**运算符号：**

也叫操作符。

通过运算符可以对一个或多个值进行运算

比如：typeof就是一个运算符，可以用来获得一个值的类型，

它会将该值的类型以字符串的形式返回

**1.算数运算符**

（除了带字符串的加法，其余都适用）当对非Number类型的值 进行运算的时候，会将其先转换为Number再进行运算

任何值和NaN做运算，结果都是NaN

**+ 加 ：**进行加法运算并将结果返回或者拼接

-如果对两个字符串进行加法运算，则会做拼接，将两个字符串拼成一个并返回

-任何的值和字符串做加法运算都会先转换为字符串，再拼成一个字符串

--我们可以利用这一特点来将任意一个数据类型转换为String

只需要为任意的数据类型+一个""即可将其转换为String

这是一种隐式的转换，由浏览器自动完成，实际上也是调用String（）函数

**- 减：** 对两个值进行减法运算

**\* 乘：**对两个值进行乘法运算

**/ 除：**对两个值进行除法运算

**% 取模：**对两个值进行除法运算，返回余数

注：前大后小，整除以后得余数

前小后大，模为小的数本身

由于对非Number类型的值 进行运算的时候，会将其先转换为Number再进行运算 我们可以利用这一特点做隐式的类型转换

可以通过为一个值进行-0 \*1 /1来将其转换为Number

原理和Number（）函数一样，使用起来更加简单。

js进行浮点数运算可能出现一个不精确的结果

当进行有理小数的加减的时候可以将小数先同乘x化为整数 进行整数的运算 最后同除x

（0.7\*10+0.8\*10）/10 = 1.5

**2.拼接运算符**

JS中拼接运算符用的是“+”

一旦遇到非法数字后，后面的一律看做拼接，如下例：



第三种混合的情况下，2+3+4正常相加为9，遇到后面的‘haha'为非法数字则从这里开始都做拼接。

**3.一元运算符**

只需要一个操作数

+正号

不会对数字产生任何影响

-负号

负号可以对进行取反

对于非Number类型的值 会将其转换为Number类型以后再运算

可以对一个其他的数据类型加上一个+正号，将其转换为Number

原理和调用Number（）函数一样

**自增++**

-通过自增可以使变量在自身的基础上增加1

-对于一个变量自增以后，原变量的值会立即自增1

-自增分成两种:后++(a++)和前++(++a)

无论是a++ 还是++a，都会立即使原变量的值自增1

不同的是a++ 和++a的值不同

a++（自增前的）的值等于原变量的值

++a 的值等于原变量自增后的新值

**自减--**

-自减可以使变量在原值的基础上自减1-自减使用—-

-自减可以使用前--(--a)后--(a--)

