Tema 2 - JavaScript

Anexo 2:
Referencia vs copia.
Expresiones regulares.

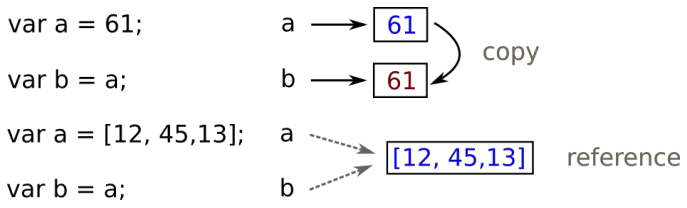
Desarrollo web en entorno cliente
IES Pere Maria Orts I Bosch

Index

Anexosemana2

Referencia vs Copia

A diferencia de los tipos primitivos como boolean, number o string, los arrays son tratados como un objeto en JavaScript, lo cual significa que tienen un puntero o referencia a la dirección de memoria que contiene el array. Cuando se copia una variable o se envía esa variable como parámetro a una función, estamos copiando la referencia y no el array, por tanto ambas variables apuntarán a la misma zona de memoria. Si por ejemplo cambiamos la información que teníamos almacenada en una de esas variables, estaremos cambiando la información de la otra también ya que son el mismo objeto.



Vamos a ver algunos ejemplos:

```
var a = 61;
var b = a; // b = 61
b = 12;
console.log(a); // Imprime 61
console.log(b); // Imprime 12

var a = [12, 45, 13];
var b = a; // b ahora referencia al mismo array que a
b[0] = 0;
console.log(a); // Imprime [0, 45, 13]
```

Otro ejemplo, pasando el array a una función:

```
function changeNumber(num) {
   var localNum = num; // Variable local. Copia el valor
   localNum = 100; // Cambia solo localNum. El valor fue copiado
}

var num = 17;
changeNumber(num);
console.log(num); // Imprime 17. No ha cambiado el valor original


function changeArray(array) {
   var localA = array; // Variable local. Hace referencia al original
   localA.splice(1,1,101, 102); // Elimina 1 item en la posición1 e inserta 101 y 102 ahí
}

var a = [12, 45, 13];
changeArray(a);
console.log(a); // Imprime [12, 101, 102, 13]. Ha sido cambiado dentro de la función
```

Expresiones regulares

Las expresiones regulares nos permiten buscar patrones en un string, por tanto buscar el trozo que coincida o cumpla dicha expresión. En JavaScript, podemos crear una expresión regular instanciando un objeto de la clase RegExp o escribiendo directamente la expresión entre dos barras '/'.

En este apartado veremos conceptos básicos sobre las expresiones regulares, podemos aprender más sobre ellas en el siguiente enlace. También, hay páginas web como esta que nos permiten probar expresiones regulares con cualquier texto.

Una expresión también puede tener uno o más modificadores como 'g' → Búsqueda global (Busca todas las coincidencias y no sólo la primera), 'i' → Case-insensitive (No distingue entre mayúsculas y minúsculas), o 'm' → Búsqueda en más de una línea (Sólo tiene sentido en cadenas con saltos de línea). En Javascript puedes crear un objeto de expresión regular estos modificadores de dos formas:

```
let reg  = new RegExp("^[0-9]{2}","gi");
let reg2  = /^[0-9]{2}/gi;
console.log(reg2 instanceof RegExp);  // Imprime true
```

Estas dos formas son equivalentes, por tanto podéis elegir cual usar.

Expresiones regulares básicas

La forma más básica de una expresión regular es incluir sólo caracteres alfanuméricos (o cualquier otro que no sea un carácter especial). Esta expresión será buscada en el string, y devolverá true si encuentra ese *substring*.

```
let str = "Hello, I'm using regular expressions";
let reg = /reg/;
console.log(reg.test(str)); // Imprime true
```

En este caso, "reg" se encuentra en "Hello, I'm using regular expressions", por lo tanto, el método test devuelve true.

Corchetes (opción entre caracteres)

Entre corchetes podemos incluir varios caracteres (o un rango), y se comprobará si el carácter en esa posición de la cadena coincide con alguno de esos.

[abc] → Algún carácter que sea 'a', 'b', ó 'c'

[a-z] → Algún carácter entre 'a' y 'z' (en minúsculas)

[0-9] → Algún carácter entre 0 y 9 (carácter numérico)

[^ab0-9] → Algún caracter excepto (^) 'a', 'b' ó número.

Pipe → | (opción entre subexpresiones)

(exp1|exp2) → La coincidencia en la cadena será con la primera expresión o con la segunda. Podemos añadir tantos pipes como queramos.

Meta-caracteres

. (punto) → Un único carácter (cualquier carácter excepto salto de línea)

\w → Una palabra o carácter alfanumérico. \W → Lo opuesto a \w

\d → un número o dígito. \D → Lo opuesto a \d

\s → Carácter de espacio. \S → Lo opuesto a \s

\b → Delimitador de palabra. Coincide el principio o final de palabra (no un carácter). Por ejemplo, /\bcase\b/, coincide con "case" pero no "uppercase" o "casein".

