20

Сохранение данных на стороне клиента

Веб-приложения могут использовать прикладные программные интерфейсы броузеров для сохранения данных локально, на компьютере пользователя. Этот механизм сохранения данных на стороне клиента выполняет роль памяти для веб-броузеров. Веб-приложения могут сохранять, например, настройки пользователя или даже полную информацию о своем состоянии, чтобы иметь возможность возобновить работу точно с того момента, на котором их работа была прервана при последнем посещении. Хранилище на стороне клиента разграничивает данные в соответствии с происхождением, поэтому страницы с одного сайта не смогут читать данные, сохраненные страницами с другого сайта. Но две страницы с одного и того же сайта смогут использовать хранилище в качестве механизма взаимодействий. Например, данные, введенные в форме на одной странице, можно отображать в таблице на другой странице. Веб-приложения могут устанавливать срок хранения своих данных: данные могут храниться временно, т. е. получить такие данные из хранилища можно, пока не будет закрыто окно или пока броузер не завершит работу, или сохраняться на жестком диске и храниться постоянно, чтобы их можно было получить месяцы или даже годы спустя.

Существует несколько разновидностей хранилищ на стороне клиента:

Web Storage

Web Storage – это прикладной программный интерфейс, определение которого первоначально было частью стандарта HTML5, но впоследствии было выделено в отдельную спецификацию. Эта спецификация все еще находится в стадии проекта, но частично (переносимым образом) реализована во всех текущих броузерах, включая IE8. Этот прикладной интерфейс содержит объекты `localStorage` и `sessionStorage`, которые, по сути, являются постоянно хранимыми ассоциативными массивами, отображающими ключи в строковые значения. Интерфейс Web Storage очень прост в использовании, он подходит для хранения больших (но не огромных) объемов данных и доступен во всех текущих броузерах, но не поддерживается старыми броузерами. Объекты `localStorage` и `sessionStorage` описываются в разделе 20.1.

Cookies

Cookies – старейший механизм хранения данных на стороне клиента, который предназначен для использования серверными сценариями. В языке JavaScript имеется довольно неудобный прикладной интерфейс, позволяющий управлять cookies на стороне клиента, но этот механизм сложен в использовании и подходит лишь для хранения небольших объемов текстовых данных. Кроме того, любые данные, хранящиеся в виде cookies, всегда передаются серверу с каждым HTTP-запросом, даже если эти данные представляют интерес только для клиента. Однако механизм cookies по-прежнему представляет определенный интерес для разработчиков клиентских сценариев, потому что он поддерживается всеми броузерами, старыми и новыми. Однако когда механизм Web Storage получит более широкое распространение, механизм cookies вернется к своей первоначальной роли в качестве механизма хранения данных для серверных сценариев на стороне клиента. Механизм cookies рассматривается в разделе 20.2.

IE User Data

Корпорация Microsoft реализовала в IE версии 5 и выше свой собственный механизм хранения данных на стороне клиента, известный как «userData». Механизм userData позволяет хранить достаточно большие объемы строковых данных и может использоваться как альтернатива механизму Web Storage в IE версии 7 и ниже. Прикладной интерфейс механизма userData рассматривается в разделе 20.3.

Offline Web Applications

Стандарт HTML5 определяет прикладной программный интерфейс «Offline Web Applications» (автономные веб-приложения), позволяющий кэшировать веб-страницы и связанные с ними ресурсы (сценарии, CSS-файлы, изображения и т. д.). Это хранилище предназначено для сохранения веб-приложений целиком, а не только их данных, и позволяет веб-приложениям устанавливать себя, давая возможность использовать их даже при отсутствии соединения с Интернетом. Автономные веб-приложения рассматриваются в разделе 20.4.

Базы данных для Веб

Разработчики, которым приходится работать с по-настоящему огромными объемами данных, предпочитают использовать базы данных, и многие производители начинают включать в свои броузеры функциональные возможности доступа к базам данных на стороне клиента. Броузеры Safari, Chrome и Opera включают прикладной интерфейс к базе данных SQL. Однако попытка стандартизации этого прикладного интерфейса потерпела неудачу, и весьма маловероятно, что он будет реализован в Firefox и IE. Существует альтернативный прикладной интерфейс доступа к базам данных, который был стандартизован под названием «Indexed Database API». Это прикладной интерфейс к простейшей объектной базе данных, не поддерживающей язык запросов. Оба прикладных интерфейса являются асинхронными и требуют использования обработчиков событий, что усложняет их использование. Они не будут описываться в этой главе, но в разделе 22.8 вы найдете краткий обзор и пример применения механизма IndexedDB API.

