HTML5 API веб-хранилища

Опубликовано Nov 01, 2014 в «[HTML](https://www.canonium.com/category/html)»

Немного понятий и фактов

Web Storage (Веб-Хранилище) — это технология хранения данных на стороне клиента прямиком в браузере.

Два объекта — две идеи

Веб-хранилище подразделяется на два типа:

**Local Storage**

Локальное хранилище — это объект localStorage, используемый для хранения данных вида «Ключ - Значение» на постоянной основе. Иначе говоря, данные хранятся до тех пор, пока их принудительно не заставят уйти, то есть удалят.

**Session Storage**

Хранилище данных сеансов — это объект sessionStorage, используемый для хранения данных вида «Ключ - Значение» на временной основе. Данные хранятся до тех пор, пока не будет закрыта вкладка или окно браузера.

Что можно хранить и для чего использовать?

Хранить можно всё то, что сможет уложиться в строку, так как единственным разрешенным типом данных для хранения является текстовый тип — строка.

Навскидку, веб-хранилище можно забивать следующими данными:

* Состояние работы приложения — Например, последний открытый документ, значение измененного поля ввода или история поиска по сайту.
* Настройки пользователя — Например, выбранная тема оформления сайта или вид отображения информации.
* Сохранение пользовательских данных при неудачной попытке отправки на сервер.
* Запись данных по мере их ввода — Например, редактор текста [StackEdit](https://stackedit.io/).

**Внимание!**

Конфиденциальную информацию пользователя не стоит хранить в веб-хранилище! Данные не шифруются и могут быть скомпрометированы.

Поддержка браузеров

Картина замечательная, даже лучше, чем можно было представить себе:

Оценить всю сложившуюся на данный момент ситуацию можно на сайте [Can I Use](http://caniuse.com/" \l "search=webstorage).

Доступный размер

В отличии от печенек (HTTP cookie), веб-хранилище предоставляет намного больший размер — 4Кб против ~5Мб.

Проведя небольшой тест на реальных устройствах, получаем следующие данные:

* Chrome 38, FireFox 33 и Opera 25 — размер хранилища **5000 Кб**.
* IE8-11 — размер хранилища **4750 Кб**.
* iOS 5-8.1 Safari, OS X Safari — размер хранилища **2500 Кб**.
* Android 4.x-5 — размер хранилища **5000 Кб**.
* Windows Phone 7.8 — размер хранилища **4750 Кб**.

Устройства проверялись с «заводскими» настройками браузера, то есть никаких настроек до тестирования не проводилось. Протестировать ваше устройство можно используя вот «[этот](http://jsfiddle.net/53xcc4LL/)» скрипт.

Работаем с данными

Рассмотрим традиционное использование объекта localStorage и не совсем традиционное.

Объектом sessionStorage мы заниматься не будем, так как единственное различие с localStorage — это срок хранения данных. Все методы и свойства объекта одинаковые, и для работы с хранилищем данных сеансов необходимо лишь заменить объект в коде.

**Пример — Запись данных в хранилище**

Записать данные в хранилище можно тремя способами:

localStorage.setItem("key", "value");

*// Или*

localStorage.key = "value";

*// Или*

localStorage["key"] = "value";

**Пример — Чтение данных из хранилища**

Считать данные из хранилища также просто как и записать:

localStorage.getItem("key");

*// Или*

localStorage.key;

*// Или*

localStorage["key"];

**Пример — Удаление данных из хранилища**

Удаление данных имеет всего один синтаксис:

localStorage.removeItem("key");

**Пример — Удаление всех данных из хранилища**

Если вдруг вам надоело записывать и считывать данные в веб-хранилище, то в любой момент вы можете остановить это, удалив все записи:

localStorage.clear();

**Пример — Количество записей в хранилище**

Для того, чтобы получить общее количество записей в хранилище, необходимо использовать стандартное свойство length:

localStorage.length;

**Пример — Получение всех записей хранилища**

Получить все данные, записанные в хранилище, можно с помощью цикла:

for (var i = 0; i < localStorage.length; i++) {

var key = localStorage.key(i);

console.log(key + ' = ' + localStorage[key]);

}

В цикле перебираются все элементы хранилища, получаются их ключи и значения.

**Пример — Хранение не строковых данных**

Если есть необходимость хранить в локальном хранилище не только строковые данные, то нужно учесть, что придётся заранее позаботиться об их преобразовании:

*// В хранилище:*

*// a = "11"*

*// b = "19"*

*// c = "25"*

Мы хотим сумму всех имеющихся элементов. Получаем:

var sum = 0;

for (var i = 0; i < localStorage.length; i++) {

var key = localStorage.key(i);

sum += Number(localStorage[key]);

}

console.log(sum);

Просто используем для этого функцию Number.

Если имеем дело с датой, то придется использовать объект Date:

*// В хранилище:*

*// date = "2014-10-30"*

var data = localStorage["date"].split('-');

var data = new Date(data[0], data[1], data[2]);

console.log(data.getDate() + ' ' + data.getMonth() + ' ' + data.getFullYear());

Если перед записью есть возможность отформатировать дату в текстовую строку, то лучше сделать так:

*// Запись*

var currentTime = new Date();

localStorage["date"] = currentTime.getFullYear() + '/' + currentTime.getMonth()

+ '/' + currentTime.getDate();

*// Чтение*

var data = new Date(localStorage["date"]);

console.log(data.getDate() + ' ' + data.getMonth() + ' ' + data.getFullYear());

**Пример — Записываем в хранилище массив**

Вы не забыли, что веб-хранилище дружит только со строками? Нам нужно больше функционала. Исправляем.

Схема махинаций:

* Преобразование массива в строку.
* Запись строки в хранилище.
* Чтение строки из хранилища.
* Преобразование строки в массив.

