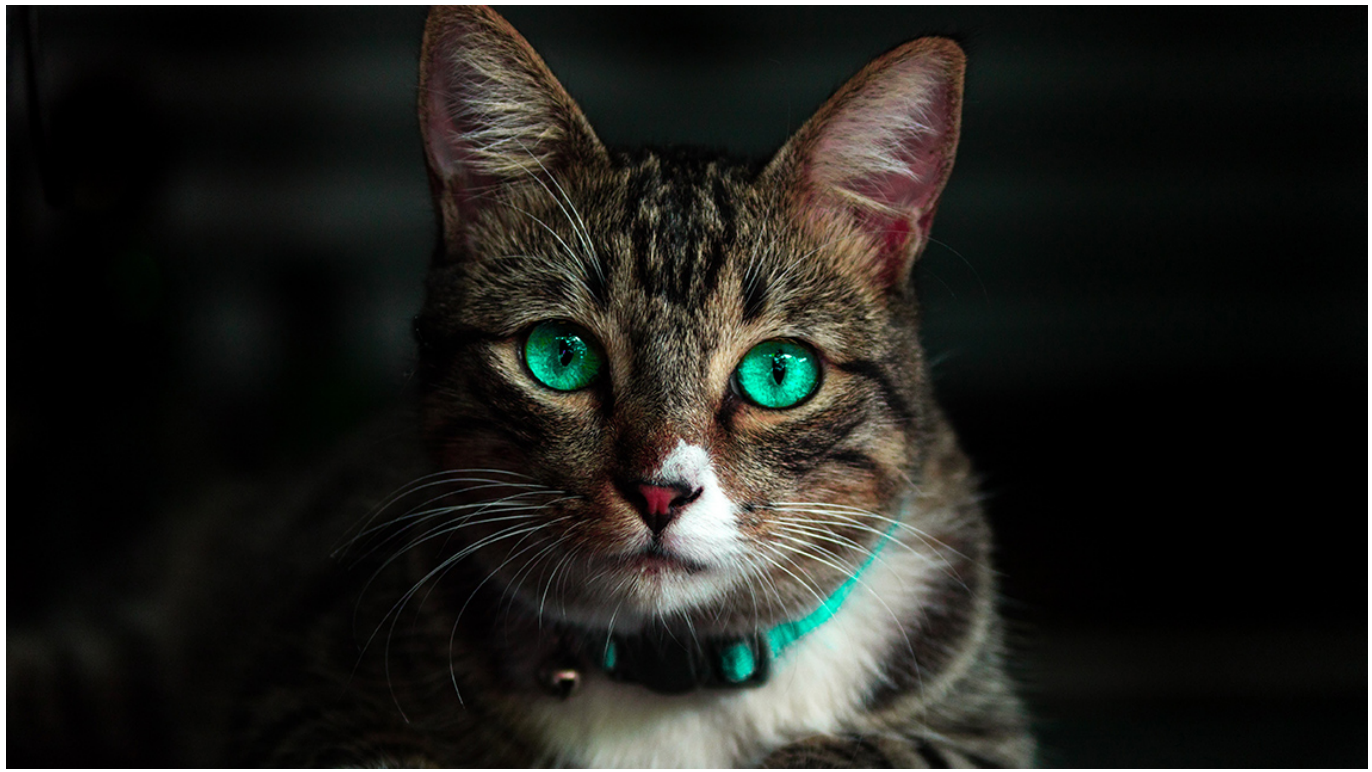
32 | JavaScript语法（四）：新加入的**运算符，哪里有些不一样呢？ | 极客时间

winter 2019-04-06



你好，我是 winter。

上一节课我们已经给你介绍了表达式的一些结构，其中关于赋值表达式，我们讲完了它的左边部分，而留下了它右边部分，那么，我们这节课一起来详细讲解。

在一些通用的计算机语言设计理论中，能够出现在赋值表达式右边的叫做：右值表达式（RightHandSideExpression），而在 JavaScript 标准中，规定了在等号右边表达式叫做条件表达式（ConditionalExpression），不过，在 JavaScript 标准中，从未出现过右值表达式字样。

JavaScript 标准也规定了左值表达式同时都是条件表达式（也就是右值表达式），此外，左值表达式也可以通过跟一定的运算符组合，逐级构成更复杂的结构，直到成为右值表达式。

关于这块的知识，我们有时会看到按照运算符来组织的讲解形式。

这样讲解形式是因为：对运算符来说的“优先级”，如果从我们语法的角度来看，那就是“表达式的结构”。讲“乘法运算的优先级高于加法”，从语法的角度看就是“乘法表达式和加号运算符构成加法表达式”。

