<!DOCTYPE html>

<html>

<head>

<title></title>

<meta charset="utf-8">

<script src="[js/jquery-1.5.2.min.js](view-source:http://www.linkexchanger.su/examples_2011/webstorage/js/jquery-1.5.2.min.js)" type="text/javascript"></script>

<style type="text/css">

button { width:150px; }

input { width:260px; }

#result {

width:410px; height:200px;

overflow:auto;

border:1px solid #369;

padding:0 5px;

}

p {

font-size:12px; color:#666;

margin:0; border-bottom:1px dotted #ccc;

}

p.er { color:#c00; }

</style>

<script type="text/javascript">

$(function(){

var result = $('#result').empty();

// локальное хранилище

if(window.localStorage)

{

window.localStorage.setItem('myOtherData', 'Какие-то произвольные данные');

} else {

result.prepend('<p class="er">Браузер не поддерживает локальное хранилище.</p>');

$('#btn').attr('disabled','disabled');

}

// добавить/изменить данные

$('#btn').bind('click',function(){

window.localStorage.setItem('myText', $('input').val());

$('#result').prepend('<p>' + $('input').val() + '</p>');

});

// добавляем обработчик события storage

window.addEventListener('storage', function(e){

$('#result').prepend('<p>Ключ (свойство key): ' + e.key +

'<br />Новое значение (свойство newValue): ' + e.newValue +

'<br />Старое значение (свойство oldValue): ' + e.oldValue +

'<br />Адрес ресурса (свойство url): ' + e.url +

'<br />Ссылка на хранилище (свойство storageArea): ' + e.storageArea.length + '</p>');

}, true);

});

</script>

</head>

<body>

<h1>Файл example-c.html</h1>

<ul>

<li><a href="[example-a.html](view-source:http://www.linkexchanger.su/examples_2011/webstorage/example-a.html)">example-a</a></li>

<li><a href="[example-b.html](view-source:http://www.linkexchanger.su/examples_2011/webstorage/example-b.html)">example-b</a></li>

<li>example-c</li>

<li><a href="[example-d.html](view-source:http://www.linkexchanger.su/examples_2011/webstorage/example-d.html)">example-d</a></li>

</ul>

<input type="text" value=""><button id="btn">Изменить</button>

<div id="result"></div>

</body>

</html>

Web Storage наверное, одна из самых простых для понимания спецификаций HTML5.

Обычно **Web Storage** рассматривают как развитие технологии *cookie*\*. Действительно, при всей универсальности (поддерживаются производителями браузеров с незапамятных времен), файлам *cookie* присущи серьезные недостатки:  
- ограниченный, и очень маленький размер файлов. Обычно не более 4 Кбайт;  
- передача от браузера к серверу и обратно при каждом запросе;  
Это только основные недостатки, углубляться в остальные не будем, чтобы не отвлекаться от темы.

Куки (от англ. cookie — печенье) — небольшой фрагмент данных, созданный веб-сервером или веб-страницей и хранимый на компьютере пользователя в виде файла, который веб-клиент (обычно веб-браузер) каждый раз пересылает веб-серверу в HTTP-запросе при попытке открыть страницу соответствующего сайта. Применяется для:  
- аутентификации пользователя;  
- хранения персональных предпочтений и настроек пользователя;  
- отслеживания состояния сессии доступа пользователя;  
- ведения статистики о пользователях.

**Что получаем от использования Web Storage?**  
Во-первых, общий размер сохраняемых данных до 5 Мбайт (по крайней мере это рекомендации производителям браузеров);  
Во-вторых, данные обоих хранилищ (локальное хранилище и хранилище сеанса) не гоняются от клиента к серверу и обратно;  
В-третьих, в настоящее время Web Storage уже [поддерживается](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B1%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80%D0%BE%D0%B2_%28HTML5%29#.D0.A1.D0.B2.D1.8F.D0.B7.D0.B0.D0.BD.D0.BD.D1.8B.D0.B5_.D1.81.D0.BF.D0.B5.D1.86.D0.B8.D1.84.D0.B8.D0.BA.D0.B0.D1.86.D0.B8.D0.B8) последними версиями основных браузеров (к сожалению, отличия в реализации все равно существуют);

Большой минус – число поклонников древнего программного обеспечения среди пользователей Интернета снижается не слишком быстро и разработчик должен взвалить на свои плечи обязанность проверить, поддерживает ли браузер эту технологию.