Прикладной интерфейс к файловой системе

В главе 18 было показано, что современные броузеры поддерживают объект `File`, позволяющий с помощью объекта `XMLHttpRequest` выгружать файлы, выбранные пользователем. Проекты родственных стандартов определяют прикладной интерфейс для организации частной, локальной файловой системы и выполнения операций чтения и записи в файлы, находящиеся в этой файловой системе. Эти появляющиеся прикладные интерфейсы описываются в разделе 22.7. Когда они получат более широкое распространение, у веб-приложений будет возможность использовать механизмы хранения данных, основанные на файлах, уже знакомые многим программистам.

Сохранность, безопасность и конфиденциальность

Веб-броузеры часто предлагают пользователям сохранить пароли и сохраняют их на диске в зашифрованном виде. Но ни один из механизмов хранения данных на стороне клиента, описываемый в этой главе, никак не связан с шифрованием: все, что вы будете сохранять, будет сохраняться на жестком диске в незашифрованном виде. То есть хранящиеся данные могут быть извлечены чересчур любопытным пользователем, имеющим доступ к компьютеру, или злонамеренным программным обеспечением (например, разнообразными шпионскими программами), находящимся на компьютере. По этой причине ни один из механизмов хранения данных на стороне клиента никогда не должен использоваться для хранения паролей, номеров банковских счетов или другой конфиденциальной информации. Запомните: тот факт, что пользователь вводит какую-то информацию в поля форм при взаимодействии с вашим веб-сайтом, еще не означает, что он хочет сохранить копию этой информации на диске. Возьмите в качестве примера номер кредитной карты. Это конфиденциальная информация, которую люди предпочитают сохранять в тайне в своих бумажниках. Сохранить эту информацию с помощью механизма хранения данных на стороне клиента – это все равно что написать номер кредитной карты на бумажке и приклеить ее к клавиатуре.

Кроме того, учтите, что многие пользователи не доверяют веб-сайтам, использующим cookies или другие механизмы хранения данных на стороне клиента для целей, которые напоминают «слежение». Применяйте механизмы хранения, описываемые в этой главе, для повышения удобства работы с вашим сайтом, но не используйте их как механизм сбора конфиденциальной информации. Если появится слишком большое количество сайтов, дискредитирующих механизмы хранения данных на стороне клиента, пользователи будут отключать их или часто очищать хранилища, что сделает невозможным и их использование, и работу сайтов, опирающихся на их применение.

20.1. Объекты localStorage и sessionStorage

Броузеры, реализующие положения проекта спецификации «Web Storage», определяют в объекте Window два свойства: localStorage и sessionStorage. Оба свойства ссылаются на объект Storage — постоянно хранимый ассоциативный массив, отображающий строковые ключи в строковые значения. Объекты Storage действуют подобно обычным объектам в языке JavaScript: достаточно просто присвоить свойству объекта строку, и броузер автоматически сохранит ее. Разница между localStorage и sessionStorage заключается лишь в сроке хранения и области видимости: они определяют, как долго будут храниться данные и кому они будут доступны.

Ниже мы подробнее поговорим о сроке хранения и области видимости. А пока рассмотрим несколько примеров. Следующий фрагмент использует свойство localStorage, но он точно так же мог бы работать и со свойством sessionStorage:

```
var name = localStorage.username;       // Получить сохраненное значение.
name = localStorage["username"];        // Эквивалентная форма обращения, как к массиву
if (!name) {
    name = prompt("Как вас зовут?");    // Задать пользователю вопрос.
    localStorage.username = name;       // Сохранить ответ.
}

// Выполнить итерации по всем хранящимся парам имя/значение
for(var name in localStorage) {         // Итерации по всем хранящимся именам
    var value = localStorage[name];     // Получить значение для каждого из них
}
```

Объекты Storage также определяют методы для сохранения, извлечения и удаления данных. Эти методы рассматриваются в разделе 20.1.2.