Пффф. Легко!

На входе у нас есть массив:

var myArray = ["One", "Two", ["One", "Two"], "four"];

Преобразуем его в строку, используя JSON.stringify, и записываем в хранилище:

localStorage["myArray"] = JSON.stringify(myArray);

Для внутреннего удовлетворения посмотрим, что выведет console.log():

console.log(localStorage["myArray"]);

*// Вывод: ["One","Two",["One","Two"],"four"]*

Спустя сто и один день, мы обращаемся к хранилищу и хотим заполучить наш массив обратно:

var data = JSON.parse(localStorage["myArray"]);

И снова нужно успокоить в себе параноика тем, что ожидаемое получено в необходимой форме:

console.log(data);

*// Вывод:*

*// ["One", "Two", Array[2], "four"]*

*// 0: "One"*

*// 1: "Two"*

*// 2: Array[2]*

*// 3: "four"*

*// length: 4*

*// \_\_proto\_\_: Array[0]*

Таким путём можно пойти и при необходимости записи объектов в веб-хранилище:

*// Входные данные*

var article = {

user: 'mrmlnc',

date: '30-10-2014',

star: '-1'

};

*// Записываем*

localStorage["post\_0"] = JSON.stringify(article);

*// Считываем*

var post = JSON.parse(localStorage["post\_0"]);

*// Выводим*

console.log(post);

Очень удобно и практично.

К слову, редактор [StackEdit](https://stackedit.io/) хранит так настройки и сами записи:

{

"layoutOrientation":"horizontal",

"editMode":"ltr",

"lazyRendering":true,

"editorFontClass":"font-rich",

"fontSizeRatio":1,

"maxWidthRatio":1,

"cursorFocusRatio":0.5,

"defaultContent":"\n\n\n> Written with [StackEdit](https://stackedit.io/).",

...

}

В реальной жизни это выглядит так:

**Пример — Отслеживание изменения хранилища**

Веб-хранилище генерирует событие onStorage на каждое своё изменение: добавление, изменение или удаление.

Вешаем прослушку на событие:

*// Функция обработчика события*

function storageChanged(storageEvent) {

alert('Событие Storage от' + storageEvent.url + '!');

};

window.addEventListener('storage', storageChanged, false);

Кстати, объект, который передается функции обработчика событий, содержит в себе немного полезной информации:

**key**

Ключ измененной, добавленной или удаленной записи.

**oldValue**

Значение ключа до изменения.

**newValue**

Значение ключа после изменения.

**url**

URL страницы, которая породила событие onStorage.

**storageArea**

localStorage или sessionStorage, в зависимости от того, где произошло изменение.

Стоит учитывать тот факт, что событие вызывается на всех страницах, кроме той, что его породило. Пример работы можно посмотреть на сайте [html5demos](http://html5demos.com/storage-events).

Выводы

По сути дела, это круто!

Web Storage — это просто ещё один шаг к оффлайн HTML5 приложениям. Уже сейчас вы можете хранить настройки своих веб-приложений в удобном для вас виде прямиком в браузере. Не маловажной ступенью дальнейшего развития вашего проекта будет сохранение введенных данных пользователя, при неудачной попытке отправки на сервер или валидации, и последующего использования их при повторном заполнении. Именно для этого можно использовать объект sessionStorage. Согласитесь — не это ли будущее отзывчивых приложений? Я уже не представляю себе повторный ввод всех полей в форме, если была допущена ошибка ранее и страница обновилась.

Кратко о плюсах веб-хранилища:

* Достаточно большой объем хранилища.
* Приличная скорость.
* Неограниченное время жизни.
* Удобный интерфейс общения с данными.

Поддержка HTML 5 DOM Storage

[Internet Explorer](https://habrahabr.ru/hub/ie/)

Одним из значительных нововведений в Internet Explorer 8 является поддержка технологии DOM Storage, которая представляет собой часть новых технологий грядущего стандарта HTML 5. Dom Storage (или как его еще называют Web Storage) – это механизм, который призван предоставить разработчику возможность хранить набор данных значительного объема на стороне клиента и получать к ним доступ с помощью специального API. На данный момент, полная поддержка DOM Storage реализована в браузерах Firefox 3.5 (с 2.0 существует частичная поддержка), Safari 4.0 и Internet Explorer 8, в котором эта поддержка появилась с версии beta2. Рассмотрим, что представляет собой эта технология, для чего она нужна и как работает.

Необходимость

Необходимость в хранилище данных на стороне клиента, которое предоставляет браузер назрела давно. Последние несколько лет web-технологии все больше сдвигаются со стороны сервера в сторону клиента, все больше вычислений, обработки данных и операций производится на компьютере у пользователя, а не на web-сервере. Частично проблему с хранением данных решал механизм **cookie**, но как известно он имеет ряд существенных ограничений и даже минусов:

* cookie имеет ограничение по размеру, Internet Explorer до 8 версии позволял хранить в cookie до 4 килобайт данных, в восьмой версии эта планка поднята до 10 килобайт, но все равно такой размер – это существенный недостаток;
* данные cookie участвуют в формировании каждого запроса к серверу, то есть при каждом запросе к серверу все cookie автоматически отправляются вместе с запросом, что увеличивает трафик;
* cookie сопоставлены с web-сайтом и, если пользователь работает с сайтом через две вкладки, он оперирует одними и теми же данными cookie. Этот момент может нарушить правильную работу сайта и ограничивает применение cookie.

C другой стороны механизм DOM Storage в Internet Explorer предлагает следующие возможности:

* до 10 мегабайт для хранения данных для каждого сайта (5 Mb в Firefox);
* доступ только на стороне клиента, данные DOM Storage не отправляются вместе с запросами;
* два механизма **localStorage** и **sessionStorage** позволяют гибко управлять данными, контекст sessionStorage и его данные существуют только для одной вкладки и если пользователь закроет ее или откроет еще одну то, данные из вкладки доступны не будут.

