JavaScript极速入门 —— 操作符

主讲人与课程设计：耕耕

本节课提纲

1. 一元操作符
2. 位操作符（了解即可）
3. 布尔操作符
4. 乘性操作符
5. 加性操作符
6. 关系操作符
7. 相等操作符
8. 条件操作符
9. 赋值操作符
10. 逗号操作符

条件操作符（小白们）

条件操作符应该算是ECMAScript 中最灵活的一种操作符

```
var max;
if (num1 > num2) {
    max = num1;
} else {
    max = num2;
}
```

```
var max = (num1 > num2) ? num1 : num2;
```

赋值操作符（小白们）

简单的赋值操作符由等于号（=）表示，其作用就是把右侧的值赋给左侧的变量

```
var num = 10;
num = num + 10;
```

```
var num = 10;
num += 10;
```

还有：
乘/赋值（*=）；
除/赋值（/=）；
…

逗号操作符（小白们）

使用逗号操作符可以在一条语句中执行多个操作

```
var num1=1, num2=2, num3=3;
```

```
var num1 = 1;
var num2 = 2;
var num3 = 3;
```

逗号操作符多用于声明多个变量；但除此之外，逗号操作符还可以用于赋值。在用于赋值时，逗号操作符总会返回表达式中的最后一项

```
var num = (5, 1, 4, 8, 0); // num 的值为0
```

一元操作符（小白们）

只能操作**一个值**的操作符叫做一元操作符

如++，--，+，-

```
var num1 = 10;
console.log(num1++);

var num2 = 10;
console.log(num1--);

var num3 = 10;
console.log(++num1);

var num4 = 10;
console.log(--num1);

console.log(+1);
console.log(-1);
```

```
var num5 = 10;
var test1 = num5++ + 1;

var num6 = 10;
var test2 = ++num6 + 1;
```

一元操作符（大神们）

所有这些操作符对**任何值**都适用，也就是它们不仅适用于整数，还可以用于字符串、布尔值、浮点数值和对象

转换规则(上一节课内容)：先进行Number()转换，然后在进行运算

```
var s1 = "2";
var s2 = "z";
var b = false;
var f = 1.1;
var o = {
    valueOf: function() {
        return -1;
    }
};
s1++; // 值变成数值3
s2++; // 值变成NaN
b++; // 值变成数值1
f--; // 值变成0.10000000000000009（由于浮点舍入错误所致）
o--; // 值变成数值-2
```

Number 类型(转化规则，上一节课内容)

如果是Boolean 值，**true** 和**false** 将分别被转换为**1** 和**0**。
　　如果是**数字值**，只是简单的传入和**返回**。
　　如果是**null** 值，返回**0**。
　　如果是**undefined**，返回**NaN**。
　　如果是**[]**，返回**0**。
　　如果是**[9]**，返回**9**。
　　如果是**function(){}**，返回**NaN**。
　　如果是字符串，遵循下列规则:
　　如果字符串中只包含数字（包括前面带正号或负号的情况），则将其转换为十进制数值，即"1"会变成1，"123"会变成123，而"011"会变成11（注意：前导的零被忽略了）；
　　如果字符串中包含有效的浮点格式，如"1.1"，则将其转换为对应的浮点数值（同样，也会忽略前导零）；
　　如果字符串中包含有效的十六进制格式，例如"0xf"，则将其转换为相同大小的十进制整数值;
　　如果字符串是空的（不包含任何字符），则将其转换为0;
　　如果字符串中包含除上述格式之外的字符，或者其他情况的数组，则将其转换为**NaN**。
　　如果是对象，则调用对象的**valueOf**()方法，然后依照前面的规则转换返回的值。如果没有valueOf()，调用对象的**toString**()方法，然后再次依照前面的规则转换返回的字符串值

Number 类型(对象类型转换，**上节课内容**)

1. 如果对象没有valueOf()和toString()，返回NaN

2. 看两者是否其中一个返回原始值，如果是，则返回并转换，否则报错， 先valueOf()后toString()

就像一对兄弟，如果大哥在，要等大哥享用完，弟弟才能用；
兄弟可以独立存在，也可以一起存在，只有弟弟有错

位操作符（了解）

位操作符用于在最基本的层次上，即按内存中表示数值的位来操作数值

按位与（&）
按位或（|）
按位异或（^）
按位非（~），大神们
左移（<<）
右移（>>）
无符号右移（>>>）

```
var num1 = 25; //     二进制00000000000000000000000011001
var num2 = ~num1; // 二进制11111111111111111111111100110
alert(num2); // -26

// 上面代码等价于这个
var num1 = 25;
var num2 = -num1 - 1;

var test = "256";
console.log(~~test); // 转化为整数了
```

布尔操作符（小白们）

布尔操作符一共有3 个：非（NOT）、与（AND）和或（OR）

如 !，&&，||

! 布尔值的反值，如果一个值是true，非操作就是false

```
alert(!false); // true
```

&&左右两边都为true的时候，才是true

```
var result = true && false; // false
```

||左右两边都为false的时候，才是false

```
var result = true && false; // true
```

布尔操作符（大神们）

所有这些操作符对**任何值**都适用，也就是它们不仅适用于整数，还可以用于字符串、布尔值、浮点数值和对象

转换规则(上一节课内容)：先进行Boolean()转换，然后在进行运算

拓展：
&& 碰到假的直接返回那个假的表达式的结果，如果全都为真，返回最后一个表达式的结果
|| 碰到真的直接返回那个真的表达式的结果，如果全都为假，返回最后一个表达式的结果

```
function stop() {
    console.log(111);
}

var found = true;
var result = found && stop();
```

