**Глава 6. ПРОГРАММНЫЙ ИНТЕРФЕЙС WEB STORAGE API**

**6.1. Назначение Web Storage API и принципы его работы**

До появления HTML5 для сохранения данных на стороне клиента применялся механизм cookie, позволяющий создавать небольшие (до 4КБ) файлы. Данные в файле cookie организованы в форме пар «ключ-значение». Некоторые значения ключа зарезервированы и устанавливают срок хранения файла (expires), домен (domain) и путь (path) доступа, а также секретность (secure). Обычно, файл cookie создается браузером при обработке ответа сервера на запрос клиента. При выполнении запроса, браузер отправляет вместе с ним все присланные ранее этим доменом файлы cookie. Допустимое количество cookie варьирует от 20 до 50 на один домен, в зависимости от браузера. Сформировать cookie можно тремя способами: на стороне сервера и переслать его браузеру (в заголовке) для сохранения вместе с ответом; на стороне клиента с помощью html-тега <meta> или сценария на JavaScript. Третий способ подробно описан в [16].

В HTML5 появился новый механизм, позволяющий web-приложению сохранять данные на стороне клиента – Web Storage (иногда в русскоязычной литературе применяют термин «браузерное хранилище»). Современные браузеры обеспечивают хранение с помощью Web Storage до 5-10 МБ на каждый домен в зависимости от типа браузера. Как и в случае с файлами cookies, данные хранятся в виде пары «ключ-значение» и доступны только создавшему их домену.

Существует два основных типа хранилища данных: локальное хранилище (local Storage) и сессионное хранилище (session Storage).

Локальное хранилище, позволяет хранить данные на постоянной основе: даже если останавливается работа браузера, при следующем сеансе связи, web-приложение может извлечь из хранилища данные, записанные туда ранее.

Сессионное хранилище позволяет хранить данные только в рамках одного сеанса связи сервером. Обычно сеанс соответствует одной вкладке браузера. Если, сеанс связи разорван, данные в сессионном хранилище разрушаются.

Применить Web Storage API очень просто. Интерфейс включает всего четыре функции позволяющие: обновить и прочитать данные в двух типах хранилищ.

**6.2. Применение Web Storage API**

Рассмотрим применение Web Storage API в простом одностраничном web-приложении, которое сохраняет, изменяет и удаляет данные в локальном браузерном хранилище.

Код html-страницы приложения представлен на рис. 6.1. Теги <section> на этом рисунке свернуты, их содержимое поясняется далее.



Рис. 6.1. Код html-страницы web-приложения, демонстрирующего работу с локальным браузерным хранилищем

Весь JavaScript-код приложения находится файле WebStorage.js , который загружается элементом <script>, расположенным в начале страницы (рис. 6.1).

**6.2.1. Проверка работоспособности Web Storage API**

На рис. 6.2 представлен фрагмент html-страницы (рис. 6.1), отражающий поддержку браузером Web Storage API и содержимое его хранилища.

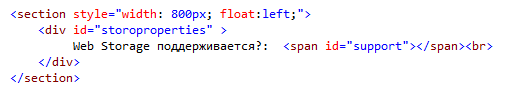


Рис. 6.2. Код html-страницы (рис. 6.1):

cодержимое первого элемента <section>

Содержимое элемента <span> формируется и заполняется анонимной функцией, обрабатывающей событие load (загрузка html-страницы) объекта window (рис. 6.3). Вначале функция проверяет наличие преопределенного объекта Storage, свидетельствующего о том, что локальное хранилище поддерживается браузером, интерпретирующим данный сценарий. Напомним, что аналогичную проверку можно выполнить с помощью JavaScript-библиотеки Modernizr, рассмотренную в 1.8.

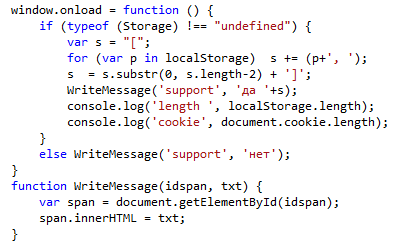


Рис. 6.3. Фрагмент файлы WebStorage.js: анонимная функция, выполняющаяся при загрузке html-страницы (рис. 6.1)

После успешной проверки объекта Storage, функция (рис. 6.3) в цикле выбирает значения ключей всех хранящихся в локальном хранилище данных и формирует строку для вывода. Непосредственно вывод строки в элемент <span> осуществляется функцией WriteMessage.

