JavaScript

Brendan Eich 布兰登艾奇

1995年 10天完成JavaScript的设计

最初是LiveScript 后来和Sun合作 改名成JavaScript

运行在客户端的脚本语言

不需要编译 运行过程由js解释器进行逐行解释运行

浏览器分成：渲染引擎 和 JS引擎

JS的组成

ECMAScript

DOM

BOM

书写位置

|  |
| --- |
| 行内    内嵌    引入 |

JS注释

// 单行 ctrl+/

/\* 多行 \*/ shift+alt+a

输入输出语句

弹窗 alert()

打印到控制台console.log()

弹出输入框 接收用户输入prompt()

变量

声明变量 赋值

var age = 18;

只声明 不赋值 结果是 undefined

不声明 不赋值 报错

不声明 直接赋值 可以正常使用

变量名命名规范

字母下划线美元符号开头

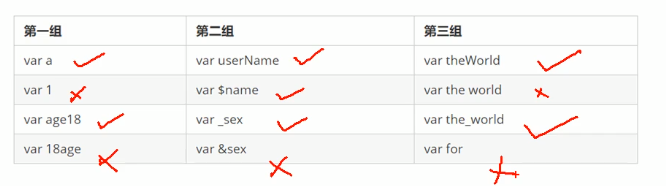
严格区分大小写

不能是关键字或者保留字

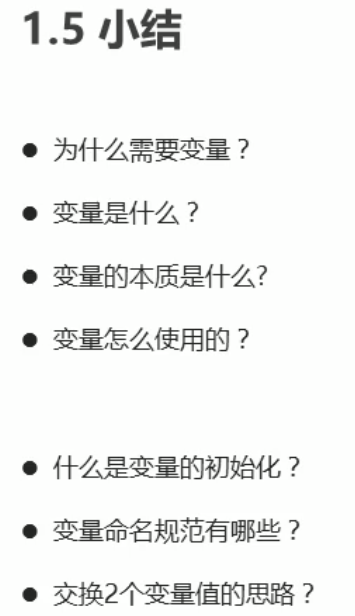
最好遵守驼峰命名法

在线翻译 爱词霸

尽量别用name



交换两个变量的值

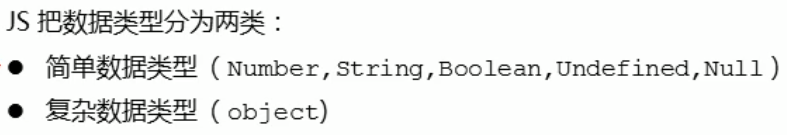


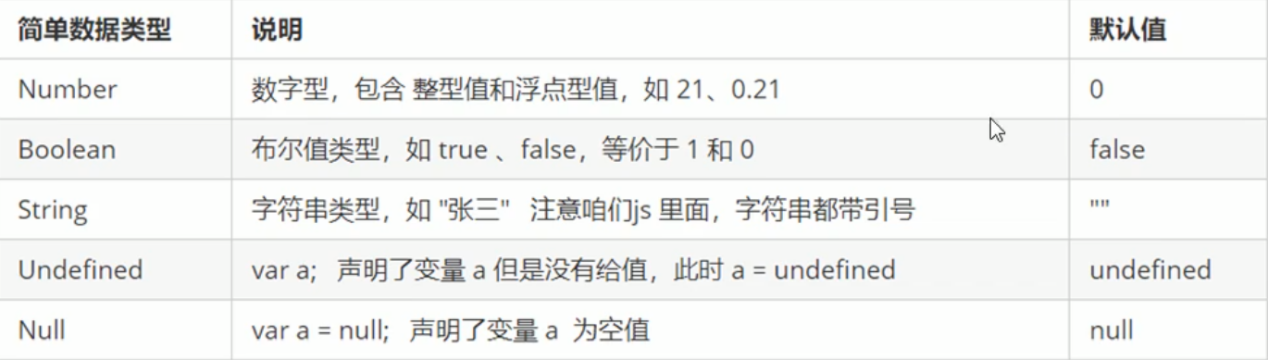
数据类型

JS是一种弱类型或者说动态语言

js的变量数据类型只有程序在运行过程中，根据等号右边的值来确定

相同的变量可以有不同的数据类型





最大值 Number.MAX\_VALUE

