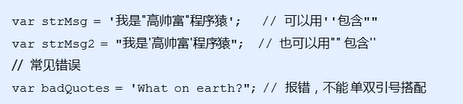
字符串型可以是引号中的任意文本,其语法为双引号""和单引号’’



因为HTML标签里面的属性使用的是双引号，JS这里我们更推荐使用单引号。

（1）.字符串引号嵌套：

JS可以用单引号嵌套双引号，或者用双引号嵌套单引号(外双内单，外单内双)



（2）.字符串转义符：

类似HTML里面的特殊字符，字符串中也有特殊字符，我们称之为转义符。

转义符都是\开头的，常用的转义符及其说明如下:

![(W]M(FHJEQ@5Z6O%6](D]Z6](data:image/png;base64,)

（3）.字符串长度：字符串是由若干字符组成的，这些字符的数量就是字符串的长度。通过字符串的length属性可以获取整个字符串的长度。

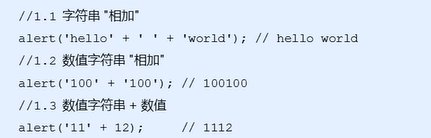
QW{UWR{@LL`}_I4MINGOS92

（4）.字符串拼接：

·多个字符串之间可以使用＋号进行拼接，其拼接方式为字符串＋任何类型=拼接之后的新字符串。

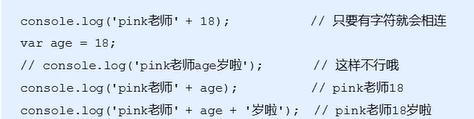
·拼接前会把与字符串相加的任何类型转成字符串，再拼接成一个新的字符串。

·变量不要写到字符串里面，是通过和字符串相连的方式实现的。



+号总结口诀：数值相加，字符相连

（5）.字符串拼接加强：



·我们经常会将字符串和变量来拼接，因为变量可以很方便地修改里面的值

·变量是不能添加引号的，因为加号的变量会变成字符串

·如果变量两侧都有字符串拼接，口诀“引引加加”，删掉数字，变量写加中间

**③布尔型Boolean**

布尔类型有两个值:true和false ，其中true表示真(对)，而false表示假(错)。

布尔型和数字型相加的时候，true的值为1，false的值为0。

**④Undefined和Null**

一个声明后没有被赋值的变量会有一个默认值undefined(如果进行相连或者相加时，注意结果)。一个声明变量给null值，里面存的值为空。

**第四节 获取变量数据类型**

**1.获取检测变量的数据类型**

typeof可用来获取检测变量的数据类型

**2.字面量**

字面量是在源代码中一个固定值的表示法，通俗来说，就是字面量表示如何表达这个值。

·数字字面量:8,9, 10

·字符串字面量:‘黑马程序员’,"大前端”

·布尔字面量:true , false

**第五节 数据类型转换**

**1.什么是数据类型转换**

使用表单、prompt获取过来的数据默认是字符串类型的，此时就不能直接简单的进行加法运算，而需要转换变量的数据类型。通俗来说，就是把一种数据类型的变量转换成另外一种数据类型。

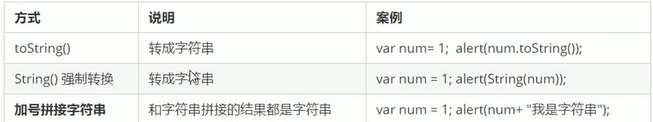
我们通常会实现3种方式的转换:

·转换为字符串类型

·转换为数字型

·转换为布尔型

**2.转换为字符串**



·toString0和String0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。

**3.转换为数字型**



·注意parseInt和parseFloat单词的大小写，I和F要大写。

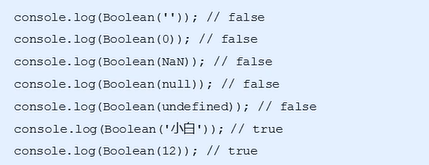
·隐式转换是我们在进行算数运算的时候，JS自动转换了数据类型。

**4.转换为布尔型**

ZA6]}EBQRV0M(JRA6US3L1K

·代表空、否定的值会被转换为false ，如""、0、NaN、 null、 undefined

·其余值都会被转换为true



**5.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。

**③保留字**

保留字∶实际上就是预留的“关键字”，意思是现在虽然还不是关键字，但是未来可能会成为关键字，同样不能使用它们当变量名或方法名。

包括:boolean、byte、char、class、const、debugger、double、enum、export、extends,fimal、float、goto、implements、import、int、interface、long、mative、package、private、protected、public、 short、statli、 super、synchronized.throws、transient,volatile等。

**第六节：运算符**

**1.运算符**

运算符( operator ) 也被称为操作符，是用于实现赋值、比较和执行算数运算等功能的符号。

JavaScript中常用的运算符有:

·算数运算符

·递增和递减运算符

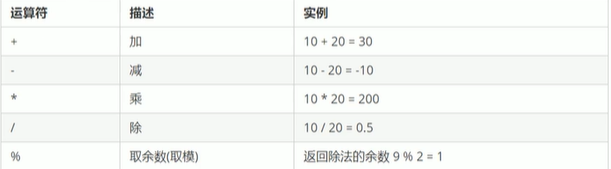
·比较运算符

·逻辑运算符

·赋值运算符

**2.算数运算符**

**①概念︰**算术运算使用的符号，用于执行两个变量或值的算术运算。



**②浮点数的精度问题**：

浮点数值的最高精度是17位小数，但在进行算术计算时其精确度远远不如整数。

WN[M%%`ZX~2`R{C0GBZJA3U

所以:不要直接判断两个浮点数是否相等!

