JavaScript 2015 (III). Interpolación y mapas.

En este artículo vamos a conocer dos nuevas aportaciones del estándar 2015 de JavaScript para el manejo de datos. Se trata de las interpolaciones y los mapas. Ambos se refieren al manejo de objetos y de arreglos, es decir, variables complejas o colecciones de variables.
Estas técnicas (o sus equivalentes similares) existen, desde hace mucho tiempo, en otros lenguajes como PHP o Java. Sin embargo, en JavaScript se echaban en falta. La nueva normalización llena completamente este vacío, si bien emplea una sintaxis peculiar. No sabemos si, en posteriores versiones del lenguaje, la sintaxis se aproximará más a la de otros lenguajes, o se mantendrá como propia de Javascript (aunque todo apunta a esto último). Nosotros vamos a conocer lo que tenemos ahora y si, en el futuro evoluciona, nos adaptaremos (aunque, insisto, la tendencia parece ser a mantener la notación actual).

INTERPOLACIONES
Manteniendo el sistema de notación de objetos de JavaScript (sí, el famoso JSON, sobre el que puedes leer en este artículo) podemos creear cualquier objeto con las variables que deseemos. Lo que vamos a conocer ahora es una forma de acceder a las variables de un objeto en una notación similar a la de Angular JS, que se conoce como interpolación. Por cierto. Para los que os acordéis de Flash, no tiene nada que ver. La coincidencia del nombre es meramente casual.
Observa el siguiente listado:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>JS 6</title>
</head>
<body>
 <div id="capaDeResultados"></div>
 <script language="javascript">
  var persona = {nombre: "Pedro", apellidos: "García Gómez", nacimiento: "12-01-1954", residencia: "Murcia"};
  document.getElementById("capaDeResultados").innerHTML = `${persona.nombre} ${persona.apellidos} nació el
${persona.nacimiento} y vive en ${persona.residencia}.`;
 </script>
</body>
</html>
```

El resultado que ves en pantalla es:
Pedro García Gómez nació el 12-01-1954 y vive en Murcia.
Fíjate en la línea 10. Definimos una colección de variables englobadas en un objeto según la notación de objetos de JavaScript. En la línea 11 creamos una frase en la que insertamos, en los puntos donde lo necesitamos, las variables del objeto. Y aquí viene la peculiaridad de la sintaxis de JS6. La frase debe estar encerrada entre acentos graves. Los delimitadores que ves en los extremos de la frase no son comillas simples, sino el acento grave (`). Además, observa que nos referimos a cada variable con la notación ${objeto.variable}, es decir, la precedemos con el signo $ y la encerramos entre llaves. Cómo ves, esto recuerda un poco a la expansión de variables PHP, aunque la sintaxis es diferente.
Por cierto. Si estás pensando que el uso del signo $ puede suponer algún conflicto si tu página emplea jQuery, olvídalo. Al ir todo el

contenido acotado por acentos graves, queda perfectamente diferenciado el uso del signo $ en cada contexto. De hecho, observa esta versión del código anterior:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>JS 6</title>
</head>
<body>
 <div id="capaDeResultados"></div>
 <script language="javascript" src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
 <script language="javascript">
  var persona = {nombre: "Pedro", apellidos: "García Gómez", nacimiento: "12-01-1954", residencia: "Murcia"};
  $("#capaDeResultados").html(`${persona.nombre} ${persona.apellidos} nació el ${persona.nacimiento} y vive en
${persona.residencia}.`);
 </script>
</body>
</html>
```

Si la pruebas, verás que funciona exactamente igual, sin ningún conflicto.

Por supuesto, podemos usar la interpolación de variables no solo sobre las de un único objeto, sino, también, sobre las varibles de un arreglo de objetos, sin más que introducir el índice del objeto al que nos referimos en cada caso. Observa el siguiente código:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>JS 6</title>
</head>
<body>
 <div id="capaDeResultados"></div>
 <script language="javascript">
  var personas = [
   {nombre: "Pedro", apellidos: "García Gómez", nacimiento: "12-01-1954", residencia: "Murcia"},
   {nombre: "María", apellidos: "Echevarría Fuentes", nacimiento: "01-05-1967", residencia: "Bilbao"},
   {nombre: "Laura", apellidos: "Torres Borrego", nacimiento: "26-02-1986", residencia: "Jaén"},
   {nombre: "Luis", apellidos: "Martín De La Hoz", nacimiento: "15-07-1975", residencia: "Madrid"}
  ];
  for (var i in personas){
   document.getElementById("capaDeResultados").innerHTML += `${personas[i].nombre} ${personas[i].apellidos} nació el
${personas[i].nacimiento} y vive en ${personas[i].residencia}.<br>`;
  }
 </script>
</body>
</html>
```

El resultado en pantalla es el siguiente:

Pedro García Gómez nació el 12-01-1954 y vive en Murcia.
María Echevarría Fuentes nació el 01-05-1967 y vive en Bilbao.
Laura Torres Borrego nació el 26-02-1986 y vive en Jaén.
Luis Martín De La Hoz nació el 15-07-1975 y vive en Madrid.

Fíjate en la notación empleada en la interpolación en la línea 17: ${objeto[indice.variable]}. Cómo el índice recorre tódo el objeto personas, en cada caso se obtienen los valores de las variables de uno de los elementos.

MAPAS

Los mapas son estructuras de datos creados a partir de la clase Map, nativa de JS6. A estas estructuras se le pueden añadir propiedades (variables, para entendernos), recuperar el valor de estas, comprobar si existen y eliminarlas, si ya no nos hacen falta. De hecho, su estructura recuerda a las matrices asociativas de PHP. Lo verás claro en el siguiente código:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>JS 6</title>
</head>
<body>
 <div id="capaDeResultados"></div>
 <script language="javascript">
  var persona = new Map();
  persona.set("nombre", "Pedro");
  persona.set("apellidos", "García Gómez");
  persona.set("nacimiento", "12-01-1954");
  persona.set("residencia", "Murcia");
  console.log (persona);
  document.getElementById("capaDeResultados").innerHTML += "Nombre: " + persona.get("nombre") + "<br>";
  document.getElementById("capaDeResultados").innerHTML += "Apellidos: " + persona.get("apellidos") + "<br>";
  document.getElementById("capaDeResultados").innerHTML += "F. Nacimiento: " + persona.get("nacimiento") + "<br>";
  document.getElementById("capaDeResultados").innerHTML += "Vive en: " + persona.get("residencia") + "<br>";
 </script>
</body>
</html>
```