最小值 Number.MIN\_VALUE

无穷大 Infinity

无穷小 -Infinity

非数值 NaN

方法：isNaN() 判断是否为非数字

字符串型String

可以是单引号也可以是双引号

单双引号的嵌套（外单内双 外双内单）

转义字符



字符串长度

length属性：**str.length**

字符串拼接

字符串+任何类型=新拼接的字符串

prompt接收的是字符串类型

Boolean布尔型

true 相当于 1

false 相当于 0

undefined 声明了 未赋值

null 声明了 并赋值为null

typeof 检测数据类型

eg:console.log(typeof num)

控制台颜色判断数据类型

蓝色：数字型

黑色：字符串

灰色：undefined null

字面量

数据类型的转换

转化为字符串

toString()

String()强制转化

**加号拼接字符串** 隐式转换

**转化为数字型**

**parseInt(string)**



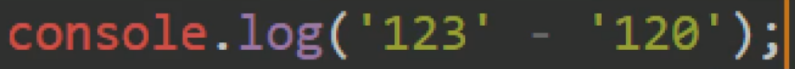


**paeseFloat(string)**

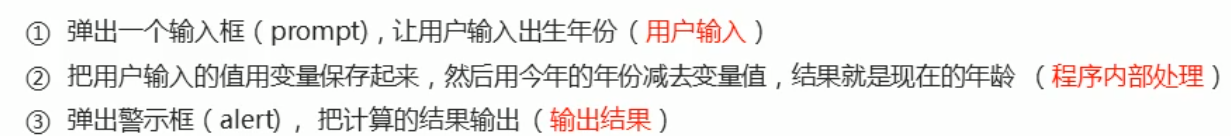
Number() 强制转换

js隐式转换 (- \* /)

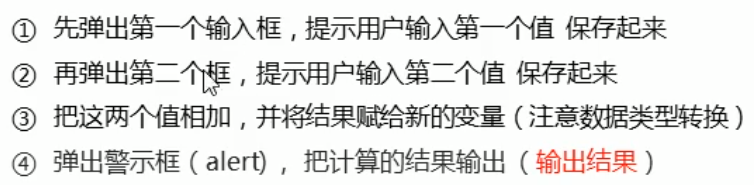




案例——计算年龄



案例——简单加法器



转化为布尔值

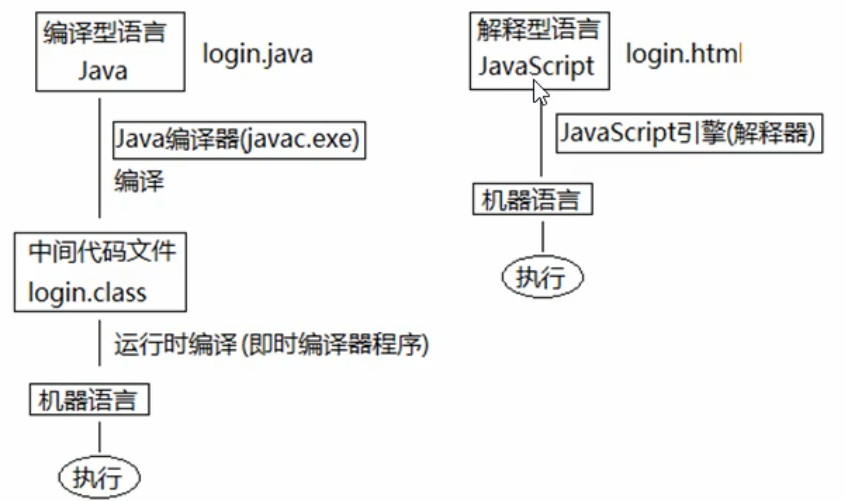
Boolean()

