JavaScript – Almacenamiento web HTML

Introducción

Con el almacenamiento web HTML las aplicaciones web pueden almacenar datos localmente dentro del navegador del usuario.

Antes de HTML5, los datos de la aplicación tenían que almacenarse en cookies.

El almacenamiento web es más seguro y se pueden almacenar grandes cantidades de datos localmente, sin afectar el rendimiento del sitio web.

A diferencia de las cookies, el límite de almacenamiento es mucho mayor (al menos 5 MB), y la información nunca se transfiere al servidor.

El almacenamiento web queda definido por su origen (por su dominio y el protocolo utilizado). Si el usuario cambia de página, el almacén de datos es distinto.

Objetos de almacenamiento web HTML

El almacenamiento web HTML proporciona dos objetos para almacenar datos en el cliente:

- `window.localStorage`: almacena datos sin fecha de caducidad.

- `window.sessionStorage`: almacena datos para una sesión (los datos se pierden cuando se cierra la pestaña del navegador).

Antes de usar el almacenamiento web es aconsejable comprobar la compatibilidad del navegador con localStorage y sessionStorage:

```
if (typeof(Storage) !== "undefined") {
  // Code for localStorage/sessionStorage.
} else {
  // Sorry! No Web Storage support..
}
```

Métodos y propiedades disponibles:

En la siguiente tabla se describen los métodos y propiedades para el objeto `window.sessionStorage`. El objeto `window.localStorage` utiliza los mismos.

Método o propiedad de sessionStorage	Descripción
`sessionStorage.setItem('clave', 'valor');`	Guarda la información **valor** a la que se podrá acceder invocando a clave. Por ejemplo, *clave* puede ser nombre y *valor* puede ser Carlos.
`sessionStorage.getItem('clave')`	Recupera el *value* de la clave especificada. Por ejemplo, si *clave* es nombre puede recuperar "Carlos".
`sessionStorage[clave]=valor`	Igual que setItem
`sessionStorage.length`	Devuelve el número de items guardados por el objeto sessionStorage actual.
`sessionStorage.key(i)`	Cada item se almacena con un índice que comienza por cero y se incrementa unitariamente por cada item añadido. Con esta sintaxis rescatamos la clave correspondiente al item con índice i.
`sessionStorage.removeItem(clave)`	Elimina un item almacenado en sessionStorage
`sessionStorage.clear()`	Elimina todos los items almacenados en sessionStorage, quedando vacío el espacio de almacenamiento.

El objeto de almacenamiento local (localStorage)

El objeto **localStorage** almacena los datos sin fecha de caducidad. Los datos no se eliminarán cuando se cierre el navegador y estarán disponibles al día, semana o año siguiente.

Almacenar y recuperar información

Se utilizan los métodos `localStorage.setItem()` y `localStorage.getItem()`

Los pares de nombre/valor siempre se almacenan como cadenas. Hay que convertirlos a otro formato cuando sea necesario.

Ejemplo para:

- Crear un par de nombre/valor de almacenamiento local con nombre="apellido" y valor="Smith"
- Recuperar el valor de "apellido" e insértelo en el elemento con id="resultado"

```
// Store
// Crear un par de nombre/valor de almacenamiento local con nombre="apellido" y valor="Peláez"
localStorage.setItem("apellido", "Peláez");

// Retrieve
// Recuperar el valor de "apellido" e insertarlo en el elemento con id="resultado"
document.getElementById("resultado").innerHTML = localStorage.getItem("apellido");
```

El ejemplo anterior también podría escribirse así:

```
// Store
localStorage.apellido = "Peláez";
// Retrieve
document.getElementById("resultado").innerHTML = localStorage.apellido;
```

Eliminar información

Se utiliza el método `localStorage.removeItem()`. La sintaxis para eliminar el elemento localStorage "apellido" es la siguiente:

```
localStorage.removeItem("apellido");
```

Eliminar todos los datos

Para eliminar todos los datos y dejar limpio el almacenamiento local de nuestro dominio y protocolo (origen):

```
localStorage.clear();
```

Ejemplo:

Ejemplo que cuenta el número de veces que un usuario ha hecho clic en un botón. En este código, la cadena de valor se convierte a Number para poder incrementar el contador:

```
if (localStorage.clickcount) {
  localStorage.clickcount = Number(localStorage.clickcount) + 1;
} else {
  localStorage.clickcount = 1;
}

document.getElementById("resultado").innerHTML = "You have clicked the button " +
localStorage.clickcount + " time(s).";
```

El objeto de almacenamiento (sessionStorage)

El objeto `sessionStorage` es igual al objeto localStorage, excepto que almacena los datos para una sola sesión. Los datos se eliminan cuando el usuario cierra la pestaña específica del navegador.

