# 二、运算符

[运算符？](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators)

运算符（Operators，也翻译为操作符），是发起运算的最简单形式。

运算符的分类见仁见智，我们的课程对运算符进行如下分类：

数学运算符(Arithmetic operators)

比较运算符(Comparison operators)

逻辑运算符(Logical operators)

赋值运算符(Assignment operators)

按位运算符(Bitwise operators)

条件 (三元) 运算符(Conditional operator)

## 2.1 数学运算符

+ - \* / % ()­

■ 这里面要知道%的意思，我们昨天已经算过很多了，不赘述。

■ 要知道计算顺序，先乘除求余，然后加减，我们昨天已经算过很多了，不赘述。

■ 隐式转换，所有带有字符串的运算都会尽可能的转为数字进行计算，加号比较特殊。

数学运算符的正统，是number和number的数学运算，结果是number。出于面试的考虑，有一些奇奇怪怪的数学运算：

数学运算中，**只有纯字符串、布尔值、null能够帮你进行隐式转换**：

ECMAScript规范对于运算的规范？ <http://www.ecma-international.org/ecma-262/6.0/>

JavaScript秘密花园的翻译？http://bonsaiden.github.io/JavaScript-Garden/zh/#types.casting

|  |
| --- |
| 1. //隐式转换：就是没有写parseInt()、parseFloat()自己帮你转格式 3. console.log(3 \* **"8"**); //24 4. console.log(**"3"** \* **"8"**); //24 5. console.log(**"48"** / "2"); //24 6. console.log(**"24"** % 55); //24 |

|  |
| --- |
| 1. console.log(3 \* **null**); //0 隐式转换的时候**null将被转为0** 2. console.log(3 \* **false**); //0 隐式转换的时候**false将被转为0** 3. console.log(3 \* **true**); //3 隐式转换的时候**true将被转为1** |

**不纯的字符串和undefined是不能帮你进行隐式转换的**，结果都是NaN

|  |
| --- |
| 1. console.log(3 \* **"8天"**); //NaN 数学运算中，不纯的字符串没法隐式转换 2. console.log(3 \* **undefined**); //NaN 数学运算中，undefined不能进行隐式转换 |

加法比较特殊，因为+同时是加法和连字符的符号，所以加法在面对字符串的时候没有隐式转换

|  |
| --- |
| 1. //加法没有隐式转换 2. console.log(3 + **"8"**); //38 3. console.log(3 + **undefined**); //NaN 4. console.log(3 + **null**); //3 5. console.log(3 + **false**); //3 6. console.log(3 + **true**); //4 |

总结：

无论哪种运算，只要出现了undefined参与运算，结果都是NaN。

然后"4"、false、true、null都能进行隐式转换。

加号比较特殊，面对"4"没有隐式转换的

特殊数值的计算，就是NaN、Infinity参与的运算，我们看《高3》来学习。

我们就举几个例子：

|  |
| --- |
| 1. Infinity + 1000 //Infinity 2. Infinity - 1000 //Infinity 3. Infinity / 1000 //Infinity 4. Infinity - Infinity //NaN 5. Infinity /Infinity //NaN 6. Infinity \* Infinity //Infinity 7. 0 / 0 //NaN 8. 6 / 0 //Infinity 9. NaN / 8 //NaN |

## 关系运算符

[比较操作符？](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Comparison_Operators" \l "Equality)

> 大于

< 小于

>= 大于等于

<= 小于等于

== 等于

!= 不等于

=== 全等于

!== 不全等

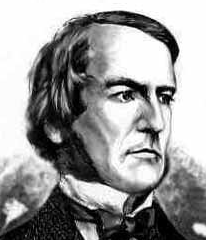
关系运算符的正统，number和number进行数学运算，得到的答案boolean。

[Boolean?](https://developer.mozilla.org/zh-CN/docs/Glossary/Boolean)

|  |
| --- |
| 1. console.log(8 > 5); 2. console.log(7 < 4); |

 关系运算的结果，是boolean类型的。

true和false叫做布尔值。boolean类型，boolean类型只有两个值，就是true和false。表示真、假。

 [布尔](http://baike.baidu.com/link?url=uH7zOTr9WwkgiGRQpPihS5m_TrqIFI43I94bIIQidge6CuFuCNecu0NNFC4g2b6jE9q5QuWevY9VVqwR7m3MOWveexE-D0PRkHKU7pzAbOIJ4Q0gUycYESGKNyzBtb9y4xerHZGkf0XIsxgRxREcJhgp6MOU_5M58yDxdUGf4S4a4xXpV1m6zrARGbxqCe9O)？