После загрузки страницы, фрагмент (рис. 6.2) в окне браузера выглядит примерно так, как на рис. 6.4. В квадратных скобках перечислены значения ключей, хранящихся в браузерном хранилище пар «ключ-значение».



Рис. 6.4. Отображение браузером фрагмента html-страницы, представленной на рис. 6.2

**6.2.2. Сохранение и извлечение простых данных**

Следующий фрагмент html-страницы (рис. 6.5, 6.6) демонстрирует сохранение и восстановление простых (не составных) данных.

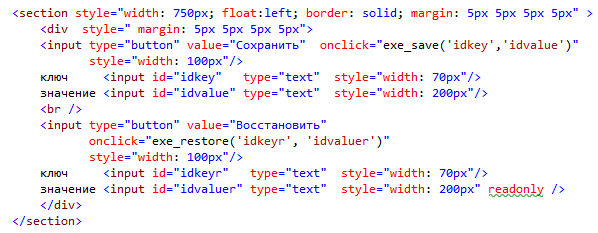


Рис. 6.5. Фрагмент кода html-страницы (рис. 6.1): запись и чтение простых данных в локальном браузерном хранилище

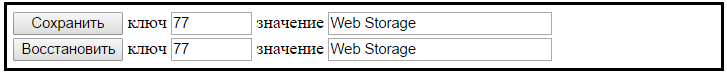


Рис. 6.6. Отображение браузером фрагмента html-страницы, представленного на рис. 6.5

Фрагмент html-страницы на рис. 6.5, 6.6 содержит два элемента <input type=”button”>, вызывающие при событии click (нажатие клавиши) функции exe\_save и exe\_restore (рис. 6.7).

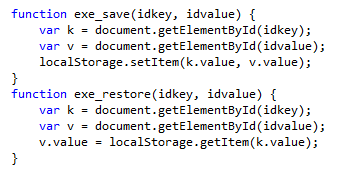


Рис. 6.7. Фрагмент файлы WebStorage.js:

функции exe\_save и exe\_restore

Функция exe\_save вызывается при нажатии клавиши «Сохранить» и принимает два параметра: идентификаторы элементов <input>, предназначенных для ввода ключа и данных. Собственно запись данных в браузерное хранилище осуществляется методом setItem, встроенного объекта localStorage. Метод принимает два параметра: ключ и значение.

Функция exe\_restore вызывается при нажатии клавиши «Восстановить» и тоже принимает два параметра: идентификатор элемента <input>, предназначенных для ввода ключа и идентификатор элемента <input readonly>, служащий для отображения извлеченных из хранилища данных. Для чтения данных из браузерного хранилища применятся метод getItem, встроенного объекта localStorage. Метод принимает два один параметр: ключ, а возвращает к точке вызова прочитанные из хранилища данные.

Если остановить работу приложения (или даже браузера), а потом запустить приложение снова и попытаться извлечь данные с помощью клавиши «Восстановить» предварительно введя ключ, то элемент с идентификатором idvaluer (рис. 6.5) отобразит соответствующие ключу данные. Браузер будет хранить данные домена до в локальном хранилище тех пор, пока они не будут удалены или изменены.

Повторная запись данных с существующим ключом приводит к замене, старых данных на новые.

**6.2.3. Сохранение и извлечение составных данных**

Третий элемент <section> (рис. 6.1) демонстрирует сохранение и извлечение составных данных с помощью Web Storage API. На рис. 6.8 представлено содержимое этого элемента, а на рис. 6.9 отображение этого фрагмента браузером.



Рис. 6.8. Фрагмент кода html-страницы (рис. 6.1): запись и чтение простых данных в локальном браузерном хранилище

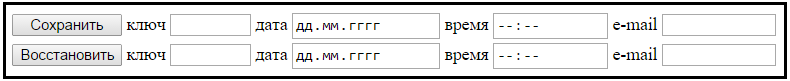


Рис. 6.9. Отображение браузером фрагмента html-страницы, представленного на рис. 6.8

Структура содержимого элемента <section> (рис. 6.8, 6.9) практически повторяет структуру предыдущего элемента (рис. 6.5, 6.6). Отличие лишь в том, что сохраняемые данные являются составные – включают три компоненты, которые вводятся с помощью трех элементов <input>.

