Material descargable

Javascript: Arrays - filter, Arrays-map

egg

👋 Introducción

Bienvenido a un nuevo paso del curso de javascript. En esta ocasión vamos a enfocarnos en aprender qué es el método `filter` y map y para qué sirve cada uno.

Teoría de `filter`

El método `filter` se utiliza para crear un **nuevo** array con todos los elementos que pasan la prueba implementada por la función proporcionada. La sintaxis básica del método filter es la siguiente:

```
Unset
array.filter((currentValue, index, arr) => {}, thisValue)
```

- `array`: Es el array sobre el cual se llama el método filter.
- `function(currentValue, index, arr)`: Es la función de prueba para cada elemento del array.
    - `currentValue`: Es el valor del elemento actual que está siendo procesado en el array.
    - `index` (opcional): Es el índice del elemento actual en el array.
    - `arr` (opcional): Es el array al que pertenece el elemento actual.
    - Se ejecuta por cada valor del array.
    - Debe retornar una expresión que se resuelva a un valor booleano (true - false).
- `thisValue` (opcional) (casi nunca usado): Es un valor que se pasa a la función como su valor `this`. Si este parámetro está vacío, se utiliza el valor undefined como su valor this.

El método `filter` **NO** cambia el array original sino que devuelve un **nuevo** array con los elementos que pasan la prueba. Si ningún elemento pasa la prueba, se devuelve un array vacío. En caso contrario si todos los elementos pasan la prueba, se devuelve el array completo.

Generalmente, el primer parámetro (función) es una función anónima de tipo flecha como se mostró anteriormente, pero tranquilamente puede ser una función nombrada.

Ejemplo 1:

```
Unset
const numbers = [1, 2, 3, 4, 5];

const evenNumbers = numbers.filter(number => {

 return number % 2 === 0 // Expresión booleana

});

console.log(evenNumbers); // [2, 4]

console.log(numbers); // [1, 2, 3, 4, 5]
```

En este ejemplo, se filtran los números pares.

> Nótese que el array `numbers` **NO** cambia su valor.

Ejemplo 2:

```
Unset
const numbers = [1, 2, 3, 4, 5];

const biggerThan7 = numbers.filter(number => number > 7); // Si la función anónima tiene una sola linea, puede ir sin `{}` y sin `return`, ya que este, estará implícito.

console.log(biggerThan7); // []
```

```
console.log(numbers); // [1, 2, 3, 4, 5]
```

En este ejemplo, ningún número pasa la prueba, devolviendo un array vacío.

Ejemplo 3:

```
Unset
const numbers = [1, 2, 3, 4, 5];

const positives = numbers.filter(number => number >= 0);

console.log(positives); // [1, 2, 3, 4, 5]

console.log(numbers); // [1, 2, 3, 4, 5]
```

En este ejemplo, todos los números pasan la prueba, devolviendo el mismo array.

Ejemplo 4:

Este último ejemplo muestra cómo filtrar un array de objetos.

```
Unset
const personas = [

  { nombre: "Ana", edad: 25, ciudad: "Madrid" },

  { nombre: "Juan", edad: 16, ciudad: "Barcelona" },

  { nombre: "Luis", edad: 17, ciudad: "Madrid" },

  { nombre: "Sofía", edad: 17, ciudad: "Valencia" },

  { nombre: "Carlos", edad: 22, ciudad: "Madrid" }

];
```

```
const mayoresMadrid = personas.filter(persona => { // Tener en
cuenta que `persona` es un objeto.
  // Condiciones para filtrar
  const esMayorDeEdad = persona.edad > 18;
  const esDeMadrid = persona.ciudad === "Madrid";

  // Retornar true si cumple ambas condiciones
  return esMayorDeEdad && esDeMadrid;
});

console.log(mayoresMadrid);

/*
[
 { nombre: "Ana", edad: 25, ciudad: "Madrid" },
 { nombre: "Carlos", edad: 22, ciudad: "Madrid" }
]
*/
```

Este código filtrará el array personas y devolverá un nuevo array que solo contiene las personas que son mayores de 18 años y que viven en Madrid. La función callback dentro de filter es una función de varias líneas que define las condiciones de filtrado.

Teoría de `map`

El método `map` se utiliza para crear un **nuevo** array con los resultados de la llamada a una función proporcionada aplicada a cada elemento del array original. La sintaxis básica del método `map` es la siguiente:

```
Unset
array.map((currentValue, index, arr) => {}, thisValue)
```

`array`: Es el array sobre el cual se llama el método `map`.

`function(currentValue, index, arr)`: Es la función que se aplica a cada elemento del array.

- `currentValue`: Es el valor del elemento actual que está siendo procesado en el array.
- `index` (opcional): Es el índice del elemento actual en el array.
- `arr` (opcional): Es el array al que pertenece el elemento actual.
- Se ejecuta por cada valor del array.
- Debe retornar una expresión.
- `thisValue` (opcional) (casi nunca usado): Es un valor que se pasa a la función como su valor this. Si este parámetro está vacío, el valor undefined se utiliza como su valor this.

El método `map` **NO** cambia el array original sino que devuelve un **nuevo** array con los resultados de aplicar la función a cada elemento.

Ejemplo 1:

```
Unset
const numbers = [1, 2, 3, 4, 5];

const doubled = numbers.map(number => {

  return number * 2 // Retorna una expresión.

});
```

```
console.log(doubled); // [2, 4, 6, 8, 10]

console.log(numbers); // [1, 2, 3, 4, 5]
```

En este ejemplo, cada número en el array se multiplica por 2.

Ejemplo 2:

```
Unset
const personas = [

  { nombre: "Ana", edad: 25 },

  { nombre: "Juan", edad: 16 },

  { nombre: "Luis", edad: 30 }

];


const nombres = personas.map(persona => persona.nombre) //
`return` implícito.


console.log(nombres) // ["Ana", "Juan", "Luis"]
```

En este ejemplo, obtengo solo los nombres de cada objeto en el array.

Ejemplo 3:

```
Unset
const personas = [

  { nombre: "Ana", edad: 25 },

  { nombre: "Juan", edad: 16 },
```

```
  { nombre: "Luis", edad: 30 }
];

const personasConMayoríaDeEdad = personas.map(persona => {
  return {
    ...persona,
    esMayorDeEdad: persona.edad >= 18
  };
});

console.log(personasConMayoríaDeEdad);

/*
  [
    { nombre: "Ana", edad: 25, esMayorDeEdad: true },
    { nombre: "Juan", edad: 16, esMayorDeEdad: false },
    { nombre: "Luis", edad: 30, esMayorDeEdad: true }
  ]
*/
```

Este código devuelve un nuevo array como `personas`, agregando un nuevo campo `esMayorDeEdad` a cada objeto.

La sintaxis `...persona` (`spread operator`) dentro del `map`, sirve para copiar el objeto completo `persona`, para luego agregar un nuevo campo (En este caso `esMayorDeEdad`).