*\* localStorage появился в Firefox 3.5, sessionStorage присутствовал в Firefox с версии 2.0.*

API

Согласно [черновику спецификации Web Storage](http://dev.w3.org/html5/webstorage/) браузер должен реализовать три следующих объекта для работы с локальным хранилищем:

* **storage** – представляет собой объект, который осуществляет доступ к набору данных хранилища. Согласно спецификации набор данных должен представлять собой пары строк “ключ-значение”. Данные отличные от строковых, **должны быть** приведены к строкам перед сохранением в хранилище;
* **window.sessionStorage** – возвращает объект типа storage и представляет собой хранилище пользовательского набора данных которое существует и актуально только для одной вкладки браузера до тех пор, пока она не будет закрыта;
* **window.localStorage** – похож на sessionStorage за исключением того, что данные этого хранилища сохраняются после закрытия вкладки и доступны всегда, что делает этот объект похожим на cookie. Каждый домен и субдомен имеет свой объект window.localStorage.

*\* Firefox поддерживает еще один, не описанный в стандарте, объект window.globalStorage*.  
  
По сути, при работе с данными, сохранением или получением из хранилища, разработчик оперирует с экземпляром объекта storage, который имеет ряд вспомогательных функций и свойств:

* **setItem, getItem, removeItem** – создает, получает или удаляет новый элемент данных;
* **clear** – “стирает” все данные хранилища;
* **length** – возвращает количество сохраненных элементов данных;

*\* Internet Explorer 8 предлагает еще одно полезное свойство remainingSpace, которое позволяет узнать объем в байтах, которое занимает хранилище. Пока, это свойство не включено в черновик спецификации и не является стандартным.*

Пример

Простейший пример работы с localStorage, данные сохраняются и достаются из хранилища:  
  
…  
sessionStorage.someDataKeyName = ‘данные’;  
…  
var data = sessionStorage.someDataKeyName;  
…  
  
Обратите внимание, что создавать и получать доступ к данным в хранилищах DOM Storage можно не только через индексаторы типа sessionStorage[ 'someDataKeyName' ], но и через псевдосвойства. Первая попытка записать данные в такое свойство создаст его экземпляр в хранилище.  
  
Данные объекта window.localStorage могут быть доступны как для субдомена так и для родительского домена, скажем следующий пример при работе с доменом test.example.com сработает:  
  
…  
var someStorage = localStorage[‘элемент example.com’];  
…  
  
  
Однако, к другим субдоменам test.example.com доступа не имеет, следующий пример для контекста test.example.com неверный:  
…  
var someStorage = localStorage[‘элемент mail.example.com’];  
…

Заключение

В этой статье я постарался рассмотреть относительно новый механизм DOM Storage, который является частью HTML 5 полностью поддерживается Internet Explorer 8. К сожалению, еще не все браузеры поддерживают DOM Storage, так например поддержка отсутствует у браузеров **Chrome** и **Opera**. Это в некоторой мере мешает распространению технологии, которая может стать очень полезной при разработке клиентских web-страниц с богатым функционалом.  
  
DOM Storage помогает работать с данными на стороне клиента и приходит на смену механизму cookie, используемому для этих целей ранее. Снимая ограничения определенные механизмом cookie, DOM Storage предлагает не менее простой и эффективный способ хранения данных.

Дополнительная информация

Статья MSDN с описанием DOM Storage <http://msdn.microsoft.com/en-us/library/cc197062(VS.85).aspx>  
Видео How I do на MSDN с описанием <http://msdn.microsoft.com/en-us/ie/dd535732.aspx>  
Статья John Resig о DOM Storage <http://ejohn.org/blog/dom-storage/>  
Сравнение браузеров по функционалу <http://robertnyman.com/javascript/>   
DOM Storage в wiki <http://en.wikipedia.org/wiki/DOM_storage>  
Черновик спецификации Web Storage <http://dev.w3.org/html5/webstorage/>

Локальное хранилище

localStorage — это локальное хранилище, которое используется для хранения данных в специально отведенном месте в браузере. localStorage имеет простой API для извлечения и запись данных в локальное хранилище. Он может хранить до 10 Мб данных для одного домена. В отличие от файлов cookie, Web Storage не делает каждый раз запрос HTTP.

Оглавление:

С появлением Web Storage недостатки [cookies](https://htmlhook.ru/otpravit-cookie-v-brauzer.html) для хранения данных в браузере были решены.

Недостатки cookies

Читайте также: [Примеры JavaScript](https://htmlhook.ru/javascript-primery-i-kody.html)

* Ограничение размера хранилища файлов cookie в веб-браузерах, около 4Кб.
* Cookies отправляются с каждым запросом HTTP, тем самым замедляя производительность веб-приложений.

Объекты Web Storage

Local storage: хранит данные без срока годности. Эти данные будут доступны даже если вкладки браузера открыты или закрыты.

Session storage: хранит данные за один сеанс. Материалы данных будут очищены, как только пользователь закроет браузер.

Local storage сохраняет данные в виде пары ключ/значение setItem('key', 'value').

данные локального хранилища не будут доступны в других браузерах, потому что дынные хранятся только в активном браузере.

[К началу](https://htmlhook.ru/html5-web-storage.html#toc)

Пример

Простая веб-форма позволит нам изменить цвета фона страницы и размер шрифта в заголовке.

<form onsubmit="javascript:setSettings()">

<label>Выбрать цвет для фона: </label>

<input id="favcolor" type="color" value="#ffffff" />

<label>Выбрать размер шрифта: </label>

<input id="fontwt" type="number" max="14" min="10" value="13" />

<input type="submit" value="Save" />

<input onclick="clearSettings()" type="reset" value="Стереть" />

</form>

Функция setSettings вызывает из формы событие onsubmit, которое сохранит выбранные значения в локальное хранилище.