Проект спецификации «Web Storage» определяет возможность сохранения структурированных данных (объектов и массивов), а также простых значений и данных встроенных типов, таких как даты, регулярные выражения и даже объекты File. Однако на момент написания этих строк броузеры позволяли сохранять только строки. Если потребуется сохранять и извлекать данные других типов, их можно кодировать и декодировать вручную. Например:

```
// При сохранении числа оно автоматически преобразуется в строку.
// Не забудьте выполнить обратное преобразование при извлечении из хранилища.
localStorage.x = 10;
var x = parseInt(localStorage.x);

// Преобразовать объект Date в строку при записи и обратно - при чтении
localStorage.lastRead = (new Date()).toUTCString();
var lastRead = new Date(Date.parse(localStorage.lastRead));

// Для кодирования любых простых или структурированных данных удобно
// использовать формат JSON
localStorage.data = JSON.stringify(data);    // Закодировать и сохранить
var data = JSON.parse(localStorage.data);    // Извлечь и декодировать.
```

20.1.1. Срок хранения и область видимости

Объекты localStorage и sessionStorage отличаются сроком хранения данных и областью видимости хранилища. Объект localStorage представляет долговременное

хранилище данных: срок хранения не ограничен, и данные сохраняются на компьютере пользователя, пока не будут удалены веб-приложением или пока пользователь не потребует от броузера (посредством некоторого пользовательского интерфейса, предоставляемого броузером) удалить их.

Доступность данных в объекте localStorage ограничивается происхождением документа. Как описывалось в разделе 13.6.2, происхождение документа определяется такими параметрами, как протокол, имя хоста и номер порта, поэтому все следующие URL-адреса ссылаются на документы с разным происхождением:

```
http://www.example.com      // Протокол: http; имя хоста: www.example.com
https://www.example.com     // Другой протокол
http://static.example.com   // Другое имя хоста
http://www.example.com:8000 // Другой порт
```

Все документы, имеющие одно и то же происхождение, будут совместно использовать одни и те же данные в объекте localStorage (независимо от происхождения сценария, который фактически обращается к хранилищу localStorage). Они смогут читать и изменять данные друг друга. Но документы с разными происхождениями никогда не смогут прочитать или изменить данные друг друга (даже если оба они будут выполнять сценарий, полученный с одного и того же стороннего сервера).

Обратите внимание, что видимость данных в хранилище localStorage также ограничивается конкретным броузером. Если посетить сайт с помощью Firefox, а затем вновь посетить его, например, с помощью Chrome, никакие данные, сохраненные при первом посещении, не будут доступны при втором посещении.

Данные, сохраняемые в sessionStorage, имеют другой срок хранения: они хранятся, пока остается открытым окно верхнего уровня или вкладка броузера, в которой выполнялся сценарий, сохранивший эти данные. При закрытии окна или вкладки все данные, хранящиеся в sessionStorage, удаляются. (Отметьте, однако, что современные броузеры имеют возможность повторно открывать недавно закрытые вкладки и восстанавливать последний сеанс работы с броузером, поэтому срок хранения информации об этих вкладках и связанных с ними хранилищах sessionStorage может оказаться больше, чем кажется.)

Доступность данных в хранилище sessionStorage, как и в хранилище localStorage, ограничивается происхождением документа, т. е. документы с разным происхождением никогда не смогут совместно использовать одни и те же данные в sessionStorage. Но помимо этого доступность данных в хранилище sessionStorage ограничивается также окном. Если пользователь откроет в броузере две вкладки, отображающие документы с общим происхождением, эти две вкладки будут владеть разными хранилищами sessionStorage. Сценарий, выполняющийся в одной вкладке, не сможет прочитать или изменить данные, сохраненные в другой вкладке, даже если в обеих вкладках будет открыта одна и та же страница и будет выполняться один и тот же сценарий. Обратите внимание, что разграничение хранилищ sessionStorage в разных окнах касается только окон верхнего уровня. Если в одной вкладке броузера будет находиться несколько элементов <iframe> и в этих фреймах будут отображаться документы с общим происхождением, они будут совместно использовать одно и то же хранилище sessionStorage.