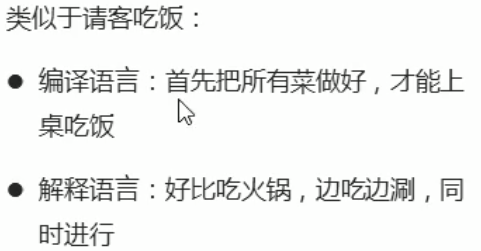
代表空、否定的值会被转换为false 如 ‘’、0、NaN、null、undefined

其他全部为true

扩展：

解释型语言和编译型语言





标识符

关键字：JS本身已经使用了的字

保留字：预留的关键字

课后作业

交换两个变量的值

获取姓名、年龄、性别，并打印用户信息

运算符

+ - \* / %

浮点数运算会有问题 不能直接判断浮点数是否相等

表达式：数字、运算符、变量组成的式子

返回值：

递增递减运算符

前置 先己后人

后置 先人后己

必须配合变量使用

比较运算符

< > <= >= == != === !==

赋值 =

判断 ==

全等 ===

逻辑运算符

与 &&

或 ||

非 !

注意：前两个有短路情况

赋值运算符

=

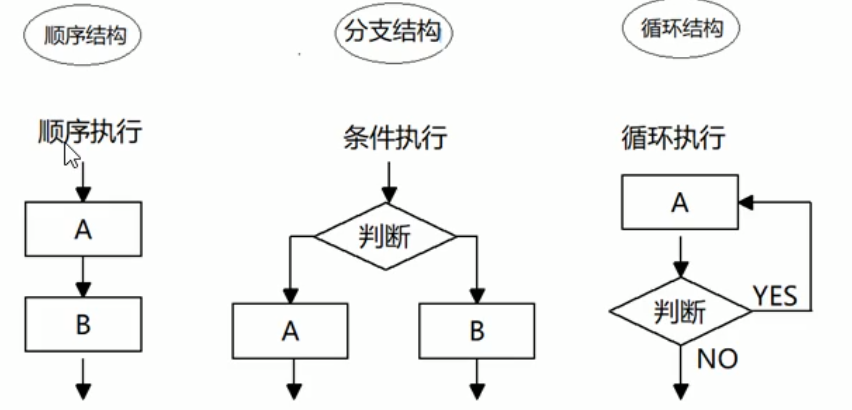
+= -= \*= /= %=

运算符优先级



与 权重高于 或

流程控制



案例——进入网吧

age>=18才可进入网吧...

案例——判断闰年

year%4==0&&year%100!=0 || year%400==0

案例——判断成绩级别

90~100\_A...

**三元表达式**

条件表达式?表达式1:表达式2;

案例——数字补0

数字小于10 补0 大于10则不需要

var result = num < 10 ? ’0’+num : num;

switch语句

注意事项 :

switch(num) num是变量

case xx: xx必须全等num才可匹配 ===

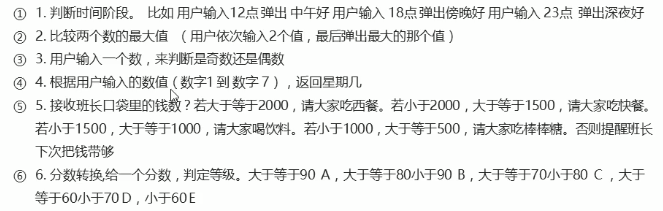
break没写 不会退出switch结构

案例——查询水果

用switch语句

总结：范围用ifelse多分支语句 准确值用switch语句

作业：



循环结构

目的：重复执行某些语句

for

双重for

while

do while

continue break

for(初始化变量;结束条件;循环变量变化){//xxxx}

for循环执行过程

1 var i=0

2 i<=100

3 执行循环体

4 i++

5 重复2~4 直到i>100

断点测试 观察程序执行过程

F12——Sources——选择对应的文件——点击行号（设置断点）——刷新浏览器（开始调试）——下一步下一步执行——鼠标放到变量上会显示其值

for循环

重复执行相同代码 固定语句

重复执行不同代码 通过i

重复执行某些操作 求和

for循环案例

1 求1~100之前所有数的平均值

2 求1~100之间所有的偶数和 奇数和

3 求1~100之间所有能被3整除的数字的和

案例——求学生成绩

输入学生人数n——分别输入n个学生的成绩——求总分和平均分

案例——打印图形

双层for循环

for(var i=0;i<10i++){

for(var j=0;j<10;j++){

str = str + ’\*’;