Se utilizan los mismos métodos que en el objeto localStorage.

Ejemplo:

El siguiente ejemplo cuenta la cantidad de veces que un usuario ha hecho clic en un botón en la sesión actual:

```
if (sessionStorage.clickcount) {
  sessionStorage.clickcount = Number(sessionStorage.clickcount) + 1;
} else {
  sessionStorage.clickcount = 1;
}
document.getElementById("resultado").innerHTML = "You have clicked the button " +
sessionStorage.clickcount + " time(s) in this session.";
```

Ejercicio: escribe el código para guardar automáticamente el contenido de un campo de texto y, si se actualiza el navegador, restaurar el contenido del campo de texto para que no se pierda lo que ya tiene escrito.

Persistencia de la información almacenada.

Hay que tener en cuenta que, si un usuario realiza una limpieza de la caché del navegador, hará que se borren los datos almacenados con localStorage.

Si el usuario no limpia la caché, los datos se mantendrán durante mucho tiempo. En cambio, hay usuarios que tienen configurado el navegador para que la caché se limpie en cada ocasión en que cierren el navegador. En este caso la persistencia que ofrece localStorage es similar a la que ofrece sessionStorage.

No podemos confiar el funcionamiento de una aplicación web a que el usuario limpie o no limpie la caché, por tanto, deberemos seguir trabajando con datos del lado del servidor siempre que deseemos obtener una persistencia de duración indefinida.

El evento storage

localStorage permite que se reconozcan datos desde distintas ventanas.

Para detectar que en una ventana que se ha producido un cambio en los datos se definió el evento **storage**: este evento se dispara cuando tiene lugar un cambio en el espacio de almacenamiento y puede ser detectado por las distintas ventanas que estén abiertas.

Para crear una respuesta a este evento podemos escribir:

```
window.addEventListener("storage", nombreFuncionRespuesta, false);
```

Donde nombreFuncionRespuesta es el nombre de la función que se invocará cuando se produzca el evento.

Diferencias entre cookies y storage

Los objetos storage juegan un papel similar a las cookies, pero por otro lado hay diferencias importantes:

- Las cookies están disponibles tanto en el servidor como en el navegador del usuario. Los objetos storage sólo están disponibles en el navegador del usuario.

- Las cookies se concibieron como pequeños paquetes de identificación, con una capacidad limitada (unos 4 Kb). Los objetos storage se han concebido para almacenar datos a mayor escala (pudiendo comprender cientos o miles de datos con un espacio de almacenamiento de varios Mb).

Hay que tener en cuenta que, de una forma u otra, ni las **cookies** ni los objetos **storage** están pensados para el almacenamiento de grandes volúmenes de información, sino para la gestión de los flujos de datos propios de la navegación web.

Ejemplo: Almacenar un array de objetos con JSON y recuperarlo

Archivos index.html y clasePersona.js completos:

Tenemos un archivo index.html y otro con la clase Persona

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <script type ="module"
src="js/app.js"></script>
</body>
</html>
```

```
export class Persona{
    constructor(nombre, apellido){
        this._nombre = nombre;
        this._apellido = apellido;
    }
    get nombre(){
        return this._nombre;
    }
    set nombre(nombre){
        this._nombre = nombre;
    }
    get apellido(){
        return this._apellido;
    }
    set apellido(apellido){
        return this._apellido = apellido;
    }
}
```

Se trata de crear objetos de la clase Persona, guardarlos como cadena y recuperarlos como array.

Vamos a utilizar los métodos

- JSON.stringify()
- JSON.parse()

```
//Guardar el objeto personas en localStorage
localStorage.setItem('personasCadena',JSON.stringify(personas));
```