**③我们怎么判断一个数能够被整除呢?**

它的余数是0就说明这个数能被整除，这就是%取余运算符的主要用途

**④请问1＋2\*3结果是?**

结果是7，注意算术运算符优先级的，**先乘除，后加减，有小括号先算小括号**里面的

**⑤表达式和返回值：**

表达式∶是由数字、运算符、变量等以能求得数值的有意义排列方法所得的组合。简单理解∶是由数字、运算符、变量等组成的式子。

表达式最终都会有一个结果，返回给我们，我们成为返回值

**3.递增运算符**

如果需要反复给数字变量添加或减去1，可以使用递增（++)和递减(-- )运算符来完成。

在JavaScript中，递增（++）和递减(-- )既可以放在变量前面，也可以放在变量后面。放在变量前面时我们可以称为前置递增（递减）运算符，放在变量后面时，我们可以称为后置递增（递减）运算符。

注意：递增和递减运算符必须和变量配合使用。

**②前置递增运算符：**++num前置递增，就是自加1，类似于num = num +1，但是++num 写起来更简单。先自加1，后返回值。

**③后置递增运算符**：num++后置递增，就是自加1，类似于num = num + 1，但是 num++写起来更简单。先返回原值，后自加1.

**④前置递增和后置递增小结**：

·前置递增和后置递增运算符可以简化代码的编写，让变量的值+1比以前写法更简单

·单独使用时，运行结果相同

·与其他代码联用时，执行结果会不同

·后置:先原值运算，后自加(先人后己)

·前置:先自加，后运算(先已后人)

·开发时，大多使用后置递增/减，并且代码独占一行，例如:num++;或者num--;

**4.比较运算符**

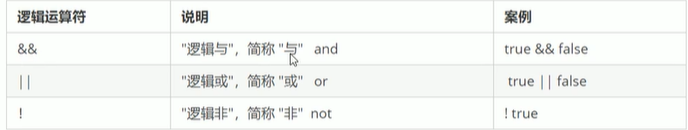
**①概念∶**比较运算符（关系运算符）是两个数据进行比较时所使用的运算符，比较运算后，会返回一个布尔值( true / false )作为比较运算的结果。





**5.逻辑运算符**

概念∶逻辑运算符是用来进行布尔值运算的运算符，其返回值也是布尔值。后面开发中经常用于多个条件的判断



逻辑与&& and两侧都为真，结果才是真，只要有一侧为假，结果就为假。

逻辑或|| or两侧都为假，结果才是假，只要有一侧为真，结果就是真

逻辑非(!)也叫作取反符，用来取一个布尔值相反的值，如true的相反值是false

**6.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

**②逻辑或**

·语法∶表达式1||表达式2

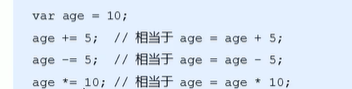
·如果第一个表达式的值为真，则返回表达式1

·如果第一个表达式的值为假，则返回表达式2

**7.赋值运算符**

概念︰用来把数据赋值给变量的运算符。





1. **运算符优先级**

****

·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

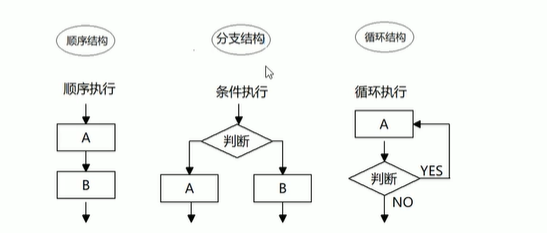
**第七节 流程控制**

**1.流程控制**

**①**在一个程序执行的过程中，各条代码的执行顺序对程序的结果是有直接影响的。很多时候我们要通过控制代码的执行顺序来实现我们要完成的功能。

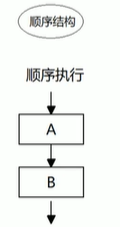
**②简单理解**︰流程控制就是来控制我们的代码按照什么结构顺字来执行

**③**流程控制主要有三种结构，分别是顺序结构、分支结构和循环结构，这三种结构代表三种代码执行的顺序。



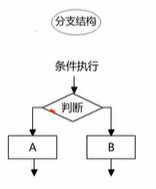
1. **顺序流程控制**

顺序结构是程序中最简单、最基本的流程控制，它没有特定的语法结构，程序会按照代码的先后顺序，依次执行，程序中大多数的代码都是这样执行的。



1. **分支流程控制if语句**

由上到下执行代码的过程中，根据不同的条件执行不同的路径代码(执行代码多选一的过程)，从而得到不同的结果



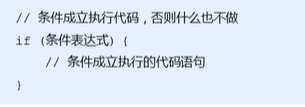
JS语言提供了两种分支结构语句：

·if语句

·switch语句

**①if语句**

·语法结构：



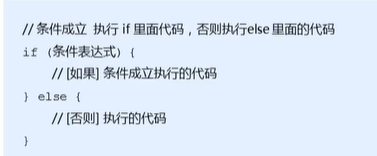
·语句可以理解为一个行为，循环语句和分支语句就是典型的语句。一个程序由很多个语句组成，一般情况下，会分割成一个一个的语句。

·执行流程：

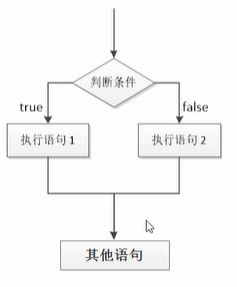


**②if ...else语句（双分支语句）**

·语法结构：



·执行流程：

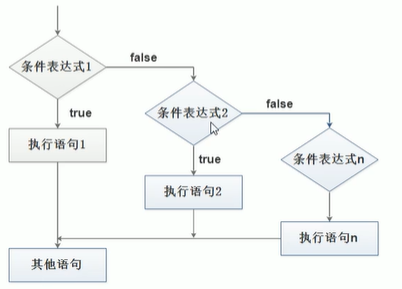


**③if ... else if语句（多分支语句）**

·语法结构：



·执行流程：



1. **三元表达式**

·三元表达式也能做一些简单的条件选择。有三元运算符组成的式子称为三元表达式。

·语法结构：

条件表达式？表达式1：表达式2；

·执行思路：

如果条件表达式结果为真，则返回表达式1的值；如果条件表达式结果为假，则返回表达式2的值。

·代码体验：

var num = 10;

var result = num >5? ‘是的’：‘不是的’；

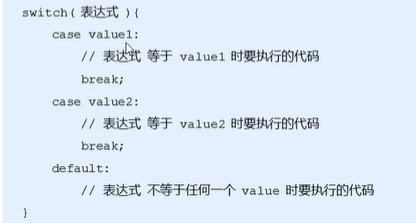
alert(result);