\n → Nueva línea

\t → Carácter de tabulación

Cuantificadores

+ → El carácter que le precede (o la expresión dentro de un paréntesis) se repite 1 o más veces.

* → Cero o más ocurrencias. Lo contrario a +, no tiene porqué aparecer

? → Cero o una ocurrencia. Se podría interpretar como que lo anterior es opcional.

{N} → Debe aparecer N veces seguidas.

{N,M} → De N a M veces seguidas.

{N,} → Al menos N veces seguidas.

^ → Principio de cadena.

$ → Final de la cadena.

Podemos encontrar más sobre las expresiones regulares de JavaScript aquí.

Ejemplos

/^[0-9]{8}[a-z]$/i → Comprueba si la cadena tiene formato de DNI. Esta expresión coincidirá con una cadena que comience (^) con 8 números, seguidos con una letra.

No puede haber nada después ($ → final de cadena). El modificador 'i' hace que no distinga entre mayúsculas y minúsculas.

/^\d{2}\/\d{2}\/(\d{2}|\d{4})$/ → Comprueba si una cadena contiene una fecha en el formato DD/MM/YY ó DD/MM/YYYY. El metacaracter \d es equivalente a usar [0-9], y el año no puede tener tres números, o son 2 o 4.

/\b[aeiou]\w*\b/i → Comprueba si hay alguna palabra en la cadena que comienza por una vocal seguida de cero o más caracteres alfanuméricos (números, letras o '_').

Métodos para las expresiones regulares en JavaScript

A partir de un objeto RegExp, podemos usar dos métodos para comprobar si una cadena cumple una expresión regular. Los dos métodos son test y exec.

El método test() recibe una cadena e intenta de encontrar una la coincidencia con la expresión regular. Si el modificador global 'g' se especifica, cada vez que se llame al método se ejecutará desde la posición en la que se encontró la última coincidencia, por tanto podemos saber cuántas veces se encuentran coincidencias con esa expresión. Debemos recordar que si usamos el modificador 'i' no diferencia entre mayúsculas y minúsculas.

```
let str = "I am amazed in America";
let reg = /am/g;
console.log(reg.test(str)); // Imprime true
console.log(reg.test(str)); // Imprime true
console.log(reg.test(str)); // Imprime false, hay solo dos coincidencias

let reg2 = /am/gi; // "Am" será lo que busque ahora
console.log(reg.test(str)); // Imprime true
console.log(reg.test(str)); // Imprime true
console.log(reg.test(str)); // Imprime true. Ahora tenemos 3 coincidencias con este nuevo patrón
```

Si queremos más detalles sobre las coincidencias, podríamos usar el método exec(). Este método devuelve un objeto con los detalles cuando encuentra alguna coincidencia. Estos detalles incluyen el índice en el que empieza la coincidencia y también el string entero.

```
let str = "I am amazed in America";
let reg = /am/gi;
console.log(reg.exec(str)); // Imprime ["am", index: 2, input: "I am amazed in America"]
console.log(reg.exec(str)); // Imprime ["am", index: 5, input: "I am amazed in America"]
console.log(reg.exec(str)); // Imprime ["Am", index: 15, input: "I am amazed in America"]
console.log(reg.exec(str)); // Imprime null. No hay más coincidencias
```

Una mejor forma de usarlo sería:

```
let str = "I am amazed in America";
let reg = /am/gi;
let match;
while(match = reg.exec(str)) {
  console.log("Patrón encontrado!: " + match[0] + ", en la posición: " + match.index);
}
/* Esto imprimirá:
```

```
* Patrón encontrado!: am, en la posición: 2
* Patrón encontrado!: am, en la posición: 5
* Patrón encontrado!: Am, en la posición: 15 */
```

De forma similar, hay métodos de la clase String (sobre la cadena esta vez) que podemos usar, estos admiten expresiones regulares como parámetros (vamos, a la inversa). Estos métodos son match (Funciona de forma similar a exec) y replace.

El método match devuelve un array con todas las coincidencias encontradas en la cadena si ponemos el modificador global → g, si no ponemos el modificador global, se comporta igual que exec().

```
let str = "I am amazed in America";
console.log(str.match(/am/gi)); // Imprime ["am", "am", "Am"]
```

El método replace devuelve una nueva cadena con las coincidencias de la expresión regular reemplazadas por la cadena que le pasamos como segundo parámetro (si el modificador global no se especifica, sólo se modifica la primera coincidencia). Podemos enviar una función anónima que procese cada coincidencia encontrada y nos devuelva la cadena con los reemplazos correspondientes.

```
let str = "I am amazed in America";
console.log(str.replace(/am/gi, "xx")); // Imprime "I xx xxazed in xxerica"

console.log(str.replace(/am/gi, function(match) {
   return "-" + match.toUpperCase() + "-";
      })); // Imprime "I -AM- -AM-azed in -AM-erica"
```

Anexo 2: Referencia vs copia. Expresiones regulares.

Programación con JavaScript
Cefire 2021
Autor: Arturo Bernal Mayordomo