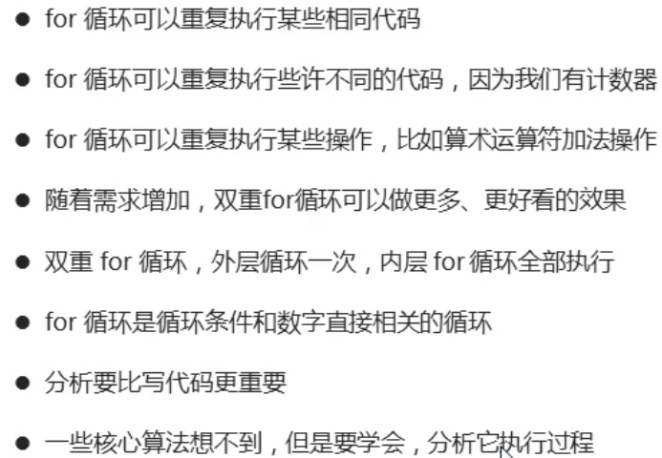
}

str = str + ‘\n’;

}

案例 1正方形 2 倒三角形 3 九九乘法表

for循环小结



while循环

while(条件表达式){

//循环体

}

do while循环

do{

//循环体

}while(条件表达式);

# 至少会执行一次

continue关键字

跳出**本次**循环，继续下一次循环。

break关键字

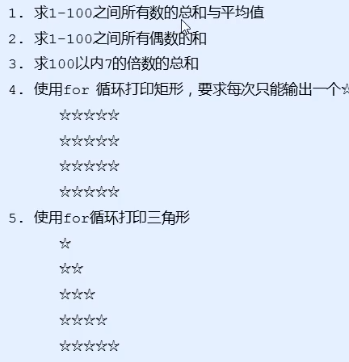
跳出**整个**循环

JS命名规则

变量用名词 方法用动词

空格等

做作业





JS数组

|  |  |
| --- | --- |
| 数组的概念    数组创建  new关键字 var arr = new Array();  字面量创建 var arr = [1,2,3];  提问    获取数组元素：arr[index] index是索引 从0开始  越界访问输出undefined  **遍历数组**  **arr.length**返回数组长度    总结  什么是遍历  怎样遍历  for里面的i是什么  怎么获取数组长度  数组索引号和数组长度的关系？    案例——数组求和、平均数  for循环遍历相加 和数组长读  案例——求数组中的最大值  案例——数组转换成字符串  数组新增元素  修改length长度（数组长度是可读写属性），默认是undefined  根据索引号扩充（追加数组元素）（注意下标是否已经有值）  不要直接给数组名赋值  案例——新建一个数组，存放1~100 100个整数  使用for循环  案例——筛选数组：选出大于10的数字，放入新的数组  可以newArr[newArr.length]=arr[i]  案例——删除指定数组元素：去掉所有的0  案例——翻转数组：1234 --> 4321  数组排序（冒泡排序）   |  | | --- | | for(var i = 0;i <= arr.length - 1;i ++){  for(var j = 0;j <= arr.length - i - 1;j ++){  if(arr[j]>arr[j+1])  {//进行交换}  }  } | |

