Lesson | Wednesday

Intermediate JavaScript (/intermediate-javascript) / Object-Oriented JavaScript (/intermediate-javascript/object-oriented-javascript) / Further Exploration: Local Storage

Text

This lesson is a **further exploration**. That means it's optional — and the content here is not required for the independent project.

Using Local Storage for Persistence

Currently, all of our data is being stored in a global variable. When we refresh the page, all our variables are cleared. That means our data gets cleared, too. This is frustrating, especially since we will need to refresh our page constantly as we update our code.

While we're not ready to use databases yet, we do have another solution for temporary persistence called **local storage**. Local storage allows us to store some data in the browser. This data will persist even when we reload our page — in fact, it will persist until we clear our browser's cache or specifically do something to clear local storage!

Local storage is one part of the Web Storage API, which is a part of the larger collection of browser Web APIs like `document`, `window`, and others. In fact, we access local storage through the `window`

object.

Local storage is a little bit like a **cookie** but there are some key differences. Most of us are familiar with cookies. They are little bits of data stored in our browser. Whenever our browser navigates to a website, servers can read these cookies. Cookies are often innocuous but they can be annoying, too — they are a way for servers to track our website activity and behaviors.

Local storage is different. It can only be read client-side — by our browser. Servers can not see what's in our local storage. Local storage also has considerably more space than a cookie.

We can store our mock database in local storage. Note that this has very limited utility — local storage is limited to just one domain on one machine, so if you opened the project on another machine, the data stored on the first machine would not be there.

However, local storage is a nice way to achieve temporary persistence in our projects — at least until we start using databases. It also has many real-world uses as well. It can save certain information about our preferences, for instance, that the server does not need to see. Try navigating to a few different sites you use regularly and check the `window.localStorage` property.

Let's take a look at how we can use local storage in our own applications. Once we've learned the basics, we'll see how we can use it to store our address book.

Adding, Getting and Removing Items

Open the DevTools console in Chrome and type in `window.localStorage`. You'll see an object and it may even have key-value pairs in it. Just like with the `document` object, we can also access `localStorage` directly without first accessing the `window` object. Which syntax to use in order to access local storage is up to personal preference.

In the following code snippet, the local storage is currently empty:

```
> window.localStorage;
Storage {length: 0}
> localStorage;
Storage {length: 0}
```

Local storage can only hold key-value pairs of strings. We can add a key-value pair like this:

```
> localStorage.setItem("name", "Jasmine");
> localStorage;
Storage { name: "Jasmine", length: 1}
```

We can use `localStorage.setItem()` to add a key-value pair. Then, when we check the value of `localStorage`, we'll see our new key-value pair. Note that we'll also see a `length` property. This is the number of items in local storage for that particular domain. So if we navigate from www.someimaginarysite111.com (one domain) to www.someotherimaginarysite222.com (a different domain), the second site will have a different local storage object.

To actually retrieve an item, we can do this:

```
> localStorage.getItem("name");
"Jasmine"
```

We can remove an item by doing the following:

```
> localStorage.removeItem("name");
```

Note that we don't need to remove an item first to add another one. We can just call `localStorage.setItem()` to override a key-value pair that has the same key.

Finally, if we wanted to clear all the items in local storage for our particular domain, we'd use:

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

Using Local Storage with Objects

Local storage has a pretty big limitation in terms of our address book. It's only for storing strings, not objects!

However, we can work around that. JavaScript provides methods to turn objects into strings and vice versa.

Here's how we can turn an object into a string. Open the DevTools console and add the following code. We'll start by using our `Contact` constructor and create a test contact.

```
> function Contact(firstName, lastName, phoneNumber) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.phoneNumber = phoneNumber;
}
> const testContact = new Contact("Ada", "Lovelace", "808-5
55-1111");
```

Next, we can use the `JSON.stringify()` method to turn our object into a string. We've very briefly discussed JSON in previous lessons — it's short for JavaScript Object Notation and it just means storing data in JavaScript objects. So we've already been working with JSON quite a bit!

```
> const stringifiedContact = JSON.stringify(testContact);
> localStorage.setItem('contact', stringifiedContact);
```

If we check `localStorage`, we'll see the value associated with the key `contact` is the following:

```
> localStorage.getItem('contact');
"{"firstName":"Ada","lastName":"Lovelace","phoneNumber":"80
8-555-1111"}"
```

Now let's say we want to retrieve this information and work with it as an object again. We can do so with `JSON.parse()`:

```
> const contactString = localStorage.getItem('contact');
> const contactObject = JSON.parse(contactString);
> contactObject;
{firstName: "Ada", lastName: "Lovelace", phoneNumber: "808-
555-1111"}
```

As we can see, the value of `contactObject` is now an object again. However, be careful. A very important piece of information has been lost in this whole translation process. Can you see what it is?

Well, `contactObject` is no longer an object of the `Contact` type. It's just a basic JavaScript object of the `Object` type. That means none of the prototypal methods specific to `Contact` objects can be used with a contact retrieved from local storage.

Once again, there are ways around this. For instance, it might make sense to create a method that translates these basic objects back into `Contact` objects.

However, we are going to leave the next steps for intrepid explorers that would like to check out local storage on their own. You can try updating the address book application or another application you're working on to use local storage — or you could try using local storage for another project in the next few sections of Intermediate JavaScript. Once again, this is completely optional — and there are more important JavaScript concepts to focus on over local storage right now. That being said, if you're feeling comfortable with the material and need more challenges, check it out.

More to Explore on MDN

`localStorage` is a part of the Web Storage API. This API has another storage option called `sessionStorage` that saves data just for the duration that a web browser is open. Once closed, the data is erased. To learn more, visit MDN:

- Web Storage API (https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API)
- `localStorage` (https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage)
- `sessionStorage` (https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage)
- `JSON` (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON)

Previous (/intermediate-javascript/object-oriented-javascript/switch-cases)
Next (/intermediate-javascript/object-oriented-javascript/further-exploration-aliases)

Lesson 29 of 33
Last updated March 23, 2023

(http://www.epicodus.com)