JavaScript笔记

js:解释型语言，是脚本语言

目录

[一、对话框/断点调试 4](#_Toc71896411)

[（一）对话框 4](#_Toc71896412)

[（二）断点调试 4](#_Toc71896413)

[二、字符型和数字型的转换 5](#_Toc71896414)

[（一） number -> string 5](#_Toc71896415)

[（二） string -> number 5](#_Toc71896416)

[三、运算符 5](#_Toc71896417)

[（一） 5](#_Toc71896418)

[（二）逻辑运算符 5](#_Toc71896419)

[1.短路运算（逻辑中断） 5](#_Toc71896420)

[2. 赋值运算符 6](#_Toc71896421)

[3.运算优先级 6](#_Toc71896422)

[四、流程控制 6](#_Toc71896423)

[（一）分支流程控制 6](#_Toc71896424)

[（二）循环流程控制 7](#_Toc71896425)

[五、数组 7](#_Toc71896426)

[六、函数 8](#_Toc71896427)

[（一）形参与实参 8](#_Toc71896428)

[1.函数形参实参个数应该匹配，但是JS里面不匹配也可以 8](#_Toc71896429)

[（二）return 8](#_Toc71896430)

[（三）arguments 8](#_Toc71896431)

[（四）函数的两种声明方式 9](#_Toc71896432)

[七、作用域 9](#_Toc71896433)

[（一）全局变量和局部变量 9](#_Toc71896434)

[1.全局变量 9](#_Toc71896435)

[2.局部变量 9](#_Toc71896436)

[3.块级作用域 9](#_Toc71896437)

[（二）作用域链 9](#_Toc71896438)

[七、预解析 11](#_Toc71896439)

[（一） 11](#_Toc71896440)

[（二）分类 11](#_Toc71896441)

[（三）案例 11](#_Toc71896442)

[四、对象 12](#_Toc71896443)

[（一）定义 12](#_Toc71896444)

[（二）创建对象的三种方式 13](#_Toc71896445)

[1. 利用对象字面量创建对象 { } 13](#_Toc71896446)

[2. 利用New Object创建对象 13](#_Toc71896447)

[3. 利用构造函数创建对象 14](#_Toc71896448)

[（三）对象的使用 15](#_Toc71896449)

[1. 调用对象的属性 15](#_Toc71896450)

[2. 调用对象的方法 15](#_Toc71896451)

[（四）对象的遍历 16](#_Toc71896452)

[1. for in 遍历对象 16](#_Toc71896453)

[五、内置函数 16](#_Toc71896454)

[（一）Math函数—常用对象 16](#_Toc71896455)

[（二）Data日期对象（必须使用new） 17](#_Toc71896456)

[1. 使用Data 17](#_Toc71896457)

[2. 日期的格式化 17](#_Toc71896458)

[3. 获得Data总的毫秒数（时间戳） 18](#_Toc71896459)

[4. 秒与天、时、分的转换 18](#_Toc71896460)

[（三）Array数组对象 18](#_Toc71896461)

[1.创建数组 18](#_Toc71896462)

[2. 检测是否为数组 19](#_Toc71896463)

[3. 添加删除数组元素的方法 19](#_Toc71896464)

[4. 数组排序 20](#_Toc71896465)

[5. 数组索引方法 21](#_Toc71896466)

[6. 数组转换为字符串 21](#_Toc71896467)

[7. 数组的连接、截取与删除或替换 22](#_Toc71896468)

[（四）字符串对象 24](#_Toc71896469)

[1. 基本包装类型 24](#_Toc71896470)

[2. 字符串的不可变 24](#_Toc71896471)

[3. 根据字符 返回位置 24](#_Toc71896472)

[4. 根据位置返回字符 24](#_Toc71896473)

[5.字符串操作方法 26](#_Toc71896474)

# 一、对话框/断点调试

## （一）对话框

1. prompt : 是弹出网页对话框，要求输入 如 prompt("请输入您的姓名：");

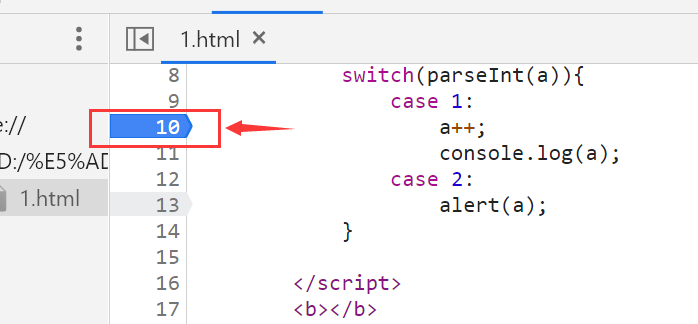
2. alert : 是弹出对话框，不可输入 如 alert('您的姓名是：' + \_name + '\n')

