JS

2021年10月9日 7:51

```
//使用var 声明的变量，具有变量提升的效果
```

```
// a = 10;
// console.log(a);
// a = 20;
// console.log(a);
//使用var 声明的变量，具有变量提升的效果    function
```

```
//使用var 声明的变量，具有变量提升的效果
var a = 10;
a = a + 1;
console.log(a);
//let 不存在变量提升
let b = 20;
b = b + 1;
console.log(b);
```

```
//调试debugger
var a;
console.log(a);
a =10;
```

有var的声明是会变量提升 但计算机执行代码的时候是会把我们的var a = 10;拆成两步执行，也就是
var a;
a = 10;
这样执行

- **Var 和 let 的区别：**
  - var和let都可以声明一个变量
  - 但是var存在变量提升，let不存在变量提升
- let不可以重复定义， var重复定义会覆盖前面

```
var a = 1;
var b = 1;
var c = 1;
a = a + 1;
console.log('这是a:' + a);
b++;
console.log('这是b:' + b);
++c;
console.log('这是c:' + c);
```

```
var a = 1;
var b = 0;
b = ++a;
//++a 和 a++
console.log(b);
console.log(a);
```

const 声明一个常量
1.声明后不允许更改
2.且声明的时候就必须赋值（实例化）不允许为空

```
> var a = new Date();
< undefined
> a.getTime()
< 1633676727319
> 时间戳
```

```
> NaN == NaN
< false
> undefined == undefined
< true
> null == null
< true
```

```
> var a = 100;
  var b = '20';
< undefined
> a>b
< true
> var a = '100';
  var b = '20';
< undefined
> a>b
< false
>
```

```
> var a =10;
< undefined
> isNaN(a)
< false
> var a  ='a'
< undefined
> isNaN(a)
< true
> var a = NaN;
< undefined
> isNaN(a)
< true
```

```
> var a = 10;
  var b = '1';
  var c =a+b;
  console.log(c)

101
```

```
var a; //声明，定义一个变量
//变量只声明未赋值就是undefined
console.log(a);
```

3.1 数字

和其他编程语言[译注1]不同，JavaScript不区分整数值和浮点数值。JavaScript中的所有数字均用浮点数值表示。JavaScript采用IEEE 754标准[注1]定义的64位浮点格式表示数字，这意味着它能表示的最大值是 $\pm 1.7976931348623157 \times 10^{308}$，最小值是 $\pm 5 \times 10^{-324}$。

按照JavaScript中的数字格式，能够表示的整数范围是从－9 007 199 254 740 992～9 007 199 254 740 992 (即 $-2^{53} \sim 2^{53}$)，包含边界值。如果使用了超过此范围的整数，则无法保证低位数字的精度。然而需要注意的是，JavaScript中实际的操作（比如数组索引，以及第4章讲到的位操作符）则是基于32位整数。

```
//类型转换
var a = 10;
var b = 'a';
var c = a + b; //"10"+"a"
console.log(c);

var d = '5';
var e = a + d; // "10"+"10" "1010"
console.log(e);
var f = a - d; // 10 -5   5
console.log(f);
var g = a - b; Number(a) -Number(b)
console.log(g); //NaN（Not a Number）
```

任何东西与NaN加减乘除 都等于NaN

三元运算符
语法 表达式？ 条件满足执行表达式1: 不满足执行表达式 2:
示例: var age = prompt('请输入你的年龄')
var str = age>18?'成年':'未成年';
console.log(str);

```
var d = '5';
var e = a + d; // "10"+"10"   1010"
console.log(e);
var f = a - d; // 10 -5   5
console.log(f);
var g = a - b;
Number(a) - Number(b)
console.log(g); //NaN（Not a Number）
var h = a / 0;
console.log(h);
```

