Universitat Oberta de Catalunya

Aula

M4.250 - Introducción a la programación en JavaScript I aula 1

Expresiones y operadores

Inicio:	Fin:	Dedicación:
01/10/19	**10/10/19**	**10 h**

Introducción

En este apartado veremos el artículo " Expresiones y operadores ".

Aunque ya hemos estado haciendo operaciones con JavaScript, ha llegado el momento de hacer un recorrido por todos los operadores que podemos encontrar en JavaScript. También repasaremos que es una expresión y veremos algunas expresiones particulares.

Operadores
Evaluación mínima

Operadores binarios: Son aquellos que operan con dos operandos. Un ejemplo es la suma: 2 + 5

Operadores unarios: Son aquellos que operan con un operando. Un ejemplo es el signo menos indicando un valor negativo: -2

En JavaScript existe además un operador ternario que veremos más adelante, pero que, evidentemente, opera con tres operandos.

Operadores de asignación

Asusta un poco ver que en JavaScript existen 13 operadores de asignación. En realidad podríamos decir que solo hay uno y que los demás son una abreviación que nos permite hacer dos operaciones en una: la asignación y otra operación.

Una asignación especial nos permite poner en tres variables diferentes los valores que hay en una tabla (array) de tres elementos:

```
var tabla = [0,1,2];
var [uno, dos, tres] = tabla;
console.log(uno);
```

En este ejemplo se crearán tres variables. En la variable uno se guardará un 0, en la variable dos un 1 y en la variable tres un 2.

Operadores de comparación

A comentar que en un mayor o menor igual, el símbolo de mayor que o el de menor que siempre van delante del símbolo igual: >= o <=

La igualdad (===) y la desigualdad (!==) estrictas suelen ser más recomendables que las no estrictas (== y !=).

Un error habitual es intentar hacer dos comparaciones a la vez. Por ejemplo, nos piden saber el mayor de tres números guardados en tres variables a, b y c. Podríamos pensar en hacer algo como esto:

```
if (a > b > c) {
   console.log("a es el más grande");
}
```

Sin embargo esto es incorrecto. Las operaciones de comparación se hacen de dos en dos. Por tanto, al calcular a > b > c en realidad se harán estas operaciones (vamos a suponer que a = 9; b = 1; c = 4):

1. a > b (true)
2. true > c (false)

El problema es que no se pueden hacer las dos comparaciones a la vez, primero hay que hacer la primera y después la segunda. Por en medio JavaScript, al comparar un número con un valor booleano, lo que hace es convertir el booleano a número con esta conversión: true equivale a 1 y false a 0...

Así que a > b > c nunca hará lo que instintivamente podemos pensar que hace.

Igual queda más claro viendo y probando el código: https://goo.gl/WbabrF

Operadores aritméticos

A remarcar dos operadores unarios: ++ y --

Estos operadores incrementan en uno o en menos uno los valores de la variable sobre la que operan. Se pueden poner delante (pre) o detrás (post) de la variable y su funcionamiento varia.

Prueba este ejemplo:

```
let a = 5;
let b = 7
console.log(a++ + b);
console.log(a);
```

```
a = 5;
console.log(++a + b);
console.log(a);
```

Está aquí: https://goo.gl/EJFaLh

Como se puede comprobar, en el primer caso, primero se hace la operación y después se incrementa la variable a . En el segundo caso, primero se hace el incremento y después la operación.

Operadores bit a bit i operadores de desplazamiento binario

Estos operadores operan las variables como si su contenido fuesen números binarios.

Operadores lógicos

Los operadores lógicos son 3: y (&&), o (||) y no (!).

Evaluación mínima

Se basa en que en un operador && si un operando es falso el resultado final siempre es falso. Y que en un operador || si un operando es cierto, el resultado final siempre será cierto.

Como las expresiones se evalúan de izquierda a derecha, en el caso de un && si el primer valor es falso, el segundo ya no se evalúa.

Y con el operando || y cierto pasa lo mismo. Esto es especialmente importante en un caso como este:

```
let a = true;
let c = 5;
if ( a && c++<5) {
  console.log("El c++ no se ejecutará siempre");
}
```

En este caso si a es true, la instrucción c++ se ejecutará. Pero si es false no.

Operador de cadena

El operador + nos permite concatenar dos cadenas de caracteres.

Operador condicional (ternario)

Este operador nos permite ejecutar una instrucción dependiendo de que se cumpla o no una condición.

Tenemos tres operandos: una condición y dos expresiones. Las expresiones pueden ser desde un valor hasta una asignación o una comparación. En el siguiente ejemplo se pueden ver tres usos de este operador ternario:

```
let a = 1, b = 0, c = 0;
(a === 1) ? b = 1 : c = 1;
console.log(b,c);
console.log((a===1)?"a es 1":"a no es 1");
```

```
c = (a > 1) ? a : b;
```

Intenta explicar qué se hace en cada uso del operador ternario en este programa.

Resto de operadores

Entre los operadores que quedan en este capítulo están:

- **delete** Borra un objeto.
- **typeof** Nos dice el tipo de una variable.
- **in** Para saber si una determinada propiedad se encuentra en un objeto determinado.
- **instanceof** Para saber si un objeto tiene un tipo determinado.

Estos operadores tendrán sentido cuando hablemos de los objetos, así que, sabemos que están aquí con los demás operadores, pero los usaremos entonces.

Precedencia de operadores

La preferencia de los operadores es muy importante dado que nos indica de que manera se evaluará una expresión.

A misma prioridad, los operadores se evaluarán de izquierda a derecha. Así en este caso:

```
3 + 5 + 8 + 9
```

Primero se calcularía 3 + 5 (8) después 8 + 8 (16) y finalmente 16 + 9. Es importante tener en cuenta que siempre, todas las operaciones se hacen secuencialmente, no se hacen a la vez.

Veamos un nuevo caso:

```
4 + 6 / 2
```

¿Cuál es el resultado correcto?

Si evaluamos de izquierda a derecha, las operaciones que se hacen son:

1. 4 + 6 (10)
2. 10 / 2 (5)

Con lo que el resultado será 5.

Pero esto no es correcto . Si miramos la tabla de prioridades veremos que la división tiene más prioridad que la suma. Por tanto el orden en que se harán estas operaciones será este:

1. 6/2 (3)

2. 4 + 3 (7)

Por tanto el resultado final en realidad será 7.

Expresiones

Podemos definir una expresión cómo un conjunto de instrucciones que devuelven un valor.

Hemos estado trabajando con expresiones así que ya tenemos experiencia. Hay sin embargo algunas, llamadas primarias, que no hemos visto:

- **this** La usaremos con los objetos.
- **paréntesis** Cambian la prioridad de las operaciones, tal como hacemos en matemáticas.
- **new** y **super** Útil con los objetos.
- **...** (operador de propagación) Útil con los arrays.

Estos operadores que ahora vemos pero que no trabajamos, los recuperaremos más adelante. Sin embargo están aquí para tener ordenados todos los operadores a modo de referencia.

Recursos de aprendizaje

Fuentes de información