3. '\n' ： 代表在对话框内换行 ，同理还有'\\'

## （二）断点调试

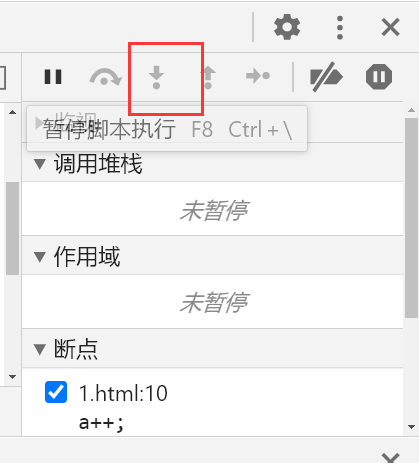
1.点击F12打开控制台，再点击源代码

2.打开代码。然后点击需要暂停的一行的左边

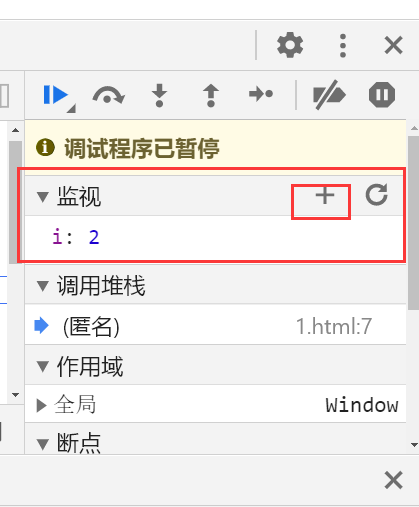


3.刷新网页，即可发现网页只运行到暂停的那一步

4.可以点击如图按钮或者F11进行下一步操作



5.在监视处可以添加需要观察的变量变化（或者直接鼠标放变量上看也能看到变量的值）



6.取消断点：再次点击代码左侧并刷新页面

# 二、字符型和数字型的转换

## （一） number -> string

string() : 强制转换 ，如 num=123 , string(123)

!! 隐式转换 : 如 132 + ' ' 结果为字符型 其中 + 号最重要，只要任何数字+上字符型，都为字符型

## （二） string -> number

2.1 parseInt() : 如 age = '123w' parseInt(age) = 123

2.2 parseFloat() : 如 age = '1.23wz3.45' parseFloat(age) = 1.23

但是 age 字符型是字母开头，则使用这两个都会返回NaN（not a number）

# 三、运算符

注意：尽量不要用浮点数进行直接运算，会有误差；同理浮点数不能直接比较

## （一）

1. ++a 前置自增运算符 : 先自加1，在返回值 如 ++age 等于 age = age + 1

2. a++ 后置自增运算符 : 先返回原值，后自加1 如 age = 10 , a = age++ + 10 , a = 20 , age = 11

前后自增如果单独使用，效果是一样的

3. == : 等于符号，只要求值一致，会转型 如 18 == '18' , 返回true

4. === : 全等于，要求两边数据类型和值完全一致 如 18 === '18' , 返回false

## （二）逻辑运算符

### 1.短路运算（逻辑中断）

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的表达式的值

1.1 逻辑与 : &&

语法 : 表达式1 && 表达式2

如果表达式1 true ，返回表达式2 ；如果表达式1 false ，返回表达式1

如 : 123 && 456 , 返回456 ; 0 && 456 , 返回0

(假命题：0 '' null undefine NaN)

1.2 逻辑或 : ||

语法 : 表达式1 || 表达式2

如果表达式1 true ，返回表达式1 (表达式1执行，后面的表达式不执行！)；如果表达式1 false ，返回表达式2

如 : 123 || 456 || 789 , 返回123 ; 0 || 456 || 822 , 返回456

### 2. 赋值运算符

2.1 += : 如 num += 2 等于 num = num + 2

2.2 -= \*= /=同理

### 3.运算优先级

A : ()

B : 一元运算符 只涉及一个的算数运算 常用符号：++ -- !

