**Веб хранилище**

HTML5 Web Storage - обзор веб-хранилища

Веб-программирование --- HTML5 --- Обзор Web Storage

В интернете информацию можно сохранять в двух местах: на веб-сервере и на веб-клиенте (т.е. компьютере посетителя страницы). Определенные типы данных лучше хранить в одном из этих мест, а другие типы — в другом.

Правильным местом для хранения конфиденциальных и важных данных будет веб-сервер. Например, если вы положите какие-либо товары в свою корзину в онлайновом магазине, данные о вашей потенциальной покупке сохраняются на веб-сервере. На вашем компьютере сохраняется лишь несколько байтов данных для отслеживания, содержащих информацию о вас (или, вернее, о вашем компьютере), чтобы веб-сервер знал, какая из корзин ваша. Даже с новыми возможностями HTML5 изменять эту систему нет надобности — она надежная, безопасная и эффективная.

Но хранение данных на сервере не всегда является лучшим подходом, т.к. иногда легче хранить второстепенную информацию на компьютере пользователя. Например, имеет смысл хранить локально пользовательские настройки (скажем, параметры, которые определяют способ отображения веб-страницы) и состояние приложения (снимок текущего состояния веб-приложения), чтобы посетитель мог продолжить его выполнение с того же самого места позже.

До HTML5 единственным способом локального хранения данных было использование механизма файлов cookies, который первоначально был разработан для обмена небольшими объемами идентифицирующей информации между веб-серверами и браузерами. Файлы cookies подходят идеально для хранения небольших объемов данных, но модель JavaScript для работы с ними несколько неуклюжа. Система файлов cookies также вынуждает разработчика возиться со сроками действия и бесполезно пересылать данные туда и обратно по интернету с каждым запросом страницы.

В HTML5 вводится лучшая альтернатива файлам cookies, которая позволяет легко и просто сохранять информацию на компьютере посетителя. Эта информация может храниться на клиентском компьютере неограниченное время, не отправляется на веб-сервер (если только разработчик не сделает это сам), может быть большого объема и для работы с ней требуется всего лишь пара простых, эффективных объектов JavaScript.

Эта возможность называется веб-хранилищем (Web Storage) и особенно хорошо подходит для применения с автономным режимом работы вебсайтов, т.к. позволяет создавать самодостаточные автономные приложения, которые могут сохранять всю требуемую им информацию даже при отсутствии подключения к интернету.

Функциональность веб-хранилища HTML5 позволяет веб-странице сохранять данные на компьютере посетителя. Эта информация может быть кратковременной, которая удаляется после выключения браузера, или долговременной, которая остается доступной при последующих посещениях веб-страницы.

Сохраняемая в веб-хранилище информация в действительности сохраняется не в интернете, а на компьютере посетителя веб-страницы. Иными, словами, веб-хранилище означает хранение данных не в интернете, а хранение данных из интернета.

Существуют два типа веб-хранилищ, которые так или иначе связаны с двумя объектами:

**Локальное хранилище**

Использует объект localStorage для хранения данных для всего веб-сайта на постоянной основе. Это означает, что если веб-страница сохранит данные в локальном хранилище, эти данные будут доступны для пользователя, когда он возвратится на эту веб-страницу на следующий день, на следующей неделе или в следующем году.

Конечно же, большинство браузеров также предоставляет пользователю возможность очистить локальное хранилище. В некоторых браузерах она реализована как стратегия "все или ничего", и посредством ее удаляются все локальные данные, во многом подобно тому, как удаляются cookies-файлы. (В действительности, в некоторых браузерах система cookies и локальное хранилище взаимосвязаны, так что единственным способом удалить локальные данные будет удаление cookies.) А другие браузеры могут предоставлять пользователю возможность просмотра данных для каждого отдельного веб-сайта и удалять данные для выбранного сайта или сайтов.

**Хранилище данных сеансов**

Использует объект sessionStorage для временного хранения данных для одного окна или вкладки браузера. Эти данные доступны лишь до тех пор, пока пользователь не закроет окно или вкладку, после чего сеанс заканчивается и данные удаляются. Но данные сеанса сохраняются, если пользователь переходит на другой веб-сайт, а потом возвращается обратно при условии, что это происходит в том же окне браузера.

