

Цели занятия

- научиться хранить данные в браузере;
- разобраться с тем, как работает LocalStorage и SessionStorage и в чём их разница;
- познакомиться с cookies и для чего они используются;

Полезные материалы

- [Подробнее про LocalStorage, SessionStorage](#)
- [JSON — самый популярный формат обмена данными](#)
- [Больше информации про cookie](#)

Хранилища данных в браузере

Периодически разработчикам необходимо сохранять часть информации в браузере пользователя, чтобы использовать её при следующем визите. Конечно, такое можно сделать с помощью бэкенда, но тогда придётся добавлять регистрацию и авторизацию на сайт, чтобы разделять пользователей между собой.

В некоторых случаях это избыточно и можно обойтись встроенными в браузер хранилища данных — LocalStorage и SessionStorage. Оба этих хранилища имеют одинаковый принцип работы и одинаковые методы, но отличаются тем, как они хранят данные.

- LocalStorage — сохранённые данные будут доступны даже после перезагрузки страницы, браузера или компьютера.
- SessionStorage — сохранённые данные доступны после перезагрузки страницы, но если закрыть вкладку или браузер, то данные будут удалены.

Из-за этих особенностей чаще всего используется именно LocalStorage, но всегда стоит помнить, что пользователь может в любой момент вручную очистить каждое из этих хранилищ.

Методы для работы с хранилищами

Объекты хранилища `localStorage` и `sessionStorage` предоставляют одинаковые методы и свойства:

- `setItem(key, value)` – сохранить пару ключ/значение;
- `getItem(key)` – получить данные по ключу `key`;
- `removeItem(key)` – удалить данные с ключом `key`;
- `clear()` – удалить всё;
- `key(index)` – получить ключ на заданной позиции;
- `length` – количество элементов в хранилище.

Например, если запустить этот код...

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

...и закрыть/открыть браузер или открыть ту же страницу в другом окне, то можно получить данные следующим образом:

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

Перебор ключей

Иногда может возникнуть задача, когда необходимо получить не какое-то определенное значение из хранилища по ключу, а сразу все сохраненные в нем значения.

В таком случае у нас есть два варианта сделать это:

- проходимся по хранилищу с помощью цикла

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

- получаем все ключи с помощью Object.keys

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

Как хранить сложные данные?

Оба этих хранилища умеют работать только со строками, поэтому если вы захотите сохранить в них какой-то объект, то он будет принудительно приведёт к строке [object Object].

Поэтому для этого используется [формат JSON](#), который позволяет трансформировать объект любой сложности в строку, а затем сделать обратный процесс — получить из этой строки корректный объект.

На примере:

```
let user = { name: "Yury" };  
  
// Обратите внимание на метод JSON.stringify  
// Он переводит объект в строку, сохраняя данные  
localStorage.setItem("user", JSON.stringify(user));
```

```
// А теперь обратный процесс — создаём из json-строки объект
let savedUser = JSON.parse(localStorage.getItem("user"));
alert(savedUser.name);
```

Событие storage

Когда обновляются данные в `localStorage` или `sessionStorage`, генерируется событие `storage` со следующими свойствами:

- `key` — ключ, который обновился (`null`, если вызван `.clear()`).
- `oldValue` — старое значение (`null`, если ключ добавлен впервые).
- `newValue` — новое значение (`null`, если ключ был удалён).
- `url` — адрес документа, где произошло обновление.
- `storageArea` — объект `localStorage` или `sessionStorage`, где произошло обновление.

Важно: событие срабатывает на всех остальных объектах `window`, где доступно хранилище, кроме того окна, которое его вызвало.

Представьте, что у вас есть два окна с одним и тем же сайтом. Хранилище `localStorage` разделяется между ними.

Вы можете открыть эту страницу в двух окнах браузера, чтобы проверить приведённый ниже код.

Теперь, если оба окна слушают `window.onstorage`, то каждое из них будет реагировать на обновления, произошедшие в другом окне.

```
// срабатывает при обновлениях, сделанных в том же хранилище из  
других документов  
  
window.addEventListener("storage", (event) => {  
    alert(event.key + ":" + event.newValue + " at " + event.url);  
});  
  
localStorage.setItem("now", Date.now());
```

Что такое cookie?

При разработке сайтов часть информации (например, токен авторизации или данные пользователя) нужно хранить и читать как в браузере, так и на сервере. Для этого используют **Cookie** (произносится «куки»).

Эти данные отправляются на сервер при каждом запросе и поэтому данными можно управлять как на стороне сервера, так и из браузера, но хранятся они в браузере пользователя, поэтому он их может всегда удалить.

В рамках данного курса мы не будем работать с cookie, поэтому эта тема рассматривается поверхностно.

Как создать cookie?

Все куки хранятся в свойстве `document.cookie`. Это свойство представляет собой строку в формате `имя=значение`, где пары имён и значений разделяются знаком `;`. При этом взаимодействие с полем весьма необычное — если присвоить `document.cookie` новое

значение, то оно не заменит полностью старую строку, а добавит или изменит значение по ключу.

Запись в `cookie` работает с помощью присвоения значения новой куки в поле `document.cookie`. За один раз можно записать лишь одно значение.

Вот так можно добавить значение 1 по ключу `counter`:

```
document.cookie = "counter=1";  
  
console.log(document.cookie); // 'counter=1'
```

При присвоении свойству куки с другим именем, получим два записанных значения:

```
document.cookie = "sidebar=false";  
  
console.log(document.cookie); // 'counter=1; sidebar=false;'
```

При повторной записи в то же поле другого значения оно будет перезаписано.

```
document.cookie = "sidebar=true";  
  
console.log(document.cookie); // 'counter=1; sidebar=true;'
```

Как читать cookie?

Для получения значений, записанных в куки, можно просто вывести содержимое `document.cookie`. Но таким способом мы получим сразу все имеющиеся куки.

Чтобы получить значение конкретной куки, нам нужно будет прочитать строки и разобрать её по значениям. Например, так:

```
function getCookie() {  
  return document.cookie.split("; ").reduce((acc, item) => {  
    const [name, value] = item.split("=");  
  
    return { ...acc, [name]: value };  
  }, {});  
}  
  
const cookie = getCookie();  
  
console.log(cookie.counter); // 1  
console.log(cookie.sidebar); // true
```