Проверить, поддерживает браузер Web Storage или нет.

function setSettings() {

if ('localStorage' in window && window['localStorage'] !== null) {

// объект Local Storage для хранения данных

} else {

alert('Данные не сохранятся, ваш браузер не поддерживает Local storage');

}

}

Для браузеров которые не поддерживают LocalStorage, можно подключить библиотеку *JavaScript Modernizr*.

<script type="text/javascript" src="modernizr.min.js"></script>

if (Modernizr.localstorage) {

// объект Local Storage для хранения данных

} else {

alert('Данные не сохранятся, ваш браузер не поддерживает Local storage');

}

Лучше всегда добавлять блоки try/catch для хранения кода при сохранении данных. QUOTA\_EXCEEDED\_ERR исключение, если лимит хранилища превышает 5Мб.

function setSettings() {

if ('localStorage' in window && window['localStorage'] !== null) {

try {

var favcolor = document.getElementById('favcolor').value;

var fontwt = document.getElementById('fontwt').value;

localStorage.setItem('bgcolor', favcolor);

localStorage.fontweight = fontwt;

} catch (e) {

if (e == QUOTA\_EXCEEDED\_ERR) {

alert('Переполнение хранилища!');

}

}

} else {

alert('Данные не сохранятся, ваш браузер не поддерживает Localstorage');

}

}

Если изменить фон и размер шрифта. Ключ getItem('Key') покажет сохранённые данные, и некоторое время фон, размер шрифта не изменится.

function applySetting() {

if (localStorage.length != 0) {

document.body.style.backgroundColor = localStorage.getItem('bgcolor');

document.body.style.fontSize = localStorage.fontweight + 'px';

document.getElementById('favcolor').value = localStorage.bgcolor;

document.getElementById('fontwt').value = localStorage.fontweight;

} else {

document.body.style.backgroundColor = '#FFFFFF';

document.body.style.fontSize = '14px'

document.getElementById('favcolor').value = '#FFFFFF';

document.getElementById('fontwt').value = '14';

}

}

Функция length возвращает общее количество значений в хранилище.

Приведенные выше функции могут быть вызваны событием onload для тега bodyследующим образом:

body onload="applySetting()"

Функция clear() или removeItem('key') очистит локальное хранилище. В примере ниже, функция вызывается событием click, кнопка очистить.

function clearSettings() {

localStorage.removeItem("bgcolor");

localStorage.removeItem("fontweight");

document.body.style.backgroundColor = '#FFFFFF';

document.body.style.fontSize = '14px'

document.getElementById('favcolor').value = '#FFFFFF';

document.getElementById('fontwt').value = '14';

}

[localStorage](https://demo.htmlhook.ru/javascript/Web-Storage/localStorage.htm)

[К началу](https://htmlhook.ru/html5-web-storage.html#toc)

Хранение событий

При удалении данных запускается событие window. Мы можем добавить вызов событий и обработку изменений, если это необходимо.

window.addEventListener('storage', storageEventHandler, false);

function storageEventHandler(event) {

applySetting();

}

Атрибуты событий:

* key свойство, которое изменилось
* newValue новое заданное значение
* oldValue ранее сохраненное значение
* url полный адрес события, где оно произошло
* storageArea объекты localStorage или sessionStorage

Помните, что событие останется без изменений, если не будет никаких изменений в данных. Такие же методы API применяются для хранения сессии, кроме того методы должны быть выполнены объектом SessionStorage.

Проверить есть данные в локальном хранилище, можно с помощью средств разработчика. Например, в Мазиле нажмите правой кнопкой мыши на веб-странице и выберите Исследовать элемент Настройки инструментов.

Если в хранилище есть данные, Вы увидите данные хранящиеся в виде пары ключ/значение.

[К началу](https://htmlhook.ru/html5-web-storage.html#toc)

Поддержка браузерами localStorage

Таблица поддержки браузерами: “Локальное хранилище”

Заключение

Теперь вы можете использовать Web Storage для хранения настроек пользователя, информацию сеанса т.д. Вы также можете попробовать создать приложение, которое сможет работать в автономном режиме, а данные изменившиеся во время автономной работы, могут быть отправлены в виде пакета обновлений когда пользователь подключится к сети.

 10.08.2013  21.11.2016 автор Виктор Клим [JavaScript](https://htmlhook.ru/javascript) [api](https://htmlhook.ru/tag/api)

Записи по метке: api

* [DOM API](https://htmlhook.ru/dom-api.html)
* [classList](https://htmlhook.ru/classlist.html)
* [Быстрые ссылки](https://htmlhook.ru/bystrye-ssylki.html)
* [Web Notifications API](https://htmlhook.ru/web-notifications-api.html)
* [Атрибут data-\*](https://htmlhook.ru/atributy-dannyh-html5.html)

Добавить комментарий

Начало формы

Ваш e-mail не будет опубликован. Обязательные поля помечены \*

Лекция 9:

**Хранилище Web: более удобное и мощное хранилище клиентских данных**

**A**

 |

[версия для печати](http://www.intuit.ru/intuit?destination=studies%2Fcourses%2F679%2F535%2Fprint_lecture%2F23139)

[< Лекция 8](http://www.intuit.ru/studies/courses/679/535/lecture/23138) || **Лекция 9** || [Лекция 10 >](http://www.intuit.ru/studies/courses/679/535/lecture/23141)

**Аннотация:**Рассматриваются сессионные (Session Storage) и локальные (Local Storage) хранилища данных на стороне клиента. Сравнение технологий хранилищ HTML5 с технологией Cookie. Помещение и извлечение данных из сессионного и локального хранилищ. Удаление данных. Лимит хранилища. Использование событий хранилища. Вопросы безопасности и соответствующие рекомендации.