С точки зрения кода веб-страницы, как локальное хранилище, так и хранилище данных сеансов работают абсолютно одинаково. Разница состоит лишь в длительности хранения данных.

Использование локального хранилища предоставляет наилучшую возможность для сохранения требуемой информации для последующих посещений вебстраницы пользователем. А хранилище сеансов служит для хранения данных, которые нужно передавать от одной страницы другой. (В хранилище сеансов можно также хранить временные данные, используемые только на одной странице, но для этой цели прекрасно работают обычные переменные JavaScript.)

Как локальное хранилище, так и хранилище сеансов связано с доменом веб-сайта. Таким образом, если сохранить в локальном хранилище данные для страницы www.professorweb.ru/index.html, эти данные будут доступны для страницы www.professorweb.ru/contact.html, т.к. обе эти страницы имеют один и тот же домен. Но эти данные не будут доступны для страниц других доменов.

Кроме этого, т.к. веб-хранилище расположено на компьютере (или мобильном устройстве) данного пользователя, оно связано с этим компьютером, и веб-страница, открытая на данном компьютере и хранящая данные в его локальном хранилище, не имеет доступа к информации, которую она сохранила на другом компьютере. Подобным образом веб-страница создает отдельное локальное хранилище, если вы войдете в систему под другим именем пользователя или запустите другой браузер.

Хотя спецификация HTML5 не устанавливает никаких жестких правил в отношении максимального объема хранилища, большинство браузеров ограничивают его 5 Мбайт. В этот объем можно упаковать много данных, но его будет недостаточно, если вы хотите использовать локальное хранилище для оптимизации производительности и кэшировать в нем изображения или видео большого объема (и, по правде говоря, локальное хранилище не предназначено для таких целей).

Для хранения большого объема данных все еще развивающийся стандарт базы данных IndexedDB допускает локальное хранение намного большего объема — обычно 50 Мбайт для начала и больше, по согласию пользователя.

**Сохранение данных**

Прежде чем поместить фрагмент информации в локальное хранилище или хранилище сеансов, ему необходимо присвоить описательное имя. Это имя называется ключом (key) и нужно для того, чтобы данные можно было извлечь в будущем.

***Синтаксис для сохранения фрагмента данных следующий:***

***localStorage[keyName] = data;***

Допустим, например, что нам нужно сохранить фрагмент текста, присвоив ему имя текущего пользователя. Для этих данных мы можем использовать ключ username:

// JS

localStorage["username"] = "Ivan Petrov";

Конечно же, сохранять фрагмент статического текста не имеет смысла. Как правило нам требуется сохранять какие-либо переменные данные, например текущую дату, результат математического вычисления или текстовые данные, введенные пользователем в поля формы. Далее приведен пример сохранения введенных пользователем текстовых данных:

<fieldset>

<legend>Веб-хранилище</legend>

<label for="localData">Этот текст сохранится в локальном хранилище:</label>

<input id="localData"><br>

<label for="sessionData">Этот текст сохранится в хранилище данных сессии:</label>

<input id="sessionData">

</fieldset>

<div>

<button onclick="saveData()">Сохранить</button>

<button onclick="loadData()">Загрузить</button>

</div>

function saveData() {

// Получаем значения текстовых полей

var localData = document.getElementById("localData").value;

var sessionData = document.getElementById("sessionData").value;

// Сохраняем текст, введенный в текстовом поле, в локальном хранилище

localStorage["localData"] = localData;

// Сохраняем текст, введенный в текстовом поле, в хранилище сессий

sessionStorage["sessionData"] = sessionData;

}

function loadData() {

// Загружаем сохраненные данные из хранилищ

var localData = localStorage["localData"];

var sessionData = sessionStorage["sessionData"];

// Отображаем эти данные в текстовых полях

if (localData != null) {

document.getElementById("localData").value = localData;

}

if (sessionData != null) {

document.getElementById("sessionData").value = sessionData;

}

}

Страница содержит два текстовых поля: для локального хранилища (вверху) и для хранилища сеансов (внизу). Нажатие кнопки "Сохранить" сохраняет текст, введенный в текстовые поля, а нажатие кнопки "Загрузить" выводит в полях соответствующие сохраненные данные.

