

Almacenando datos en el navegador: `LocalStorage`, `sessionStorage`

1

2º DAW - Desarrollo Web en Entorno Cliente

Almacenando datos en el navegador

LocalStorage, sessionStorage

- JavaScript puede enviar peticiones de red al servidor y cargar nueva información siempre que se necesite. Por ejemplo, Los objetos de almacenaje web localStorage y sessionStorage permiten guardar pares de clave/valor en el navegador. Lo que es interesante sobre ellos es que los datos sobreviven a una recarga de página (en el caso de sessionStorage) y hasta un reinicio completo de navegador (en el caso de localStorage).
- Ya tenemos cookies. ¿Por qué tener objetos adicionales? Al contrario que las cookies, los objetos de almacenaje web no se envían al servidor en cada petición. Debido a esto, podemos almacenar mucha más información. La mayoría de los navegadores modernos permiten almacenar, como mínimo, 5 megabytes de datos y tienen opciones para configurar estos límites.
- También diferente de las cookies es que el servidor no puede manipular los objetos de almacenaje via cabeceras HTTP, todo se hace via JavaScript. El almacenaje está vinculado al origen (al triplete dominio/protocolo/puerto). Esto significa que distintos protocolos o subdominios tienen distintos objetos de almacenaje, no pueden acceder a otros datos que no sean los suyos.
- Ambos objetos de almacenaje proveen los mismos métodos y propiedades:
 - `setItem(clave, valor)` – almacenar un par clave/valor.
 - `getItem(clave)` – obtener el valor por medio de la clave.
 - `removeItem(clave)` – eliminar la clave y su valor.
 - `clear()` – borrar todo.
 - `key(índice)` – obtener la clave de una posición dada.
 - `length` – el número de ítems almacenados.
- Como puedes ver, es como una colección Map (`setItem/getItem/removeItem`), pero también permite el acceso a través de index con `key(index)`.

Demo de localStorage

Las principales funcionalidades de localStorage son:

- Es compartido entre todas las pestañas y ventanas del mismo origen.
- Los datos no expiran. Persisten a los reinicios de navegador y hasta del sistema operativo.

Por ejemplo, si ejecutas este código...

```
1 localStorage.setItem('test', 1);
```

... y cierras/abres el navegador, o simplemente abres la misma página en otra ventana, puedes coger el ítem que hemos guardado de este modo:

```
1 alert( localStorage.getItem('test') ); // 1
```

Solo tenemos que estar en el mismo dominio/puerto/protocolo, la url puede ser distinta.

localStorage es compartido por todas las ventanas del mismo origen, de modo que si guardamos datos en una ventana, el cambio es visible en la otra.

Acceso tipo Objeto

También podemos utilizar un modo de acceder/guardar claves del mismo modo que se hace con objetos, así:

```
1 // guarda una clave
2 localStorage.test = 2;
3
4 // coge una clave
5 alert( localStorage.test ); // 2
6
7 // borra una clave
8 delete localStorage.test;
```

Esto se permite por razones históricas, y principalmente funciona, pero en general no se recomienda por dos motivos:

1. Si la clave es generada por el usuario, puede ser cualquier cosa, como `length` o `toString`, u otro método propio de `localStorage`. En este caso `getItem/setItem` funcionan correctamente, pero el acceso de simil-objeto falla;

```
1 let key = 'length';
2 localStorage[key] = 5; // Error, no se puede asignar 'length'
```

2. Existe un evento ***storage***, que se dispara cuando modificamos los datos. Este evento no se dispara si utilizamos el acceso tipo objeto. Lo veremos más tarde en este capítulo.

Iterando sobre las claves

Los métodos proporcionan la funcionalidad get / set / remove. ¿Pero cómo conseguimos todas las claves o valores guardados?

Desafortunadamente, los objetos de almacenaje no son iterables. Una opción es utilizar iteración sobre un array:

```
1 for(let i=0; i<localStorage.length; i++) {  
2   let key = localStorage.key(i);  
3   alert(`${key}: ${localStorage.getItem(key)}`);  
4 }
```

Otra opción es utilizar el loop específico para objetos for key in localStorage tal como hacemos en objetos comunes. Esta opción itera sobre las claves, pero también devuelve campos propios de localStorage que no necesitamos:

```
1 // mal intento  
2 for(let key in localStorage) {  
3   alert(key); // muestra getItem, setItem y otros campos que no nos interesan  
4 }
```

... De modo que necesitamos o bien filtrar campos des del prototipo con la validación hasOwnProperty:

```
1 for(let key in localStorage) {  
2   if (!localStorage.hasOwnProperty(key)) {  
3     continue; // se salta claves como "setItem", "getItem" etc  
4   }  
5   alert(`${key}: ${localStorage.getItem(key)}`);  
6 }
```

... O simplemente acceder a las claves “propias” con **Object.keys** y iterar sobre éstas si es necesario:

```
1 let keys = Object.keys(localStorage);  
2 for(let key of keys) {  
3   alert(`${key}: ${localStorage.getItem(key)}`);  
4 }
```

Esta última opción funciona, ya que **Object.keys** solo devuelve las claves que pertenecen al objeto, ignorando el prototipo.

Solo strings

Hay que tener en cuenta que tanto la clave como el valor deben ser strings.