Нажатие клавиш «Сохранить» и «Восстановить» приведет к вызову JavaScript-функций соответственно exe\_save1 и exe\_restore1 (рис. 6.10).

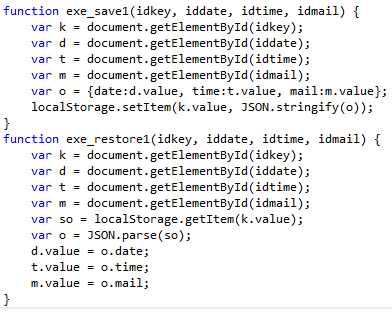


Рис. 6.10. Фрагмент файлы WebStorage.js:

функции exe\_save1 и exe\_restore1

Функция exe\_save1 принимает четыре параметра – идентификаторы элементов для ввода ключа, даты, времени и адреса электронной почты. Для того, чтобы объединить три компоненты, создается JavaScript-объект, содержащий три свойства с именами: date, time, mail.

При сохранении простых данных функцией setItem, значение заданное вторым параметром преобразуется к строке (если оно таковой не является). Для этого применяется стандартный метод toString, которым обладают все встроенные типы. При восстановлении программист должен предусмотреть возможность обратного преобразования данных.

При сохранении с помощью setItem объекта, разработчик должен либо предусмотреть метод toString в этом объекте, либо предварительно преобразовать (часто говорят «сериализовать») его к строке. Причем преобразование должно быть обратимым.

Чаще всего для сериализации JavaScript-объектов применяется формат JSON (JavaScript Object Notation). JSON – текстовый формат, предназначенный для обмена данными, основанный на JavaScript. C полным описанием JSON можно ознакомиться в [17]. Практически все современные браузеры имеют встроенную поддержку JSON. Применение JSON в сценариях JavaScript подробно описано в [18].

В функции exe\_save1 для сериализации JavaScript-объекта применяется функция JSON.stringify, которая принимает в качестве параметра JavaScript-объект, а возвращает строку в формате JSON. Эта строка запоминается в браузерном хранилище с помощью функции setItem.

Функция exe\_restore1 такие же четыре параметра, что и функция exe\_save1, причем первый параметр используется для считывания значения ключа, а три остальных применяются для отображения извлеченных из хранилища данных. Извлечение данных осуществляется с помощью функции getItem. После этого данные преобразуются из JSON-формата к своему первоначальному виду с помощью функции JSON.parse и присваиваются свойству value элементов <input> .

**6.2.4. Сохранение и извлечение изображения элемента CANVAS**

Предпоследний элемент <section> (рис. 6.1) демонстрирует сохранение и извлечение изображений элемента <canvas> с помощью Web Storage API. На рис. 6.11 представлено содержимое этого элемента, а на рис. 6.12 отображение этого фрагмента браузером.

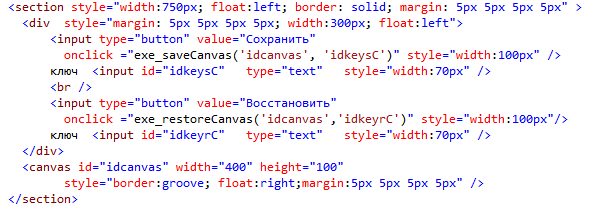


Рис. 6.11. Фрагмент кода html-страницы (рис. 6.1): запись и чтение изображения элемента <canvas> в локальном браузерном хранилище



Рис. 6.12. Отображение браузером фрагмента html-страницы, представленного на рис. 6.11

На рис. 6.13 представлен программный код, добавленный в функцию-обработчик загрузки html-страницы (рис. 6.3). С помощью метода addEventListener объекта элемента <canvas> задантся функция-обработчик события click (вызывается при щелчке мышью в области окна элемента <canvas>), в результате выполнения которой на холсте отображается красный круг радиусом 30 пикселей с центром в точке щелчка мышью. Обратите внимание: координаты точки холста передаются в функцию-обработчик в качестве параметра, а для вычисления координат этой точки относительно левого правого угла холста применяются свойства offaetLeft и offsetTop элемента <canvas>.