```
// recuperar de localStorage en crudo (texto sin más)
let personasAlmacenadas = localStorage.getItem('personasCadena');
```

```
// recuperar de localStorage parseando (obteniendo el array de objetos)
// let personas2 = JSON.parse(localStorage.getItem('personasCadena'));
```

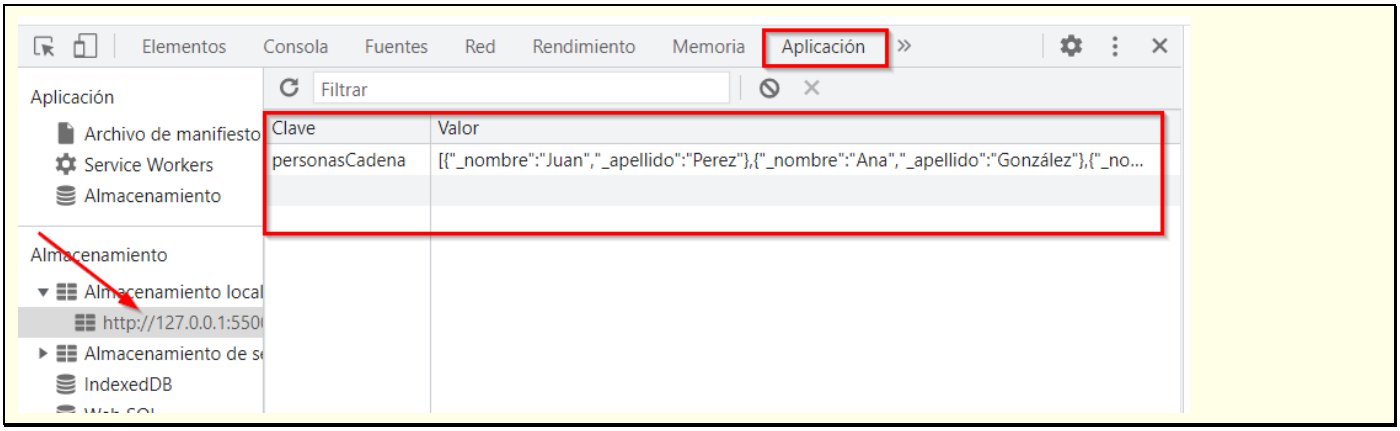
```
let personas2=JSON.parse(personasAlmacenadas);
```

Archivo app.js completo:

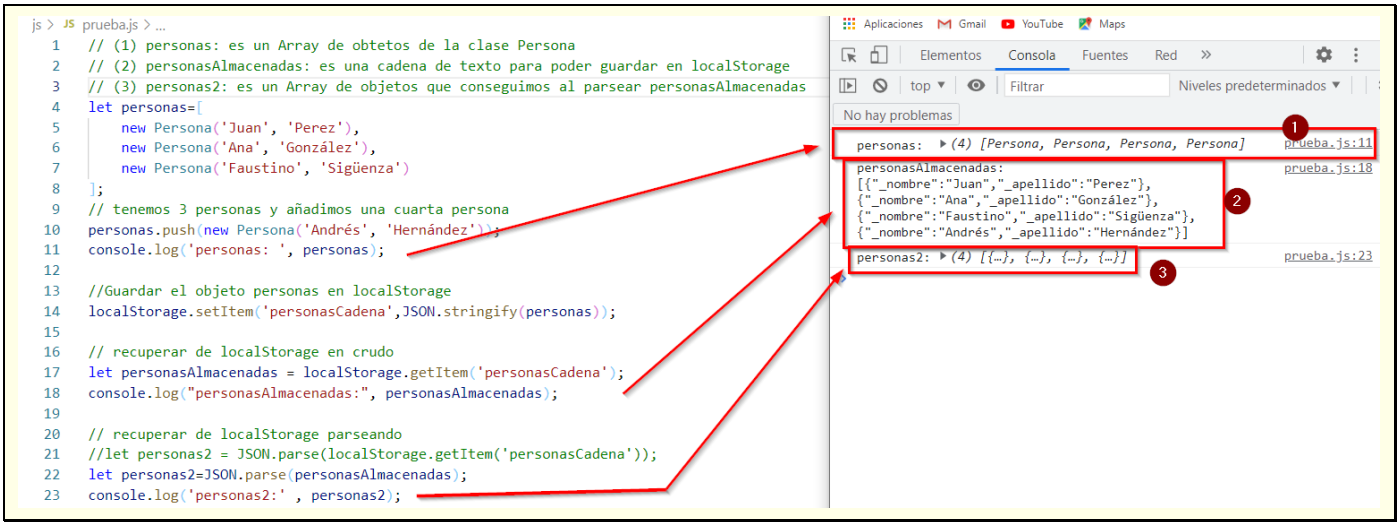
```
import {Persona} from './clases/clasePersona.js';