-无论是--a还是a--都会立即使原变量自减1

不同的是--a和a--的值是不同的,

a--（自减前的）的值等于原变量的值

--a 的值等于原变量自减后的新值

**4.关系运算符**

通过关系运算符可以比较两个值之间的大小关系，

如果关系成立它会返回true，如果关系不成立则返回false

**< 小于**

-判断符号左侧的值是否小于右侧的

-如果关系成立，返回true，如果关系不成立则返回false

**> 大于**

-判断符号左侧的值是否大于右侧的

-如果关系成立，返回true，如果关系不成立则返回false

**<= 小于等于**

-判断符号左侧的值是否小于等于右侧的

-如果关系成立，返回true，如果关系不成立则返回false

**>= 大于等于**

-判断符号左侧的值是否大于等于右侧的

-如果关系成立，返回true，如果关系不成立则返回false

**非数值的情况**

-数值与数值比较，直接比较大小

-数值和非数值进行比较时，会将其转换为数值再进行比较

-任何值和NaN作比较 都返回false

-如果符号两侧的值都是字符串时，不会将其转换为数字进行比较，而是比较ASCII码的大小

0~9的ASCII码 48 ~ 57

a-z的ascii码 97 ~ 122

A-Z的ascii码 65 ~ 90

-比较字符编码是一位一位进行比较，如果第一位就能完成比较，就返回true或false。如果两位一样，则比较下一位。

**所以利用它来对英文进行排序、比较中文时没有意义**

-注意：

如果比较两个字符串型的数字，可能会得到不可预期的结果

在比较两个字符串型的数字时，一定一定要转型

**相等**

-判断符号左侧的值是否等于右侧的

-如果关系成立，返回true，如果关系不成立则返回false

**使用==来做相等运算**

-当使用==来比较两个值时，如果值的类型不同,

-则会自动进行类型转换，将其转换为相同的类型然后在比较

--undefined衍生自null
，所以这两个值做相等判断时，会返回true

--NaN不和任何值相等，包括它本身 NaN == NaN返回false

可以通过isNaN()函数来判断一个值是否是NaN

如果该值是NaN则返回true，否则返回false

**使用===来做全等运算**

-用来判断两个值是否全等，它和相等类似，不同的是它不会做自动的类型转换

如果两个值的类型不同,直接返回false

**不相等**

-判断符号左侧的值是否不等于右侧的

-如果关系成立，返回true，如果关系不成立则返回false

**使用!=来做不相等运算**

-不相等也会对变量进行自动的类型转换，如果转换后相等它也会返回false

**使用!==来做全不等运算**

-用来判断两个值是否全不等，它和不等类似，不同的是它不会做自动的类型转换

如果两个值的类型不同,直接返回true

**5.赋值运算符**

**=**

将符号右侧的值赋值给符号左边的变量

**+=**

a += 5等价于a = a + 5

**-=**

a -= 5等价于a = a - 5

**\*=**

a \*=5等价于a = a \* 5

**/=**

a /= 5等价于a = a / 5

**%=**

a %= 5等价于a = a %5

**6.逻辑运算符**

&&和||是短路运算符 &&找假 ||找真 找到就不往后找了

**&& 逻辑与**

- &&可以对符号两侧的值进行与运算并返回结果

-运算规则

-两个值中只要有一个值为false就返回false,

只有两个值都为true时，才会返回true

- JS中的“与”属于短路的与

如果第一个值为false，则不会看第二个值

**| | 逻辑或**

-| |可以对符号两侧的值进行或运算并返回结果

-运算规则:

-两个值中只要有一个true，就返回true

如果两个值都为false，才返回false

-JS中的“或”属于短路的或

如果第一个值为true，则会检查第二个

**! 非**

- !可以用来对一个值进行非运算

-所谓非运算就是值对一个布尔值进行取反操作，true变false，false变true

-如果对一个值进行！！两次取反，它不会变化

-对于一个非布尔值的数进行取反的时候，会将其转换为布尔值后再进行取反操作，所以我们可以利用该特点进行隐式类型转换，将一个任意其他数据类型取两次反，将其转换为布尔值，原理和Boolean（）函数一样

**逻辑运算符两边非布尔值的情况**

-对于非布尔值进行&& 、| |运算时，会将其转换为布尔值，然后再进行运算，并且返回原值：

&&与运算 如果两个值都为true，则返回后面的值

如果两个值中有false，则返回第一个false

所以得出规律

如果第一个值是true，一定会返回后面那个值

第一个值是false ，一定会返回第一个false；

|| 或运算 如果两个值最后结果为true，则返回靠前的true

如果两个值都为false，则返回第二个false

所以得出规律

如果第一个值是true，则直接返回第一个值

第一个值是false ，一定会返回第二个值；

真&&真 为真

真&&假 为假

假&&假 为假

真| |真 为真

真| |假 为真

假| |假 为假

**7.条件运算符**

条件运算符也叫三元运算符/三目运算符

**语法:**

条件表达式?语间1:语句2;

**执行的流程:**

条件运算符在执行时,首先对条件表达式进行求值，

-如果该值为true，则执行语句1，并返回执行结果

-如果该值为false，则执行语句2，并返回执行结果

-如果条件的表达式的求值结果是一个非布尔值，

-会将其转换为布尔值然后在运算

var max = a > b ? a : b;//获取a和b中的最大值

max = max > c ? max : c;//获取a b c中的大值

var max = a > b ? (a > c ? a :c) : (b : c ? b : c);//这种写法不推荐，不方便阅读

**三目运算改写:**

if(条件表达式){

执行语句1

}else{

执行语句2

}

改为三目运算符(三元运算符)

条件表达式?执行语句1:执行语句2

**8.位移运算符**

<<、>>、<<<、>>>

A<<B把A转换成二进制后，向左移动b位后，转换为10进制

eg.