注：二元运算符是非B类，二元运算符例：1 + 1 ； 三元运算符也是非B类，例子：条件表达式 ? 表达式1 : 表达式2 如果条件表达式结果为真则返回表达式1，反之则返回表达式2

C : 算术运算符 先\*/后+-

D : 关系运算符 > >= < <=

E : 相等运算符 == != === !==

F : 逻辑运算符 先&& 后||

G : 赋值运算符 =

H : 逗号运算符 ，

# 四、流程控制

顺序流程控制；分支流程控制；循环流程控制

## （一）分支流程控制

if else if :

if(条件表达式1) {

语句1;

} else if(条件表达式2) {

语句2;

} else if(条件表达式3) {

语句3;

} else if(条件表达式4) {

语句4;

} else {

最后的语句;

}

//如果条件表达式1满足，则执行语句1，且退出if语句；如果条件表达式1不满足，往下走，执行第一个'else if'，如果满足条件表达式2，则执行语句2并结束整个if语句；以此类推;如果上面的所有表达式都不成立，则执行else。

！注意 ：分支语句是多选一，所有语句中只能有一个语句执行，因此一般都需多个if连用判断。

## （二）循环流程控制

1. 一行输出：需要字符串，可以在循环里面写 var str = ‘ ‘ str = str + ‘(需要的东西)’

# 五、数组

1. 创建：var arr = []； 也可以写成var arr = new Array();

2. 在数组中可以放任意的数据类型，如整型、字符型、布尔型 var arr = [ 1 , 2 , ‘阿泽’ , true]

3. arr.length 可以直接获得arr数组的长度

4. 数组新增元素：修改length的长度，多余n个的会在控制台输出empty \* n

修改索引号，如arr[3] = [ ‘one’ , ‘tow’ , ‘three’ ]; arr[4] = ‘ABC’ 即可增加成功；同理再输入arr[0] = ‘pink’ , 此时 arr[4] = [ ‘pink’ , ‘tow’ , ‘three’ , ‘ABC’ ]

注：不要直接给数组名赋值，否则整个数组都会被覆盖

5. 冒泡排序：

方法一： 方法二：

int a[5]={100,40,50,21,90}; int a[5]={100,40,50,21,90};

int i,j; int i,j;

int temp; int temp;

for(i=0;i<5;i++) for(i=0;i<5;i++)

{ {

for(j=0;j<5-i-1;j++) for(j=i+1;j<5;j++)

if(a[j]> a[j+1]) if(a[i]> a[j])

{ {

temp=a[j]; temp=a[i];

a[j]=a[j+1]; a[i]=a[j];

a[j+1]=temp; a[j]=temp;

} }

} }

for(i=0;i<5;i++) for(i=0;i<5;i++)

{ {

printf("%5d",a[i]); printf("%5d",a[i]);

} }

# 六、函数

## （一）形参与实参

### 1.函数形参实参个数应该匹配，但是JS里面不匹配也可以

1.1 实参 > 形参 ： 会取到形参的个数 如：getSum(num1,num2) return num1+num2 getSum(1,2,3) 返回1+2=3

1.2 实参 < 形参 ： 形参因为有的没有接受值，所以是undefined，因此一个数值+undefined结果是NaN

## （二）return

1. return可以终止函数，即return之后的代码不被执行

2. 返回的结果只能是最后一个值 如return 1，3； 显示反回3 因此返回多个值时常返回数组

3. 如果函数没有return，则返回undefined

4. return也可退出循环（但是不常用）

## （三）arguments

1. arguments的使用：放在函数里面，里面存储了所有传递过来的实参

2. arguments是一个伪数组

2.1 具有数组length的属性

2.2 按照索引的方式存储 如arguments=[2] ；因此也可以按照数组的方式遍历

2.3 没有真正数组的一些方法 如pop() push()

3. 只有函数才有argumens对象，且每个函数都内置好了这个argumens

## （四）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）：function fn(){}

2. 函数表达式（匿名函数）：var 变量名 = function (){} 如 var fun = function(){} fun();

2.1 fun是变量名，不是函数名

2.2 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

2.3 函数表达式也可以进行传递参数

# 七、作用域

## （一）全局变量和局部变量

### 1.全局变量

1.1 是在全局作用下的变量

1.2 只有在网页关闭的时候才会销毁，比较占内存资源。

### 2.局部变量

2.1 在局部作用域下的变量，常为函数中的声明的变量和形参（注：在函数内部没有声明却直接赋值的变量属于全局变量）

2.2 当程序被执行的时候才会初始化，当程序执行完毕就会销毁，更节省内存空间。

### 3.块级作用域

3.1 JS没有块级作用域，在es6的时候才会新增

3.2 定义：if {} ； for{} 花括号里面定义的变量作用域就算块级作用域。

## （二）作用域链

1. 定义：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那个值，这种结构我们称为作用域链。

2. 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。

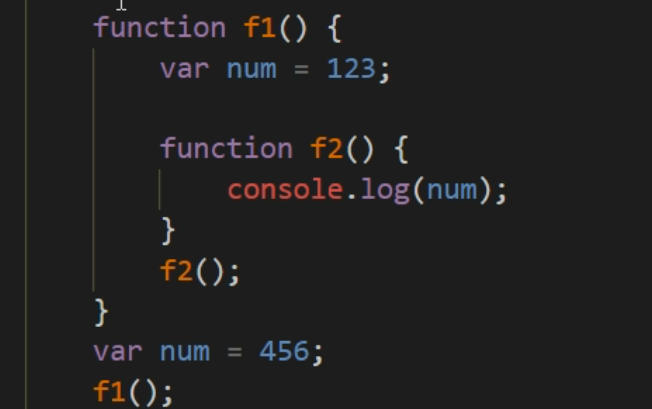
3. 根据在内部函数可以访问外部函数变量的这种机制，用链式查找决定哪些数据能被内部函数访问，就称作作用域链。

