表达式与运算符

2020年02月26日, 星期三 14:20

1.运算符

运算符的优先级决定了表达式中运算执行的先后顺序，优先级高的运算符最先被执行。

2.字面量

字面量，就是表示自身的常量

```
12                  //数字12
"hello world"       //字符串
{ x: 1, y: 2 }      //对象字面量
[1, 2, 3, 4]        //数组字面量
```

3.表达式

表达式将产生一个值，用于需要值的地方

能产生值的保留字：undefined、 null、 true、 false

4.函数表达式

```
function add(a, b) {
    return a + b;
}
var add = function(a, b) {
    return a + b;
};
```

函数声明

函数表达式

JavaScript 解析器识别函数声明的条件是以 function 关键字开始，只要在 **function 关键字的前面有任何其他的元素**，就会从函数声明转变为函数表达式。

注意：函数声明时，函数名字不可缺少

函数表达式：

```
> function fn(){}
< undefined
> !function(){}
< false
> (function(){})
< ƒ (){}
```

函数表达式时，推荐使用括号，如上图第三个，因为只有括号不会进行运算。

5.函数调用表达式

无return语句时，返回undefined

6.立即执行表达式

```
(function(a, b) {
    return a + b
})(1, 2);
```

在 function 前面加！、+、 - 甚至是逗号等到都可以起到识别为函数表达式的效果。

在这些运算符中加**括号**是最安全的做法，因为它不会改变函数的返回值。

7.

```
function Student(name, age) {
    this.name = name;
    this.age = age;
}
var stu1 = Student("Lily", 18);
var stu2 = new Student("Lucy", 20);
console.log(typeof stu1);
```

分析代码，在控制台输出结果为（）。

A string　　B undefined

C null　　D object

2.html

答案：BD

stu1无return，因此返回undefined

stu2返回的是一个实例对象，因此是object类型

8.逻辑运算符

逻辑运算符两边的操作数都是布尔类型

对于&&来说， 除了两侧都为真时为真，其他情况都为假（与表达式）

对于||来说， 除了两侧都为假时为假，其他情况都为真（或表达式）

```
console.log(2>1&&4<5);
console.log(true&&(!2));
console.log(false&&("2" == 2));
console.log(false&&false);
```

```
true
false
false
false
```

```
console.log(2>1||4<5);
console.log(true||(!2));
console.log(false||("2" == 2));
console.log(false||false);
```

```
true
true
true
false
```

当逻辑运算符 && 和 || 两侧的操作数不是布尔类型时：

```
console.log(2 && 4);
console.log({x: 2} && {name: "Jack"});
console.log(2 || 4);
console.log({x: 2} || {name: "Jack"});
```

- 首先将左操作数转换成布尔类型
- 对转换后的左操作数进行逻辑判断（true or false）
- 根据短路原则返回原始左操作数或原始右操作数

短路原则：（忽略对右操作数的判断）

对于 &&，转换后的左操作数若为 true，则直接返回原始右操作数，若为 false 则直接返回原始左操作数

对于 ||，转换后的左操作数若为 true，则直接返回原始左操作数，若为 false 则直接返回原始右操作数

```
console.log(2&&4);
console.log(0&&4);
console.log({x:2}&&{name:"Jame"});
console.log(null&&"hello");
console.log({}&&"world");
```

```
4
0
▶ Object {name: "Jame"}
null
world
```

```
console.log(2||4);
console.log(0||4);
console.log({x:2}||{name:"Jame"});
console.log(null||"hello");
console.log({}||"world");
```

```
2
4
▶ Object {x: 2}
hello
▶ Object {}
```

短路原则应用：

- 遵循短路特性，**使用 || 来设置函数参数的默认值**
  - 函数定义时可以给参数指定默认值，调用时若未传参数则该参数的值取它定义时的默认值

```
//定义一个计算圆面积的函数area_of_circle(), 它有两个参数:

//r: 表示圆的半径
//pi: 表示π的值, 如果不传, 则默认3 .14


function area_of_circle(r, pi) {

}
```