**Ключевые слова:**[Web](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword1), [хранение данных](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword3), [w3c](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword5), [session](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword6), [storage](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword7), [local storage](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword8), [межсайтовый скриптинг](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword9), [скрытое поле](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword10), [flash](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword11), [домен](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword18),[ПО](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword19), [извлечение данных](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword21), [ключ](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword22), [значение](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword23), [this](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword25), [sentence](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword26), [предупреждающее сообщение](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword30), [атрибут](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword33), [локальная функция](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword39), [example](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword40), [org](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword41),[доступ](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword53), [поле](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword56), [пространство](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword64), [браузер](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword69), [злоумышленник](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword70), [DNS](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword71), [SSL](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword73), [сайт](http://www.intuit.ru/studies/courses/679/535/lecture/23139?page=1#keyword74)

Шветанк Диксит · 2 марта 2010 г.

**Введение**

Приложения *Web* становятся все более развитыми с каждым днем, со все более продуманным использованием JavaScript, а также появляющихся стандартов и технологий. Мы все в большей степени полагаемся на эти приложения, многие из них становятся частью нашей повседневной жизни. Одной из областей, в которой разработка приложений *Web* отстает от потребностей, является*хранение данных* на стороне клиента. То есть, до сих пор.

Хранилище *Web* (<http://dev.w3.org/html5/webstorage/>) является спецификацией *W3C*, которая предоставляет функции для сохранения данных на стороне клиента до конца сеанса (*Session* *Storage* – сессионное хранилище), или после завершения сеанса (*Local Storage* – локальное хранилище). Это значительно более мощное средство, чем традиционные cookies, и более простое в использовании. В этой статье мы посмотрим, почему это так, и как можно его использовать.

**Существующая проблема: cookies могут разрушаться**

Прежде чем двигаться дальше, давайте коротко разберем, почему текущий способ хранения данных на стороне клиента — cookies — является проблемой:

* Маленький размер: Cookies обычно имеют максимальный размер около 4KB, что не слишком хорошо подходит для хранения сложных данных любого вида.
* С помощью cookies трудно отслеживать две или больше транзакций на одном и том же сайте, которые могут происходить в двух или более различных вкладках.
* Cookies могут использоваться злонамеренно с помощью так называемой техники *межсайтового скриптинга*, что приводит к проблемам безопасности.

Другие (менее популярные) альтернативы для cookies включают методы, использующие строки запросов, *скрытые поля* форм, совместно используемые локальные объекты на основе *flash*, и т.д. Каждый со своим собственным набором проблем, связанным с безопасностью, легкостью использования, ограничениями на размер, и т.д. Поэтому до сих пор мы используем достаточно плохие способы хранения данных на стороне пользователя. Нам требуется улучшенный способ, и здесь на помощь приходит Хранилище*Web*.

**Хранилище Web**

Спецификация Хранилища *Web* *W3C* (<http://dev.w3.org/html5/webstorage/>) была разработана для улучшения способа хранения данных на стороне клиента. Она имеет два различных типа хранилища: *Session* *Storage* (Сессионное хранилище) и *Local Storage*(Локальное хранилище).

Как сессионное, так и локальное хранилище будут иметь возможность хранить около 5Mb данных на *домен*, что значительно больше чем cookies. *По* мере дальнейшего чтения вы больше узнаете о них, и о том, что делает хранилище *Web* более удобным механизмом хранения.

**Сессионное хранилище**

Сессионное хранилище имеет единственное предназначение: Запоминать все данные сеанса и забывать их, как только закрывается используемая вкладка (или окно).

Задание и *извлечение данных*

Чтобы задать пару *ключ* *значение* в сессионном хранилище, необходимо написать просто строку следующего вида:

sessionStorage.setItem(yourkey, yourvalue);

Чтобы снова извлечь данные, необходимо написать:

var item = sessionStorage.getItem(yourkey);

Чтобы сохранить *значение* "*This* is a sample *sentence*" в сессионном хранилище, можно написать:

sessionStorage.setItem(1, 'This is a sample sentence');

Здесь значением ключа будет 1, но это не значит, что это вообще первое *значение*. Число 1 просто преобразуется в строку '1', которая используется в качестве ключа, но это не помещает эту пару *ключ* *значение* в первую позицию.

А чтобы извлечь это предложение в *предупреждающем сообщении* JavaScript, необходимо написать:

var item = sessionStorage.getItem(1);

alert(item);

Другим примером setItem() может быть:

sessionStorage.setItem('name', 'john');

а извлечь *значение* можно с помощью

var name = sessionStorage.getItem('name');

Удаление данных

Существуют также методы для удаления данных из сессионного хранилища. Метод removeItem() используется для удаления определенного объекта из списка:

var item = sessionStorage.removeItem(yourkey);

Помните, что можно также сослаться просто на *ключ* объекта и удалить его из списка следующим образом:

var items = sessionStorage.removeItem(1);

Метод clear() используется для удаления всех объектов в списке; он применяется следующим образом:

sessionStorage.clear();

Можно использовать также *атрибут* length, чтобы определить число пар *ключ*/*значение* в хранилище, следующим образом:

var no\_of\_items = sessionStorage.length;

**Локальное хранилище**

Локальное хранилище используется, если требуется, чтобы данные сохранялись более чем для одного сеанса. Простым примером использования будет подсчет количества посещений пользователем страницы *Web*. Когда страница использует локальное хранилище, страница (или окно) можно закрыть и снова открыть, и, тем не менее, показать сохраненные данные — т.е. обеспечивается постоянное хранение.