20.1.2. Прикладной программный интерфейс объекта Storage

Объекты localStorage и sessionStorage часто используются как обычные объекты языка JavaScript: присваивание значения свойству приводит к сохранению строки, а чтение свойства – к ее извлечению из хранилища. Но эти объекты определяют также более формальный прикладной интерфейс, основанный на методах. Сохранить значение можно с помощью метода setItem(), передав ему имя и значение. Извлечь значение можно с помощью метода getItem(), передав ему имя. Удалить значение можно с помощью метода removeItem(), передав ему имя. (В большинстве броузеров удалить значение можно также с помощью оператора delete, как если бы оно было обычным объектом, но этот прием не работает в IE8.) Удалить все хранящиеся значения можно вызовом метода clear() (без аргументов). Наконец, перечислить имена всех хранящихся значений можно с помощью свойства length и метода key(), передавая ему значения от 0 до length-1. Ниже приводится несколько примеров использования объекта localStorage. Этот программный код с тем же успехом мог бы использовать объект sessionStorage:

```
localStorage.setItem("x", 1); // Сохранить число под именем "x"
localStorage.getItem("x");    // Извлечь значение

// Перечислить все хранящиеся пары имя-значение
for(var i = 0; i < localStorage.length; i++) { // length дает количество пар
    var name = localStorage.key(i);            // Получить имя i-й пары
    var value = localStorage.getItem(name);    // Получить значение этой пары
}

localStorage.removeItem("x"); // Удалить элемент "x"
localStorage.clear();         // Удалить все остальные элементы
```

Несмотря на то что обычно удобнее сохранять и извлекать данные, обращаясь к свойствам, тем не менее иногда может потребоваться использовать эти методы. Во-первых, метод clear() не имеет эквивалента и является единственным способом удаления всех пар имя/значение в объекте Storage. Аналогично метод removeItem() является единственным переносимым способом удаления одной пары имя/значение, потому что IE8 не позволяет использовать оператор delete для этой цели.

Если производители броузеров реализуют все положения спецификации и позволят сохранять в объекте Storage другие объекты и массивы, появится еще одна причина использовать методы, такие как setItem() и getItem(). Объекты и массивы являются изменяемыми значениями, поэтому объекту Storage необходимо будет создавать копию сохраняемого значения, чтобы все последующие изменения в оригинале не коснулись хранящегося значения. Точно так же объект Storage должен будет создавать копию и при извлечении значения, чтобы никакие изменения в извлеченном значении не коснулись хранящегося значения. Когда такое копирование будет реализовано, использование интерфейса свойств может показаться запутывающим. Взгляните на следующий (гипотетический, пока броузеры не поддерживают возможность сохранения структурированных значений) фрагмент:

```
localStorage.o = {x:1}; // Сохранить объект, имеющий свойство x
localStorage.o.x = 2;   // Попытка установить свойство хранящегося объекта
localStorage.o.x        // => 1: свойство x не изменилось
```

Во второй строке в примере выше выполняется безуспешная попытка присвоить новое значение свойству хранящегося объекта, потому что интерпретатор сначала извлечет копию хранящегося объекта, присвоит свойству копии новое значение и затем удалит копию. В результате изменения не коснутся хранящегося объекта. Меньше шансов ошибиться, если использовать метод `getItem()`:

```
localStorage.getItem("o").x = 2; // Не предполагает сохранение значения 2
```

Наконец, еще одна причина отдать предпочтение более явному прикладному интерфейсу на основе методов заключается в возможности имитировать этот прикладной интерфейс поверх других механизмов сохранения данных в броузерах, которые пока не поддерживают спецификацию «Web Storage». В следующих разделах будут представлены примеры реализации интерфейса объекта `Storage` с применением cookies и userData в IE. При использовании прикладного интерфейса на основе методов можно писать программный код, который будет использовать свойство `localStorage`, когда оно доступно, и возвращаться к использованию других механизмов в противном случае. Такой программный код мог бы начинаться следующими строками:

```
// Определить, какой механизм хранения будет использоваться
var memory = window.localStorage ||
             (window.UserDataStorage && new UserDataStorage()) ||
             new CookieStorage();
// Затем отыскать требуемый элемент в хранилище
var username = memory.getItem("username");
```