Si fueran de cualquier otro tipo, como un número o un objeto, se convertirían a cadena de texto automáticamente:

```
1 localStorage.user = {name: "John"};
2 alert(localStorage.user); // [object Object]
```

A pesar de eso, podemos utilizar **JSON** para almacenar objetos:

```
1 localStorage.user = JSON.stringify({name: "John"});
2
3 // en algún momento más tarde
4 let user = JSON.parse( localStorage.user );
5 alert( user.name ); // John
```

También es posible pasar a texto todo el objeto de almacenaje, por ejemplo para debugear:

```
1 // se ha añadido opciones de formato a JSON.stringify para que el objeto se lea mejor
2 alert( JSON.stringify(localStorage, null, 2) );
```

sessionStorage

El objeto `sessionStorage` se utiliza mucho menos que `localStorage`. Las propiedades y métodos son los mismos, pero es mucho más limitado:

- `sessionStorage` solo existe dentro de la pestaña actual del navegador.
 - Otra pestaña con la misma página tendrá un almacenaje distinto.
 - Pero se comparte entre iframes en la pestaña (asumiendo que tengan el mismo origen).
- Los datos sobreviven un refresco de página, pero no cerrar/abrir la pestaña.

Vamos a verlo en acción. Ejecuta este código...

```
1 sessionStorage.setItem('test', 1);
```

... Y recarga la página. Aún puedes acceder a los datos:

```
1 alert( sessionStorage.getItem('test') ); // después de la recarga: 1
```

... Pero si abres la misma página en otra pestaña, y lo intentas de nuevo, el código anterior devuelve `null`, que significa que no se ha encontrado nada.

Esto es exactamente porque `sessionStorage` no está vinculado solamente al origen, sino también a la pestaña del navegador. Por esta razón `sessionStorage` se usa relativamente poco.

Evento storage

Cuando los datos se actualizan en localStorage o en sessionStorage, se dispara el evento storage con las propiedades:

- key – la clave que ha cambiado, (null si se llama .clear()).
- oldValue – el anterior valor (null si se añade una clave).
- newValue – el nuevo valor (null si se borra una clave).
- url – la url del documento donde ha pasado la actualización.
- storageArea – bien el objeto localStorage o sessionStorage, donde se ha producido la actualización.

El hecho importante es: el evento se dispara en todos los objetos window donde el almacenaje es accesible, excepto en el que lo ha causado. Vamos a desarrollarlo.

Imagina que tienes dos ventanas con el mismo sitio en cada una, de modo que localStorage es compartido entre ellas. Quizá quieras abrir esta página en dos ventanas distintas para probar el código que sigue.

Si ambas ventanas están escuchando el evento **window.onstorage**, cada una reaccionará a las actualizaciones que pasen en la otra.

```
1 // se dispara en actualizaciones hechas en el mismo almacenaje, desde otros documentos
2 window.onstorage = event => { // también puede usar window.addEventListener('storage',
3   if (event.key !== 'now') return;
4   alert(event.key + ':' + event.newValue + " at " + event.url);
5 };
6
7 localStorage.setItem('now', Date.now());
```


Hay que tener en cuenta que el evento también contiene: `event.url` – la url del documento en que se actualizaron los datos.

También que **`event.storageArea`** contiene el objeto de almacenaje – el evento es el mismo para `sessionStorage` y `localStorage` --, de modo que `storageArea` referencia el que se modificó. Podemos hasta querer cambiar datos en él, para “responder” a un cambio.

Esto permite que distintas ventanas del mismo origen puedan intercambiar mensajes.

Los navegadores modernos también soportan la API de Broadcast channel API, la API específica para la comunicación entre ventanas del mismo origen. Es más completa, pero tiene menos soporte. Hay librerías que añaden polyfills para ésta API basados en `localStorage` para que se pueda utilizar en cualquier entorno.

RESUMEN

- Los objetos de almacenaje web `localStorage` y `sessionStorage` permiten guardar pares de clave/valor en el navegador.
 - Tanto la clave como el valor deben ser strings.
 - El límite es de más de 5mb+, dependiendo del navegador.
 - No expiran.
 - Los datos están vinculados al origen (dominio/puerto/protocolo).

| <code>localStorage</code> | <code>sessionStorage</code> |
|---|---|
| Compartida entre todas las pestañas y ventanas que tengan el mismo origen | Accesible en una pestaña del navegador, incluyendo iframes del mismo origen |
| Sobrevive a reinicios del navegador | Muere al cerrar la pestaña |

- API:
 - `setItem(clave, valor)` – guarda pares clave/valor.
 - `getItem(clave)` – coge el valor de una clave.
 - `removeItem(clave)` – borra una clave con su valor.
 - `clear()` – borra todo.
 - `key(índice)` – coge la clave en una posición determinada.
 - `length` – el número de ítems almacenados.
 - Utiliza `Object.keys` para conseguir todas las claves.
 - Puede utilizar las claves como propiedades de objeto, pero en ese caso el evento `storage` no se dispara
- Evento `storage`:
 - Se dispara en las llamadas a `setItem`, `removeItem`, `clear`.
 - Contiene todos los datos relativos a la operación (`key/oldValue/newValue`), la url del documento y el objeto de almacenaje.
 - Se dispara en todos los objetos `window` que tienen acceso al almacenaje excepto el que ha generado el evento (en una pestaña en el caso de `sessionStorage` o globalmente en el caso de `localStorage`).

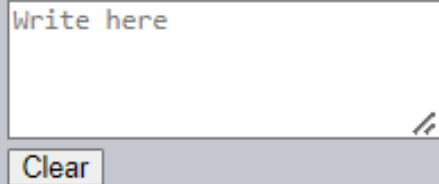
TAREA

Guardar automáticamente un campo de formulario

Crea un campo textarea que "autoguarde" sus valores en cada cambio.

Entonces, si el usuario cierra accidentalmente la página y la abre de nuevo, encontrará su entrada inacabada en su lugar.

Como esto:

A screenshot of a web form. It features a text area with the placeholder text "Write here" and a small icon in the bottom right corner. Below the text area is a button labeled "Clear". The form is set against a light blue background.

[Abrir un entorno controlado para la tarea.](#)