函数

|  |
| --- |
| 函数的概念  功能相似的代码，可以重复执行，为了简化，封装成函数  如：求和函数（x~y）、  函数的使用  声明函数-->调用函数  声明函数  function 函数名([参数列表]){// 函数体}  调用函数  函数名([参数列表]);  函数的封装  一个或者多个功能通过函数的方式封装起来  类似于快递打包  案例——函数求1-100累加和  思考  函数的作用？声明函数的关键字？如何调用函数？函数的封装是什么？  带参数的函数  function 函数名([参数列表]){// 函数体}  函数的参数分为形参和实参  声明的时候写的是形参，调用的时候写的是实参  案例——求x到y之间所有的数字的和  形参和实参的个数的匹配问题  1 形参和实参的个数一致，正常执行。  2 实参的个数多余形参的个数时，会取到形参的个数  3 实参的个数小于形参的个数时，少的就会被定义为undefined  建议采取1  函数的返回值  return语句  function 函数名([参数列表]){return 需要返回的值;}  结果返回给调用者  案例——利用函数求两个数之间较大的那一个  return n1>n2?n1:n2;  案例——求数组中的最大值  return另一个功能：终止函数  return语句之后的代码就不会再执行了（永远不会）  return只能返回一个值。  函数没有return 返回的是undefined  此时比较break continue return三者的区别？  通过榨汁机看透函数  arguments的使用  存储了传递的所有的实参  当不确定有多少个参数传递的时候，就可以会用arguments来获取  实际上arguments是函数的一个内置对象    arguments的展示形式是一个伪数组  伪数组并不是真正的数组，但是它有length属性，可以通过下标访问元素，但是没有真正数组的一些方法：pop() push()等  可以遍历arguments  案例——利用函数求任意个数的最大值（不确定个数）  利用arguments  案例——翻转数组  案例——冒泡排序    案例——判断闰年  返回true或false  函数可以调用另外一个函数（函数之间的相互调用）  案例——输入年份，输入当年2月份的天数（调用上一个案例）  函数声明其实有两种方式  第一种是上面那些  第二种：var 变量名 = function(){};  调用方式是一样的  第一种是命名函数，第二种是匿名函数（那个是变量名，不是函数名） |

JavaScript作用域

|  |  |
| --- | --- |
| 作用域：代码名字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，减少命名冲突。  js的作用域（es6前）：全局作用域+局部作用域  全局作用域：整个script标签，或者是一个单独的js文件  局部作用域：在函数内部的（函数作用域）  不同作用域下的变量名可以相同（不冲突）  变量作用域：根据作用域的不同，分为全局变量和局部变量  全局变量：全局作用域里的变量  局部变量：在函数内部的（函数作用域）变量。只能在函数内部使用  注意：在函数内部，没有声明就赋值的变量也是全局变量  函数的形参也是局部变量  执行效率来看，全局变量只有浏览器关闭的时候才销毁，比较占内存  块级作用域   |  | | --- | | 现阶段JS没有块级作用域  JS在es6之后就会新增块级作用域  块级作用域就是{}包含起来的代码段 |   作用域链  var num =10  function fun1(){var num = 20; function fun2(){ console.log(num) } }  查找num的时候会根据链式查找（内部作用域访问外部变量）(就近原则)  最终输出20  案例——结果是几？    123  案例——结果是几？    a=4 b=’22’ |

JavaScript预解析

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1 console.log(num);  2 console.log(num);  var num=10;  3 fn();  function fn(){}  4 fun()  var fun =function(){} | 报错  undefined  正常  报错 |   1js引擎运行js分为两步 预解析 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   2 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 就是把所有变量的生命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的作用域最前面，不调用函数   函数表达式无法提升（上面第四点）  案例  ------>  输出结果：undefined  案例2  --->  结果：undefined 20  案例3  --->  结果：undefined 9  案例4 |