Веб-хранилище также поддерживает менее распространенный синтаксис свойств. Согласно правилам этого синтаксиса, мы обращаемся к ячейке хранения с именем **username** как **localStorage.username**, а не **localStorage["username"]**. Оба типа синтаксиса равнозначны, и использование того или другого является вопросом личного предпочтения.

Веб-хранилище не работает без веб-сервера

В своих исследованиях веб-хранилища вы можете столкнуться с неожиданной проблемой. Во многих браузерах веб-хранилище работает только для страниц, предоставленных веб-сервером. При этом не важно, где расположен сервер, в интернете или на вашем собственном компьютере, самое главное, просто чтобы страницы не запускались с локального жесткого диска (например, двойным щелчком по значку файла страницы).

Эта особенность является побочным эффектом способа, которым браузеры выделяют место в локальное хранилище. Как ранее говорилось, браузеры ограничивают локальное хранилище для каждого веб-сайта 5 Мбайт, для чего им нужно ассоциировать каждую страницу, которая хочет использовать локальное хранилище, с доменом вебсайта.

Что же происходит, если открыть страницу, которая использует веб-хранилище, с локального жесткого диска? Все зависит от браузера. Браузер Internet Explorer, похоже, полностью утрачивает поддержку веб-хранилища. Объекты localStorage и sessionStorage исчезают, и попытка использовать их вызывает ошибку JavaScript.

В браузере **Firefox** объекты **localStorage** и **sessionStorage** остаются на месте и, вроде бы, поддерживаются (даже **Modernizr** определяет, что поддерживаются), но все, что отправляется на хранение, исчезает неведомо куда. В браузере Chrome опять же что-то другое — большая часть функциональности веб-хранилища работает как следует, но некоторые возможности (например, событие **onStorage)** не работают.

Подобные проблемы возникают и с использованием интерфейса **File API**. Поэтому вы избавите себя от многих хлопот, если поместите тестируемую страницу на тестовый сервер, чтобы избежать всех этих неопределенностей.

**Поддержка веб-хранилища браузерами**

Веб-хранилище является одной из наиболее поддерживаемой возможностью HTML5, с хорошим уровнем поддержки в каждом основном браузере. В таблице ниже приведены минимальные версии основных браузеров, поддерживающих веб-хранилище:

Поддержка браузерами локального хранилища и хранилища данных сеансов Браузер IE Firefox Chrome Safari Opera Safari iOS Android

Минимальная версия 8 3.5 5 4 10.5 2 2

Все эти браузеры предоставляют возможность локального хранилища и хранилища данных сеанса. Но для поддержки события **onStorage** требуются более поздние версии браузеров, например IE 9, Firefox 4 или Chrome 6.

Самой проблемной является версия IE 7, которая не поддерживает веб-хранилище вообще. В качестве обходного решения можно эмулировать веб-хранилище посредством файлов **cookies**. Это не совсем идеальное решение, но оно работает. Хотя официального сценария для закрытия этого пробела не существует, несколько хороших отправных точек можно найти на странице HTML5 Cross Browser (в разделе "Web Storage").

HTML5 Web Storage - работа с веб-хранилищем

**Веб-программирование --- HTML5 --- Работа с Web Storage**

На данном этапе мы познакомились с основами работы с веб-хранилищем — как помещать информацию в него и извлекать эту информацию оттуда. Но нам нужно знать еще несколько подробностей и полезных методов, прежде чем мы сможем применять его. В последующих разделах мы изучим, как удалять информацию из веб-хранилища и просматривать всю находящуюся в нем в данный момент информацию. Также мы рассмотрим, как работать с разными типами данных, сохранять пользовательские объекты и реагировать на изменения в коллекции хранящихся данных.