// (1) personas: es un array de objetos de la clase persona
// (2) personasAlmacenadas: es una cadena de texto para poder guardar en localStorage
// (3) personas2: es un array de objetos que conseguimos al parsear personasAlmacenadas

let personas=[
    new Persona('Juan', 'Pérez'),
    new Persona('Ana', 'Gónzález'),
    new Persona('Faustino', 'Sigüenza')
];

// Tenemos 3 personas y añadimos una cuarta
personas.push(new Persona('Andrés', 'Hernández'));
console.log('personas: ', personas);

// Guardamos el objeto personas en localStorage
localStorage.setItem('personasCadena', JSON.stringify(personas));

// Recuperamos de localStorage en crudo, como texto
let personasAlmacenadas = localStorage.getItem('personasCadena');
console.log("personasAlmacenadas: ", personasAlmacenadas);

// Recuperamos de localStorage parseando
//let personas2 = JSON.parse(localStorage.getItem('personasCadena'))
let personas2 = JSON.parse(personasAlmacenadas);
console.log('personas2:', personas2);
```

Inspeccionando el almacenamiento en localStorage:

Se puede visualizar, editar y eliminar el contenido de localStorage inspeccionando el documento:

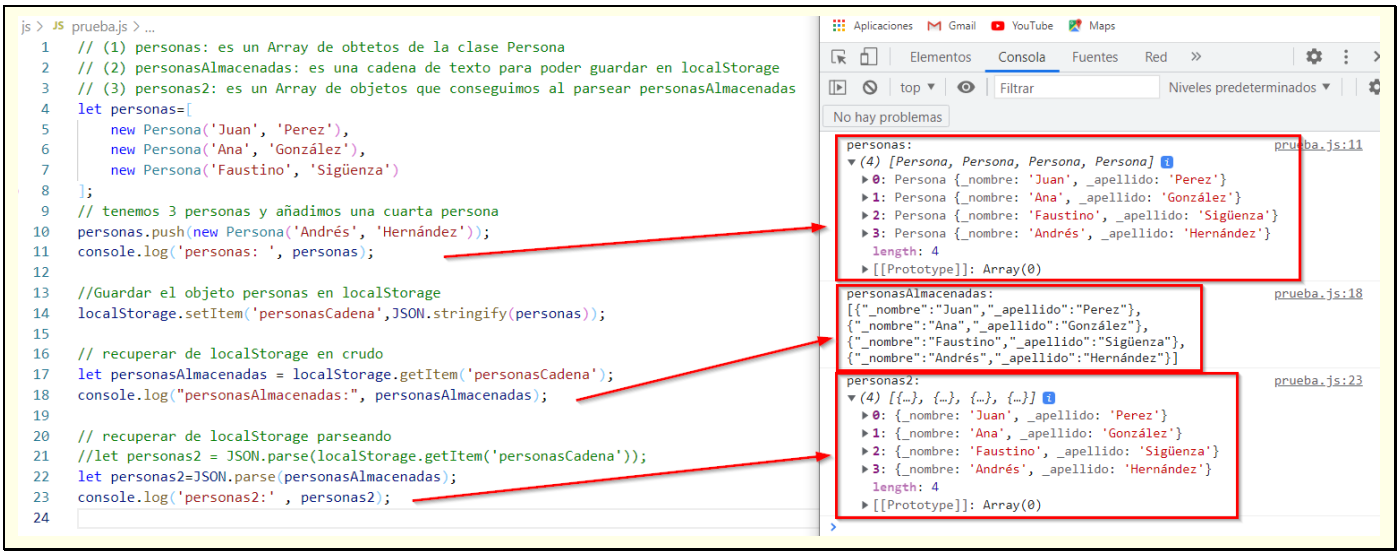


Mirando en la consola:



Problemilla:

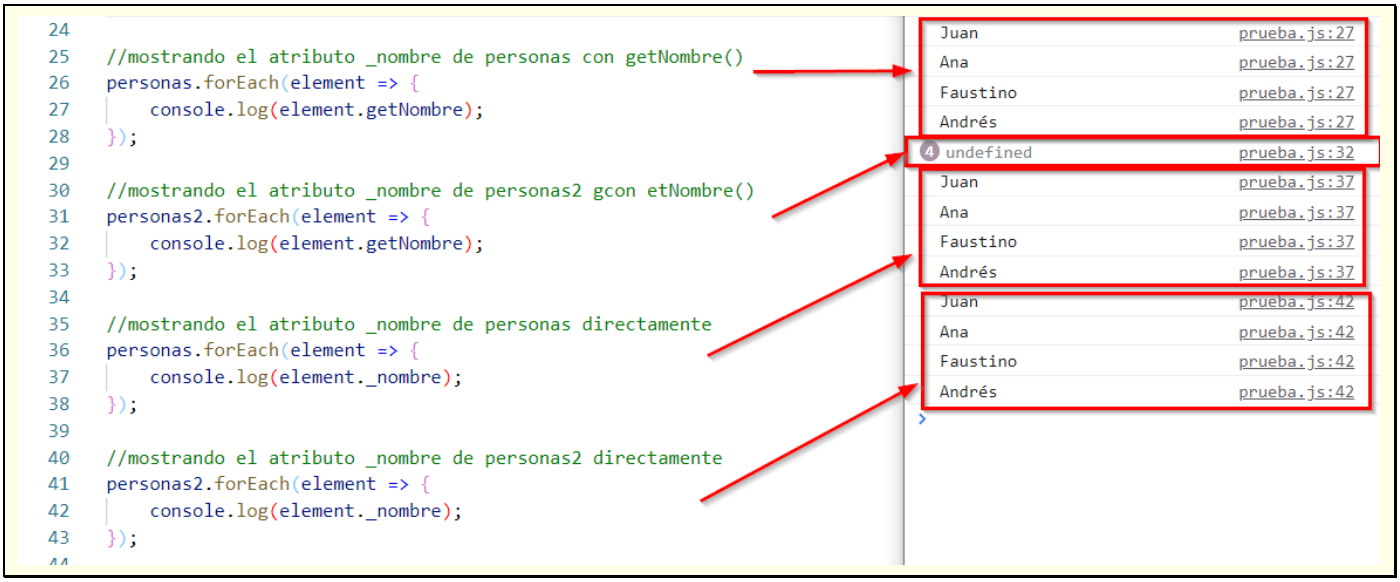
Desplegando los arrays en la consola observamos:



Tenemos un pequeño problema:

- Guardamos un array de objetos de la clase Persona (el llamado **personas**)
- Recuperamos un array de objetos sin clase (al que hemos llamado **personas2**)

Los datos son los mismos, pero el segundo array (personas2) no puede utilizar los métodos de la clase.

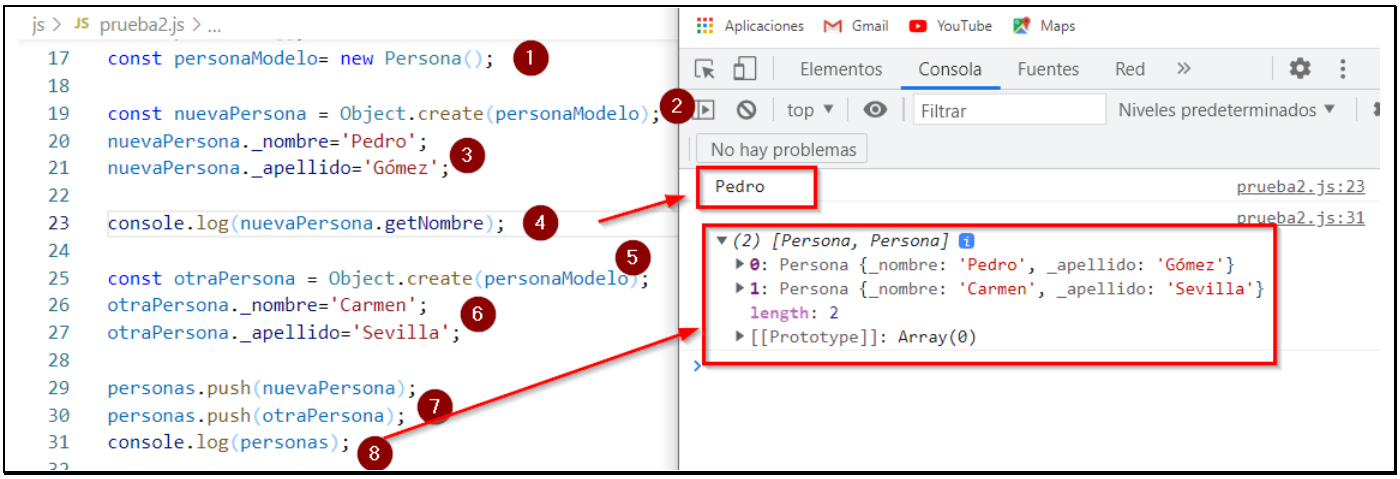


En el ejemplo anterior no se pueden utilizar los métodos get() para obtener los valores de los atributos de los elementos en personas2, debemos obtener los atributos directamente.

Usando Object.create()

Para resolver el caso anterior, podemos utilizar Object.create(*objetoModelo*) para añadir a un array **personas3** los objetos de la clase Persona que hemos recuperado de localStorage.

A continuación, ponemos un ejemplo:



(1) Creamos un objeto modelo de la clase que queramos tener. No hace falta que tenga datos.
(2) Creamos un nuevo objeto utilizando el objeto modelo existente.
(3) (6) Asignamos contenido a los atributos.
(4) Ya podemos utilizar los métodos de la clase.
(5) Podemos crear más objetos de la clase.
(7) Metemos los objetos de la clase Persona en el array personas
(8) Comprobamos el contenido del array.

Obteniendo el array de objetos de la clase Persona:

Se va a resolver el problemilla:

Archivo app.js resultante:

```
import {Persona} from './clases/clasePersona.js';
```

```
// (1) personas: es un array de objetos de la clase persona
// (2) personasAlmacenadas: es una cadena de texto para poder guardar en localStorage
// (3) personas2: es un array de objetos que conseguimos al parsear personasAlmacenadas

let personas=[
    new Persona('Juan', 'Pérez'),
    new Persona('Ana', 'Gónzález'),
    new Persona('Faustino', 'Sigüenza')
];

// Tenemos 3 personas y añadimos una cuarta
personas.push(new Persona('Andrés', 'Hernández'));
console.log('personas: ', personas);

// Guardamos el objeto personas en localStorage
localStorage.setItem('personasCadena', JSON.stringify(personas));

// Recuperamos de localStorage en crudo, como texto
let personasAlmacenadas = localStorage.getItem('personasCadena');
console.log("personasAlmacenadas: ", personasAlmacenadas);

// Recuperamos de localStorage parseando
//let personas2 = JSON.parse(localStorage.getItem('personasCadena'))
let personas2 = JSON.parse(personasAlmacenadas);
console.log('personas2:', personas2);

// Recuperar el array de objetos de la clase Persona:
// const personaModelo = new Persona(); // (**)
const personas3=[];

personas2.forEach((p) =>{
    // let nuevaPersona= Object.create(personaModelo); // (**)
    let nuevaPersona=new Persona() // (**)
    nuevaPersona.nombre = p._nombre;
    nuevaPersona.apellido = p.apellido;
    console.log(nuevaPersona);
    personas3.push(nuevaPersona);
})

console.log('personas3: ',personas3);
```

(**) Se puede optar por:

a) Utilizar personaModelo como modelo de nuevaPersona, o
b) Crear nuevaPersona como una instancia de la clase Persona.