对于右值表达式来说，我们可以理解为以左值表达式为最小单位开始构成的，接下来我们就来看看左值表达式是如何一步步构成更为复杂的语法结构。

更新表达式 UpdateExpression

左值表达式搭配 ++ -- 运算符，可以形成更新表达式。

```
-- a;
++ a;
a --
a ++
```


更新表达式会改变一个左值表达式的值。分为前后自增，前后自减一共四种。

我们要注意一下，这里在 ES2018 中，跟早期版本有所不同，前后自增自减运算被放到了同一优先级。

一元运算表达式 UnaryExpression

更新表达式搭配一元运算符，可以形成一元运算表达式，我们看下例子：

```
delete a.b;

void a;
```

```
typeof a;

- a;

~ a;

! a;

await a;
```

□复制代码

他的特点就是一个更新表达式搭配了一个一元运算符。

乘方表达式 ExponentiationExpression

乘方表达式也是由更新表达式构成的。它使用**号。

```
++i ** 30
2 ** 30 // 正确
-2 ** 30 // 报错
```

□复制代码

我们看一下例子，-2 这样的一元运算表达式，是不可以放入乘方表达式的，如果需要表达类似的逻辑，必须加括号。

这里我们需要注意一下结合性，** 运算是右结合的，这跟其它正常的运算符（也就是左结合运算符）都不一样。

我们来看一个例子。

```
4 ** 3 ** 2
```

□复制代码

事实上，它是这样被运算的：

```
4 ** (3 ** 2)
□复制代码
```

而不是这样被运算的：

```
(4 ** 3) ** 2
□复制代码
```

我们来实际在代码中执行一下试试。最终结果是 262144， 而不是 4096。

在最后加一句它的特殊。

乘法表达式 MultiplicativeExpression

到这里，我们进入了比较熟悉的表达式类型，乘方表达式可以构成乘法表达式，用乘号或者除号、取余符号连接就可以了，我们看看例子：

```
x * 2;
□复制代码
```

乘法表达式有三种运算符：

```
*
/
%
□复制代码
```

它们分别表示乘、除和取余。它们的优先级是一样的，所以统一放在乘法运算表达式中。

加法表达式 AdditiveExpression

加法表达式是由乘法表达式用加号或者减号连接构成的。我们看下例子:

```
a + b * c
□复制代码
```

加法表达式有加号和减号两种运算符。

```
+
-
□复制代码
```

这就是我们小学学的加法和减法的意思了。不过要注意，加号还能表示字符串连接，这也比较符合一般的直觉。

移位表达式 ShiftExpression

移位表达式由加法表达式构成，移位是一种位运算，分成三种：

```
<< 向左移位
>> 向右移位
>>> 无符号向右移位
□复制代码
```

移位运算把操作数看做二进制表示的整数，然后移动特定位数。所以左移 n 位相当于乘以 2 的 n 次方，右移 n 位相当于除以 2 取整 n 次。

普通移位会保持正负数。无符号移位会把减号视为符号位 1，同时参与移位：

```
-1 >>> 1
□复制代码
```

这个会得到 2147483647，也就是 2 的 31 次方，跟负数的二进制表示法相关，这里就不详细讲解了。

在 JavaScript 中，二进制操作整数并不能提高性能，移位运算这里也仅仅作为一种数学运算存在，这些运算存在的意义也仅仅是照顾 C 系语言用户的习惯了。

关系表达式 RelationalExpression

移位表达式可以构成关系表达式，这里的关系表达式就是大于、小于、大于等于、小于等于等运算符号连接，统称为关系运算。

```
<=

>=

<

>

instanceof

in
```

复制代码

需要注意，这里的 <= 和 >= 关系运算，完全是针对数字的，所以 <= 并不等价于 < 或 ==。例如

```
null <= undefined
//false
null == undefined
//true
```

请大家务必不要用数学上的定义去理解这些运算符。

相等表达式 EqualityExpression

在语法上，相等表达式是由关系表达式用相等比较运算符（如 ==）连接构成的。所以我们可以像下面这段代码一样使用，而不需要加括号。

```
a instanceof "object" == true
□复制代码
```

