Материалы занятия

Курс: Разработка интерфейса на JavaScript

Дисциплина: Основы JavaScript

Тема занятия №20: Сохранение данных в браузере. Куки. Локальное хранилище

Куки, document.cookie

Куки – это небольшие строки данных, которые хранятся непосредственно в браузере. Они являются частью HTTP-протокола, определённого в спецификации RFC 6265.

Куки обычно устанавливаются веб-сервером при помощи заголовка Set-Cookie. Затем браузер будет автоматически добавлять их в (почти) каждый запрос на тот же домен при помощи заголовка Cookie.

Один из наиболее частых случаев использования куки – это аутентификация:

1. При входе на сайт сервер отсылает в ответ HTTP-заголовок Set-Cookie для того, чтобы установить куки со специальным уникальным идентификатором сессии («session identifier»).
2. Во время следующего запроса к этому же домену браузер посылает на сервер HTTP-заголовок Cookie.
3. Таким образом, сервер понимает, кто сделал запрос.

Мы также можем получить доступ к куки непосредственно из браузера, используя свойство document.cookie.

Куки имеют множество особенностей и тонкостей в использовании, и в этой главе мы подробно с ними разберёмся.

Чтение из document.cookie

Хранит ли ваш браузер какие-то куки с этого сайта? Посмотрим:

```
// На javascript.info мы используем сервис Google Analytics
// поэтому какие-то куки должны быть
alert( document.cookie ); // cookie1=value1; cookie2=value2;
```

Значение document.cookie состоит из пар ключ=значение, разделённых ;. Каждая пара представляет собой отдельное куки.

Чтобы найти определённое куки, достаточно разбить строку из document.cookie по ;, и затем найти нужный ключ. Для этого мы можем использовать как регулярные выражения, так и функции для обработки массивов.

Оставим эту задачу читателю для самостоятельного выполнения. Кроме того, в конце этой главы вы найдёте полезные функции для работы с куки.

Запись в document.cookie

Мы можем писать в document.cookie. Но это не просто свойство данных, а аксессор (геттер/сеттер). Присваивание к нему обрабатывается особым образом. Запись в **document.cookie** обновит только упомянутые в ней куки, но при этом не затронет все остальные.

Например, этот вызов установит куки с именем user и значением John:

```
document.cookie = "user=John"; // обновляем только куки с именем 'user'
alert(document.cookie); // показываем все куки
```

Если вы запустите этот код, то, скорее всего, увидите множество куки. Это происходит, потому что операция document.cookie= перезапишет не все куки, а лишь куки с вышеупомянутым именем user.

Технически, и имя и значение куки могут состоять из любых символов, для правильного форматирования следует использовать встроенную функцию encodeURIComponent:

```
// специальные символы (пробелы), требуется кодирование
let name = "my name";
let value = "John Smith"

// кодирует в my%20name=John%20Smith
document.cookie = encodeURIComponent(name) + '=' + encodeURIComponent(value);

alert(document.cookie); // ...; my%20name=John%20Smith
```

Ограничения

Существует несколько ограничений:

- После encodeURIComponent пара name=value не должна занимать более 4Кб. Таким образом, мы не можем хранить в куки большие данные.
- Общее количество куки на один домен ограничивается примерно 20+. Точное ограничение зависит от конкретного браузера.

У куки есть ряд настроек, многие из которых важны и должны быть установлены. Эти настройки указываются после пары ключ=значение и отделены друг от друга разделителем , вот так:

```
document.cookie = "user=John; path=/; expires=Tue, 19 Jan 2038 03:14:07 GMT"
```

path

- path=/mypath

URL-префикс пути, куки будут доступны для страниц под этим путём. Должен быть абсолютным. По умолчанию используется текущий путь.

Если куки установлено с path=/admin, то оно будет доступно на страницах /admin и /admin/something, но не на страницах /home или /adminpage.

Как правило, указывают в качестве пути корень path=/, чтобы наше куки было доступно на всех страницах сайта.

domain

- domain=site.com

Домен определяет, где доступен файл куки. Однако на практике существуют определённые ограничения. Мы не можем указать здесь какой угодно домен.

Нет никакого способа разрешить доступ к файлам куки из другого домена 2-го уровня, поэтому **other.com** никогда не получит куки, установленный по адресу **site.com**.

Это ограничение безопасности, позволяющее нам хранить конфиденциальные данные в файлах куки, которые должны быть доступны только на одном сайте.

По умолчанию куки доступны лишь тому домену, который его установил.