Проверка сводится к проверке существования объектов хранилища сеансов *window.sessionStorage* и локального хранилища *window.localStorage*.

Вы можете открыть файл [example-a.html](http://www.linkexchanger.su/examples_2011/webstorage/example-a.html), чтобы увидеть пример в действии. Попробуйте открыть пример в разных браузерах.

Выполняем проверку *window.sessionStorage*

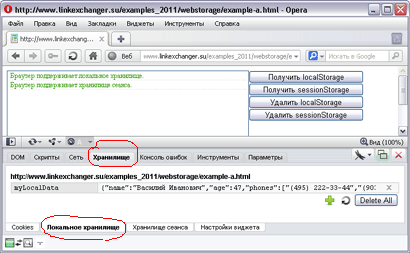
|  |  |  |
| --- | --- | --- |
| 1 | if(window.sessionStorage) | |
| 2 | { |

|  |  |  |
| --- | --- | --- |
| 3 | alert('Браузер поддерживает хранилище сеанса'); | |
| 4 | } else { |

|  |  |  |
| --- | --- | --- |
| 5 | alert('Браузер не поддерживает хранилище сеанса'); | |
| 6 | } |

Аналогичным образом можно выполнить и проверку существования *window.localStorage*.

Если мы убедились, что используемый браузер поддерживает эту технологию, то настало время «живьем» посмотреть на этот самый Web Storage и узнать, чем отличается локальное хранилище от хранилища сеанса.

Возьмем для примера браузер Opera. Если нажать комбинацию клавиш Ctrl+Shift+I, откроется панель «Dragon Fly» – встроенное в браузер средство отладки. Можете попробовать и Google Chrome – используйте точно такое же сочетание клавиш.  


Посмотрите – здесь доступно для просмотра, изменения и удаления содержимое обоих хранилищ. Теперь о том, почему хранилищ два и чем они отличаются.

Если приложению требуются данные, которые продолжают существовать после того, как будет закрыта вкладка или окно браузера, тогда следует использовать *localStorage*. Здесь уместно сохранять данные, содержащие например, пользовательские настройки каких-либо интерфейсов.

Если же данные должны храниться в течение времени, которое необходимо для просмотра нескольких страниц (диалоговые окна, многостраничные формы) – подойдет *sessionStorage*. В этом случае данные перестанут существовать, как только пользователь закроет вкладку или окно браузера.

В остальном оба хранилища выглядят братьями-близнецами. По крайней мере установка, получение и удаление данных для обоих типов хранилищ не отличается ничем, кроме названия объекта – *sessionStorage* или *localStorage*.

Попробуем сохранить какие-либо данные в хранилище сеансов:

|  |  |  |
| --- | --- | --- |
| 1 | window.sessionStorage.setItem('mySessionKey', | |
| 2 | 'Какие-то данные в хранилище сеанса'); |

Метод **setItem** принимает два аргумента – строку с именем ключа и строку с данными, которые в дальнейшем можно будет извлечь по соответствующему им ключу. Причем сделать это очень просто:

|  |  |
| --- | --- |
| 1 | var d = window.sessionStorage.getItem('mySessionKey'); |

Передаем методу **getItem** в качестве аргумента имя ключа, и в переменной **d** получаем строку ‘*Какие-то данные в хранилище сеанса’*. Так же просто можно удалить ставшие ненужными данные:

|  |  |
| --- | --- |
| 1 | window.sessionStorage.removeItem('mySessionKey'); |

После этой операции, при попытке получить данные по ключу **mySessionKey** из хранилища сеанса, будем получать *null*.