1. **分支流程控制switch语句**

·语法结构：

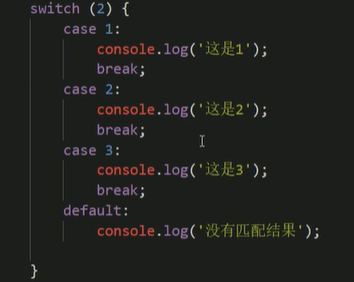
switch语句也是多分支语句，它用于基于不同的条件来执行不同的代码。当要针对变量设置一系列的具体特定值的选项时，就可以使用switch。

·语法结构：



·执行思路﹑利用我们的表达式的值和 case后面的选项值相匹配如果匹配上，就执行该case里面的语句如果都没有匹配上，那么执行default里面的语句。

·代码验证：



·注意事项：

①在开发里面，表达式经常写成变量。

②表达式的值和case里面的值相匹配的时候是全等，必须是值和数据类型一致才可以。

③如果当前的case里面没有break，则不会退出switch，而是继续执行下一个case。

1. **switch语句和if...else if语句的区别**

①一般情况下，它们两个语句可以相互替换。

②switch...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，常用于范围判断(大于、等于某个范围)。

③switch语句进行条件判断后直接执行到程序的条件语句，效率更高。而if..else语句有几种条件，就得判断多少次。

④当分支比较少时，if...else语句的执行效率比switch语句高。

⑤当分支比较多时，switch语句的执行效率比较高，而且结构更清晰。

**第八节 循环**

**1.循环**

**·**循环目的：在实际问题中，有许多具有规律性的重复操作，因此在程序中要完成这类操作就需要重复执行某些语句。

1. **JS中的循环**

在JS中，主要有三种类型的循环语句：

**·**for循环

**·**while循环

**·**do...while循环

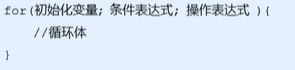
1. **for循环**

**求和公式：sum+=i;**

在程序中，一组被重复执行的语句被称之为循环体，能否继续重复执行，取决于循环的终止条件。由循环体及循环的终止条件组成的语句，被称之为循环语句。

**·语法结构**

for循环主要用于把某些代码循环若干次，通常跟计数有关系。其语法结构如下∶



**·断点调试：**

断点调试是指自己在程序的某一行设置一个断点，调试时，程序运行到这一行就会停住，然后你可以一步一步往下调试，调试过程中可以看各个变量当前的值，出错的话，调试到出错的代码行即显示错误，停下。

**断点调试可以帮我们观察程序的运行过程。**

浏览器中按F12--> sources -->找到需要调试的文件-->在程序的某一行设置断点。

Watch:监视，通过watch可以监视变量的值的变化，非常的常用。

F11:程序单步执行，让程序一行一行的执行，这个时候，观察watch中变量的值的变化。

代码调试的能力非常重要，只有学会了代码调试，才能学会自己解决bug的能力。

1. **双重for循环**

**·**很多情况下，单层for循环并不能满足我们的需求，比如我们要打印一个5行5列的图形、打印一个倒直角三角形等，此时就可以通过循环嵌套来实现。

**·**循环嵌套是指在一个循环语句中再定义一个循环语句的语法结构，例在for循环语句中，可以再嵌套一个for循环，这样的for循环语句我们称之为双重for循环。

**·外层循环循环一次，里面的循环执行全部。**

1. **for循环小结**

·for 循环可以重复执行某些相同代码

·for循环可以重复执行些许不同的代码，因为我们有计数器

·for循环可以重复执行某些操作，比比如算术运算符加法操作

·随着需求增加，双重for循环可以做更多、更好看的效果

·双重for循环，外层循环一次，内层 for循环全部执行

·for循环是循环条件和数字直接相关的循环

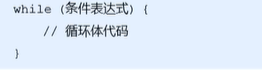
·分析要比写代码更重要

·一些核心算法想不到，但是要学会，分析它执行过程

·举一反三，自己经常总结，做一些相似的案例

1. **while循环**

·while语句可以在条件表达式为真的前提下，循环执行指定的一段代码，直到表达式不为真时结束循环.while语句的语法结构如下:



·执行思路:

①先执行条件表达式，如果结果为true，则执行循环体代码;如果为false，则退出循环，执行后面代码。

②执行循环体代码。

③循环体代码执行完毕后，程序会继续判断执行条件表达式，如条件仍为true，则会继续执行循环体，直到循环条件为false 时，整个循环过程才会结束。

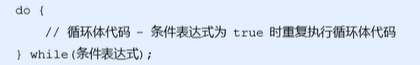
④里面应该也有计数器，初始化变量。

⑤里面应该也有操作表达式，完成计数器的更新，防止死循环。

1. **do while 循环**

·do..while语句其实是while语句的一个变体。该循环会先执行一次代码块，然后对条件表达式进行判断，如果条件为真，就会重复执行循环体，否则退出循环。

·语法结构：



·执行思路：

①先执行一次循环体代码

②再执行条件表达式，如果结果为true，则继续执行循环体代码，如果为false，则退出循环，继续执行后面代码

·注意∶先再执行循环体，再判断，我们会发现do...while循环语句至少会执行一次循环体代码

1. **循环小结**

·JS中循环有for . while、do while。

·三个循环很多情况下都可以相互替代使用。

·如果是用来计次数，跟数字相关的，三者使用基本相同，但是我们更喜欢用for。

·while和do...while可以做更复杂的判断条件，比for循环灵活一些。

·while和do..while执行顺序不一样，while先判断后执行，do...while先执行一次，再判断执行。

·while和do..while执行次数不一样,do...while至少会执行一次循环体，而while可能一次也不执行。

·实际工作中，我们更常用for循环语句，它写法更简洁直观，所以这个要重点学习。

1. **continue关键字**

·continue关键字用于立即跳出本次循环，继续下一次循环（本次循环体中continue之后的代码就会少执行一次)。

**10.break关键字**

break关键字用于立即跳出整个循环（循环结束)。

**11.JS命名规范以及语法格式**

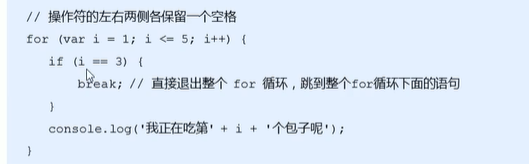
①标识符命名规范：

·变量、函数的命名必须要有意义

·变量的名称一般用名词

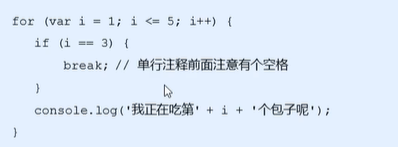
·函数的名称一般用动词

②操作符规范



操作符的左右要空格

③单行注释规范



注释符号后面空一格写注释

④快捷键调整格式规范：shift+alt+f

1. **数组**

·数组是指一组数据的集合，其中的每个数据被称作元素，在数组中可以存放任意类型的元素。数组是一种将一组数据存储在单个变量名下的优雅方式。

1. **数组的创建方式**

JS中创建数组有两种方式:

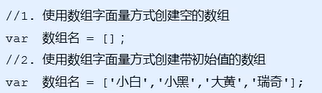
·利用new创建数组

·利用数组字面量创建数组

1. **利用new创建数组**

TOG2RIIRHUDJ81M@IG2ETB0

**3.利用数组字面量创建数组**



·数组的字面量是方括号[]

·声明数组并赋值称为数组的初始化

·这种字面量方式也是我们以后最多使用的方式

**4.数组元素的类型**

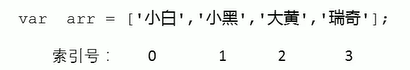
数组中可以存放任意类型的数据，例做字符串,数字，布尔值等。

~)24$_A5A)N0K5NG56@PSCG

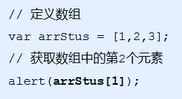
**5.获取数组中的元素**

①数组的索引

索引(下标)︰用来访问数组元素的序号（数组下标从0开始)。



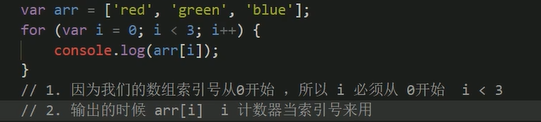
数组可以通过索引来访问、设置、修改对应的数组元素，我们可以通过“数组名[索引]”的形式来获取数组中的元素。这里的访问就是获取得到的意思



**6.遍历数组**

·遍历:就是把数组中的每个元素从头到尾都访问一次。

·从代码中我们可以发现，从数组中取出每一个元素时，代码是重复的，有所不一样的是索引值在递增。可以通过循环的方式把数组里面的所有元素取出来。



**i是计数器，当索引号使用，从0开始。**

**7.数组长度**

使用“数组名.length”可以访问数组元素的数量(数组长度)。

·数组的长度是元素个数，不要和索引号混淆。

·数组名.length是动态监测元素的个数。

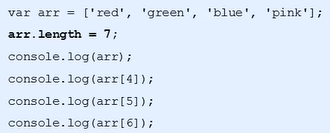
**8.数组中新增元素**

可以通过修改length长度以及索引号增加数组元素

①通过修改length长度新增数组元素

·可以通过修改length长度来实现数组扩容的目的

·length属性是可读写的

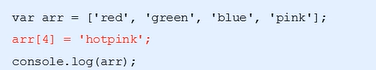


其中索引号是4，5，6的空间没有给值，就是声明变量未给值，默认值就是undefined。

②通过修改数组索引新增数组元素

·可以通过修改数组索引的方式追加数组元素

·不能直接给数组名赋值,否则会覆盖掉以前的数据



这种方式也是我们最常用的一种方式。

**9.冒泡排序**

·冒泡排序:是一种算法，把一系列的数据按照一定的顺序进行排列显示(从小到大或从大到小)。

·冒泡排序是一种简单的排序算法，它重复地走访过要排序的数列，一次比较两个元素，如果他们的顺序错误就把他们交换过来。走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。这个算法的名字由来是因为越小的元素会经由交换慢慢“浮”到数列的顶端。



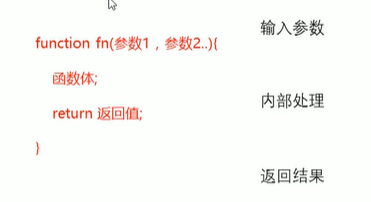
**第十节 函数**

**1.函数的概念**

·在JS里面，可能会定义非常多的相同代码或者功能相似的代码，这些代码可能需要大量重复使用。

·虽然 for循环语句也能实现一些简单的重复操作，但是比较具有局限性，此时我们就可以使用JS中的函数。

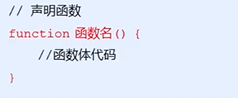
·函数∶就是封装了一段可被重复调用执行的代码块。通过此代码块可以实现大量代码的重复使用。



**2.函数的使用**

·函数使用分为两步:声明函数和调用函数

①声明函数



·function是声明函数的关键字,必须小写

·由于函数一般是为了实现某个功能才定义的，所以通常我们将函数名命名为动词，比如getSum

②调用函数

(V%QZYMA`J20(WF{~Y1XBPL

·调用的时候千万不要忘记添加小括号

·口诀:函数不调用，自己不执行。分

·注意:声明函数本身并不会执行代码，只有调用函数时才会执行函数体代码。

**3.函数的封装**

·函数的封装是把一个或者多个功能通过函数的方式封装起来，对外只提供一个简单的函数接口。

·简单理解∶封装类似于将电脑配件整合组装到机箱中(类似快递打包)

**4.函数的参数**

**①形参和实参**

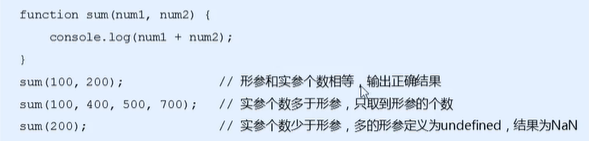
在声明函数时，可以在函数名称后面的小括号中添加一些参数，这些参数被称为形参，而在调用该函数时，同样也需要传递相应的参数，这些参数被称为实参。