Пожалуйста, обратите внимание, что по умолчанию файл куки также не передаётся поддомену, например forum.site.com.

```
// если мы установим файл куки на веб-сайте site.com...
document.cookie = "user=John"

// ...мы не увидим его на forum.site.com
alert(document.cookie); // нет user
```

…Но это можно изменить. Если мы хотим разрешить поддоменам типа forum.site.com получать куки, установленные на site.com, это возможно.

Чтобы это произошло, при установке файла куки в site.com, мы должны явно установить параметр domain для корневого домена: domain=site.com. После этого все поддомены увидят такой файл cookie.

Например:

```
// находясь на странице site.com
// сделаем куки доступным для всех поддоменов *.site.com:
document.cookie = "user=John; domain=site.com"

// позже

// на forum.site.com
alert(document.cookie); // есть куки user=John
```

По историческим причинам установка domain=.site.com (с точкой перед site.com) также работает и разрешает доступ к куки для поддоменов. Это старая запись, но можно использовать и её, если нужно, чтобы поддерживались очень старые браузеры.

Таким образом, опция domain позволяет нам разрешить доступ к куки для поддоменов.

expires, max-age

По умолчанию, если куки не имеют ни одного из этих параметров, то они удалятся при закрытии браузера. Такие куки называются сессионными («session cookies»).

Чтобы помочь куки «пережить» закрытие браузера, мы можем установить значение опций expires или max-age.

- expires=Tue, 19 Jan 2038 03:14:07 GMT

Дата истечения срока действия куки, когда браузер удалит его автоматически.

Дата должна быть точно в этом формате, во временной зоне GMT. Мы можем использовать date.toUTCString, чтобы получить правильную дату. Например, мы можем установить срок действия куки на 1 день.

```
// +1 день от текущей даты
let date = new Date(Date.now() + 86400e3);
date = date.toUTCString();
document.cookie = "user=John; expires=" + date;
```

Если мы установим в expires прошедшую дату, то куки будет удалено.

- max-age=3600

Альтернатива expires, определяет срок действия куки в секундах с текущего момента.

Если задан ноль или отрицательное значение, то куки будет удалено:

```
// куки будет удалено через 1 час
document.cookie = "user=John; max-age=3600";

// удалим куки (срок действия истекает прямо сейчас)
document.cookie = "user=John; max-age=0";
```

secure

Куки следует передавать только по HTTPS-протоколу.

То есть, куки, по умолчанию, опираются на доменное имя, они не обращают внимания на протоколы.

С этой настройкой, если куки будет установлено на сайте https://site.com, то оно не будет доступно на том же сайте с протоколом HTTP, как http://site.com. Таким образом, если в куки хранится конфиденциальная информация, которую не следует передавать по незашифрованному протоколу HTTP, то нужно установить этот флаг.

```
// предполагается, что сейчас мы на https://
// установим опцию secure для куки (куки доступно только через HTTPS)
document.cookie = "user=John; secure";
```

samesite

Это ещё одна настройка безопасности, применяется для защиты от так называемой XSRF-атаки (межсайтовая подделка запроса).

Чтобы понять, как настройка работает и где может быть полезной, посмотрим на XSRF-атаки.

Приложение: Функции для работы с куки

Вот небольшой набор функций для работы с куки, более удобных, чем ручная модификация document.cookie.

Для этого существует множество библиотек, так что они, скорее, в демонстрационных целях. Но при этом полностью рабочие.

getCookie(name)

Самый короткий способ получить доступ к куки – это использовать регулярные выражения.

Функция getCookie(name) возвращает куки с указанным name:

```
// возвращает куки с указанным name,
// или undefined, если ничего не найдено
function getCookie(name) {
  let matches = document.cookie.match(new RegExp(
    "(?:^|; )" + name.replace(/([\.$?*|{}\(\)\[\]\\\/\+^])/g, '\\$1') + "=([^;]*)"
  ));
  return matches ? decodeURIComponent(matches[1]) : undefined;
}
```

Здесь new RegExp генерируется динамически, чтобы находить; name=<value>.