JS对象

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 什么是对象  具体的事物  JS对象是一组无序的相关属性和方法的集合  属性是事物的特征（名词）  方法是事物的行为（动词）  如：手机：大小、颜色、重量； 打电话、发短信、玩游戏  为什么需要对象  如：保存一个人的完整信息  创建对象（三种方式）  1 字面量创建对象   |  | | --- | | 对象字面量是花括号{}  var obj={uname:”小叶”,age:18}  调用：1）obj.uname 2）obj[“uname”] |   变量、属性、函数、方法  前两都是用来存储数据的。变量：单独存在 属性：在对象里  后两都是实现某种功能的。函数：单独存在 方法：在对象里  2 new Object()创建对象   |  | | --- | | var obj= new Object();  obj.uname=”小叶”;  obj.sex=’男’； |   3 利用构造函数创建对象   |  |  |  | | --- | --- | --- | | 前面两种，一次只能创建一个对象  如：需要创建四大天王的对象  声明   |  | | --- | | function 构造函数名（）{  this.属性=值;  this.方法=function(){}  } |   使用   |  | | --- | | new 构造函数名(); |   构造函数名字首字母大写  var ldh = new Star();  此时可以看看typedef ldh  只要new了，就是调用函数创建了对象  必须加new  构造函数和对象的区别  对象是具体的事物，特指某一个；构造函数是模板，泛指一大类。  利用构造函数创建对象的过程称为对象的实例化  new关键字执行过程  1 在内存中创建一个空的对象  2 this就会指向这个空对象  3 开始执行构造函数中的代码，添加属性和方法  4 返回这个对象 |   遍历对象（for-in）   |  | | --- | | var obj = {  name:’y’,  sex:’man’,  age:18  }  for (var k in obj){  k是属性名  obj[j]是属性值  } |   小结  对象属于复杂数据类型  对象就是一组无序的相关属性和方法的集合  构造函数泛指一大类，对象特指某个实例  for..in..遍历对象 |

内置对象

|  |
| --- |
| JS中的对象分为：自定义对象、内置对象、浏览器对象  前两种属于JS基础内容，基于ECMAScript 第三个属于JS独有  内置对象就是JS语言自带的一些对象，帮助人们快速开发  内置对象举例：Math、Date、Array、String |

查文档

|  |
| --- |
| MDN/W3C  1 查阅改方法的功能  2 查看方法的参数和说明  3 查看方法的返回值和说明  4 通过demo进行测试 |

Math对象

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 数学常数和函数的属性和方法  Math不是一个构造器，所有的方法和属性都是静态的  如：Math.PI、Math.max()、  案例——利用对象封装自己的Math求最大值和最小值   |  | | --- | | var myMath = {  PI:3.1415,  max:function(){  var max = arguments[0];  for(){}  return max;  }  min:function(){  var min= arguments[0];  for(){}  return min;  }  }  console.log(myMath.PI); |   其他常用的   |  |  | | --- | --- | | Math.floor() | 向下取整 | | Math.ceil() | 向上取整 | | Math.abs() | 绝对值 | | Math.round() | .5往大取 | | Math.random() | 返回随机数，浮点数x∈[0,1)，无参数 |   得到两个数之间的随机整数[x,y]   |  | | --- | | x = Math.ceil(x);  y = Math.floor(y);  return Math.floor(Math.random()\*(y-x+1))+x  可以应用在随机点名 |   案例——猜数字游戏 |

Date日期对象

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 必须使用new调用  var date = new Date()  无参数返回当前系统时间  参数(‘1999-11-25 12:12:12’)  参数（1999,11,25,12,12,12,）  参数（2020,10,10）  参数（‘2019/5/1’）  常用方法   |  |  | | --- | --- | | getFullYear() | 年 | | getMonth() | 月份 0-11 | | getDate() | 日期 几号 | | getDay() | 0-6 周日0 | | getHours() | 时 | | getMinutes() | 分 | | getSeconds() | 秒 |   Date总的毫秒数（三种方式都可）  获取日期的总的毫秒形式  1970-1-1  时间戳（当前距离1970-1-1总的毫秒数）  var date = new Date();  date.valueOf()  date.getTime()  var date = +new Date() （推荐）  Date.now (H5新增)  案例——倒计时  不够10，前面补0  时间戳相减  parseInt()转换成整数  时间戳之差/1000 = 单位是秒times  times/60/60/24=天  times/60/60%24=时  times/60%60=分  times%60=秒  以上都需要取整 |