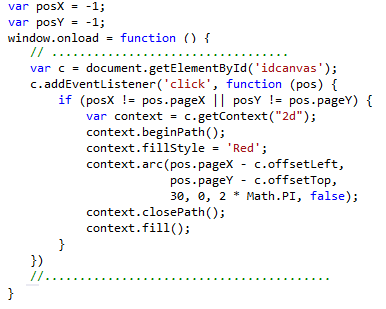


Рис. 6.13. Фрагмент файлы WebStorage.js: обработчик события onclick для элемента <canvas>

Нажатие клавиш «Сохранить» и «Восстановить» приведет к вызову JavaScript-функций с именами соответственно exe\_saveCanvas и exe\_restoreCanvas (рис. 6.14).

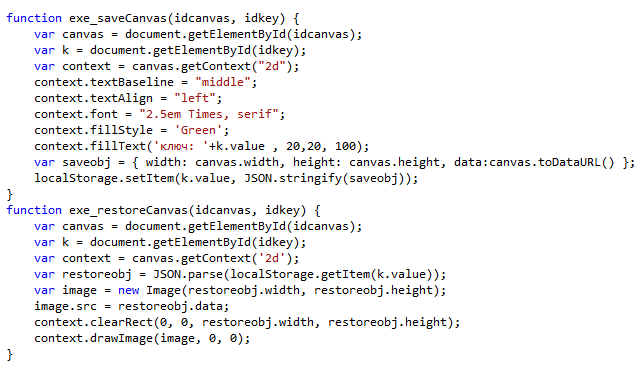


Рис. 6.14. Фрагмент файлы WebStorage.js:

функции exe\_saveCanvas и exe\_restoreCanvas

Функция exe\_saveCanvas принимает два параметра: идентификатор элемента <canvas> и идентификатор элемента <input>, используемого для ввода значения ключа. Функция выводит текст на холст и формирует объект, содержащий три свойства: размеры холста (свойства width и height) и строка символов, содержащая закодированный с помощью BASE 64 двоичный код изображения на холсте (data). Для формирования такой строки применяется метод toDataURL объекта элемента <canvas>. Далее объект сериализуется с помощью функции JSON.stringify и помещается (функция setItem) в локальное хранилище браузера.

Извлечение изображения элемента <canvas> из локального хранилища и отображение его на холсте осуществляется функцией restoreCanvas, принимающей такие же параметры, что и функция exe\_saveCanvas. Она извлекает сериализованное изображение из хранилища (метод getItem), десереализует (функция JSON.parse) и создает на его основе объект Image, а затем на очищенный (методом clearRect) холст помещает это изображение (метод drawImage).

**6.2.5. Удаление данных из локального хранилища**

Последний элемент <section> (рис. 6.1) демонстрирует удаление данных из локального браузерного хранилища . На рис. 6.15 представлено содержимое этого элемента, а на рис. 6.16 отображение этого фрагмента браузером.

\\User\sql2008\HTML5\6\16.png

Рис. 6.16. Отображение браузером фрагмента html-страницы, представленного на рис. 6.15

Представленный на рис. 6.15, 6.16 фрагмент, позволяет ввести значение ключа и нажатием клавиши «Удалить» удалить ассоциированные с ключом данные из локального хранилища.

При нажатии клавиши «Удалить» вызывается функция exe\_delete (рис. 6.17), принимающая единственный параметр – идентификатор элемента <input>, предназначенного для ввода ключа.

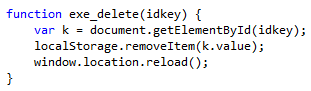


Рис. 6.17. Фрагмент файлы WebStorage.js:

функция exe\_delete

Собственно удаление данных осуществляется методом removeItem, принимающим единственный параметр – значение ключа, хранящихся данных.

**6.2.6. Работа с сессионным хранилищем данных**

Принцип работы с сессионным браузерным хранилищем данных ничем не отличается от работы с локальным хранилищем. Отличий только два: для записи, извлечения и удаления данных применяются методы sessionStorage.setIem, sessionStorage.getIem и sessionStorage.removeIem; данные хранятся только в рамках одной сессии, после того как окно браузера будет закрыто, данные удаляются.