20.1.3. События объекта Storage

При изменении данных, хранящихся в `localStorage` или `sessionStorage`, броузер генерирует событие «storage» во всех объектах `Window`, в которых доступны эти данные (но не в окне, где выполнялось сохранение). Если в броузере открыты две вкладки со страницами с общим происхождением и в одной из страниц производится сохранение значения в `localStorage`, в другой вкладке будет сгенерировано событие «storage». Не забывайте, что область видимости данных, хранящихся в `sessionStorage`, ограничивается окном верхнего уровня, поэтому при изменении данных в `sessionStorage` события «storage» будут генерироваться только при наличии нескольких фреймов. Обратите также внимание, что события «storage» генерируются, только когда содержимое хранилища действительно изменяется. Присваивание хранимому элементу его текущего значения, как и попытка удалить несуществующий элемент, не возбуждают событие.

Регистрация обработчиков события «storage» выполняется с помощью метода `addEventListener()` (или `attachEvent()` в IE). В большинстве броузеров для этой цели можно также использовать свойство `onstorage` объекта `Window`, но на момент написания этих строк данное свойство не поддерживалось в Firefox.

Объект события, связанный с событием «storage», имеет пять основных свойств (к сожалению, они не поддерживаются в IE8):

`key`

Имя или ключ сохраняемого или удаляемого элемента. Если был вызван метод `clear()`, это свойство будет иметь значение `null`.

`newValue`

Новое значение элемента или `null`, если был вызван метод `removeItem()`.

`oldValue`

Старое значение существующего элемента, изменившегося или удаленного, или значение `null`, если был создан новый элемент.

`storageArea`

Это свойство будет хранить значение свойства `localStorage` или `sessionStorage` целевого объекта `Window`.

`url`

URL-адрес (в виде строки) документа, сценарий которого выполнил операцию с хранилищем.

Наконец, обратите внимание, что объект `localStorage` и событие «storage» могут служить широковещательным механизмом, с помощью которого броузер может отправлять сообщения всем окнам, в которых в настоящий момент открыт один и тот же веб-сайт. Например, если пользователь потребует от веб-сайта прекратить воспроизводить анимационные эффекты, сценарий может сохранить соответствующий параметр настройки в `localStorage`, чтобы соблюсти это требование при последующих посещениях сайта. При сохранении параметра будет сгенерировано событие, что позволит другим окнам, отображающим тот же сайт, также удовлетворить это требование. В качестве другого примера представьте веб-приложение графического редактора, позволяющее пользователю отображать палитры с инструментами в отдельных окнах. При выборе пользователем некоторого инструмента приложение могло бы сохранять в `localStorage` признак выбранного инструмента и тем самым рассылать другим окнам извещения о том, что был выбран новый инструмент.

20.2. Cookies

Cookies – это небольшие фрагменты именованных данных, сохраняемые веб-броузером и связанные с определенными веб-страницами или веб-сайтами. Cookies первоначально предназначались для разработки серверных сценариев и на низшем уровне реализованы как расширение протокола HTTP. Данные в cookies автоматически передаются между веб-броузером и веб-сервером, благодаря чему серверные сценарии могут читать и записывать значения, сохраняемые на стороне клиента. В этом разделе будет показано, как клиентские сценарии могут работать с cookies, используя свойство `cookie` объекта `Document`.

Прикладной интерфейс для работы с cookies является одним из старейших, а это означает, что он поддерживается всеми броузерами. К сожалению, этот прикладной интерфейс слишком замысловат. В нем отсутствуют методы: операции чтения, записи и удаления cookies осуществляются с помощью свойства `cookie` объекта `Document` с применением строк специального формата. Срок хранения и область видимости можно указать отдельно для каждого cookie с помощью атрибутов. Эти атрибуты также определяются посредством записи строк специального формата в то же самое свойство `cookie`.

Почему «cookie»?

Особого смысла у термина «cookie» (булочка) нет, тем не менее появился он не «с потолка». В туманных анналах истории компьютеров термин «cookie», или «magic cookie», использовался для обозначения небольшой порции данных, в частности, привилегированных или секретных данных, вроде пароля, подтверждающих подлинность или разрешающих доступ. В JavaScript cookies применяются для сохранения информации о состоянии и могут служить средством идентификации для веб-броузера, хотя они не шифруются и не могут считаться безопасными (впрочем, это не относится к передаче их через защищенное соединение по протоколу `https:`).