- **局变量和局部变量：**
  - 局部可以访问全局变量
  - 而全局访问不了局部变量
  - 如果一个变量没有声明直接赋值，
  - 那么这个变量会被系统补充声明，
  - 并且提升为全局变量。
  - 一个变量即被全局定义也被局部定义，在局部里，局部变量优先

```
null == undefined     // 这两值被认为相等
"0" == 0              // 在比较之前字符串转换成数字
0 == false            // 在比较之前布尔值转换成数字
"0" == false          // 在比较之前字符串和布尔值都转换成数字
```

```
//全局变量和局部变量   //可以访问全局变量，全局访问不了局部变量
var a = 10;
var b = 20;

function aa() {
    var c = 30;
    var d = 40;
```

```
if(false){
 console.log(true);
}else{
 console.log(false);
}
false

if(undefined){
 console.log(true);
}else{
 console.log(false);
}
false

if(null){
 console.log(true);
}else{
 console.log(false);
}
false

if(0){
 console.log(true);
}else{
 console.log(false);
}
false

if(-0){
 console.log(true);
```

```
var b = 20;

function aa() {
    var c = 30;
    var d = 40;
    console.log(a);
}
aa();
```

转换里面
如果比较里面 如果有加号 更偏向于字符串
如果比较大小更偏向于数字

```
//全局变量和局部变量   局部可以访问全局变量，全局访问不了局部变量
var a = 10;
var b = 20;
var d;

function aa() {
    var c = 30;
    d = 40;
    var a = 100;
    console.log(a);
}
aa();
console.log(a);
//如果一个变量没有声明直接赋值，那么这个变量会被系统补充声明，并提升为全局变量
//一个变量既被全局定义也被局部定义，在局部里，局部变量优先
```

这里的局部声明的变量这样写的话：里面这个a就是100，而要是把 var 这个去掉的话 外面全局的这个 a 也会变成这个 100


```
//使用var 声明的变量，具有变量提升的效果    function 函数声明提前
//定义一个函数：   function 函数名(){ js表达式，函数体 }
//调用/执行函数：  函数名()
```

++a 和 a++ 的区别：

```
var a = 0;
var b = a++; //var b;   b = a;   a = a+1;
var c = ++a; //var c;   a = a+1; c = a;
var d = a++ +2 + ++a;
```

D 的算法是 从后往前算
先算++a+2=a+1+2=6;
然后再加一个a++ a++是先赋值 在运算
所以 加上a++=6赋值给d
所以最后这个 d =6，a=7;

```
var a = 10;
++a;
var b = ++a + 2;
console.log(b);

var c = 10;
c++;
var d = c++ + 2;
console.log(d);

var e = 10;
var f = e++ + ++e;
console.log(f);
</script>
```

- 使用递增（++）、递减（--）运算符可以快速地对变量的值进行递增和递减操作，它属于一元运算符，只对一个表达式进行操作。
  - 前置递增（递减）运算符：递增和递减运算符写在变量前面，返回的是计算后的结果
  - 后置递增（递减）运算符：递增和递减运算符写在变量后面，返回的是计算前的结果
  - 递增和递减运算符的优先级高于“+”“-”等运算符

```
    //对象的格式：  key:value 访问对象的方法：  1,对象名.属性名 2,对象名['属性名']
var student1 = {
    name: '陈泽锦',
    age: 23,
    sex: '男'
}
var student2 = {
    name: '刘泽宇',
    age: 25,
    sex: '女'
}
```

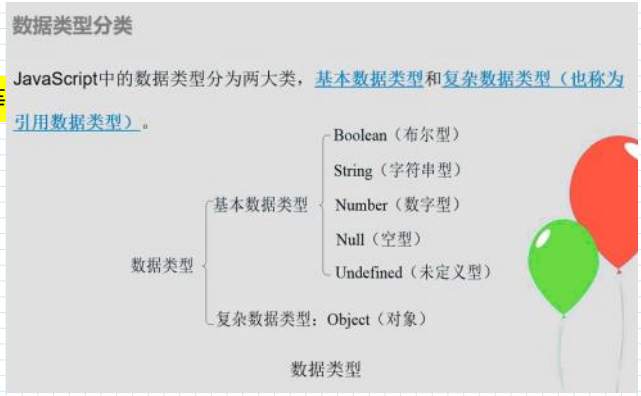
- 数组的格式： var 数组名 =[数组元素,数组元素,.....]
- 获取数组元素   数组名【下标】(计算机计数是从0从左到右开始数的)

```
false

if(-0){
 console.log(true);
}else{
 console.log(false);
}
false

if(''){
 console.log(true);
}else{
 console.log(false);
}
false

if(" "){
 console.log(true);
}else{
 console.log(false);
}
true

if(' '){
 console.log(true);
}else{
 console.log(false);
}
true

if(NaN){
 console.log(true);
}else{
 console.log(false);
}
false
```