Сохранение и *извлечение данных* в локальном хранилище работает аналогично сессионному хранилищу: оно использует такие же имена функций setItem() и getItem(). Чтобы сохранить предложение в локальном хранилище, нужно написать что-нибудь следующего вида:

localStorage.setItem(1, 'This is a sample sentence');

а чтобы извлечь его:

var data = localStorage.getItem(1);

Также как и сессионное хранилище, локальное хранилище поддерживает *атрибут* length, и функции removeItem() иclear().

Как в сессионном хранилище, так и в *локальном функция* clear() имеет одну задачу – удалить все значения из списка. Это означает, что если вызвать, например, функцию localStorage.clear(), то она удалит все локальное хранилище из этого источника. Поэтому все данные локального хранилища из, скажем, (такие как www.*example*.*org*, www.*example*.*org*:80, www.*example*.*org*/abc/, www.*example*.*org*/xyz/) будут удалены. Тем не менее, хранилище, скажем, для abc.*example*.*org* этим затронуто не будет. Однако для сессионного хранилища она будет очищать хранилище только для текущей сессии.

**Простой пример**

Чтобы проиллюстрировать хранилище *Web* в действии, был создан небольшой пример, который использует, как локальное, так и сессионное хранилище. Посмотрите демонстрационную страницу хранилища *Web*, чтобы увидеть его в действии (<http://people.opera.com/shwetankd/external/demos/webstorage_demo.htm>). Пример попросит ввести две строки, одну для сессионного хранилища, и другую для локального хранилища. Затем можно открыть *Storage* Inspector в Opera Dragonfly, чтобы получить *доступ* к хранилищу *Web*. Можно заметить, что если закрыть страницу, а затем снова ее открыть, данные, введенные для локального хранилища, сохраняются, в то время как для сессионного хранилища, это будет не так.

**Использование событий хранилища**

Спецификация предоставляет также событие хранилища (<http://dev.w3.org/html5/webstorage/#the-storage-event>), которое будет порождаться, когда область хранилища изменяется. Оно имеет различные полезные атрибуты, такие как:

* storageArea: Говорит, какой это тип хранилища (сессионное или локальное)
* key: Изменяющийся ключ.
* oldValue: Старое значение ключа.
* newValue: Новое значение ключа.
* url: URL страницы, ключ которой изменился.

Если вызвать метод clear(), то атрибуты key, oldValue и newValue устанавливаются пустыми. Здесь имеется модифицированная версия упомянутого ранее демонстрационного примера страницы (<http://people.opera.com/shwetankd/external/demos/webstorage_demo2.htm>), в этот раз использующая события хранилища, чтобы пользователи могли знать об изменениях в значениях. Если ввести *значение*, а затем снова его изменить, то можно будет видеть предупреждение, упоминающее новое и старое значения.

**Где можно получить детальный доступ к данным хранилища Web в браузере?**

В Opera 10.50+ существует несколько способов, как это можно сделать. Можно ввести opera:webstorage, а такжеopera:config#PersistentStorage в *поле* адреса, чтобы получить *доступ* к высокоуровневым данным хранилища *Web* (какой лимит хранилища, где оно находится, и т.д.), но для разработчиков существует лучший способ получить подробную информацию о хранилище *Web* для конкретной страницы — с помощью *Storage* Inspector в Opera Dragonfly, который предоставляет значительно более подробную информацию.

Opera 10.50+ имеет новую и улучшенную утилиту отладки Opera Dragonfly (которая была опубликована как проект с открытым исходным кодом - <http://www.opera.com/dragonfly/>). Среди исправлений, усовершенствований и новых свойств появился *Storage*Inspector (Инспектор хранилища). Он создает для разработчиков отдельную вкладку для доступа к информации о cookies и локальном и сессионном хранилище страницы. Откройте Opera Dragonfly и щелкните на вкладке *Storage*, чтобы получить к ней*доступ*.

**Что нужно помнить при использовании хранилища Web**

Хранилище на источник: Все данные из одного и того же источника будут совместно использовать одно *пространство* хранения. Источником является тройка схема/хост/порт (или глобально уникальный идентификaтор). Например, [http://www.example.org](http://www.example.org/) и[http://abc.example.org](http://abc.example.org/) являются двумя отдельными источниками, также как [http://example.org](http://example.org/) и [https://example.org](https://example.org/), а также[http://example.org:80](http://example.org/) и [http://example.org:8000](http://example.org:8000/)

Лимит хранилища: В настоящее время большинство браузеров, которые реализовали хранилище *Web*, включая Opera, определяют лимит хранилища как 5 Mb на *домен*. Можно изменить этот лимит хранилища для каждого домена отдельно, сохраняя какие-то данные из домена в сеансовом или локальном хранилище, а затем переходя к opera:webstorage. Этот *домен* появится тогда в списке, и можно будет щелкнуть на кнопке, чтобы получить *доступ* к статистике и параметрам, включая размер данных, сохраненных для этого домена, какой имеется лимит хранилища, и что *браузер* будет делать, когда лимит будет исчерпан.

Вопросы безопасности и соответствующие рекомендации: Хранилище назначается на основе источника. *Злоумышленник* может использовать подмену *DNS*, чтобы представить себя определенным доменом, которым он фактически не является, получая тем самым *доступ* к области хранилища этого домена на компьютере пользователя. Можно использовать *SSL*, чтобы предотвратить такие действия, чтобы пользователи могли быть абсолютно уверены, что просматриваемый *сайт* находится в том же домене.

Где не надо использовать: Если два различных пользователя используют различные пути доступа на одном домене, они могут получить *доступ* к области хранения всего источника, и поэтому к данным друг друга. Поэтому в настоящее время не рекомендуется использовать хранилище *Web* на своих страницах пользователям свободных хостов, которые имеют свои сайты в различных каталогах одного и того же домена (например, freehostingspace.org/user1/ и freehostingspace.org/user2/ ).