Обратите внимание, значение куки кодируется, поэтому getCookie использует встроенную функцию decodeURIComponent для декодирования.

setCookie(name, value, options)

Устанавливает куки с именем name и значением value, с настройкой path=/ по умолчанию (можно изменить, чтобы добавить другие значения по умолчанию):

```
function setCookie(name, value, options = {}) {

  options = {
    path: '/',
    // при необходимости добавьте другие значения по умолчанию
    ...options
  };

  if (options.expires instanceof Date) {
    options.expires = options.expires.toUTCString();
  }

  let updatedCookie = encodeURIComponent(name) + "=" + encodeURIComponent(value);

  for (let optionKey in options) {
    updatedCookie += "; " + optionKey;
    let optionValue = options[optionKey];
    if (optionValue !== true) {
      updatedCookie += "=" + optionValue;
    }
  }

  document.cookie = updatedCookie;
}

// Пример использования:
setCookie('user', 'John', {secure: true, 'max-age': 3600});
```

deleteCookie(name)

Чтобы удалить куки, мы можем установить отрицательную дату истечения срока действия:

```
function deleteCookie(name) {
  setCookie(name, "", {
    'max-age': -1
  })
}
```

Операции обновления или удаления куки должны использовать те же путь и домен. Обратите внимание: когда мы обновляем или удаляем куки, нам следует использовать только такие же настройки пути и домена, как при установке куки.

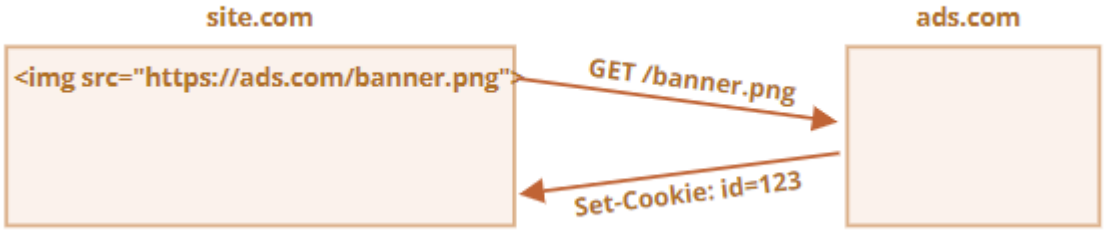
Всё вместе: cookie.js.

Приложение: Сторонние куки

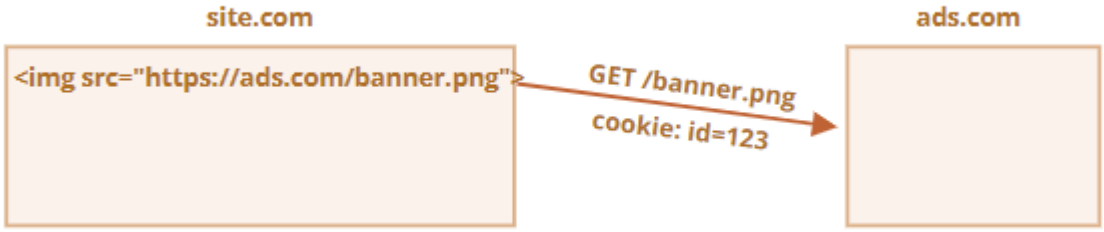
Куки называются сторонними, если они размещены с домена, отличающегося от страницы, которую посещает пользователь.

Например:

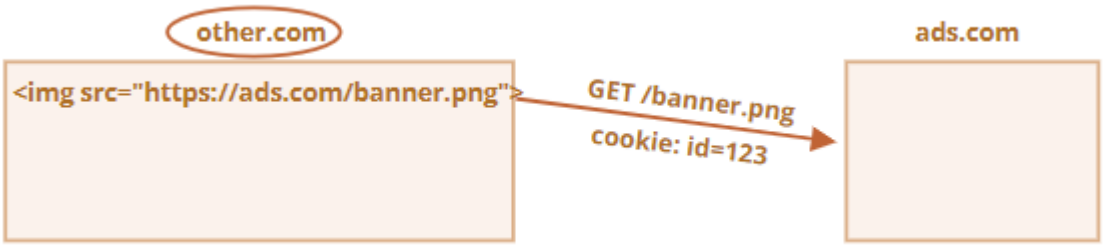
1. Страница site.com загружает баннер с другого сайта: <img src="https://ads.com/banner.png">.
2. Вместе с баннером удалённый сервер ads.com может установить заголовок Set-Cookie с куки, например, id=1234. Такие куки создаются с домена ads.com и будут видны только на сайте ads.com:



3. В следующий раз при доступе к ads.com удалённый сервер получит куки id и распознает пользователя:



4. Что ещё более важно, когда пользователь переходит с site.com на другой сайт other.com, на котором тоже есть баннер, то ads.com получит куки, так как они принадлежат ads.com, таким образом ads.com распознает пользователя и может отслеживать его перемещения между сайтами:



Сторонние куки в силу своей специфики обычно используются для целей отслеживания посещаемых пользователем страниц и показа рекламы. Они привязаны к исходному домену, поэтому ads.com может отслеживать одного и того же пользователя на разных сайтах, если оттуда идёт обращение к нему.