参数的作用:在函数内部某些值不能固定，我们可以通过参数在调用函数时传递不同的值进去。

②函数形参和实参个数不匹配问题





·注意:在JavaScript中，形参的默认值是undefined。

**5.函数的返回值**

①return语句：

有的时候，我们会希望函数将值返回给调用者，此时通过使用return语句就可以实现。

·语法格式：function 函数名（）{

Return 需要返回的结果；

}

函数名（）；

·我们函数只是实现某种功能，最终的结果需要返回给函数的调用者，函数名(）通过return实现的

·只要函数遇到return，就把后面的结果返回给函数的调用者，函数名() = return后面的结果

②return终止函数

return语句之后的代码不被执行。

③return的返回值

return只能返回一个值。如果用逗号隔开多个值，以最后一个为准。

④函数没有return返回undefined

函数都是有返回值的

1. 如果有return则返回return后面的值

2．如果没有return则返回undefined

⑤break，continue，return的区别

·break :结束当前的循环体（如for、while )。

·continue :跳出本次循环，继续执行下次循环(如for、while )。

·return :不仅可以退出循环，还能够返回return语句中的值，同时还可以结束当前的函数体内的代码。

**6.arguments的使用**

·当我们不确定有多少个参数传递的时候，可以用arguments来获取。在JavaScript中,arguments实际上它是当前函数的一个内置对象。所有函数都内置了一个arguments对象，arguments对象中存储了传递的所有实参。

·arguments展示形式是一个伪数组，因此可以进行遍历。伪数组具有以下点：

·具有length属性

·按索引方式储存数据

·不具有数组的push , pop等方法

**7.函数的两种声明方式**

①利用函数关键字自定义函数（命名函数）

②函数表达式(匿名函数)

·语法格式：var 变量名 = function（）{};

·注意：

①变量名不是函数名。

②函数表达式声明方式跟声明变量差不多，只不过变量里面存的是值，而函数表达式里面存的是函数。

③函数表达式也可以进行传递参数。

**8.函数可以调用另外一个函数**

·因为每个函数都是独立的代码块，用于完成特殊任务，因此经常会用到函数相互调用的情况。

**第十一节 作用域**

**1.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。

·JavaScript作用域:就是代码名字（变量)在某个范围内起作用和效果目的是为了提高程序的可靠性更重要的是减少命名冲突。

·JS的作用域：全局作用域、局部作用域

①全局作用域：整个script标签，或者是一个单独的JS文件。

②局部作用域：在函数内部就是局部作用域，这个代码的名字只在函数内部起效果和作用。

**2.变量作用域的分类**

·在JavaScript中，根据作用域的不同，变量可以分为两种:全局变量、局部变量。

①全局变量：在全局作用域下的变量，在全局下都可以使用。

·在全局作用域下声明的变量叫做全局变量（在函数外部定义的变量)。

·全局变量在代码的任何位置都可以使用。

·在全局作用域下var 声明的变量是全局变量。

·特殊情况下，在函数内不使用var声明的变量也是全局变量(不建议使用)。

②局部变量：在局部作用域下的变量，在函数内部的变量，只能在函数内部使用。

·在局部作用域下声明的变量叫做局部变量(在函数内部定义的变量)。

·局部变量只能在该函数内部使用。

·在函数内部var 声明的变量是局部变量。

·函数的形参实际上就是局部变量。

③全局变量和局部变量的区别

·全局变量︰在任何一个地方都可以使用，只有在浏览器关闭时才会被销毁，因此比较占内存空间。

·局部变量∶**只在函数内部使用**，当其所在的代码块被执行时，会被初始化﹔当代码块运行结束后，就会被销毁，因此更节省内存空间。

**3.作用域链**

·只要是代码，就至少有一个作用域。

·写在函数内部的局部作用域。

·如果函数中还有函数，那么在这个作用域中就又可以诞生一个作用域。

·根据在内部函数可以访问外部函数变量的这种机制，用链式查找决定哪些数据能被内部函数访问，就称作作用域链。

·作用域链:内部函数访问外部函数的变量，采取的是链式查找的方式来决定取哪个值，这种结构我们称为作用域链。

·站在目标出发，就近原则，一层一层往外查找。

**第十二节 预解析**

· JavaScript代码是由浏览器中的JavaScript解析器来执行的。JavaScript解析器在运行JavaScript代码的时候分为两步∶预解析和代码执行。

①预解析js引擎会把js里面所有的 var 还有 function提升到当前作用域的最前面。

②代码执行按照代码书写的顺序从上往下执行。

·预解析分为变量预解析（变量提升）和函数预解析（函数提升)

①变量提升就是把所有的变量声明提升到当前的作用域最前面，不提升赋值操作。

②函数提升就是把所有的函数声明提升到当前作用域的最前面，不调用函数。

**第十三节 对象**

·在JavaScript中，对象是一组无序的相关属性和方法的集合，所有的事物都是对象，例如字符串、数值、数组、函数等。

·对象是由属性和方法组成的。

·属性:事物的特征，在对象中用属性来表示(常用名词)

·方法:事物的行为，在对象中用方法来表示(常用动词)

**1.为什么需要对象**

JS中的对象表达结构更清晰、更强大。

**2.创建对象的三种方式**

在JavaScript中，现阶段我们可以采用三种方式创建对象( object) :

·利用字面量创建对象

·利用new Object创建对象

·利用构造函数创建对象

**3.利用字面量创建对象**

·对象字面量:就是花括号{}里面包含了表达这个具体事物（对象）的属性和方法。

·{}里面采取键值对的形式表示

·键︰相当于属性名

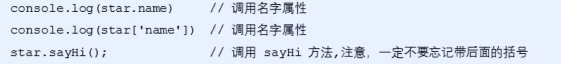
·值︰相当于属性值，可以是任意类型的值（数字类型、字符串类型、布尔类型，函数类型等)

·对象的调用

·对象里面的属性调用:对象.属性名，这个小点.就理解为“的”。

·对象里面属性的另一种调用方式:对象[‘属性名’]，注意方括号里面的属性必须加引号，我们后面会用。

·对象里面的方法调用:对象.方法名（），注意这个方法名字后面一定加括号。



**4.变量、属性、函数、方法总结**

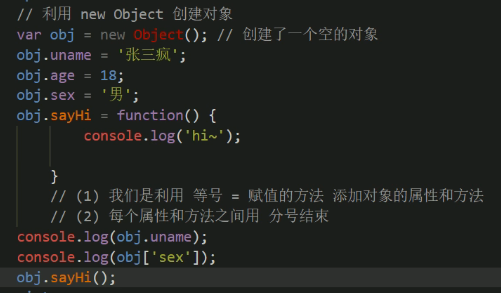
·变量:单独声明赋值，单独存在

·属性∶对象里面的变量称为属性，不需要声明，用来描述该对象的特征

·函数:单独存在的，通过“函数名（）”的方式就可以调用

·方法:对象里面的函数称为方法，方法不需要声明，使用“对象.方法名（）”的方式就可以调用，方法用来描述该对象的行为和功能。

**5.利用new Object创建对象**



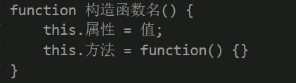
**6.利用构造函数创建对象**

·因为我们一次创建一个对象，里面很多的属性和方法是大量相同的我们只能复制，因此我们可以利用函数的方法重复这些相同的代码，我们就把这个函数称为构造函数。又因为这个函数不一样，里面封装的不是普通代码，而是对象。