Хранилище *Web* не является частью спецификации HTML5: Это целая спецификация сама *по* себе (<http://dev.w3.org/html5/webstorage/>).

Лекция 10:

# Web Workers за работой

**A**

 |

[версия для печати](http://www.intuit.ru/intuit?destination=studies%2Fcourses%2F679%2F535%2Fprint_lecture%2F23141)

[< Лекция 9](http://www.intuit.ru/studies/courses/679/535/lecture/23139) || **Лекция 10** || [Лекция 11 >](http://www.intuit.ru/studies/courses/679/535/lecture/23142)

**Аннотация:**Введение в технологию многопоточного выполнения кода Web Workers. Принципы работы и случаи использования. Какие стандартные объекты JavaScript доступны для Web Workers. Присущие ему ограничения. Поддержка в современных браузерах. Дополнительные ссылки по теме.

**Ключевые слова:**[Web](http://www.intuit.ru/studies/courses/679/535/lecture/23141?page=1#keyword1), [процессор](http://www.intuit.ru/studies/courses/679/535/lecture/23141?page=1#keyword2), [множества](http://www.intuit.ru/studies/courses/679/535/lecture/23141?page=1#keyword4), [браузер](http://www.intuit.ru/studies/courses/679/535/lecture/23141?page=1#keyword6), [API](http://www.intuit.ru/studies/courses/679/535/lecture/23141?page=1#keyword12), [worker](http://www.intuit.ru/studies/courses/679/535/lecture/23141?page=1#keyword13)

Дэниел Дэвис · 1 июля 2010 г.

### Введение

Представьте следующее. Вы являетесь любимым руководителем малоизвестной страны ScravaJipt, безраздельно властвуя над всем, что обозреваете. У вас есть главный слуга, который ухаживает за вами, покупает одежду, нажимает кнопки на мобильном телефоне. Но иногда он со всем не справляется. Он бывает перегружен всеми этим повседневными заботами, так что приходится передавать часть работы специалистам (один для нажатия кнопок, еще один для покупки рубашек, другой для покупки брюк). К счастью для него имеется множество работников, на которых можно положиться. Аналогично, к счастью для разработчиков *web*существуют виртуальные цифровые специалисты, которые могут взять на себя некоторые задачи, когда *процессор* JavaScript становится перегруженным. Познакомьтесь с *Web* Workers, одной из *множества* технологий, которые, совместно с HTML5 формируют следующее поколение открытой *Web*.

### Введение в технологию Web Workers

Встречали ли вы страницу, которая отображается частично, но не отвечает ни на какие щелчки? Или страницу, которая замораживается или приводит к аварийной остановке *браузер*?

Причина была, скорее всего, в JavaScript. Страницы *Web* становятся все в большей степени перегружены JavaScript, иногда в такой степени, что не могут двигаться. Вездесущность JavaScript является благом для разработчиков, но это означает, что язык может выполняться на широком множестве устройств, включая такие, которым не хватает мощности для сегодняшних приложений *Web*. Существует несколько способов оптимизации JavaScript, но он все равно не будет работать так же быстро, как машинный код.

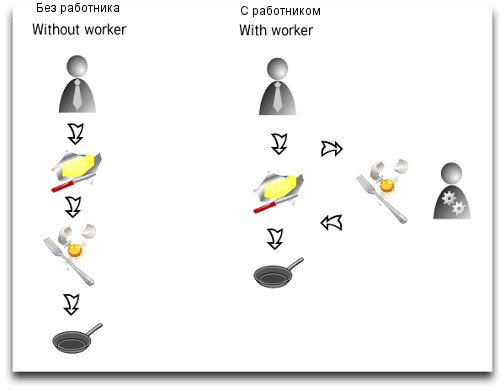
Разработчики *Web* не собираются (и не должны) в связи с этим сокращать свое использование JavaScript. Вместо этого, стандарты*Web* и браузеры, которые их реализуют, продвигаются вперед, чтобы выдержать нагрузку. Технология *Web* Workers является этому примером, вместе с различными другими *API* JavaScript, которые разработаны для повышения возможностей браузера.

#### Как работают Web Workers

Большинство современных языков программирования являются мультипоточными, т.е. они могут выполнять несколько процессов одновременно. Превращение JavaScript в мультипоточный язык потребует значительных архитектурных изменений и фундаментального переосмысливания, поэтому Web Workers предлагают способ обойти эту проблему, позволяя расширить язык таким образом, что он может казаться в некоторых случаях мультипоточным. Другими словами может эффективно выполняться более одного процесса, но с некоторыми ограничениями. На самом деле достаточно много ограничений, поэтому они будут полезны только в определенных ситуациях.

#### Когда я могу их использовать?

Возвращаясь к нашей аналогии со "специалистами" можно сказать, что Web Workers могут делать только одну вещь, но они делают ее очень хорошо. Они прекрасно справляются с выполнением быстрых вычислений, но не могут сделать более сложную работу, такую как доступ к DOM. Если сравнить web-приложение с кухней, то основной поток JavaScript будет шеф-поваром, собирающимся приготовить омлет. Если он делает все самостоятельно, он должен взбить яйцо, подготовить сковородку, растопить масло и, наконец, приготовить омлет. Если он хочет повысить эффективность, он может получить помощь от кухонного работника. Работник может взбить яйцо, позволяя шеф-повару подготовить сковороду и масло, и затем приготовить омлет. Работнику не разрешается прикасаться к сковороде или готовить омлет – он просто выполняет задачу, в то время как шеф-повар продолжает с другой работой.