== 我们叫做“相等判断”，它会帮你进行一个隐式转换，尽可能的得到true的答案：

|  |
| --- |
| 1. console.log(5 == "5"); //true |

===我们叫做“全等判断”，不仅仅比较数值是否相等，还比较类型是否相等

[相等和全等？](http://dorey.github.io/JavaScript-Equality-Table/)

!= 是==的反面，如果==运算是true，那么!=就是false

!==是===的反面，如果===运算是true，那么!==就是false

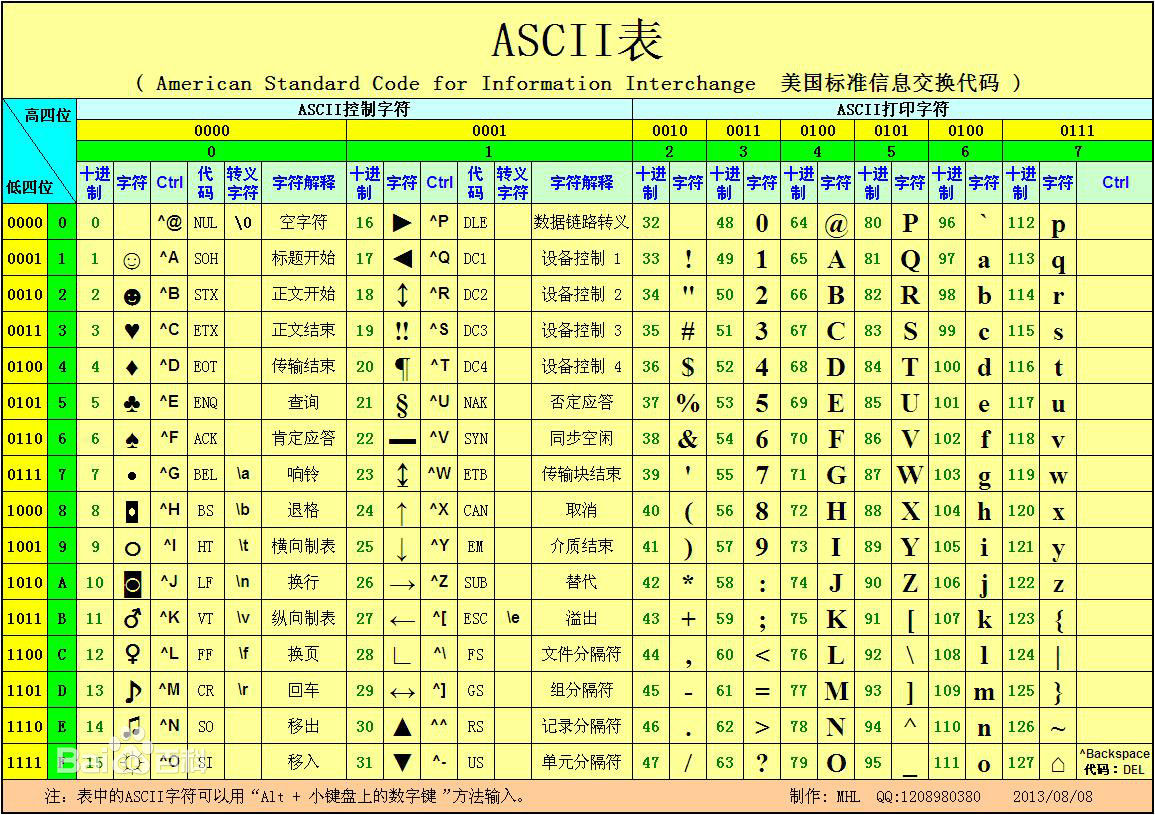
|  |
| --- |
| 1. console.log(5 != "5"); //false ， 脑子要反着想一下，5=="5"结果是t，所以就是f 2. console.log(5 !== "5"); //true ， 脑子要反着想一下，5==="5"结果是f，所以就是t |

正统的运算讲完了，number和number进行关系运算，结果是boolean。

现在我们讲一丢丢不正统的关系运算，为了面试，划一下重点，更多的自己通过《高三》P50、P51、P52：

1. string 和 string 也能够进行关系运算，比较的就是字符编码顺序。

[字符串和编码？](http://www.liaoxuefeng.com/wiki/0014316089557264a6b348958f449949df42a6d3a2e542c000/001431664106267f12e9bef7ee14cf6a8776a479bdec9b9000)

[](http://www.liaoxuefeng.com/wiki/0014316089557264a6b348958f449949df42a6d3a2e542c000/001431664106267f12e9bef7ee14cf6a8776a479bdec9b9000)