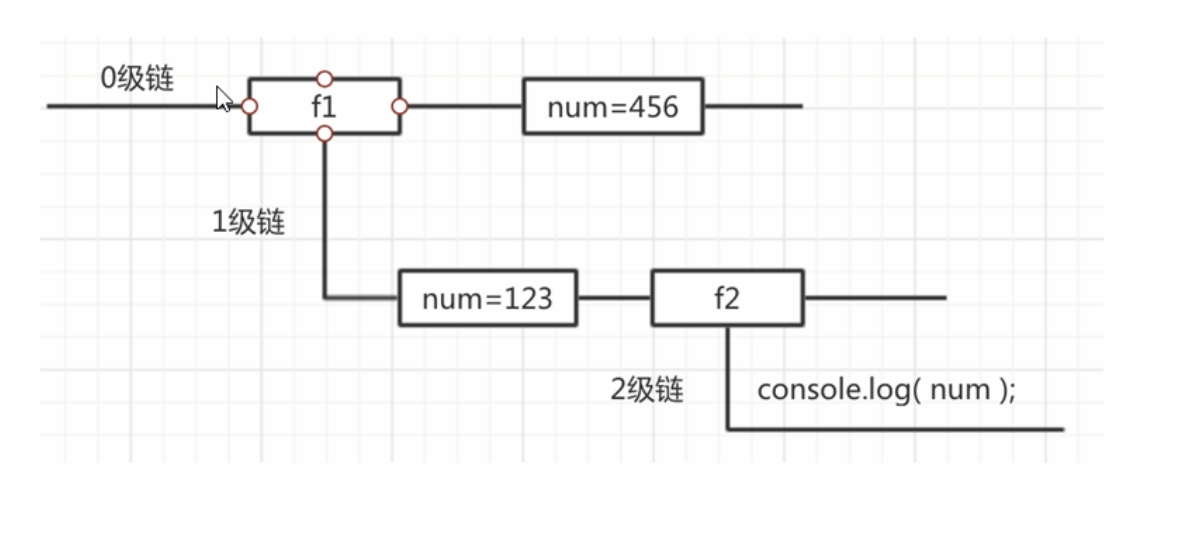
如： 例一



先是最里层函数fun里面寻找num；没有就向上一个作用域（也就是fn函数）寻找num，如果有就使用该num的数值并停止寻找；如果也没有就向再上一个作用域（也就是全局）寻找num。

例二





站在目标出发，一层一层的往外查找。 结果是打印了123。

总结：就近原则

# 七、预解析

# （一）

我们js引擎运行js分为两步: 预解析 代码执行

1. 预解析：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

# （二）分类

1. 变量预解析（变量提升）

定义：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

如： console.log(num); var num = 10;

结果是undefined,而非报num is undefinded的错误

原因：代码相当于执行了 var num ; console.log(num); num = 10;

2. 函数预解析（函数提升）

定义：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

# （三）案例

--------------------------------------------------------------

例一：

预解析前

var num = 10 ;

function fun ( ) {

console.log(num) ;

var num = 20 ;

}

fun ( ) ;

--------------------------------------------------------------

例二：

预解析前

--------------------------------------------------------------

预解析后

var num ;

function fun ( ) {

var num ;

console.log(num) ;

num = 20 ;

}

num = 10 ;

fun ( ) ;

结果：undefined

--------------------------------------------------------------

预解析后

f1 ( ) ;

console.log(c) ;

console.log(b) ;

console.log(a) ;

function f1 ( ) {

var a = b = c = 9 ;

console.log(a) ;

console.log(b) ;

console.log(c) ;

}

--------------------------------------------------------------

例三：

预解析前

var a = 18 ;

f1 ( ) ;

function f1 ( ) {

var b = 9 ;

console.log (a) ;

console.log (b) ;

var a = '123' ;

}

--------------------------------------------------------------

function f1 ( ) {

var a ;

a = b = c = 9 ;

//相当于var a = 9; b = 9 ; c = 9 ; b和c直接赋值没有var声明当全局变量看

console.log(a) ;

console.log(b) ;

console.log(c) ;