В следующих подразделах сначала будут описаны атрибуты, которые определяют срок хранения и область видимости cookies, а затем будет продемонстрировано, как сохранять и извлекать значения cookies в сценариях на языке JavaScript. Завершится этот раздел примером реализации на основе cookies прикладного интерфейса, имитирующего объект `Storage`.

Как определить, когда поддержка cookies включена

Cookies пользуются дурной славой у многих пользователей Всемирной паутины из-за недобросовестного использования cookies, связанных не с самой веб-страницей, а с изображениями на ней. Например, сторонние cookies позволяют компаниям, предоставляющим услуги рекламного характера, отслеживать перемещение пользователей с одного сайта на другой, что вынуждает многих пользователей по соображениям безопасности отключать режим сохранения cookies в своих веб-броузерах. Поэтому, прежде чем использовать cookies в сценариях JavaScript, следует проверить, не отключен ли режим их сохранения. В большинстве броузеров это можно сделать, проверив свойство `navigator.cookieEnabled`. Если оно содержит значение `true`, значит, работа с cookies разрешена, а если `false` – запрещена (хотя при этом может быть разрешено использование временных cookies, срок хранения которых ограничивается продолжительностью сеанса работы броузера). Свойство `navigator.cookieEnabled` не является стандартным, поэтому если сценарий вдруг обнаружит, что оно не определено, придется проверить, поддерживаются ли cookies, попытавшись записать, прочитать и удалить тестовый cookie, используя прием, описываемый ниже.

20.2.1. Атрибуты cookie: срок хранения и область видимости

Помимо имени и значения каждый cookie имеет необязательные атрибуты, управляющие сроком его хранения и областью видимости. По умолчанию cookies

являются временными – их значения сохраняются на период сеанса веб-броузера и теряются при закрытии броузера. Обратите внимание, что этот срок хранения не совпадает со сроком хранения данных в `sessionStorage`: доступность cookies не ограничивается единственным окном, поэтому период их хранения по умолчанию совпадает с периодом работы процесса броузера, а не какого-то одного окна. Чтобы cookie сохранялся после окончания сеанса, необходимо сообщить броузеру, как долго (в секундах) он должен храниться, указав значение атрибута *max-age*. Если указать срок хранения, броузер сохранит cookie в локальном файле и удалит его только по истечении срока хранения.

Видимость cookie ограничивается происхождением документа, как и при использовании хранилищ `localStorage` и `sessionStorage`, а также строкой пути к документу. Область видимости cookie может регулироваться посредством атрибутов *path* и *domain*. По умолчанию cookie связывается с создавшей его веб-страницей и доступен этой странице, а также другим страницам из того же каталога или любых его подкаталогов. Если, например, веб-страница *http://www.example.com/catalog/index.html* создаст cookie, то этот cookie будет также видим страницам *http://www.example.com/catalog/order.html* и *http://www.example.com/catalog/widgets/index.html*, но невидим странице *http://www.example.com/about.html*.

Этого правила видимости, принятого по умолчанию, обычно вполне достаточно. Тем не менее иногда значения cookie требуется использовать на всем многостраничном веб-сайте независимо от того, какая страница создала cookie. Например, если пользователь ввел свой адрес в форму на одной странице, целесообразно было бы сохранить этот адрес как адрес по умолчанию. Тогда этим адресом можно будет воспользоваться при следующем посещении тем же пользователем этой страницы, а также при заполнении им совершенно другой формы на любой другой странице, где требуется ввести адрес, например для выставления счета. Для этого в cookie можно определить атрибут *path*. И тогда любая страница того же веб-сервера с URL-адресом, начинающимся с указанного значения, сможет использовать этот cookie. Например, если для cookie, установленного страницей *http://www.example.com/catalog/widgets/index.html*, в атрибуте *path* установлено значение «/catalog», этот cookie также будет виден для страницы *http://www.example.com/catalog/order.html*. А если атрибут *path* установлен в значение «/», то cookie будет видим для любой страницы на веб-сервере *http://www.example.com*.