**Удаление элементов**

Задача удаления хранящихся в веб-хранилище данных проста до предела. Для удаления отдельного ненужного элемента используется метод **removeItem()**, которому передается соответствующий ключ:

// JS

localStorage.removeItem("localData");

А если же требуется удалить все локальные данные, сохраненные веб-сайтом, используется более радикальный метод **clear()**:

// JS

localStorage.clear();

**Поиск всех сохраненных элементов**

Чтобы извлечь отдельный элемент данных из веб-хранилища, нам нужно знать его ключ. Но есть еще один ловкий прием. Посредством метода **key()** можно извлечь все элементы, хранящиеся в локальном хранилище или в хранилище сеансов (для текущего веб-сайта), даже если нам не известны их ключи. Этот метод полезен при отладке или когда нужно просто просмотреть, какие данные сохраняют другие страницы веб-сайта и под какими ключами.

Ниже показан простой пример, в котором нажатие кнопки активирует функцию **findAllItems()**, которая сканирует коллекцию элементов в локальном хранилище. Далее приводится полный код примера:

<button onclick="findAllItems()">Содержимое локального хранилища</button>

<ul id="itemList"></ul>

function findAllItems() {

// Получаем элемент <ul> для списка элементов данных

var itemList = document.getElementById("itemList");

// Очищаем список

itemList.innerHTML = "";

// Перебираем все элементы данных в цикле

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

// Получаем ключ текущего элемента

var key = localStorage.key(i);

// Получаем сам элемент, хранящийся под этим ключом

var item = localStorage[key];

// Заполняем список

var newItem = document.createElement("li");

newItem.innerHTML = key + ": " + item;

itemList.appendChild(newItem);

}

}

// Сохранить какие-нибудь данные для примера

window.onload = function() {

localStorage.username = "Alexandr";

localStorage.password = "12345";

localStorage.work = "programmer";

localStorage.lang = "C#";

}

**Сохранение чисел и дат**

До сих пор в наших исследованиях веб-хранилища мы обходили стороной один важный аспект. А именно — все данные, сохраняемые посредством объектов **localStorage** и **sessionStorage**, автоматически преобразуются в текст.

Со значениями, которые и так являются текстовыми (например, имя пользователя, вводимое в текстовое поле), это обстоятельство не представляет никаких проблем. Но числа не такие сговорчивые.

Текст и числа легко поддаются обработке, но при сохранении в веб-хранилище других типов данных следует соблюдать осторожность. Для некоторых типов данных существуют удобные процедуры приведения типов (например Number для чисел). Но допустим, например, мы сохранили следующую дату:

var today = new Date();

Этот код сохраняет не объект даты, а текстовую строку, например Sat Jun 09 2013 13:30:46. К сожалению, не существует легкого способа для преобразования этого текста обратно в объект даты при извлечении его из хранилища. А если у нас нет объекта даты, мы не сможем манипулировать этой датой, как и ее исходным объектом, например вызывать его методы и выполнять вычисления.

Чтобы решить эту проблему, разработчик должен явно преобразовать дату в текст, а потом преобразовать извлеченный из хранилища текст обратно в правильный объект даты. Пример таких преобразований, включая и пример с числом, приводится в следующем коде:

window.onload = function() {

// Сохранить какие-нибудь данные

localStorage.mynumber = 5;

// Получить число при считывании

var x = Number(localStorage.mynumber);

// Проверяем (если не использовать приведение к типу Number,

// результатом будет строка "520")

alert(x + 20);

// Создаем новый объект даты

var today = new Date();

// Преобразуем дату в тестовую строку в формате ГГГГ/ММ/ДД

// и сохраняем эту строку

sessionStorage.session\_started = today.getFullYear() +

"/" + today.getMonth() + "/" + today.getDate();

// Теперь извлекаем из хранилища строку даты и с ее помощью

// создаем новый объект Date.

// Это возможно благодаря распознаваемому формату текста даты

today = new Date(sessionStorage.session\_started);

alert(today.getFullYear());

}

**Сохранение объектов**

В предыдущем разделе мы рассмотрели преобразование чисел и дат в текст для сохранения в веб-хранилище и обратное преобразование данных, извлеченных из веб-хранилища. Выполнение этих преобразований доступно благодаря таким возможностям языка JavaScript, как функция Number(), и преобразованиям, зашитым в объекты данных. Но существует большое количество других объектов, которые нельзя преобразовывать таким способом. Классическим примером таких объектов будет пользовательский объект.