数组Array对象

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 创建   |  | | --- | | 1 字面量  var arr = [1,2,3];  2 new  var arr1 = new Array(2); //2是长度  var arr1 = new Array(2,3);// 2 3 是其中的元素 |   检测是否为数组   |  | | --- | | 1 instanceof  var arr = [];  console.log(arr instanceof Array); // true  2 Array.isArray(arr)  var arr = [];  console.log(Array.isArray(arr));// true  (H5新增) |   添加数组元素   |  |  | | --- | --- | | push(x) | 末尾加x,可以push多个，返回新数组长度，修改原数组 | | unshift(x) | 开头加x，可以加多个，返回新数组长度，修改原数组 |   删除数组元素   |  |  | | --- | --- | | pop() | 删除最后一个元素，作为返回值返回 | | shitf() | 删除第一个元素，作为返回值返回 |   案例——筛选数组  使用push():符合条件，newArr.push[x]  数组排序   |  | | --- | | 1 reverse()翻转数组  var arr = [1,2,3]  arr.reverse(); //arr = [3,2,1]  2 sort()排序数组  var arr = [3,1,2];  arr.sort(); // arr=[1,2,3]  这个是按照ASCII码来排的  可以这样子解决  arr.sort(function(a,b){return a - b;}); //排序是升序 |   获取元素的索引   |  | | --- | | 1 indexOf(元素)  元素第一次出现的位置的索引  2 lastIndexOf(元素)  元素最后一次出现的位置的索引  没有找到，都是返回-1 |   案例——数组去重  去除重复的数组元素  遍历旧数组，查询新数组是否有，没有就存，有就过  可以使用indexOf()判断新数组是否存在指定元素(!=-1)  封装成函数    数组转换成字符串   |  |  | | --- | --- | | arr.toString() | 数组转字符串 | | arr.join(分隔符) | 分隔符默认是逗号 |   其他方法   |  |  |  | | --- | --- | --- | | concat() | 连接两个或多个数组，不影响原数组 | 返回新数组 | | slice() | 数组截取slice(begin,end) | 返回截取之后的新数组 | | splice() | 数组删除splice(begin,count) | 会影响原数组，返回删后的 | |

字符串对象String

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| var str =’mike’;  str.length  基本包装类型：简单数据类型包装成复杂数据类型  其实是这么执行的  var temp = new String(“mike”);  str = temp;  temp = null;  除此之外还有Number、Boolean  字符串不可变   |  | | --- | | var str = ‘andy’;  str = ‘red’    因此不要大量地拼接字符串，会很消耗内存 |   返回字符所在位置   |  | | --- | | 1 indexOf(字符,index)  字符在index开始，第一次出现的位置的索引，index可省略  2 lastIndexOf(字符)  字符在index开始，最后一次出现的位置的索引，index可省略  没有找到，都是返回-1 |   案例——查找一个字符串中指定字符的位置和出现的次数  indexOf()找到出现的位置  作为第二次开始查找的位置，循环查找      改进——求数组中，指定元素出现的位置和次数  根据位置返回字符（重点）   |  |  |  | | --- | --- | --- | | charAt(index) | 根据位置返回字符 | str.charAt(1) | | charCodeAt(index) | 根据位置返回字符的ASCII码 | str.charCodeAt(1) | | str[index] | 获取指定位置的字符 | H5，IE8+支持 |   案例——统计出现次数最多的字符  有一个对象 判断是否有改属性 obj[‘属性名’]      字符串操作方法（重点）   |  |  | | --- | --- | | concat(str1,str2,str3) | 相当于+ | | substr(start,length) | 重点记住这个,length不写就是取到结尾 | | slice(start,end) | [start,end) | | substring(start,end) |  | | replace(s,d) | s变成d，只换一个（写个案例，替换全部） | | split(分隔符) | 字符串转数组，默认是一个个隔开 | | toUpperCase() | 转大写 | | toLowerCase() | 转小写 |     作业 |

JS数据类型

|  |  |  |
| --- | --- | --- |
| 简单类型和复杂数据类型   |  | | --- | | 简单：基本/值  string number boolean undefined ull  复杂：引用  new出来的 Object Array Date等 |   简单数据类型null返回的是一个空对象  如果一个变量打算存为对象，可以先存为null  堆和栈  简单数据类型在栈  复杂数据类型在堆    传递参数   |  | | --- | | 1 简单类型    即不会改变a的值  2 复杂类型    1 刘德华  2 刘德华  3 张学友  4 张学友    复杂数据类型传的是地址 | |