}

f1 ( );

console.log(c) ;

console.log(b) ;

console.log(a) ;

结果：9 9 9 9 9 9 错误

--------------------------------------------------------------

预解析后

var a ;

function f1 ( ) {

var b ;

b = 9 ;

console.log (a) ;

console.log (b) ;

var a ;

a = '123' ;

}

a = 18 ;

f1 ( ) ;

结果：undefined 9

--------------------------------------------------------------

# 四、对象

## （一）定义

1. 定义：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。

2. 属性：事物的特征，在对象中用属性来表示。（常用名词）

3. 方法：事物的行为，在对象中用方法来表示。（常用动词）

## （二）创建对象的三种方式

### 1. 利用对象字面量创建对象 { }

#### 1.1 例子

var obj = {

uname : ‘张三’ ,

age : 18 ,

sex = ‘男’ ,

//以上是属性

//以下是方法（方法常用新定义的函数来执行）

sayHi : function ( ) {

console.log(‘Hi~’) ;

}

}

#### 1.2 注意事项

1.2.1 里面的属性或者方法我们采取键值对的形式键属性名：值 属性值

1.2.2 多个属性或者方法中间用逗号隔开的（最后一个可以不用逗号隔开）

1.2.3 方法冒号后面跟的是一个匿名函数

### 2. 利用New Object创建对象

#### 2.1 例子

var obj = new Object ( ) ;

obj.uname = ‘张三’ ;

obj.age = 18 ;

obj.sex = ‘男’ ;

obj.sayHi = function ( ) {

console.log( ‘Hi~’ );

} ;

#### 2.2 注意事项

2.2.1 我们是利用 等号 赋值的方法 添加对象的属性和方法

2.2.2 每个属性和方法之间用 分号 结束

2.2.3 创建的时候new Object 的Object的O要大写

### 3. 利用构造函数创建对象

#### 3.1 由来

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

#### 3.2定义

3.2.1 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

3.2.2 构造函数是泛指一大类；而对象是特指一个具体的事物

3.2.3 对象的实例化：利用构造函数创建对象的过程就叫对象的实例化

#### 3.3 例子

//标准

function 构造函数名 ( ) {

this.属性 = 值 ;

this.方法 = function ( ) { }

}

//使用

new 构造函数名 ( ) ;

----------------------------------------------------------------------------------

//真实例子

function Star ( uname , age , sex ) {

this.name = uname ;

this.age = age ;

this.sex = sex ;

this.sing = function ( sang ) {

console.log (uname + '的歌曲' + sang ) ;

}

} //这里面的name、age、sex都是属性，而unmae、age、sex都是形参

var ldh = new Star( ‘刘德华’ , 18 , ‘男’ ) ;

var zxy = new Star( ‘张学友’ , 20 , ‘男’ ) ;

console.log( ldh.name ) ; //结果是 刘德华

ldh.sing ( ‘冰雨’ ) ; //结果是 刘德华的歌曲冰雨

#### 3.4 new关键字执行过程

3.4.1 new构造函数可以在内存中创建一个空的对象；

3.4.2 this就会指向刚才创建的空对象；

3.4.3 执行构造函数里面的代码，给这个空对象添加属性和方法；

3.4.4 返回这个对象（因此不需要return）

#### 3.5 注意事项

3.5.1 构造函数名首字母要大写（惯用方法，便于区分普通函数）

3.5.2 构造函数不需要return就可以返回结果

3.5.3 var ldh = new Star ( ) 这个调用函数返回的是一个对象，即console.log( typeof ldh ) 结果是object

3.5.4 调用构造函数时，必须使用new

3.5.5 只要调用了new Star ( ) ,就创建了一个新对象

3.5.6 属性和方法之前必须添加this

## （三）对象的使用

构造函数需要先建立一个变量来等于这个对象，才能使用以下方法来显示属性或方法

如：var 变量 = new 构造函数名 ( ) ; console.log( 变量.属性 ) ;

### 1. 调用对象的属性

1.1 方法1： 对象名.属性名

如：console.log( obj.age ) ;

1.2 方法2： 对象名[‘属性名’]

如：console.log( obj [ ‘uname’ ] ) ; //必须打引号

//得到的都是对象的值

### 2. 调用对象的方法

2.1 方法1： 对象名.方法名 （不能忘记小括号）

如：obj.sayHi( ) ;