字符编码顺序，就是数字、大写字母、小写字母

|  |
| --- |
| 1. "a" < "b" //true 2. "A" < "B" //true 3. "A" < "a" // true ，大写字母在字符集里面是在小写字母前面 4. "1" < "A" //true ，数字在字母前端 5. "blank" < "blue" //true 因为一位一位比，直到比出大小 6. "25678" < "3" //true 因为是string和string比，比的是字符编码顺序 |

② 与数字进行关系运算时，纯数字字符串被转为数字，null转换为0，true转换转为1， false转换为0

null不能进行和0的相等判定

|  |
| --- |
| 1. null < 0.00001 //true 2. null > -0.0001 //true 3. null == 0 //false 具体原因，我们后面讲解Object的时候介绍 4. false == 0 //true 5. true == 1 //true |

③ NaN不等于自己，不全等于自己

|  |
| --- |
| 1. NaN == NaN //false 2. NaN === NaN //false 3. NaN != NaN //true 4. NaN !== NaN //true |

④ string 和 number比，string会被隐式转换为number

|  |
| --- |
| 1. "25" < 3 //false |

需要注意的是，我们已经了解了一些不正统的运算，所以不要出洋相，不能连续使用关系运算符！！

比如我们想验证3大于2,2大于1：

表达式：

|  |
| --- |
| 1. 3 > 2 > 1 |

的值是多少？

解：原式=(3>2) >1 = true > 1 = false (因为true会被当做1来与1进行比较)

也就是说，不能连续使用关系运算符！！因为一旦连续使用了，实际上还是从左至右计算，所以就有上一步的boolean参与了下一步的运算。

**剧透一下，如果想要使用连续关系运算，1<2 && 2<3**

## 2.3 逻辑运算符

[逻辑运算符](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Logical_Operators" \l "Logical_NOT)就三个：

&& 逻辑与运算

|| 逻辑或运算

! 逻辑非运算

正统来说，参与逻辑运算的是boolean和boolean，得到的结果也是boolean

值按照真值表来定

&& 逻辑与，“且”

|  |  |  |
| --- | --- | --- |
| a && b | | |
| A | b | 结果 |
| 真 | 真 | 真 |
| 真 | 假 | 假 |
| 假 | 真 | 假 |
| 假 | 假 | 假 |

“都真才真”，“有假就假”。

命题1：“地球是圆的” 真的

命题2：“宋仲基很帅” 真的

命题1 **且** 命题2 真的

命题1：“1+1=3” 假的

命题2：“地球是方的” 假的

命题1 **且** 命题2 假的

|  |
| --- |
| 1. //逻辑运算符 2. console.log(true **&&** true); //t 3. console.log(true **&&** false); //f 4. console.log(false **&&** true); //f 5. console.log(false **&&** false); //f |

|| 逻辑或，“或者”的意思

|  |  |  |
| --- | --- | --- |
| a || b | | |
| A | b | 结果 |
| 真 | 真 | 真 |
| 真 | 假 | 真 |
| 假 | 真 | 真 |
| 假 | 假 | 假 |

“有真就真”，“都假才假”

命题1： 1 + 1 = 2

命题2：“王老师很帅”

命题1 或者 命题2 总结果是真

|  |
| --- |
| 1. console.log(true **||** true); //t 2. console.log(true **||** false); //t 3. console.log(false **||** true); //t 4. console.log(false **||** false); //f |

!就是“逻辑非”，相反的

|  |
| --- |
| 1. console.log(**!**true); //f 2. console.log(**!**false); //t 3. console.log(**!!!!!!!!!**false); //t |