Если бы Web Workers мог готовить, вот как бы он помог приготовить омлет

Использование Web Workers такое же. Если JavaScript содержит какие-то интенсивные вычисления с ресурсами, можно передать это Web *Worker* для обработки, в то время как основной процесс продолжает выполняться. Можно использовать более одного Web*Worker*, и Web *Worker* может делать более одной задачи. Давайте приготовим пример, чтобы увидеть это в действии.

#### Покажите мне просто код!

Сохраняйте спокойствие, мы прибываем на место! Работник (*worker*) сам является просто некоторым кодом JavaScript в своем собственном файле. Также как концепция Web Workers является выполнением кода в отдельном потоке, так и код самого работника должен находиться в отдельном файле, или нескольких файлах, если используется больше одного работника. В нашем примере давайте начнем с создания пустого текстового файла, который назовем *worker*.js.

В нашем основном потоке JavaScript мы используем работника, создавая новый объект *Worker*:

Основной поток JavaScript

var worker = new Worker('worker.js');

Как и с кухонным помощником, мы передаем работнику что-то, он делает что-то с этим в фоновом режиме, и затем что-то нам возвращает. Коммуникация с работником осуществляется с помощью метода postMessage:

Основной поток JavaScript

// Создаем новый объект работника

var worker = new Worker('worker.js');

// Посылаем простое сообщение, чтобы запустить работника

worker.postMessage();

Можно также передать работнику переменную:

Основной поток JavaScript

// Создаем новый объект работника

var worker = new Worker('worker.js');

// Посылаем сообщение, чтобы запустить работника

// и передаем ему переменную

var info = 'Web Workers';

worker.postMessage(info);

В работнике, т.е. внутри файла *worker*.js, мы используем событие onmessage для получения сообщения из основного потока и выполнения какой-то работы. Если передается переменная, то можно получить к ней доступ с помощью event.data следующим образом:

worker.js

// Получаем сообщение из основного потока

onmessage = function(event) {

// Выполняем что-то

var info = event.data;

};

Отправка сообщений из работника назад в основной поток использует те же методы:

worker.js

// Получить сообщение из основного потока

onmessage = function(event) {

// Сделать что-то

var info = event.data;

var result = info + ' rise up!';

postMessage(result);

};

Main JavaScript thread

// Создать новый объект worker

var worker = new Worker('worker.js');

// Послать сообщение, чтобы запустить работника и

// передать ему переменную

var info = 'Web Workers';

worker.postMessage(info);

// Получить сообщение от работника

worker.onmessage = function (event) {

// Сделать что-то

alert(event.data);

};

При желании можно загрузить следующий демонстрационный пример использования Web Workers (<http://dev.opera.com/articles/view/web-workers-rise-up/WebWorkers_demo.zip>).

Opera создана как однопоточный браузер с поддержкой множества платформ, поэтому текущая реализация Web Workers разделяет выполнение кода в одном потоке UI. Однако другие браузеры могут иметь мультипоточную архитектуру, которая позволяет одновременное выполнение различных потоков кода.

#### О чем надо помнить

Это, очевидно, очень простой пример, но когда вы задаете работникам более сложные задачи, такие как управление большими массивами, или вычисление точек в трехмерном пространстве для отображения в основном потоке, то это становится очень мощным средством. Основное, что надо помнить, однако, состоит в том, что работники не могут получить доступ к DOM. В примере выше, например, мы не можем вызвать внутри работника alert() или даже document.getElementById() - он может только получать и возвращать переменные, хотя это могут быть строки, массивы, объекты JSON, и т.д.

Вот сводка того, что доступно и недоступно для Web Workers.

Могут использовать:

* объект navigator
* объект location (только чтение)
* метод importScripts() (для доступа к файлам сценариев в том же домене)
* объекты JavaScript, такие как Object, Array, Date, Math, String
* XMLHttpRequest
* методы setTimeout() и setInterval()

Не могут использовать:

* DOM
* Порождающую работника страницу (только через postMessage() )

#### Поддержка браузеров

На момент написания не все браузеры поддерживали Web Workers, поэтому их надо использовать с осторожностью. Вместо того чтобы пытаться отследить, какие версии браузеров поддерживают и не поддерживают, проще включить проверку внутри сценария. Чтобы определить, поддерживает ли браузер пользователя Web Workers, можно проверить существование свойства*Worker* объекта window:

// Проверка, что имеется поддержка Web Workers

if (!!window.Worker) {

// Ура, можно передать утомительную работу!

}

Web Workers особенно подходят в тех ситуациях, где нежелательно заставлять пользователя ждать, пока выполняется какой-то код. Основной поток вычислений может сконцентрироваться на работе с UI, отображая его как можно быстрее, в то время как Web Workers могут в фоновом режиме обрабатывать данные, используя AJAX для коммуникации с сервером. Все счастливы, и все прекрасно в стране ScravaJipt.

### Дополнительные ссылки по теме Web Workers

* Последняя опубликованная версия спецификации Web Workers (<http://www.w3.org/TR/workers/>)
* Реми Шарп, Простая демонстрация Web Workers (<http://html5demos.com/worker>)
* Брэндон Аарон, Пример использования Web Workers (<http://brandonaaron.net/examples/connecting-the-dots-with-web-workers/1>)
* Николас К. Закас, Введение в API Web Workers (<http://answers.oreilly.com/topic/1358-introducing-the-web-workers-api/>)
* Марк Пилгрим, Обнаружение поддержки HTML5 и связанных API JavaScript (<http://diveintohtml5.org/detect.html>)

**[Дальше >>](http://www.intuit.ru/studies/courses/679/535/lecture/23142)**

[< Лекция 9](http://www.intuit.ru/studies/courses/679/535/lecture/23139) || **Лекция 10** || [Лекция 11 >](http://www.intuit.ru/studies/courses/679/535/lecture/23142)

[Яндекс.Директ](https://direct.yandex.ru/?partner)Конец формы