Сохранять и получать значения можно и иным способом, без вызовов методов **setItem** и **getItem**. Например:

|  |  |
| --- | --- |
| 1 | // сохраняем |
| 2 | window.sessionStorage.mySessionKey2 = 'Какие-то другие данные в хранилище сеанса.'; | |

|  |  |
| --- | --- |
| 3 | // получаем |
| 4 | alert(window.sessionStorage.mySessionKey2); | |

Двигаемся дальше… Во-первых, используем в следующем примере локальное хранилище, а во-вторых, попробуем сохранить в нем не просто банальную строку, а объект, значениями свойств которого являются строки, числа, массивы и даже другие объекты.

Вообще-то, спецификация Web Storage подразумевает сохранение данных не только строкового типа. Но, по крайней мере в текущих версиях браузеров, эти возможности ограничены. Поэтому без некоторых хитростей тут не обойтись. Вернее, пока не обойтись…

Вы можете открыть файл [example-b.html](http://www.linkexchanger.su/examples_2011/webstorage/example-b.html), чтобы увидеть пример в действии. Попробуйте открыть пример в разных браузерах.

Поставим перед собой задачу сохранить в локальном хранилище под именем **myLocalData** вот такой объект:

|  |  |
| --- | --- |
| 1 | var localData = { |
| 2 | 'name': 'Василий Иванович', | |

|  |  |
| --- | --- |
| 3 | 'age': 47, |
| 4 | 'phones': ['(495) 222-33-44 begin\_of\_the\_skype\_highlighting              (495) 222-33-44      end\_of\_the\_skype\_highlighting begin\_of\_the\_skype\_highlighting              (495) 222-33-44      end\_of\_the\_skype\_highlighting', '(903) 987-55-66'], | |

|  |  |  |
| --- | --- | --- |
| 5 | 'profession': { 'one':'токарь', 'two':'слесарь', 'three':'сантехник' } | |
| 6 | } |

Казалось бы нет ничего сложного, проходили… Но, по причине, указанной выше мы сделаем несколько иначе:

|  |  |
| --- | --- |
| 1 | var data = JSON.stringify(localData); |
| 2 | window.localStorage.setItem('myLocalData', data); | |

Приведем объект JSON к строке и только потом сохраним эту строку в локальном хранилище под именем **myLocalData**.

Соответственно, при извлечении нужных нам данных проделаем обратную операцию:

|  |  |
| --- | --- |
| 1 | var local = $.parseJSON(window.localStorage.getItem('myLocalData')); |

[ см. [$.parseJSON(json)](http://api.jquery.com/jQuery.parseJSON/) ]  
после чего в переменной **local** будет храниться объект, к свойствам которого можно легко обратиться:

|  |  |
| --- | --- |
| 1 | local.name // Василий Иванович |
| 2 | local.phones[0] // (495) 222-33-44 | |

|  |  |
| --- | --- |
| 3 | local.profession.one // токарь |

Мы познакомились с методами **getItem**, **setItem** и **removeItem** объектов *localStorage* и *sessionStorage*, но это еще не все возможности.

Объекты *localStorage* и *sessionStorage* имеют еще полезное свойство **length**, в котором как Вы уже догадались, содержится число, которое может сказать о том, сколько пар ключ/значение находится в данный момент в хранилище.

|  |  |
| --- | --- |
| 1 | var n = window.localStorage.length; |

Вот так в переменную **n** мы записали количество пар ключ/значение, содержащихся в локальном хранилище. Есть у обоих объектов метод **clear**, по названию которого тоже можно догадаться о его предназначении:

|  |  |
| --- | --- |
| 1 | window.localStorage.clear(); |

Таким образом, можно удалить все содержимое хранилища (в данном случае локального).