Boolean 类型（逻辑类型，**上节课内容**）

boolean 类型仅包含两个值：**true** 和 **false**。

```
var nameFieldChecked = true; // yes, name field is checked
var ageFieldChecked = false; // no, age field is not checked
```

拓展：任何值可以通过Boolean函数转化为boolean值

```
var message = "Hello world!";
var messageAsBoolean = Boolean(message); //true
```

结论：**0, ‘’, null, undefined, NaN, false** 这6个值全部转化为false，其他都为true

```
var message = "Hello world!";
if (message) {
    alert("Value is true");
}
```

乘性操作符（小白们）

乘法操作符由一个星号（*）表示，用于计算两个数值的乘积

```
var result = 34 * 56;
```

除法操作符由一个斜线符号（/）表示，执行第二个操作数除第一个操作数的计算

```
var result = 66 / 11;
```

求模（余数）操作符由一个百分号（%）表示

```
var result = 26 % 5; // 等于1
```

幂运算符 ** 是最近被加入到 JavaScript 中的

```
alert(2 ** 4); // 16 (2 * 2 * 2 * 2)
```

注意：如果有一个操作数不是数值，则在后台调用Number()将其转换为数值

加性操作符（小白们）

加法操作符由一个星号（+）表示

注意规则:
如果两个操作符都是数值，执行常规的加法计算

```
var result = 1 + 2;
```

如果**有一个**操作数是字符串，把另外一个转换为字符串处理，两个字符串拼接起来

```
var result2 = 5 + "5"; // 55
```

如果两个操作数既不是字符串，也不是对象，两个操作数**Number()**转换，做加法运算

```
var result3 = ture + true // 2
```

拖展：如果两个操作数其中一个是对象，先执行valueOf()方法，如果没有就执行toString()方法(计算)；两者转换出来的为字符串的话，把另一个转换字符串，否则就是转换数字在计算**（小白们不用掌握）**

```
var a = 1 + {}; //"1[object Object]"
```

加性操作符（练习题）

```
var a1 = 1 + 2; // 3
var a2 = '1' + '2'; // 12
var a3 = 1 + null;
var a4 = [] + {};
var a5 = {} + [];
var a6 = undefined + {};
var a7 = null + {};
```

加性操作符（小白们）

减法操作符由一个星号（-）表示

注意规则:
如果两个操作符都是数值，执行常规的减法计算

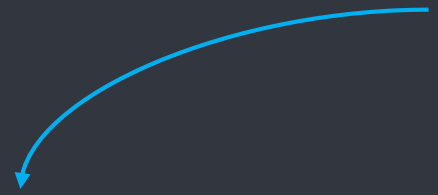
```
var result = 1 - 2;
```

如果两个操作符有不是数值的（除了对象），先Number()转换;
如果有对象先valueOf()，在toString()，并得到的结果转换数字

对象搞事，有口诀

凡是碰到二元操作符（除了Boolean）左右两边有对象，需要先执行对象**valueOf()**方法，再执行**toString()**方法

valueOf()和toString()都可以返回原始值，可以返回对象类型值，**valueOf()优先级高**



valueOf() 如果返回原始值，则直接返回转化，否则执行toString()方法，对象如果转换不了原始值就报错!

关系操作符（小白们）

小于（<）、大于（>）、小于等于（<=）和大于等于（>=）这几个关系操作符用于对两个值进行比较，比较的规则与我们在数学课上所学的一样。这几个操作符都返回一个布尔值

```
var result1 = 5 > 3; //true
var result2 = 5 < 3; //false
```

关系操作符（大神们）

如果比较一些特殊的数据类型呢?

规则:

如果两个操作数都是数值，则执行数值比较

如果两个操作数都是字符串，则比较两个字符串对应的字符编码值

如果两个都不是对象，没有字符串，则执行数值比较

如果是对象，走口诀，先valueOf()，再toString()。转换出来如果都是字符串，按照字符串比较，如果有一个不是字符串，转换数字比较!

```
var t1 = undefined > 1;
var t2 = null < 1;
var t3 = {} < 1;
var t4 = undefined >= null;
```

相等操作符（小白们）

ECMAScript 中的相等操作符由两个等于号（==）表示，如果两个操作数相等，则返回true。而不相等操作符由叹号后跟等于号（!=）表示，如果两个操作数不相等，则返回true。这两个操作符都会先转换操作数（通常称为强制转型），然后再比较它们的相等性

如果是两对象，比的对象的地址，两个对象是否是同一个地址
如果其中一个是对象，valueOf()和toString()转化为原始值进行比较
字符串，数字，布尔值比较，转化数字后比较
其他均不相等

特殊:

null 和undefined 是相等的, null == undefined //true

NaN跟谁都过不去，全是false，自己都不认识自己

相等操作符（小白们）

除了在比较之前不转换操作数之外，全等和不全等操作符与相等和不相等操作符没有什么区别。全等操作符由3 个等于号（===）表示，它只在两个操作数未经转换就相等的情况下返回true

```
var result1 = ("55" == 55); //true，因为转换后相等
var result2 = ("55" === 55); //false，因为不同的数据类型不相等
```

null === undefined //false