·构造函数就是把我们对象里面一些相同的属性和方法抽象出来封装到函数里面。

·构造函数︰是一种特殊的函数，主要用来初始化对象，即为对象成员变量赋初始值，它总与new运算符一起使用。我们可以把对象中一些公共的属性和方法抽取出来，然后封装到这个函数里面。

·语法格式：





1．构造函数名字首字母要大写

2．我们构造函数不需要return就可以返回结果

3．我们调用构造函数必须使用new

4．我们只要new star()调用函数就创建一个对象 ldh {}

5．我们的属性和方法前面必须添加 this

**7.构造函数和对象**

·构造函数，如Stars()，抽象了对象的公共部分，封装到了函数里面，它泛指某一大类( class )。

·创建对象，如new Stars()，特指某一个具体的事物，通过new关键字创建对象的过程我们也称为对象实例化。

**8.new关键字**

new在执行时会做四件事情:

1. 在内存中创建一个新的空对象。

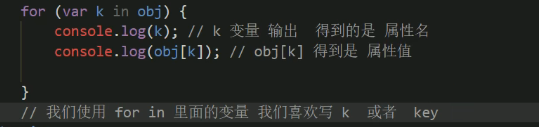
2.让 this指向这个新的对象。

3.执行构造函数里面的代码，给这个新对象添加属性和方法。

4.返回这个新对象（所以构造函数里面不需要return )。

**9.遍历对象**

for...in语句用于对数组或者对象的属性进行循环操作。



**10.小结**

1.对象可以让代码结构更清晰。

2.对象复杂数据类型object。

3.本质:对象就是一组无序的相关属性和方法的集合。

4.构造函数泛指某一大类，比如苹果，不管是红色苹果还是绿色苹果，都统称为苹果。

5.对象实例特指一个事物，比如这个苹果、正在给你们讲课的pink老师等。

6.for...in语句用于对对象的属性进行循环操作。

**第十四节 内置对象**

·JavaScript中的对象分为3种∶自定义对象、内置对象、浏览器对象。

·前面两种对象是JS基础内容，属于ECMAScript;第三个浏览器对象属于我们

JS独有的，我们JS API讲解。

·内置对象就是指JS语言自带的一些对象，这些对象供开发者使用，并提供了一些常用的或是最基本而必要的功能(属性和方法)。

·内置对象最大的优点就是帮助我们快速开发。

·JavaScript提供了多个内置对象:Math、 Date . Array、string等。

1. **查文档**

①MDN

·学习一个内置对象的使用，只要学会其常用成员的使用即可，我们可以通过查文档学习，可以通过MDN/W3C来查询。

·Mozilla开发者网络(MDN )提供了有关开放网络技术(Open Web )的信息，包括HTML、CSS和万维网及HTML5应用的API。

·MDN:https://developer.mozilla.org/zh-CN/

②如何学习对象中的方法

1．查阅该方法的功能。

2．查看里面参数的意义和类型。

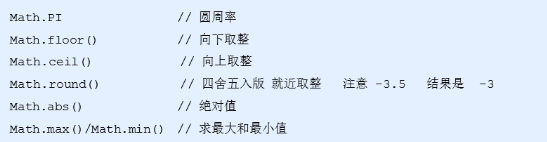
3．查看返回值的意义和类型。

4、通过demo进行测试。

**2.Math（数学）对象**

·Math数学对象不是一个构造函数﹐所以我们不需要new来调用而是直接使用里面的属性和方法即可。

·Math对象不是构造函数，它具有数学常数和函数的属性和方法。跟数学相关的运算（求绝对值，取整、最大值等)可以使用Math中的成员。



①随机数方法random( )



**3.日期对象**

·Date对象和Math对象不一样，他是一个构造函数，所以我们需要实例化后才能使用.

·Date 实例用来处理日期和时间.

·Date( )方法的使用

1.获取当前时间必须实例化

MS}1%HL$R44IM)W$L034)(Q

2. Date)构造函数的参数

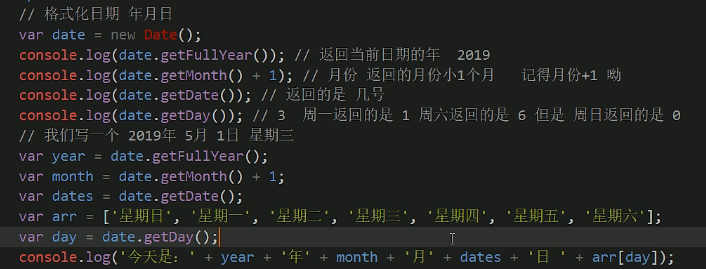
·如果括号里面有时间，就返回参数里面的时间。例如日期格式字符串为‘2019-5-1’，可以写成newDate(‘2019-5-1')或者new Date(‘2019/5/1’)

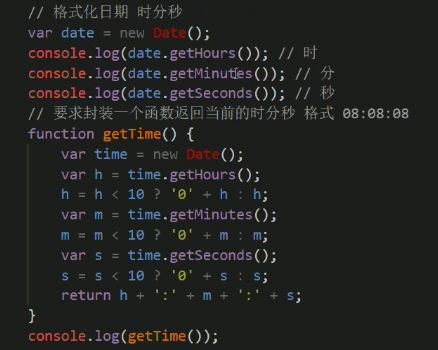
·日期格式化

我们想要2019-8-8 8:8:8格式的日期，要怎么办?