## （四）对象的遍历

### 1. for in 遍历对象

#### 1.1 例子

for ( 变量 in 对象 ) {

}

-------------------------------------------------------------------------

for ( var k in obj ) {

console.log ( k ) ; // k是变量，输出得到的是属性名（如name、age、sex）

console.log ( obj[k] ) ; //输出得到的是属性值（如张三、18、男）

}

#### 1.2 注意事项

1.2.1 变量常用k或者key

1.2.2 输出属性值时，不需要在变量两边打引号（与之前的对象调用不同）

# 五、内置函数

## （一）Math函数—常用对象

Math.max( ) ; 取最大值

Math.min( ) ; 取最小值

Math.abs( ) ; 取绝对值

Math.floor( ) ; 表示小于或等于指定数字的最大整数的数字

Math.ceil( ) ; 返回大于或等于一个给定数字的最小整数

Math.pow( ) ; 次幂，如Math.pow(2,7) 表示2得7次方

Math.round( ) ; 给定数值的四舍五入（注：Math.floor( -1.5 )得-1；Math.floor( -1.8 )得-2）

使用以上不需要先new一下

## （二）Data日期对象（必须使用new）

### 1. 使用Data

#### 1.1 如果Data的括号内没有参数，则返回系统当前时间

var data = new Data( ) ;

console.log(data);

#### 1.2 Data的括号内有参数

数字型：2021,4,23

字符型：‘2021-4-23 15:35:30’ 或者 ‘2021/4/23 15:35:30 ’ //年月份和时分必须有个空格

---------------------------------------------------------------------------------------------------

例1： var data1 = new Date(2021,4,23);

console.log(data1);

//结果：Sun May 23 2021 00:00:00 GMT+0800 (中国标准时间)

---------------------------------------------------------------------------------------------------

例2： var data2 = new Date('2021-4-23 15:35:30');

console.log(data2);

//结果：Fri Apr 23 2021 15:35:30 GMT+0800 (中国标准时间)

注：JS跑出来会有问题，用网页跑没问题

用数字型时，跑出来会比定义的月份大一个月

使用、定义时必须new

### 2. 日期的格式化

年： 例：var data = new Date();

console.log(data.getFullYear());

月： 例：console.log(data.getMonth()); //返回月份小一个月

//因此常写成console.log(data.getMonth()+1);

日： 例：console.log(data.getDate());

周几：例：data.getDay() //周一返回1，周六返回6，周日返回0

时： 例：data.getHours()

分： 例：data.getMinutes()

秒： 例：data.getSeconds()

### 3. 获得Data总的毫秒数（时间戳）

获得的不是当前毫秒数，而是距离1970年1月1日共过了多少毫秒

3.1 写法一

例：var data = new Date();

console.log(data.valueOf()); //或者data.getTime(); 也是返回一样的值

3.2 写法二

例：var data1 = +new Date();

console.log(data1);

3.3 写法三（HTML5才有的，版本过低不能使用）

例：Date.now()

3.4 其他

var a = +new Date('2020-5-1 0:0:0');

//这样写可以获得当天的时间戳

### 4. 秒与天、时、分的转换

d = parseInt(总秒数/60/60/24) //计算天数

h = parseInt(总秒数/60/60%24) //计算小时数

m = parseInt(总秒数/60%60) //计算分钟数

s = parseInt(总秒数%60) //计算当前秒数

## （三）Array数组对象

### 1.创建数组

#### 1.1 利用数组字面量

例：var arr = [ 1 , 2 , 3 ];

#### 1.2 利用new Array()

例：var arr1 = new Array(); //创建一个空数组

var arr2 = new Array(2); //这个2表示数组的长度为2，里面有两个空的数组元素

var arr3 = new Array(2 , 3); //等价于[ 2 , 3 ] 这时候存了2，3两个元素的数组

### 2. 检测是否为数组

#### 2.1 instanceof运算符

A instanceof B 可以理解为A属于B吗？ 如果是则返回true，反之返回false

例： var arr = [ ] ;

console.log( arr instanceof Array ) //返回true

#### 2.2 Array.isArray(参数)

注：是h5新增的，ie9以上版本支持

例：var arr = [ ] ;

console.log(Array.isArray( arr ) ); //返回true

### 3. 添加删除数组元素的方法

#### 3.1 添加

##### 3.1.1 push()

定义：在数组末尾添加一个或多个数组元素

注：在括号里面直接写数组元素就可以

push完毕后会有一个返回值，返回结果是新数组的长度

原数组也会发生变化

例：var arr = [ 1 , 2 , 3 ]

console.log( arr.push(4,5,’ze’) )

//结果：6

console.log( arr )

//结果：[ 1 , 2 , 3 , 4 , 5 , “ze” ]

##### 3.1.2 unshift()

定义：在数组开头添加一个或多个数组元素

注：在括号里面直接写数组元素就可以

unshift完毕后会有一个返回值，返回结果是新数组的长度

原数组也会发生变化

例：var arr = [ 1 , 2 , 3 ]

console.log( arr.unshift( ‘wang’ , 0) )