运算顺序是非、与、或

|  |
| --- |
| 1. true || false && !true || false; |

解：原式 = true || false && **false** || false

= true || false || false

= true || false

= true

逻辑运算符最最有意思的事情，就是所谓的“短路语法”。

就是你发现没有，

如果计算一个且运算的时候，比如a && b，a如果就是一个false，那么就不会管b是什么，直接输出false就行了，等于说直接输出a。

如果计算一个且运算的时候，比如 a && b ，a如果就是一个true，那么也不用管b是什么，直接把b当做结果输出就行了。

也就是说，本质上计算机进行a&&b运算的时候，不是在进行逻辑分析，这小子就想着要么扔a，要么扔b。如果a是负性的，那么直接扔出a；如果a是正性的，直接扔出b。

—— 短路语法。 要么a被短路，要么b被短路。

负性的：false，null， 0， NaN， 空字符串("")，undefined

正性的：除了上面的，全是正性的。

|  |
| --- |
| 1. false && 8 //false 因为计算机发现，且运算a已经是false了，直接输出false 2. null && 8 //null 因为计算机发现，且运算a已经是false性的了，直接扔出来null 3. true && 13 //13 因为计算机发现，且运算a是true，所以总结果就是看b，直接扔出b 4. 12 && 13 //13 因为计算机发现，12当做true，所以总结果看b，直接扔出b 5. 13 && 12 //12 因为计算机发现，13当做true，所以总结果看b，直接扔出b 6. undefined && 哈哈 //undefined 不报错，因为a已经是负性的了，所以直接扔出a，哈哈不管 7. 哈哈 && undefined //报错 8. true && NaN //NaN 扔后面 |

|| 逻辑或的短路也是类似的，a||b

计算机发现a是真，那么扔a；如果a是假，那么扔b

|  |
| --- |
| 1. 0 || 18 //18 前面假，扔后面 2. 18 || 0 //18 前面真，扔前面 3. undefined || NaN //NaN 前面假，扔后面 4. NaN || undefined //undefined 前面假，扔后面 |

|  |
| --- |
| 1. 88 || 99 && 66 || 55 |

解：原式 = 88 || 66 || 55

= 88 || 55

= 88

|  |
| --- |
| 1. undefined && ("3" != 3) || NaN && null |

解： 原式 = undefined && false || NaN && null

= undefined || NaN && null

= undefined || NaN

= NaN

总结一下短路语法：

a&&b， 计算机要么执行a要么执行b。a真执行b，a假执行a；

a||b， 计算机要么执行a要么执行b。a真执行a，a假执行b。

千万不要背，从真值表中自己推倒。

## 2.4 赋值运算符

= 赋值

+= 简便写法

-= 简便写法

\*= 简便写法

/= 简便写法

%= 简便写法

++

--

[赋值运算符？](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Assignment_Operators" \l "Assignment)

赋值运算的参与者，一定是变量。

|  |
| --- |
| 1. var a = 1; 2. a **+=** 2; //这行语句等价于**a = a + 2;** 3. console.log(a); //3 |

|  |
| --- |
| 1. var b = 6; 2. b **/=** 3; //等价于b = b / 3 3. console.log(b); //2 |

|  |
| --- |
| 1. var c = 100; 2. c **%=** 10; //等价于c = c % 10; 3. console.log(c); //0 |

|  |
| --- |
| 1. var a = "我"; 2. a **+=** "爱"; 3. a **+=** "你"; 4. console.log(a); |

[++运算符？](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators" \l "Increment)：

|  |
| --- |
| 1. var e = 10; 2. e++; //等价于e=e+1 3. console.log(e); //11 |

++可以与输出语句写在一起，++写在变量前和写在变量后不是一个意思

a++ : 先用a的原值，然后a加1；

++a ：先给a加1，然后用a的新值

|  |
| --- |
| 1. var f = 10; 2. console.log(f++); //10 ，先引用原值，然后加1 |

等价于：

|  |
| --- |
| 1. //等价于 2. var f= 10; 3. console.log(f); //先输出f 4. f++; //然后f加1 |

|  |
| --- |
| 1. var g = 10; 2. console.log(++g); //11 ， 这次是先加1，然后输出 |

++有花式玩儿法，仅面试有用：

|  |
| --- |
| 1. var a = 8; 2. console.log(4 + a++); //12 ， 先使用原来的a的值，就是4+8，输出12.然后a加1 3. console.log(a); //9 |

|  |
| --- |
| 1. var i = 9; 2. console.log(++i % 5); //0 , 先把i加1，然后使用i，10%5=0 3. console.log(i); //10 |

[运算符的计算顺序？](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Operator_Precedence)：

++ -- !贴身的 **→→→** 数学 **→→→** 比较 **→→→** 逻辑 **→→→** 赋值

|  |
| --- |
| 1. var a = 3 < 6 && 7 < 14; //true |

原式 = true && true

= true

|  |
| --- |
| 1. var a = 1 + 2 < 3 + 3 && 3 + 4 < 2 \* 7; |

原式 = 3 < 6 && 7 < 14

= 上一题

= true

|  |
| --- |
| 1. var a = false + true && 13; |

原式 = 0 + 1 && 13

= 1 && 13

= 13

|  |
| --- |
| 1. var a = 15; 2. false + a++ + true > 8 && 13 || 6 |

原式 = false + 15 + true > 8 && 13 || 6

= 16 > 8 && 13 || 6

= true && 13 || 6

= 13 || 6

= 13