El resultado en pantalla es el siguiente:

```
Nombre: Pedro
 Apellidos: García Gómez
 F. Nacimiento: 12-01-1954
 Vive en: Murcia
```

Mientras que en la consola vemos más definidamente la estructura del mapa:

```
Map {"nombre" => "Pedro", "apellidos" => "García Gómez", "nacimiento" => "12-01-1954", "residencia" => "Murcia"}
```

Lo que hacemos es crear el mapa, cómo un objeto vacío. Esto lo hacemos con la línea 10:

```
var persona = new Map();
```

Los mapas tienen el método set() para asignarles propiedades. Este método recibe el nombre de la propiedad y, separado con el signo de dos puntos, el valor que le queremos dar a la misma. Lo puedes ver en las líneas de la 11 a la 14.

Además, los mapas cuentan con el método get(), que permite recuperar el valor de una propiedad, pasándole, como argumento, el nombre de la misma. Puedes ver cómo lo usamos en las líneas de la 16 a la 19.

Los mapas son vistos por JavaScript como objetos, ya que JavaScript ve como un objeto, en el sentido estricto de la palabra, cualquier estructura de datos (un mapa, una matriz, etc.). Sin embargo, en un mapa no podemoa definir métodos, sino que sólo podemos usar los que ya existen con la clase Map.

Además de los dos métodos que ya hemos visto, contamos con otros dos muy interesantes:

El primero es el método has(). Este recibe, cómo argumento, el nombre de una propiedad y nos devuelve true si la mencionada propiedad existe en el mapa. Si no existe, nos devuelve false.

El segundo es delete(). Recibe, como argumento, el nombre de una propiedad, y la elimina del mapa. Si se ha podido elimiinar correctamente, devuelve true. En caso de que no se haya podido eliminar (por ejemplo, porque no exista tal propiedad en el mapa), devolverá false.

Observa el siguiente código:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>JS 6</title>
</head>
<body>
 <div id="capaDeResultados"></div>
 <script language="javascript">
  var persona = new Map();
  persona.set("nombre", "Pedro");
  persona.set("apellidos", "García Gómez");
  persona.set("nacimiento", "12-01-1954");
  persona.set("residencia", "Murcia");
  console.log ("El mapa recién creado, y asignadas sus propiedades: ", persona);
  /* Comprobamos si existe la propiedad "nombre" */
  console.log ("Existe la propiedad 'nombre': ", persona.has("nombre"));
  console.log ("El valor de la propiedad nombre es: ", persona.get("nombre"));
  /* Intentamos eliminar la propiedad nombre, y vemos que se puede hacer. */
  console.log ("Eliminar la propiedad 'nombre': ", persona.delete("nombre"));
  console.log ("Existe la propiedad 'nombre': ", persona.has("nombre"));
  console.log ("El mapa sin la propiedad 'nombre': ", persona);
  /* Intentamos eliminar la propiedad nombre, y vemos que no se puede hacer, porque ya no existe. */
  console.log ("Eliminar la propiedad 'nombre': ", persona.delete("nombre"));
 </script>
</body>
</html>
```

El resultado que obtenemos en la consola del navegador es el siguiente:

```
El mapa recién creado, y asignadas sus propiedades:  Map {"nombre" => "Pedro", "apellidos" => "García Gómez", "nacimiento" => "12-01-1954", "residencia" => "Murcia"}
 Existe la propiedad 'nombre':  true
 El valor de la propiedad nombre es:  Pedro
 Eliminar la propiedad 'nombre':  true
 Existe la propiedad 'nombre':  false
 El mapa sin la propiedad 'nombre':  Map {"apellidos" => "García Gómez", "nacimiento" => "12-01-1954", "residencia" => "Murcia"}
 Eliminar la propiedad 'nombre':  false
```

Si quieres saber más sobre el uso de la consola del navegador (personalmente, te recomiendo usar siempre alguna edición de Chrome), puedes echarle un vistazo a este artículo, y a este otro.