需要获取日期指定的部分，所以我们要手动的得到这种格式。





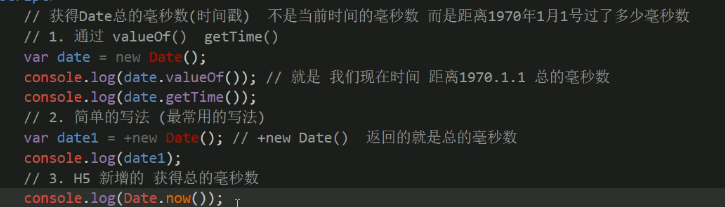


**·获取日期的总的毫秒形式（时间戳）**

Date对象是基于1970年1月1日（世界标准时间)起的亳秒数。

为什么计算机起始时间从1970年开始?

我们经常利用总的毫秒数来计算时间，因为它更精确。



1秒=1000毫秒

1.倒计时案例

①核心算法︰输入的时间减去现在的时间就是剩余的时间，即倒计时，但是不能拿着时分秒相减，比如05分减去25分，结果会是负数的。

②用时间戳来做。用户输入时间总的亳秒数减去现在时间的总的毫秒数，得到的就是剩余时间的毫秒数。

③把剩余时间总的亳秒数转换为天、时、分、秒（时间戳转换为时分秒)

转换公式如下：

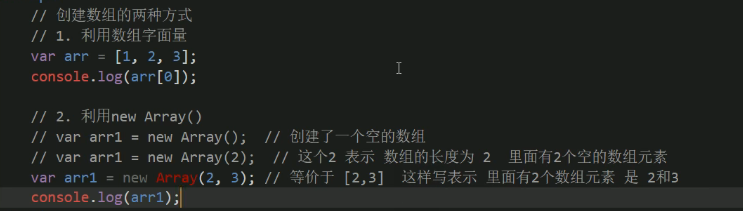
**·**d = parseInt(总秒数/60/60/24);//计算天数

**·**h = parseInt(总秒数/60/60 %24); //计算小时

**·**m = parseInt(总秒数/60%60); //计算分数

**·**s = parseInt(总秒数%60); //计算当前秒数

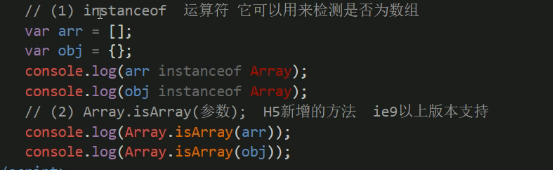
**4.数组对象**



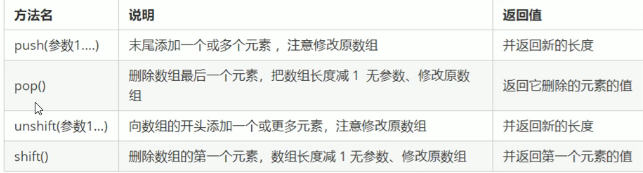
·检测是否为数组的方法

①instanceof 运算符 可以用来检测是否为数组

②Array.isArray(参数);



·添加删除数组元素的方法



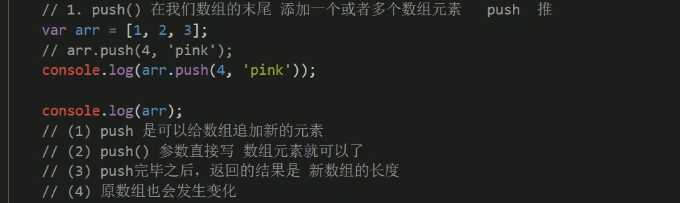
1.push(参数) 在数组的末尾添加一个或者多个数组元素

①push 是可以给数组追加新的元素

②push（ ）参数直接写数组元素就可以了

③push完毕之后,返回的结果是新数组的长度

④原数组也会发生变化



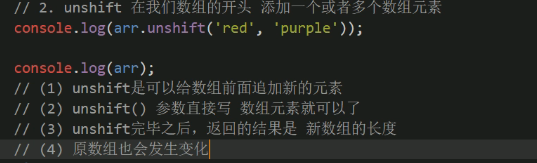
1. unshift(参数)在数组的开头添加一个或者多个数组元素

①unshift 是可以给数组前面追加新的元素

②unshift（ ）参数直接写数组元素就可以了

③unshift完毕之后,返回的结果是新数组的长度

④原数组也会发生变化



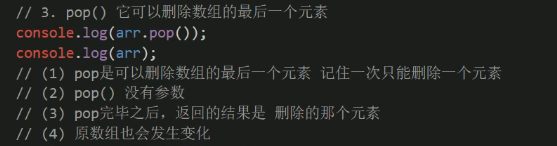
1. pop( )可以删除数组的最后一个元素

①pop 是可以删除数组的最后一个元素，记住一次只能删除一个元素

②pop（ ）括号里面没有参数

③pop完毕之后,返回的结果是删除的那个元素

④原数组也会发生变化



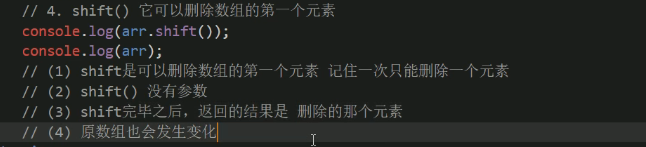
1. shift( )可以删除数组的第一个元素

①shift是可以删除数组的第一个元素，记住一次只能删除一个元素

②shift（ ）括号里面没有参数

③shift完毕之后,返回的结果是删除的那个元素

④原数组也会发生变化



·数组排序



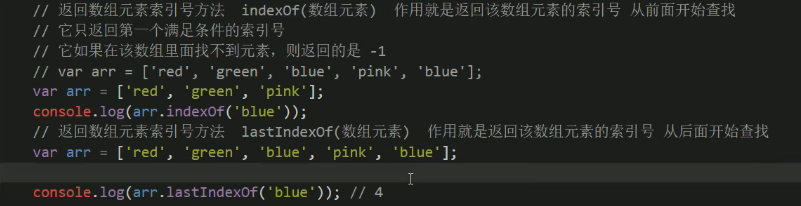


·数组索引方法



·indexOf(数组元素)，作用就是返回该数组元素的索引号从前面开始查找

·lastIndexof(数组元素)作用就是返回该数组元素的索引号从后面开始查找



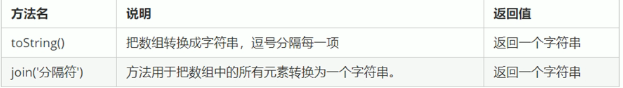
数组去重复：

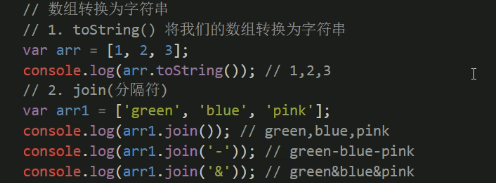
①目标∶把旧数组里面不重复的元素选取出来放到新数组中，重复的元素只保留一个，放到新数组中去重。

