# 运算符和表达式及数据类型转换

## 运算符和表达式

### 1、运算符和表达式分类

#### 2）赋值运算符和表达式

= 赋值运算 是把= 右边的值 赋值给 = 左边的变量；

=左边一定是变量（后期还有属性） =右边一定是有值的东西； \*\*\*\*\*\*\*\*\*\*\*

var a = 10;

var b = 20;

a = a + b;

a = b;

+= -= \*= /= %= 复合赋值运算符

a += b; <====> a = a + b;

a += 1; a = a + 1;

#### 3）比较（条件运算符） 和 表达式

> < >= <= == != ===全等 !== 不全等

比较运算符和表达式；最终表达式的值都是布尔值；

#### 4）逻辑运算符 常用在多个条件表达式的连接

&&与 ||或 ！非

1、通用：

&& 整个表达式的值：如果&&前面为真就取后面的值作为整个表达式的值如果&&前面为假就取前面的值作为整个表达式的值,后面的表达式根本不执行;

|| 整个表达式的值：如果||前面为真就取前面的值作为整个表达式的值，后面的表达式根本不执行;如果||前面为假就取后面的值作为整个表达式的值;

! 非真即假 非假即真 和它构成的表达式 最终都会是一个布尔值；

!:出现了！，！要求后面必须值是布尔值，不是就转化为布尔值，必须是布尔值；！6

var a = 10;

var b = 20;

var c = a || b;

2、常用：

多个条件表达式的连接的时候 ,两遍都是条件表达式才能使用

&& 一假则假

|| 一真则真

! 非真即假 非假即真

var a = 10;

var b = 20;

var c = a> b && a < 30;

#### 5）三目运算符 三元 问号冒号表达式

? :

一元 ++ -- ！ 一元运算符 a++ ++a !a !(a>b) +5 -5

a++

++a

!6

+5

-5;

一元运算符，优先级是最高的

var c = a + b++

二元 + - \* / % 二元运算符

&& || =(在二元运算符优先级很低的)

var c = a + b;

三元 ?: 是运算符当中唯一的一个三元运算符

var a = 10;

var b = 10?++a:a--;

问号冒号表达式：先看第一个表达式问号前面的值是否为真(不是布尔要转化为布尔)？ 如果为真，则取冒号前面的值作为整个表达式的值，后面的不执行，如果为假,则取冒号后面的值作为整个表达式的值,前面的不执行

## 数据类型转换

### 1、数据类型强制转换（显式）

#### 1）其它类型值转数字

Number() 强制将一个其它类型数据转化为数字类型

转不了就是NaN not a number

转化字符串

如果字符串整体来看是一个数字，那么就转化为这个数字

如果字符串整体来看不是一个数字，那么就转化为NaN

如果字符串是一个特殊的空字符串或者空白字符串，那么转化为0

转化boolean true会转化为1 false 会转化为0

转化undefined undefined 会转化为NaN

转化null null 会转化为0

#### 2）其它类型值转字符串

String() 强制将一个其它类型数据转化为字符串类型

转化字符串没有什么特殊，只要给我值，那么就会把这个值原样变为字符串输出；

#### 3）其它类型值转布尔值

Boolean() 强制将一个其它类型数据转化为boolean类型

转化数字的时候，除了0和NaN是false,其余都是true;

转化字符串的时候，除了空字符串是false,其余都是true

转化undefined和null都是false;

### 2、数据类型隐式转换

各种类型在适当的场合会发生隐式转换（隐式转换的规则和强制转换一样，只是自己没写出来）

主要是运算和条件判断过程中

### 3、数据类型手动转换（其实就是在字符串当中提取数字）

字符串：数字字符必须在字符串的前面（前面可以有空白字符）；

否则就是NaN

parseInt()

parseFloat()

## 基本数据运算（隐式转换）

### 1、同种数据类型运算、比较

数字和数字

小数运算的时候不准

数字能表示的最大值 Number.MAX\_VALUE

1/0 0/1 0/0 -1/0 1%0 分别是多少?

+

-

\*

/

比较

该怎么算怎么算 该怎么比较怎么比较 不会进行数据转化

字符串和字符串 字符串有数字型的字符串和不是数字型的,还有空字符串

+

-

\*

/

比较

+法的时候是拼接字符串 其余的运算全部转化数字

比较的时候比较的是字符的Unicode码，不会转化数字进行比较；

boolean和boolean

+

-

\*

/

比较

判等的时候不需要转化，其余全部转数字

Undefined和undefined

+

-

\*

/

比较

判等的时候不需要转化，其余全部转数字

null和null

+

-

\*

/

比较

判等的时候不需要转化，其余全部转数字

### 2、不同数据类型运算、比较

数字和字符串 + - \* / 比较

数字和boolean + - \* / 比较

数字和undefined + - \* / 比较

数字和null + - \* / 比较

字符串和boolean + - \* / 比较

字符串和undefined + - \* / 比较

字符串和null + - \* / 比较

Bool和undefined + - \* / 比较

Bool和null + - \* / 比较

Undefined和null + - \* / 比较

### 3、总结：

判等的时候：先看两边是不是同种数据类型，如果是直接看是不是一样就完了；

如果不是那么两边都转数字

当遇到null的时候;会有特殊情况发生,

特殊情况：空串和null不相等

特殊情况：false和null不相等

特殊情况: 0和null不相等

特殊情况：undefined 和 null 相等；

在其余的运算和比较情况下：

第一步：先看是不是 +

如果是+ 看有没有字符串，如果有 就是拼接字符串

第二步：再看是不是比较

如果是比较 看是不是两边都是字符串 如果是 比较的是字符串的Unicode码

最后： 全部转数字

NaN: 所有的东西和NaN进行算术运算都是NaN 所有的东西和NaN进行比较大小都是false 所有的东西和NaN都不相等（包括自己）

NaN == NaN

全等和不全等： 不会出现类型转换，他们在判等的时候，先判断类型是不是一样，如果类型一样再判断值是不是相同，如果都相同才全等；如果有一个不一样，那么就不全等；

1 == true; //true

1 === true；//false

== 是用来判值的，最后的值相等就相等（它是可以发生隐式转化的）

=== 是类型和值都要判，必须全部相等才全等；

推荐今后自己写程序使用全等；

小面试题：

怎么快速将一个字符串‘23’转化为数字23；

+‘23’；

除了+号可以转化为23 我们还有什么方法可以转化为23

作业：晚自习强化练习

1、字符串拼接：个人简介

姓名： var name = 'ly';

年龄：

性别：

期望薪资：

console.log(‘我的名字叫’ + name);// ‘我的名字叫ly’；

'我的名字叫\*\*，我的年龄是\*\*岁，我的性别是\*\*，我期望毕业薪资是\*\*K

2、数据类型转换（课堂案例）

所有的基本数据类型进行运算和比较（隐式转化）

3、求出下面表达式的值

NaN + NaN

parseInt(' 12.345.67sdjfg');

parseFloat(' 12.345.67sdjfg');

parseFloat(' iloveyou12.345.67sdjfg');

null == ''

null == null

null == undefined

null == false

NaN == NaN

'100' + NaN

Number(NaN)

Boolean(NaN)

true == ' ';

true + '123';

+'45'-false;