相等表达式由四种运算符和关系表达式构成，我们来看一下运算符：

- ==
- !=
- ===
- !==

相等表达式又包含一个 JavaScript 中著名的设计失误，那就是 == 的行为。

一些编程规范甚至要求完全避免使用 == 运算，我觉得这样规定是比较合理的，但是这里我还是尽量解释一下 == 的行为。

虽然标准中写的==十分复杂，但是归根结底，类型不同的变量比较时==运算只有三条规则：

- undefined 与 null 相等；
- 字符串和 bool 都转为数字再比较；
- 对象转换成 primitive 类型再比较。

这样我们就可以理解一些不太符合直觉的例子了，比如：

- `false == '0'` true
- `true == 'true'` false
- `[] == 0` true
- `[] == false` true
- `new Boolean('false') == false` false

这里不太符合直觉的有两点：

一个是即使字符串与 boolean 比较，也都要转换成数字；
二是对象如果转换成了 primitive 类型跟等号另一边类型恰好相同，则不需要转换成数字。

此外，== 的行为也经常跟 if 的行为（转换为 boolean）混淆。总之，我建议，仅在确认 == 发生在 Number 和 String 类型之间时使用，比如：

```
document.getElementsByTagName('input')[0].value == 100
□复制代码
```

在这个例子中，等号左边必然是 string，右边的直接量必然是 number，这样使用 == 就没有问题了。

位运算表达式

位运算表达式含有三种：

- 按位与表达式 BitwiseANDExpression
- 按位异或表达式 BitwiseANDExpression
- 按位或表达式 BitwiseORExpression。

位运算表达式关系比较紧密，我们这里放到一起来讲。

按位与表达式由按位与运算符（&）连接按位异或表达式构成，按位与表达式把操作数视为二进制整数，然后把两个操作数按位做与运算。

按位异或表达式由按位异或运算符（^）连接按位与表达式构成，按位异或表达式把操作数视为二进制整数，然后把两个操作数按位做异或运算。异或两位相同时得 0，两位不同时得 1。

异或运算有个特征，那就是两次异或运算相当于取消。所以有一个异或运算的小技巧，就是用异或运算来交换两个整数的值。

```
let a = 102, b = 324;

a = a ^ b;
b = a ^ b;
```

```
a = a ^ b;


console.log(a, b);
```

□复制代码

按位或表达式由按位或运算符（|）连接相等表达式构成，按位或表达式把操作数视为二进制整数，然后把两个操作数按位做或运算。

按位或运算常常被用在一种叫做 Bitmask 的技术上。Bitmask 相当于使用一个整数来当做多个布尔型变量，现在已经不太提倡了。不过一些比较老的 API 还是会这样设计，比如我们在 DOM 课程中，提到过的 Iterator API，我们看下例子：

```
var iterator = document.createNodeIterator(document.body, NodeFilter.SHOW_TEXT | NodeFilter.SHOW_COMMENT, null, false);

var node;

while(node = iterator.nextNode())

{

    console.log(node);

}
```

□复制代码

这里的第二个参数就是使用了 Bitmask 技术，所以必须配合位运算表达式才能方便地传参。

逻辑与表达式和逻辑或表达式

逻辑与表达式由按位或表达式经过逻辑与运算符连接构成，逻辑或表达式则由逻辑与表达式经逻辑或运算符连接构成。

这里需要注意的是，这两种表达式都不会做类型转换，所以尽管是逻辑运算，但是最终的结果可能是其它类型。

比如：

这句将会得到结果 1。

```
false && undefined;
□复制代码
```

这句将会得到 undefined。

另外还有一点，就是逻辑表达式具有短路的特性，例如：

```
true || foo();
□复制代码
```

这里的 foo 将不会被执行，这种 zhongduan 后面表达式执行的特性就叫做短路。

条件表达式 ConditionalExpression

条件表达式由逻辑或表达式和条件运算符构成，条件运算符又称三目运算符，它有三个部分，由两个运算符?和:配合使用。

```
condition ? branch1 : branch2
□复制代码
```

这里需要注意，条件表达式也像逻辑表达式一样，可能忽略后面表达式的计算。这一点跟 C 语言的条件表达式是不一样的。

条件表达式实际上就是 JavaScript 中的右值表达式了 RightHandSideExpression，是可以放到赋值运算后面的表达式。

总结

今天我们讲解了表达式的右边部分，讲到了包括更新表达式、一元运算表达式、乘方表达式、乘法表达式、移位表达式等 14 种表达式。至此为止，我们已经讲全了表达式。你如果有不熟悉的地方，可以随时回头查阅。

留一个小任务，我们试着总结下 JavaScript 中所有的运算符优先级和结合性。例如：

运算符	优先级	结合性
`*`	1	左
`/`	1	左
`+`	2	左
`-`	2	左

bd2star