②核心算法∶我们遍历旧数组，然后拿着旧数组元素去查询新数组，如果该元素在新数组里面有出现过，我们就添加，否则不添加。

③我们怎么知道该元素没有存在？利用新数组.indexOf(数组元素)如果返回时-1就说明新数组里面没有改元素。

·数组转换为字符串







const array1 = ['a', 'b', 'c'];

const array2 = ['d', 'e', 'f'];

const array3 = array1.concat(array2);

console.log(array3);

// 运算后的 : array3 ["a", "b", "c", "d", "e", "f"]

var fruits = ['Banana', 'Orange', 'Lemon', 'Apple', 'Mango'];

var citrus = fruits.slice(1, 3);

//运算后的 citrus: ['Orange','Lemon']

可以取begin索引号的值，但不能取end索引号的值，左闭右开【begin，end）

var myFish = ['angel', 'clown', 'drum', 'mandarin', 'sturgeon'];

var removed = myFish.splice(3, 1);

// 运算后的 myFish: ["angel", "clown", "drum", "sturgeon"]

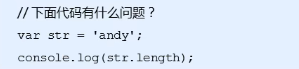
// 被删除的元素: ["mandarin"]

**5.字符串对象**

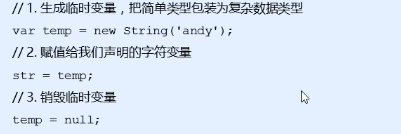
**①基本包装类型**

为了方便操作基本数据类型，JavaScript还提供了三个特殊的引用类型:String、Number和Boolean。

基本包装类型就是把简单数据类型包装成为复杂数据类型，这样基本数据类型就有了属性和方法。

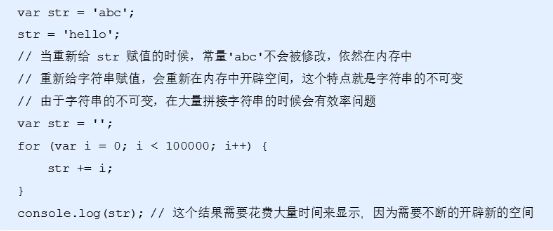


按道理基本数据类型是没有属性和方法的，而对象才有属性和方法，但上面代码却可以执行，这是因为js会把基本数据类型包装为复杂数据类型，其执行过程如下:



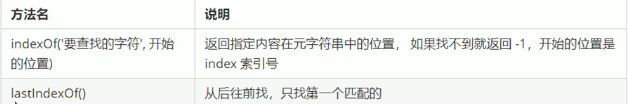
**②字符串的不可变**

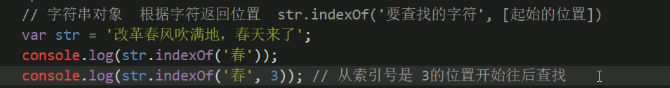
指的是里面的值不可变，虽然看上去可以改变内容，但其实是地址变了，内存中新开辟了一个内存空间。



**③根据字符返回位置**

字符串所有的方法，都不会修改字符串本身(字符串是不可变的)，操作完成会返回一个新的字符串。





①核心算法∶先查找第一个春出现的位置

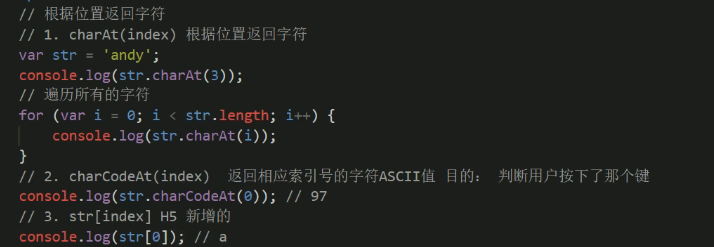
②然后只要indexOf返回的结果不是-1就继续往后查找

③因为indexOf只能查找到第一个，所以后面的查找，利用第二个参数，当前索引加1，从而继续查找

**④根据位置返回字符**







**⑤字符串操作方法**





**⑥替换字符串以及将字符串转换为数组**

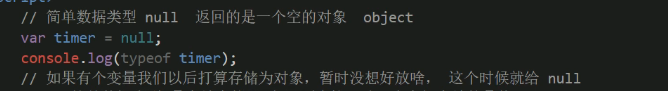


2．字符转换为数组 split('分隔符')，前面我们学过join把数组转换为字符串

1. **简单数据类型和复杂数据类型**
2. **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。**

·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string , number , boolean , undefined , null。

·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。



**2.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。复杂数据类型存放到堆里面。



·注意:JavaScript中没有堆栈的概念，通过堆栈的方式，可以让大家更容易理解代码的一些执行方式，便于将来学习其他语言。

**3.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。

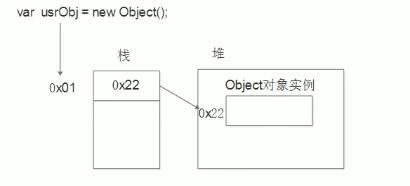


简单数据类型：是存放在栈里面，里面直接开辟一个空间存放的是值。

**4.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。

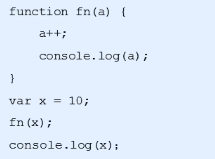
·引用类型变量（栈空间)里存放的是地址，真正的对象实例存放在堆空间中。



复杂数据类型：首先在栈里面存放地址十六进制表示，然后这个地址指向堆里面的数据。

**5.简单类型传参**

函数的形参也可以看做是一个变量，当我们把一个值类型变量作为参数传给函数的形参时，其实是把变量在栈空间里的值复制了一份给形参，那么在方法内部对形参做任何修改，都不会影响到的外部变量。



a=11; x=10;

**6.复杂类型传参**

函数的形参也可以看做是一个变量，当我们把引用类型变量传给形参时，其实是把变量在栈空间里保存的堆地址复制给了形参，形参和实参其实保存的是同一个堆地址，所以操作的是同一个对象，修改形参也会改变实参。