//结果：6

console.log( arr )

//结果：[ “wang” , 0 , 1 , 2 , 3 ]

#### 3.2 删除

##### 3.2.1 pop()

定义：可以删除数组的最后一个元素

注：pop()括号例不跟参数

pop完毕后返回的结果是删除的那个元素

原数组也会发生变化

例：var arr = [ 1 , 2 , 3 ]

console.log( arr.pop( ) );

//结果：3

console.log( arr );

//结果：[ 1 , 2 ]

##### 3.2.2 shift()

定义：可以删除数组的第一个元素

其他都可pop一样

### 4. 数组排序

#### 4.1 翻转数组

reverse()

例：var arr = [ ‘blue’ , ‘white’ , ‘pink’ ];

arr.reverse(); //即可完成翻转

#### 4.2 冒泡排序

sort() //从小到大排序

例：var arr1 = [ 3 , 5 , 1 , 7 ]

arr1.sort();

console.log(arr1);

//结果：[ 1 , 3 , 5 , 7 ]

注：如果是非个位数排序，sort没办法做到，他会默认将最高位相同的放在一起

如：var arr2 = [ 13 , 5 , 1 , 77 , 7 ]

arr2.sort();

console.log(arr2);

//结果：[ 1 , 13 , 5 , 7 , 77 ]

sort()完整用法

例：var arr3 = [ 13 , 5 , 1 , 77 , 7 ]

arr3.sort( function ( a , b ) {

return a - b; //按照升序的顺序排列

// return b - a; //按照降序的顺序排列

} );

console.log(arr3);

//结果：[ 1 , 5 , 7 , 13 , 77 ]

### 5. 数组索引方法

#### 5.1 indexOf(数组元素)

例：var arr = [ ‘blue’ , ‘white’ , ‘pink’ , ‘green’ ];

console.log( arr.indexOf( ‘blue’ ) )

//结果：0

注：这个是从前面开始索引的

如果没有找到就返回-1

如果数组中有多个一样的元素，则返回第一个满足的索引号/下标

#### 5.2 lastIndexOf(数组元素)

例：var arr = [ ‘blue’ , ‘white’ , ‘pink’ , ‘green’ ];

console.log( arr.indexOf( ‘pink’ ) )

//结果：2

注：这个是从末尾开始索引的

如果没有找到就返回-1

### 6. 数组转换为字符串

#### 6.1 toString()

例：var arr = [1, 2, 3];

console.log(arr.toString()); //结果：1，2，3

#### 6.2 join(分隔符)

如果括号里面什么都不写，默认以‘逗号’分隔

如果括号里面写了，就以括号里面来分隔

例：var arr = ['green', 'blue', 'green'];

console.log(arr.join('-')); //结果：green-blue-green

### 7. 数组的连接、截取与删除或替换

#### 7.1 连接：concat()

var new\_arr = arr.concat()

会返回新的数组

例：var alpha = ['a', 'b', 'c'];

var alphaNumeric = alpha.concat(1, [2, 3]);

console.log(alphaNumeric);

// 结果['a', 'b', 'c', 1, 2, 3]

#### 7.2 截取：slice()

arr.slice(begin, end) //begin与end都可以省略，也可以只省略end

会返回新的数组，但原数组不会改变

返回不包括begin和end下标的元素

例：const animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];

console.log(animals.slice(2));

// 结果： Array ["camel", "duck", "elephant"]

console.log(animals.slice(2, 4));

// 结果： Array ["camel", "duck"]

console.log(animals.slice(1, 5));

// 结果： Array ["bison", "camel", "duck", "elephant"]

#### 7.3删除或替换：splice()

array.splice( start , deleteCount , item1 item2 ....)

##### 7.3.1 start（必须）​：

指定修改的开始位置（从0计数）。如果超出了数组的长度，则从数组末尾开始添加内容；如果是负值，则表示从数组末位开始的第几位（从-1计数，这意味着-n是倒数第n个元素并且等价于array.length-n）；如果负数的绝对值大于数组的长度，则表示开始位置为第0位。

##### 7.2.2 deleteCount（非必须）：

整数，表示要移除的数组元素的个数。

如果 deleteCount 大于 start 之后的元素的总数，则从 start 后面的元素都将被删除（含第 start 位）。

如果 deleteCount 被省略了，或者它的值大于等于array.length - start(也就是说，如果它大于或者等于start之后的所有元素的数量)，那么start之后数组的所有元素都会被删除。

