…JavaScript LocalStorage, SessionStorage

LocalStorage and sessionStorage are web storage objects, allowing developers to save key-value pairs in the browser.

The most interesting thing about them is that the data survives a page refresh and a full restart of the browser.

Both of the storage objects include the same properties and methods:

- **setItem(key, value)** – keep the key/value pair.
- **getItem(key)** – receive the value by key.
- **removeItem(key)** –delete the key, along with its value.
- **clear()**– remove everything.
- **key(index)** – receive the key in a specific position.
- **length** – the quantity of the stored items.

So, it is a Map collection (`setItem/getItem/removeItem`), which also allows accessing by index with key(index).

LocalStorage Demo ¶

The `localStorage` has the following features:

1. It is shared between all the tabs and windows from the same origin.
2. The data remains after the browser restarts. It doesn't expire even after OS reboot.

Here is an example of using `localStorage`:

```
localStorage.setItem('test', 2);
```

So, if you run the code above and open/close the browser or even open the same page in another window, you will get it as follows:

```
<!DOCTYPE html>
```

```
<html>
  <head>
    <title>Title of the Document</title>
  </head>
  <body>
    <script>
      alert(localStorage.getItem('test'));
    </script>
  </body>
</html>
```

Try it Yourself »

The `localStorage` is shared between the overall windows with the same origin. So, setting the data in one window makes the change visible in another one.

Object-like Access ¶

A plain object can also be used without getting/setting keys, like here:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title of the Document</title>
  </head>
  <body>
    <script>
      // set key
      localStorage.test = 2;
      // get key
      alert(localStorage.test); // 2
      // remove key
      delete localStorage.test;
    </script>
  </body>
</html>
```

Try it Yourself »

This code works but, generally, it is not recommended.

Looping Over the keys ¶

Storage objects are not iterable.

One of the ways is looping them as over an array, like this:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title of the Document</title>
  </head>
  <body>
    <script>
      for(let j = 0; j < localStorage.length; j++) {
        let key = localStorage.key(j);
        alert(`${key}: ${localStorage.getItem(key)}`);
      }
    </script>
  </body>
</html>
```

Try it Yourself »

The next way is to use for key in `localStorage` loop, just as doing with regular objects.

That iterates over keys but also outputs some built-in fields that you may not need:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title of the Document</title>
  </head>
  <body>
    <script>
      // bad try
      for(let key in localStorage) {
        alert(key); // shows getItem, setItem and other built-in
things
      }
    </script>
  </body>
</html>
```

Try it Yourself »

So, it is necessary to filter fields from the prototype using `hasOwnProperty` check, like this:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title of the Document</title>
  </head>
  <body>
    <script>
      for(let key in localStorage) {
        if(!localStorage.hasOwnProperty(key)) {
          continue; // skip keys like "setItem", "getItem" etc
        }
        alert(`${key}: ${localStorage.getItem(key)}`);
      }
    </script>
  </body>
</html>
```

Try it Yourself »

Another option is to get the own keys using Object.keys and loop over them, if it is necessary, like here:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title of the Document</title>
  </head>
  <body>
    <script>
      let keys = Object.keys(localStorage);
      for(let key of keys) {
        alert(`${key}: ${localStorage.getItem(key)}`);
      }
    </script>
  </body>
</html>
```

Try it Yourself »

Only Strings ¶

Both the key and the value should be strings.

In case there is another type (for example, a number or an object), it will automatically be converted into a string, like this:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title of the Document</title>
  </head>
  <body>
    <script>
      sessionStorage.user = {
        name: "Jack"
      };
      alert(sessionStorage.user); // [object Object]
    </script>
  </body>
</html>
```

Try it Yourself »

For storing objects, you can also use JSON:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title of the Document</title>
  </head>
  <body>
    <script>
      sessionStorage.user = JSON.stringify({
        name: "Jack"
      });
      // sometime later
      let user = JSON.parse(sessionStorage.user);
      alert(user.name); // Jack
    </script>
  </body>
</html>
```

Try it Yourself »

It is also possible to stringify the overall storage object (for example, for debugging):

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title of the Document</title>
  </head>
  <body>
    <script>
      // added formatting options to JSON.stringify to make the
object look nicer
      alert(JSON.stringify(localStorage, null, 2));
    </script>
  </body>
</html>
```

Try it Yourself »

SessionStorage ¶

The sessionStorage is used not as often as `localStorage`. But their properties and methods are the same. However, sessionStorage is more limited:

- It exists only within the current browser tab.
- A different tab with the same page will have another storage.
- It is shared between iframes in the same tab.
- The data survives page refresh.

The code for running `localStorage` is the following:

```
sessionStorage.setItem('test', 2);
```

Try it Yourself »

You will still get the data after refreshing the page:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title of the Document</title>
  </head>
  <body>
    <script>
      alert(sessionStorage.getItem('test')); // after refresh: 2
    </script>
  </body>
</html>
```

Try it Yourself »

But opening the same page in another tab will return null, which means "nothing found". The reason is that `sessionStorage` is bound to both the origin and the browser tab. So, it is used sparingly.

Storage Event ¶

Once the data is updated in the `localStorage` or the `sessionStorage`, storage event occurs, with these properties:

- **key** – the changed key (`null` in case `.clear()` is called).
- **oldValue** – the old value (`null` in case the new key is added).
- **newValue** – the new value (`null` in case the key is removed).
- **url** – the document URL where the update occurred.
- **storageArea** – the `localStorage` or the `sessionStorage` object where the update occurred.

The storage event happens on all the `window` objects where the storage is accessible, except the one causing it.

Let’s take a look at an example:

```
window.onstorage = event => { // same as
window.addEventListener('storage', () => {
  if (event.key != 'now') {
    return;
  }
  alert(event.key + ':' + event.newValue + " at " + event.url);
};
localStorage.setItem('now', Date.now());
```

Also, modern browsers support the specific API, known as Broadcast channel API, used for same-origin inter-window communication. There are libraries that polyfill it, based on the `localStorage`, making it available anywhere.