Естественно, некоторым пользователям не нравится, когда их отслеживают, поэтому браузеры позволяют отключать такие куки.

Кроме того, некоторые современные браузеры используют специальные политики для таких куки:

- Safari вообще не разрешает сторонние куки.

- У Firefox есть «чёрный список» сторонних доменов, чьи сторонние куки он блокирует.

Итого

document.cookie предоставляет доступ к куки.

- Операция записи изменяет только то куки, которое было указано.
- Имя и значение куки должны быть закодированы.
- Одно куки вмещает до 4kb данных, разрешается более 20 куки на сайт (зависит от браузера).

Настройки куки:

- path=/, по умолчанию устанавливается текущий путь, делает куки видимым только по указанному пути и ниже.
- domain=site.com, по умолчанию куки видно только на текущем домене, если явно указан домен, то куки видно и на поддоменах.
- expires или max-age устанавливает дату истечения срока действия, без них куки умрёт при закрытии браузера.
- secure делает куки доступным только при использовании HTTPS.
- samesite запрещает браузеру отправлять куки с запросами, поступающими извне, помогает предотвратить XSRF-атаки.

Дополнительно:

- Сторонние куки могут быть запрещены браузером, например Safari делает это по умолчанию.
- Установка отслеживающих куки пользователям из стран ЕС требует их явного согласия на это в соответствии с законодательством GDPR.

LocalStorage, sessionStorage

Объекты веб-хранилища localStorage и sessionStorage позволяют хранить пары ключ/значение в браузере.

Что в них важно – данные, которые в них записаны, сохраняются после обновления страницы (в случае sessionStorage) и даже после перезапуска браузера (при использовании localStorage). Скоро мы это увидим.

Но ведь у нас уже есть куки. Зачем тогда эти объекты?

- В отличие от куки, объекты веб-хранилища не отправляются на сервер при каждом запросе. Именно поэтому мы можем хранить гораздо больше данных. Большинство современных браузеров могут выделить как минимум 5 мегабайтов данных (или больше), и этот размер можно поменять в настройках.
- Ещё одно отличие от куки – сервер не может манипулировать объектами хранилища через HTTP-заголовки. Всё делается при помощи JavaScript.

- Хранилище привязано к источнику (домен/протокол/порт). Это значит, что разные протоколы или поддомены определяют разные объекты хранилища, и они не могут получить доступ к данным друг друга.

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.

Как видим, интерфейс похож на Map (setItem/getItem/removeItem), но также позволяет получить доступ к элементу по индексу – key(index).

Давайте посмотрим, как это работает.

Демо localStorage

Основные особенности localStorage:

- Этот объект один на все вкладки и окна в рамках источника (один и тот же домен/протокол/порт).
- Данные не имеют срока давности, по которому истекают и удаляются. Сохраняются после перезапуска браузера и даже ОС.

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
```

Нам достаточно находиться на том же источнике (домен/протокол/порт), при этом URL-путь может быть разным.

Объект localStorage доступен всем окнам из одного источника, поэтому, если мы устанавливаем данные в одном окне, изменения становятся видимыми в другом.

Доступ как к обычному объекту

Также можно получать/записывать данные, как в обычный объект:

```
// установить значение для ключа
localStorage.test = 2;

// получить значение по ключу
alert( localStorage.test ); // 2

// удалить ключ
delete localStorage.test;
```

Это возможно по историческим причинам и, как правило, работает, но обычно не рекомендуется, потому что:

1. Если ключ генерируется пользователем, то он может быть каким угодно, включая length или toString или другой встроенный метод localStorage. В этом случае getItem/setItem сработают нормально, а вот чтение/запись как свойства объекта не пройдут:

   ```
   let key = 'length';
   localStorage[key] = 5; // Ошибка, невозможно установить length
   ```

2. Когда мы модифицируем данные, то срабатывает событие storage. Но это событие не происходит при записи без setItem, как свойства объекта. Мы увидим это позже в этой главе.

Перебор ключей

Методы, которые мы видим, позволяют читать/писать/удалять данные. А как получить все значения или ключи?

К сожалению, объекты веб-хранилища нельзя перебрать в цикле, они не итерируемы.

Но можно пройти по ним, как по обычным массивам:

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

Другой способ – использовать цикл, как по обычному объекту for key in localStorage.

Здесь перебираются ключи, но вместе с этим выводятся несколько встроенных полей, которые нам не нужны:

```
// bad try
for(let key in localStorage) {
  alert(key); // покажет getItem, setItem и другие встроенные свойства
}
```

…Поэтому нам нужно либо отфильтровать поля из прототипа проверкой hasOwnProperty:

```
for(let key in localStorage) {
  if (!localStorage.hasOwnProperty(key)) {
    continue; // пропустит такие ключи, как "setItem", "getItem" и так далее
  }
  alert(`${key}: ${localStorage.getItem(key)}`);
}
```

…Либо просто получить «собственные» ключи с помощью Object.keys, а затем при необходимости вывести их при помощи цикла:

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

Последнее работает, потому что Object.keys возвращает только ключи, принадлежащие объекту, игнорируя прототип.

Только строки

Обратите внимание, что ключ и значение должны быть строками.

Если мы используем любой другой тип, например число или объект, то он автоматически преобразуется в строку:

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

Мы можем использовать JSON для хранения объектов:

```
localStorage.user = JSON.stringify({name: "John"});

// немного позже
let user = JSON.parse( localStorage.user );
alert( user.name ); // John
```

sessionStorage

Объект sessionStorage используется гораздо реже, чем localStorage.

Свойства и методы такие же, но есть существенные ограничения:

- sessionStorage существует только в рамках текущей вкладки браузера.
  - Другая вкладка с той же страницей будет иметь другое хранилище.
  - Но оно разделяется между ифреймами на той же вкладке (при условии, что они из одного и того же источника).
- Данные продолжают существовать после перезагрузки страницы, но не после закрытия/открытия вкладки.

Давайте посмотрим на это в действии.

Запустите этот код…

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

…И обновите страницу. Вы всё ещё можете получить данные:

```
alert( sessionStorage.getItem('test') ); // после обновления: 1
```

…Но если вы откроете ту же страницу в другой вкладке и попробуете получить данные снова, то код выше вернёт null, что значит «ничего не найдено».

Так получилось, потому что sessionStorage привязан не только к источнику, но и к вкладке браузера. Поэтому sessionStorage используется нечасто.

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 – url 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.

Давайте уточним.

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.onstorage = event => { // можно также использовать window.addEventListener(
  if (event.key != 'now') return;
  alert(event.key + ':' + event.newValue + " at " + event.url);
};

localStorage.setItem('now', Date.now());
```

Обратите внимание, что событие также содержит: event.url – url-адрес документа, в котором данные обновились.

Также event.storageArea содержит объект хранилища – событие одно и то же для sessionStorage и localStorage, поэтому event.storageArea ссылается на то хранилище, которое было изменено. Мы можем захотеть что-то записать в ответ на изменения.

Это позволяет разным окнам одного источника обмениваться сообщениями.

Современные браузеры также поддерживают Broadcast channel API специальный API для связи между окнами одного источника, он более полнофункциональный, но менее поддерживаемый. Существуют библиотеки (полифилы), которые эмулируют это API на основе localStorage и делают его доступным везде.

Итого

Объекты веб-хранилища localStorage и sessionStorage позволяют хранить пары ключ/значение в браузере.

- key и value должны быть строками.
- Лимит 5 Мб+, зависит от браузера.
- Данные не имеют «времени истечения».
- Данные привязаны к источнику (домен/протокол/порт).
-

`localStorage`	`sessionStorage`
Совместно используется между всеми вкладками и окнами с одинаковым источником	Разделяется в рамках вкладки браузера, среди ифреймов из того же источника
«Переживает» перезапуск браузера	«Переживает» перезагрузку страницы (но не закрытие вкладки)

API:

- setItem(key, value) – сохранить пару ключ/значение.
- getItem(key) – получить данные по ключу key.
- removeItem(key) – удалить значение по ключу key.
- clear() – удалить всё.
- key(index) – получить ключ на заданной позиции.
- length – количество элементов в хранилище.
- Используйте Object.keys для получения всех ключей.
- Можно обращаться к ключам как к обычным свойствам объекта, в этом случае событиеstorage не срабатывает.

Событие storage:

- Срабатывает при вызове setItem, removeItem, clear.
- Содержит все данные об произошедшем обновлении (key/oldValue/newValue), url документа и объект хранилища storageArea.
- Срабатывает на всех объектах window, которые имеют доступ к хранилищу, кроме того, где оно было сгенерировано (внутри вкладки для sessionStorage, глобально для localStorage).