不会返回新的数组

先deleteCount再添加tiem

##### 7.3.3 item1, item2, ... （非必须）

要添加进数组的元素,从start 位置开始。如果不指定，则 splice() 将只删除数组元素。

##### 7.3.4 例子：

例一：

const months = ['Jan', 'March', 'April', 'June'];

months.splice(1, 0, 'Feb');

console.log(months);

// 结果： Array ["Jan", "Feb", "March", "April", "June"]

months.splice(4, 1, 'May');

console.log(months);

// 结果： Array ["Jan", "Feb", "March", "April", "May"]

-------------------------------------------------------------------------------------

例二：从倒数第 2 位开始删除 1 个元素

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

var removed = myFish.splice(-2, 1);

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

// 被删除的元素: ["mandarin"]

注：倒着数是往前删

## （四）字符串对象

### 1. 基本包装类型

定义：把简单数据类型包装成复杂数据类型（因为只有复杂数据类型才有方法与对象）

三种基本引用类型：String、Number、Boolean

如：var str = 'wangze';

console.log(str.length);

//上述等价于：

var temp = new String('wangze');

srt = temp;

temp = null; //摧毁临时变量

console.log(str.length);

### 2. 字符串的不可变

定义：指的是字符串里面的值不可变，虽然看上去内容可以改变，但其实是地址变了，内存中新开辟了一个内存空间

因此

### 3. 根据字符 返回位置

字符串.indexOf(任意值)

例：var str = '六月六，七月七';

console.log(str.indexOf('七')); //结果：4

console.log(str.indexOf('七',5)); //结果：6

注：'七'后面的数字是表示从第几个开始查找，这个是非必须的。即str.indexOf('要查找的字符'，起始的位置)

### 4. 根据位置返回字符



#### 4.1 charAt(index)

例：let str = 'wangze'

console.log(str.charAt(4)); //结果：z

//遍历所有字符

for (let i = 0; i < str.length; i++) {

console.log(str.charAt(i));

}

#### 4.2 charCodeAt(index)

目的：判断用户按了哪个键

例：let str = 'wangze'

console.log(str.charCodeAt(0)); //结果：119

#### 4.3 str[index]

h5新增，与charAt相同

例：console.log(str[4]); //结果：z

//遍历所有字符

for (let i = 0; i < str.length; i++) {

console.log(str[i]);

}

#### 特例：求字符串中重复出现次数最多的字母

var str = 'abcoefoxyozzopp';

var obj = {};

for (var i = 0; i < str.length; i++) {

var x = str.charAt(i); //x是str里面的每个字符

if (x in obj) { //x是否在对象obj里面，没有就新建，有就原属性+1

obj[x]++;

} else {

obj[x] = 1;

}

}

//console.log(obj);

var max = 0;

var letter;

for (var k in obj) {

if (obj[k] > max) {

max = obj[k];

letter = k;

}

}

console.log('出现次数最多的字母是：' + letter + ' , ' + '出现的次数为：' + max);

### 5.字符串操作方法



#### 5.1 substr

str.substr(start[, length])

**start**

开始提取字符的位置。如果为负值，则被看作 strLength + start，其中 strLength 为字符串的长度（例如，如果 start 为 -3，则被看作 strLength + (-3)）。

如果 start 为负值且 abs(start) 大于字符串的长度，则 substr 使用 0 作为开始提取的索引。

**length**

可选。提取的字符数。如果 length 为 0 或负值，则 substr 返回一个空字符串。如果忽略 length，则 substr 提取字符，直到字符串末尾。

#### 5.2 substring

str.substring(indexStart[, indexEnd])

substring 提取从 indexStart 到 indexEnd（不包括）之间的字符。特别地：

如果 indexStart 等于 indexEnd，substring 返回一个空字符串。

如果省略 indexEnd，substring 提取字符一直到字符串末尾。

如果任一参数小于 0 或为 NaN，则被当作 0。

如果任一参数大于 stringName.length，则被当作 stringName.length。

如果 indexStart 大于 indexEnd，则 substring 的执行效果就像两个参数调换了一样。

#### 5.3 替换字符

replace( ‘被替换的字符串’ , ‘替换为的字符串’ )

例：var str = 'qangze'

onsole.log(str.replace('q', 'w')); //结果：wangze

注：替换后str不变，即str=‘qangze’

只替换出现的第一个字符（因此常用循环来替换整个字符串）

#### 5.4 字符串转换为数组

split( ‘分隔符’ )

例：var str = 'wang,ze'

console.log(str.split(','));

var str = 'wang&ze'

console.log(str.split('&'));

//结果都是：[ 'wang', 'ze' ]

注：替换后str不变，即str = 'wang,ze'；因此可用str = str.split(',')来给str赋值

#### 5.5 toUpperCase 转大写

#### 5.6 toLowerCase 转小写