2<<2 == 8

2的2进制是10 看作000010

向左移动两位 得001000

重新转换为10进制得8

**运算符的优先级**

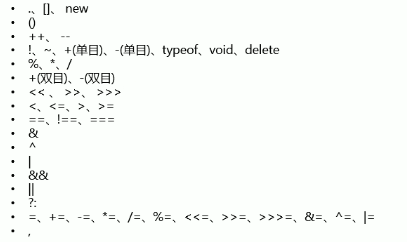
就和数学中一样，在JS中运算符也有优先级，

比如:先乘除后加减

在]S中有一个运算符优先级的表，

在表中越靠上优先级越高，优先级越高越优先计算

如果优先级一样，则从左往右计算。不需要记表，如果遇到不清楚的优先级可以用（）来人为改变计算顺序



**，运算符**

使用“，”可以分割多个语句,一般可以在声明多个变量时使用，

可以同时声明多个变量并赋值var a=1 , b=2 , c=3;

**流程控制语句：**

顺序结构，分支结构，循环结构

-js中的程序是从上到下一行一行执行的

-通过流程控制语句可以控制程序执行流程,

使程序可以根据一定的条件来选择执行

**1.条件判断语句：**

使用条件判断语句可以在执行某个语句之前进行判断，

如果条件成立才会执行语句,条件不成立则语句不执行。

**if语句：**

**语法一：**

if(条件表达式){

语句

}

if语句在执行时，会先对条件表达式进行求值判断，

如果条件表达式的值为true，则执行if后的语句，

如果条件表达式的值为false，则不会执行if后的语句。

if语句只能控制紧随其后的那个语句，

如果希望if语句可以控制多条语句,

可以将这些语句统一放到代码块中

if语句后的代码块不是必须的，但是在开发中尽量写上代码块

即使只有一条语句

**语法二∶**

if(条件表达式){

语句..

}else{

语句...

}

if...else...语句

当该语句执行时,会先对if后的条件表达式进行求值判断，

如果该值为true，则执行if后的语句

如果该值为false，则执行else后的语句

**语法三∶**

if(条件表达式){

js执行语句

}else if(条件表达式){

js执行语句

}else if(条件表达式){

}...

else{

}

当该语句执行时，会从上到下依次对条件表达式进行求值判断

如果值为true，则执行当前语句。如果值为false，则继续向下判断。

如果所有条件都不满足则执行最后一个else后的语句

该语句中只会有一个代码块被执行，执行完毕后直接结束语句

**if嵌套:**

if(条件表达式){

if(条件表达式){

js执行语句

}else if{

js执行语句

}

}else{

js执行语句

}

【注】

if(0){console.log("不会打印")}

原因: 0是false

0 -0 null "" false undefined NaN 在if条件里都判断为false

**2.条件分支语句(选择结构)︰**

根据不同的条件判断来决定程序执行的走向的结构

**switch语句:**

switch(){

case 1:

执行语句

break;//使用break；退出switch语句

case 2:

执行语句

break;

case 3:

.

.

.

default:

执行语句

break;

}

**执行流程：**

switch. ..case..语句

在执行时会依次将case后的表达式的值和switch后的条件表达式的值进行全等比较

如果比较结果为true，则从当前case处开始执行代码。

当前case后的所有代码都会执行，所以要在每个case的最后跟一个break；语句

如果比较结果为false，则继续向下比较。

【注】如果没有break,那么程序将一直执行,直到遇到break

switch语句和if语句的功能实际上有重复的，使用switch可以实现if的功能，

同样使用if也可以实现switch的功能，所以我们使用时，可以根据自己的习惯选择

switch好处是条件清楚明了，if的好处是书写方便简单。

**进制**

1. 16进制的数以0x开头

2. 8进制的数以0开头 尽量避免不用

3. 2进制的数以0b开头 不是所有浏览器都支持

可以在parseInt（）传递第二个参数，来指定数字的进制

比如 var a = "101";///这是一个字符串

parseInt（a,8）//让js把a的字符串中包含的数当成8进制数来解析。最终返回对应的10进制的值

toString(） 可以将数字指定为其他进制 返回转换后的进制 一开始默认是十进制 a.toString(2) 将a转换为二进制

**语句/代码块**

我们的程序是由一条一条语句构成的

语句是按照自上向下的顺序一条一条执行的

在js中可以使用{ }来为语句进行分组，

同一个{ }得中的语句我们称为是一组语句，

它们要么都执行，要么都不执行，

一个{ }得中的语句我们也称为叫一个代码块

在代码块的后边就不用再编写;了

js中的代码块，只具有分组的的作用，没有其他的用途

代码块内容的内容，在外部是完全可见的