JavaScript

FUNCIONES DE ARRAY: MAP, FILTER Y REDUCE

Array.prototype.map()

La función map devuelve un nuevo array con los resultados de la llamada a la función pasada como parámetro, aplicada a cada uno de los elementos del array sobre el que itera.

```
Arr.map(callback[, thisArg])

[1, 2, 3, 4].map( function(valorActual, índice, array{……..})
```

Veamos como lo haríamos con el método tradicional:

```
var numeros = [1, 2, 3, 4, 5, 6];
var cuadrados = [];
for (var i = 0; i < numeros.length; i++)
 {
   cuadrados[i] = numeros[i] * numeros[i];
 }

// [1, 4, 9, 16, 25, 36]
```

Array.prototype.map() II

```
var numeros = [1, 2, 3, 4, 5, 6];

var cuadrados = numeros.map( function (numero) {

 return numero * numero;

 });

 // [1, 4, 9, 16, 25, 36]
```

Con ECMAScript 6:

```
var numeros = [1, 2, 3, 4, 5, 6];
var cuadrados = numeros.map( numero => numero * numero );

 // [1, 4, 9, 16, 25, 36]
```

Array.prototype.filter()

La función filter filtrará los elementos de un array, devuelve un nuevo array con los ítems que pasen el filtro. Este filtro será el resultado del Callback que pasemos como parámetro, que siempre devolverá true o false.

```
Arr.filter(callback[, thisArg])

[1, 2, 3, 4].filter( function(valorActual, índice, array{……..})
```

Veamos como lo haríamos con el método tradicional:

```
var numeros = [1, 2, 3, 4, 5, 6];
var mayoresA3  = [];
for (var i = 0; i < numeros.length; i++)
 {
     if(números[i] > 3)
      mayoresA3[i] = numeros[i];
 }

// [4, 5, 6]
```

Array.prototype.filter() II

```
var numeros = [1, 2, 3, 4, 5, 6];

var mayoresA3 = numeros.filter( function (numero) {

 return numero > 3;

 });

 // [4, 5, 6]
```

Con ECMAScript 6:

```
var numeros = [1, 2, 3, 4, 5, 6];
var mayoresA3 = numeros.filter( numero => numero > 3);

 // [4, 5, 6]
```

Array.prototype.reduce()

La función reduce convierte o reduce un array hasta un único valor por medio de la función pasada como Callback.

```
Arr.reduce(callback[, initialValue])

      arr.reduce( function(valorActual, valorAnterior, índiceActual, array{……..})
```

Veamos como lo haríamos con el método tradicional:

```
      var numeros = [1, 2, 3, 4, 5, 6];
      var total = 0;
      for (var i = 0; i < numeros.length; i++)
       {
         total += numeros[i];
       }

     // 21
```

Array.prototype.reduce() II

```
var numeros = [1, 2, 3, 4, 5, 6];

var total = numeros.reduce( function (previo, actual) {

 return previo + actual;

 }, 0);

 // 21
```

Con ECMAScript 6:

```
var numeros = [1, 2, 3, 4, 5, 6];
var total = numeros.reduce( (previo, actual) => previo + actual, 0 );

 // 21
```

Encadenando funciones

```
var mascotas = [
{ nombre: 'Bobby', tipo: 'perro', age: 4 },
{ nombre: 'Morgan', tipo: 'perro', age: 2 },
{ nombre: 'Michi', tipo: 'gato', age: 3 },
{ nombre: 'Billy', tipo: 'perro', age: 1 },
{ nombre: 'Rafael', tipo: 'iguana', age: 2 }
 ];
```

Supongamos que tenemos que calcular el total de edad de los perros:

```
var TotalEdadPerros = mascotas
  .filter( function(mascota){ mascota.tipo === 'perro'})
  .map( function(mascota){ mascota.edad})
  .reduce( function(previo, actual){ previo + actual}, 0);

// 7
```