- 遵循短路特性，使用 && 防止运行报错

jQuery 动画 - animate() 方法

jQuery animate() 方法用于创建自定义动画。

```
function animate(param, speed, callback) {
    //do something

    callback && callback();
}
```

```
callback && typeof callback === "function" && callback();
```

- 遵循短路特性，使用 && 和 || 可用来实现条件语句

```
var score = 76;
if(score > 90) {
    console.log("优");
} else if(score > 75) {
    console.log("良");
} else if(score > 60) {
    console.log("及格");
} else {
    console.log("不及格");
}
```

1. 减少了代码的量
2. 增加了程序的执行效率

```
console.log((score>90&&"优")||(score>75&&"良")||(score>60&&"及格")||"不及格");
```

注：小括号优先级最高

9.相等运算符

JavaScript 有两种比较方式：严格比较运算符和宽松比较运算符。

严格相等运算符（===）

仅当两个操作数的类型相同且值相等为 true

宽松相等运算符（==）

在进行比较之前，将两个操作数转换成相同的类型

10.递增递减

递增 (++)

递增运算符为其操作数增加1，返回一个数值

如果使用后置（postfix），即运算符位于操作数的后面（如 **x++**），那么将会在**递增前返回数值**

```
> var a=1
< undefined
> a++;
< 1
> a
< 2
```

如果使用前置（prefix），即运算符位于操作数的前面（如 **++x**），那么将会在**递增后返回数值**

```
> var a=1;
< undefined
> ++a;
< 2
> a
< 2
```

eg：

```
> var x = 1;
  x = x++;
  console.log(x);
  1
```

递减（--）

递减运算符为其操作数减去1，返回一个数值

11.赋值运算符

基于右值（right operand）的值，给左值（left operand）赋值

左值：“=”运算符的左操作数

右值：“=”运算符的右操作数

```
name = "Lily"; //将变量name设置为"Lily"
person.age = 20; //将对象person的age属性设置为20
```

赋值表达式的返回值为右操作数

```
> var a = 2;
< undefined
> a = 2;
< 2
```

有var关键字时，是声明语句，没有返回值

练习1：

```
function fun() {
    var a = b = 5;
}
fun();
console.log(a, typeof a);
console.log(b, typeof b);
```

var a = b = 5；等价于 var a； b=5； a=5；

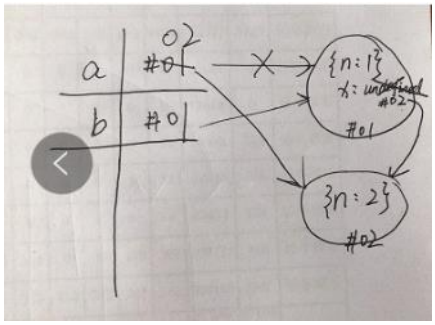
输出变量a时报错，但是console.log是在函数体外访问的。而a是在函数中通过var声明的，所以a是fun（）函数中的局部变量；b不是通过var关键字声明的，所以b是全局变量。

a是在函数内用var局部变量，b是全局变量。因此输出a时，访问不到a，作用域已经失效了，所以报错；b是全局变量，在任何地方都能访问到，所以输出结果为5，number。

练习2：

```
var a = { n: 1 };
var b = a;
a.x = a = { n: 2 };
console.log(a.x);
console.log(b);
```

```
var a = { n: 1 };
var b = a;
a.x = a = { n: 2 };
console.log(a.x);
console.log(b);
undefined
{n: 1, x: {…}}
```




```
var a = { n: 1 };
a.x = a = { n: 2 };
console.log(a.x);
```

输出结果为：undefined

复合赋值：

- x += y ⟹ x = x + y
- x -= y ⟹ x = x - y
- x *= y ⟹ x = x * y
- x /= y ⟹ x = x / y
- x %= y ⟹ x = x % y

```
var x = 2;

x += x++;
console.log(x);//?

x *= --x;
console.log(x);//?
```

12.逗号操作符

对它的每个操作数求值（从左到右），并返回最后一个操作数的值

```
var x = 1;

x = (x++, x);
console.log(x); //?

x = (2, 3);
console.log(x); //?
```