Рассмотрим еще один метод, который обеспечивает извлечение заданного ключа по его индексу – метод **key(index)**.

|  |  |
| --- | --- |
| 1 | var myKey = window.localStorage.key(index); |

Вот так можно получить значение ключа по его индексу. Ключи индексируются начиная с нуля, т.е. первому ключу соответствует индекс 0, а последнему length – 1. При попытке получить значение ключа по несуществующему индексу будет возвращено значение *undefined* (в Opera) или *null* (в Google Chrome).

Теперь немного про ошибки, которые могут возникать при сохранении данных в хранилищах. При превышении памяти хранилища попытка сохранения в нем значения приведет к возникновении ошибки *QUOTA\_EXCEEDED\_ERR* (некоторые браузеры могут предложить увеличить размер хранилища). К слову, такая же ошибка возникнет при отключении хранилища пользователем. Соответственно, в своих приложениях с использованием как локального хранилища, так и хранилища сеанса следует предусматривать обработку таких ошибок.

Рассмотрим механизм событий, реализованный в интерфейсе **HTML5 Web Storage**.

Сначала на простом примере познакомимся с объектом *StorageEvent*. Пример продемонстрирует возможность обмена данными между двумя окнами, относящимися к одному и тому же источнику(\*). Естественно, что в примере будет использовано локальное хранилище, поскольку, как Вы помните, данные хранилища сеансов «живут» только в одном окне.

\* – ресурс адресуется с использованием одного и того же сочетания схемы, хоста и порта.

Откройте файлы примеров [example-c.html](http://www.linkexchanger.su/examples_2011/webstorage/example-c.html) и [example-d.html](http://www.linkexchanger.su/examples_2011/webstorage/example-d.html) в двух разных окнах одного и того же браузера. Попробуйте ввести какие-либо произвольные данные в поле ввода в любом окне и нажмите кнопку **Изменить**. Посмотрите, что произойдет в другом окне.

Если посмотрели пример, то попробуем немного расшифровать, что же там происходит.

При нажатии на кнопку мы просто изменяем данные для ключа **myText**, находящиеся в локальном хранилище.

|  |  |
| --- | --- |
| 1 | $('#btn').bind('click',function(){ |
| 2 | window.localStorage.setItem('myText', $('input').val()); | |

|  |  |
| --- | --- |
| 3 | }); |

Зарегистрируем обработчик события **storage**, которое будет вызываться при изменениях в хранилище, вызванных любой страницей.

|  |  |
| --- | --- |
| 1 | window.addEventListener('storage', function(e){ |
| 2 | $('#result').prepend('<p>Ключ (свойство key): ' + e.key + | |

|  |  |
| --- | --- |
| 3 | '<br />Новое значение (свойство newValue): ' + e.newValue + |
| 4 | '<br />Старое значение (свойство oldValue): ' + e.oldValue + | |

|  |  |
| --- | --- |
| 5 | '<br />Адрес ресурса (свойство url): ' + e.url + |
| 6 | '<br />Ссылка на хранилище (свойство storageArea): ' + e.storageArea.length + '</p>'); | |

|  |  |
| --- | --- |
| 7 | }, true); |

Конечно, такой обработчик не сработает в IE, но нам сейчас не до кроссбраузерности.

Рассмотрим каждое из свойств:

**key** – содержит значение ключа, который был обновлен/удален;  
**url** – адрес источника в котором произошло событие;  
**newValue** – содержит новое/измененное значение (если значение было удалено, то *null*);  
**oldValue** – содержит старое значение (если значение добавлено впервые, то *null*);  
**storageArea** – ссылка на хранилище;

Несколько подробнее можно сказать разве что о *storageArea*. Его удобно использовать, когда необходимо при изменении каких-либо данных в хранилище производить операции над другими данными из того же хранилища. В примере мы используем свойство **length**, чтобы узнать количество содержащихся в хранилище элементов.

Думаю, что когда вы будете испытывать предоставленные файлы примеров, то наверняка заметите отличия (и немалые!) в реализации **спецификации Web Storage** в различных браузерах.