Чтобы сохранить пользовательский объект в веб-хранилище, нам нужно преобразовать его в текст. Это можно было бы сделать с помощью кода, что отвечало бы цели, но потребовало бы приложения больших усилий. К счастью, существует более простой, стандартный способ решения этой задачи, который называется кодированием JSON (JavaScript Object Notation - система обозначений объектов JavaScript).

Облегченный формат JSON преобразует структурированные данные — наподобие всех значений, обернутых в объект в текст. Но лучшее в JSON то, что его поддержка встроена в браузеры. Это означает, что мы можем преобразовать в текст любой объект JavaScript вместе с его данными простым вызовом метода **JSON.stringify()**, а метод **JSON.parse()** преобразует этот текст обратно в объект.

Далее приведен код примера использования этих методов с простым пользовательским объектом **UserInfo**:

// Определяем тип данных UserInfo

function UserInfo(name, family, age, login) {

this.name = name;

this.family = family;

this.age = age;

this.login = login;

}

window.onload = function() {

// Создаем объект UserInfo

var user = new UserInfo("Иван", "Петров", 28, "ivan85\_krevedko");

// Сохраняем этот объект в формате JSON

sessionStorage.userinfo = JSON.stringify(user);

// Для проверки

user = null;

// Преобразуем JSON-текст в соответствующий объект

user = JSON.parse(sessionStorage.userinfo);

alert("Привет " + user.name + " " + user.family);

}

**Реагирование на изменения в хранилище**

Кроме рассмотренных применений, веб-хранилище предоставляет способ для общения между разными окнами браузера. Это возможно благодаря тому, что каждое изменение локального хранилища или хранилища сеансов активирует событие **window.onstorage** во всех других окнах, в которых просматривается та же страница или другая страница того же самого сайта.

Таким образом, изменение локального хранилища для страницы www.professorweb.ru/index.html активирует событие onstorage в окне браузера для страницы www.professorweb.ru/about.html. (Конечно же, страницы должны просматриваться в том же браузере и на том же компьютере, но вы это уже и так знали, не так ли?)

Событие onstorage активируется при добавлении объекта в хранилище, изменении объекта, находящегося в хранилище, удалении объекта из хранилища или полной очистке хранилища. Событие не активируется, если код выполняет операцию с хранилищем, которая не имеет никакого эффекта (например, сохранение значения, которое уже находится в хранилище, или очистка пустого хранилища).

Рассмотрим пример страницы показанной на рисунке ниже. Здесь в хранилище можно добавить любое значение с любым ключом, заполнив соответствующие текстовые поля и нажав кнопку "Добавить". Добавленное значение отображается на другой, уже открытой странице.

Чтобы создать пример, показанный на этих рисунках, сначала нужно создать страницу, на которой выполняется ввод и сохранение данных. Данные сохраняются функцией addValue(), которая активируется нажатием кнопки "Добавить" и выглядит так:

function addValue() {

// Получаем значения из обоих текстовых полей

var key = document.getElementById("key").value;

var item = document.getElementById("item").value;

// Сохраняем элемент в локальном хранилище.

// Если ключ уже существует, новый элемент заменяет старый

localStorage[key] = item;

}

Вторая страница тоже не представляет ничего сложного. При загрузке страницы событию window.onstorage присваивается функция с помощью следующего кода:

window.onload = function() {

// Подключаем событие onstorage к функции storageChanged

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

};

function storageChanged(e) {

var message = document.getElementById("updateMessage");

message.innerHTML = "Обновление локального хранилища.";

message.innerHTML += "<br>Ключ: " + e.key;

message.innerHTML += "<br>Старое значение: " + e.oldValue;

message.innerHTML += "<br>Новое значение: " + e.newValue;

message.innerHTML += "<br>URL: " + e.url;

}

Этот код выглядит несколько иначе, чем код для обработки событий, с которым мы сталкивались до сих пор. Вместо установки события **window.onstorage** он вызывает функцию **window.addEventListener()**. Это сделано для того, чтобы код работал на всех современных браузерах. Если активировать событие **window.onstorage** напрямую, код будет работать на всех браузерах, за исключением Firefox.

Старожилы, наверное, помнят, что метод addEventListener() не работает на версиях Internet Explorer 8 и более ранних. В данном примере это ограничение несущественно, т.к. IE 8 не поддерживает события хранилища в любом случае.