in 用于检测对象中是否包含这个属性

= 代表赋值 ==代表判断比较值是否相等 ===判断值和类型是否绝对相等

==	判断相等
!=	判断不等
===	判断恒等
!==	判断非恒等



- 替换变量值：新声明一个新的变量，然后要给哪两个替换就用哪两个变量和新定义的这个变量来回替换形成一个轮回就可以了
- 数据类型的检测：typeof
- 浮点型转换成整数类型：加parseint（）
  - 语法 console.log(parseint(3.9));
- 浮点型转换成小数型：加parsefloat（）
  - 语法 console。Log(parsefloat(3.9))
- 实现一个小功能 输入两个数字 实现加法运算 要求 parsefloat
  - 语法 var mum1=prompt('请输入一个数字');

- 算术运算符的注意事项：
  - 进行四则混合运算时，运算顺序要遵循数学中“先乘除后加减”的原则
  - 在进行取模运算时，运算结果的正负取决于被模数（%左边的数）的符号，与模数（%右边的数）的符号无关
  - 在开发中尽量避免利用浮点数进行运算，因为有可能会因JavaScript的精度导致结果的偏差
  - 使用“+”和“-”可以表示正数或负数

在逻辑与 && 中 有一个结果是假的 结果就是假 两个条件同时满足 执行最后一个 有一个值是 0 结果就是假
在逻辑或 || 中 两个条件同时满足执行第一个
在逻辑非！中 非真即假 （用到的较少）
在JS中除了基本数据类型以外的都是对象，数据是对象，函数是对象，正则表达式是对象

- 数字型的3个特殊值：Infinity和- Infinity 和NaN，举例：
  - Infinity（无穷大）：如Number.MAX_VALUE*2
  - - Infinity（无穷小）：如-Number.MAX_VALUE*2
  - NaN（非数值）： 如'abc' – 100
  - isNaN：用来判断一个变量是否为非数字的类型，返回值为true表示非数字，false表示是数字

比较运算符用于对两个数据进行比较，其结果是一个布尔值，即true或false，常用的比较运算符及用法见下表。

运算符	运算	示例	结果
>	大于	5 > 5	false
<	小于	5 < 5	false
>=	大于或等于	5 >= 5	true
<=	小于或等于	5 <= 5	true
==	等于	5 == 4	false
!=	不等于	5 != 4	true
===	全等	5 === 5	true
!==	不全	5 !== ’5’	true

&&和&有什么区别：
一个&和两个&&都表示并且的意思，都是只有两个条件都成立，结果才为真。
区别是：

1. & 无论第一个条件是否成立，第二个条件都会被判断执行。
2. && 当第一个条件不满足时，第二个条件不会被执行。

a+=1 意思就是：a=1
特殊的：
Var a='a';
a++;
然后结果输出的：nan

js里面可以表示false的值：
1.false
2.undefined
3,。Null
4.±0
5. “”
6.NaN

即使完全相同也不相等的NaN
即使完全不同值相等的±0

表3-2：JavaScript类型转换

值	转换为：			
	字符串	数字	布尔值	对象
undefined	"undefined"	NaN	false	throws TypeError
null	"null"	0	false	throws TypeError
true	"true"	1		new Boolean(true)
false	"false"	0		new Boolean(false)
""(空字符串)		0	false	new String("")
"1.2"(非空,数字)		1.2	true	new String("1.2")
"one"(非空,非数字)		NaN	true	new String("one")
0	"0"		false	new Number(0)
-0	"0"		false	new Number(-0)
NaN	"NaN"		false	new Number(NaN)
Infinity	"Infinity"		true	new Number(Infinity)
-Infinity	"-Infinity"		true	new Number(-Infinity)
1(无穷大,非零)	"1"		true	new Number(1)
{}(任意对象)	参考3.8.3节	参考3.8.3节	true	
[](任意数组)	""	0	true	

- switch判断
  - case匹配条件的意思
  - break结束的意思

```
var a = 1;
switch (a) {
    case 0:
        console.log('这是0');
        break;
    case 1:
        console.log('这是1');
        break;
    case 2:
        console.log('这是2');
        break;

    default:
        break;
}
```

所有基本数据类型都不会往堆里面存储 只会往栈里面存
而所有复杂类型的

{}(任意对象)	参考3.8.3节	参考3.8.3节	true
[](任意数组)	""	0	true
[9](1个数字元素)	"9"	9	true
['a'](其他数组)	使用join()方法	NaN	true
function(){}(任意函数)	参考3.8.3节	NaN	true

所有基本数据类型都不会往堆里面存储　只会往栈里面存
而所有复杂类型的

Push 就是往数组末尾追加元素 数组名.push(新加的元素);
concat 是合并某几个数组

表4-1：JavaScript 运算符

运算符	操作	A	N	类型
++	前/后增量	R	1	lval[①]→num
--	前/后减量	R	1	lval→num
-	求反	R	1	num→num
+	转换为数字	R	1	num→num
~	按位求反	R	1	int→int
!	逻辑非	R	1	bool→bool
delete	删除属性	R	1	lval→bool
typeof	检测操作数类型	R	1	any→str
void	返回undefined值	R	1	any→undef
*、/、%	乘，除，求余	L	2	num,num→num
+、-	加，减	L	2	num,num→num
+	字符串连接	L	2	str,str→str
<<	左移位	L	2	int,int→int
>>	有符号右移	L	2	int,int→int
>>>	无符号右移	L	2	int,int→int
<、<=、>、>=	比较数字顺序	L	2	num,num→bool
<、<=、>、>=	比较在字母表中的顺序	L	2	str,str→bool
instanceof	测试对象类	L	2	obj,func→bool
in	测试属性是否存在	L	2	str,obj→bool
==	判断相等	L	2	any,any→bool
!=	判断不等	L	2	any,any→bool
===	判断恒等	L	2	any,any→bool
!==	判断非恒等	L	2	any,any→bool
&	按位与	L	2	int,int→int
^	按位异或	L	2	int,int→int
|	按位或	L	2	int,int→int
&&	逻辑与	L	2	any,any→any
||	逻辑或	L	2	any,any→any
?:	条件运算符	R	3	bool,any,any→any
=	变量赋值或对象属性赋值	R	2	lval,any→any
*=、/=、%=、+=、-=、&=、^=、|=、<<=、>>=、>>>=	运算且赋值	R	2	lval,any→any
,	忽略第一个操作数，返回第二个操作数	L	2	any,any→any

① lval是left-value的简写，意思是“左值”。

使用for循环用 '*' 完成在控制台打印直角三角形

```
var output = '';
for (var j = 0; j < 5; j++) {
    for (var i = 0; i < j + 1; i++) {
        output += '*'
    }
    output += '\n';
}
console.log(output);
```

使用for循环用 '*' 完成在控制台打印等腰三角形

```
var output = "";
for (var j = 0; j < 5; j++) {
    for (var m = 0; m < 4 - j; m++) {
        output += ' '
    }
    for (var i = 0; i < 2 * j + 1; i++) {
        output += '*'
    }
    output += '\n'
}
console.log(output);
```

While 和 do while的区别：
while只要条件不满足 它就不回执行
Do while 不管条件满不满足 它都会至少执行一次

```
var a = 0;
// while (a < 10) {
//     console.log(a);
//     a++;
// }
do {
    console.log(a);
    a++;
} while (a > 10);
//do while 不管条件是否满足，至少执行一次
```