Функция **storageChanged()** выполняет простую задачу. Она берет обновленную информацию и выводит ее на странице в элементе <div>.

Как видно, событие onstorage предоставляет несколько единиц информации, в том числе ключ измененного значения, старое значение, новое значение и URL страницы, на которой изменение было выполнено. Если событие onstorage активировано вставкой нового элемента, значение свойства **e.oldValue** равно либо **null** (в большинстве браузеров), либо пустой строке (в Internet Explorer).

Если открыто несколько страниц одного и того же веб-сайта, событие **onstorage** активируется в каждой из этих страниц, за исключением страницы, на которой было осуществлено изменение (в данном примере это будет страница StorageEvents1.html). Но, как всегда, Internet Explorer не следует этому правилу и активирует событие **onstorage** и на странице, выполняющей изменения.

Alexandr Erohin ✯ alexerohinzzz@gmail.com © 2011 – 2017

**HTML5 Web Storage - пример приложения**

Веб-программирование --- HTML5 --- Web Storage - пример приложения

На данном этапе у вас может сложиться впечатление, что работа с веб-хранилищем не представляет ничего особенного: нужно всего лишь выбрать ключ для данных и заключить его в квадратные скобки (или использовать синтаксис свойств). По большему счету вы будете правы. Но локальному хранилищу можно дать дополнительное, более практическое применение, не прилагая для этого дополнительных усилий.

Возьмем, например, игру-лабиринт на основе Canvas, которую мы рассмотрели в статье "HTML5 Canvas - простая игра". На определенном этапе игры пользователь может отвлечься, но при этом он, вероятно, захочет не просто бросить игру, а сохранить ее текущее состояние, чтобы возобновить ее с этого момента. В таком случае имеет смысл сохранить текущее состояние игры, чтобы пользователь мог возвратиться к ней позже.

Существует несколько способов реализации такой возможности. Можно просто сохранять новую позицию после каждого хода. Поскольку локальное хранилище обладает высоким быстродействием, такой подход вполне реализуем. Либо можно реагировать на событие страницы **onbeforeunload** и запрашивать пользователя, не желает ли он сохранить текущее состояние.

Код, который предлагает сохранить информацию о состоянии игры, выглядит так:

window.onbeforeunload = function(e) {

// Проверяем, существует ли объект localStorage (т.к. нет смысла

// предлагать сохранять состояние, если мы не сможем это сделать)

if (localStorage)

{

// Выводим запрос о сохранении состояния

if (confirm("Вы хотите сохранить позицию значка, \n" +

"чтобы загрузить ее при следующем входе в игру?")) {

localStorage["mazeGame\_currentX"] = x;

localStorage["mazeGame\_currentY"] = y;

}

}

}

Используйте длинный ключ сохраняемых данных наподобие **mazeGame\_currentX**. В конце концов разработчик должен обеспечить однозначность ключей, чтобы один и тот же ключ случайно не использовался двумя (или, еще хуже, несколькими) веб-страницами для сохранения разных фрагментов данных. При наличии только одного контейнера для сохранения всех данных породить конфликт ключей проще всего, что является одной из кричащих слабостей системы веб-хранилища. Во избежание подобного рода проблем следует разработать план для создания логичных и описательных ключей.

При следующей загрузке страницы можно проверить, существует ли эта информация:

// Возможность локального хранилища поддерживается?

if (localStorage) {

// Пытаемся получить данные

var savedX = localStorage["mazeGame\_currentX"];

var savedY = localStorage["mazeGame\_currentY"];

// При нулевых значениях переменных не считываем никаких данных.

// В противном случае устанавливаем новые координаты

// по сохраненным данным

if (savedX != null) x = Number(savedX);

if (savedY != null) y = Number(savedY);

}

В данном примере показано, как сохранять состояние приложения. Если бы мы не хотели надоедать пользователю каждый раз при выходе из игры, можно было бы добавить флажок для автоматического сохранения состояния игры при выходе. При таком подходе состояние игры будет сохраняться автоматически, если этот флажок установлен. Конечно же, здесь нам также нужно было бы сохранять состояние флажка, что дает нам пример сохранения настроек приложения.