Установка атрибута *path* в значение «/» определяет такую же область видимости cookie, как для хранилища `localStorage`, а также говорит о том, что броузер должен передавать имя и значение cookie на сервер при запросе любой веб-страницы с этого сайта. Имейте в виду, что атрибут *path* не должен восприниматься как своеобразный механизм управления доступом. Если веб-странице потребуется прочитать cookies, принадлежащие какой-то другой странице на том же веб-сайте, она может просто загрузить эту страницу в скрытый элемент `<iframe>` и прочитать все cookies, принадлежащие документу во фрейме. Политика общего происхождения (раздел 13.6.2) предотвращает подобное «подглядывание» за cookies, установленных веб-страницами с других сайтов, но оно считается вполне допустимым для документов с одного и того же сайта.

По умолчанию cookies доступны только страницам с общим происхождением. Однако большим веб-сайтам может потребоваться возможность совместного использования cookies несколькими поддоменами. Например, серверу *order.example.com* может потребоваться прочитать значения cookie, установленного сервером

catalog.example.com. В этой ситуации поможет атрибут *domain*. Если cookie, созданный страницей с сервера *catalog.example.com*, имеет в атрибуте *path* значение «/», а в атрибуте *domain* – значение «.example.com», этот cookie будет доступен всем веб-страницам в поддоменах *catalog.example.com*, *orders.example.com* и в любых других поддоменах в домене *example.com*. Если атрибут *domain* не установлен, его значением по умолчанию будет имя веб-сервера, на котором находится страница. Обратите внимание, что в атрибут *domain* нельзя записать значение, отличающееся от домена вашего сервера.

Последний атрибут cookie – это логический атрибут с именем *secure*, определяющий, как значения cookie передаются по сети. По умолчанию cookie не защищен, т. е. передается по обычному незащищенному HTTP-соединению. Однако если cookie помечен как защищенный, он передается, только когда обмен между броузером и сервером организован по протоколу HTTPS или другому защищенному протоколу.

20.2.2. Сохранение cookies

Чтобы связать временное значение cookie с текущим документом, достаточно присвоить его свойству `cookie` строку следующего формата:

```
имя=значение
```

Например:

```
document.cookie = "version=" + encodeURIComponent(document.lastModified);
```

При следующем чтении свойства `cookie` сохраненная пара имя/значение будет включена в список cookies документа. Значения cookie не могут содержать точки с запятой, запятые или пробельные символы. По этой причине для кодирования значения перед сохранением его в cookie, возможно, потребуется использовать глобальную JavaScript-функцию `encodeURIComponent()`. В этом случае при чтении значения cookie надо будет вызвать соответствующую функцию `decodeURIComponent()`.

Записанный таким способом cookie сохраняется в течение сеанса работы веб-броузера, но теряется при его закрытии пользователем. Чтобы создать cookie, сохраняющийся между сеансами броузера, необходимо указать срок его хранения (в секундах) с помощью атрибута `max-age`. Это можно сделать, присвоив свойству `cookie` строку следующего формата:

```
имя=значение; max-age=число_секунд
```

Следующая функция устанавливает cookie с дополнительным атрибутом `max-age`:

```
// Сохраняет пару имя/значение в виде cookie, кодируя значение с помощью
// encodeURIComponent(), чтобы экранировать точки с запятой, запятые и пробелы.
// Если в параметре daysToLive передается число, атрибут max-age
// устанавливается так, что срок хранения cookie истекает через
// указанное число дней. Если передать значение 0, cookie будет удален.
function setCookie(name, value, daysToLive) {
    var cookie = name + "=" + encodeURIComponent(value);
    if (typeof daysToLive === "number")
        cookie += "; max-age=" + (daysToLive*60*60*24);
    document.cookie = cookie;
}
```

Аналогичным образом можно установить атрибуты path, domain и secure, дописав к значению cookie строки следующего формата перед его записью в свойство cookie:

```
; path=путь
; domain=домен
; secure
```

Чтобы изменить значение cookie, установите его значение снова, указав то же имя, путь, домен и новое значение. При изменении значения cookie можно также переопределить срок его хранения, указав новое значение в атрибуте max-age.

Чтобы удалить cookie, установите его снова, указав то же имя, путь, домен и любое произвольное (возможно пустое) значение, а в атрибут max-age запишите 0.

20.2.3. Чтение cookies

Когда свойство cookie используется в JavaScript-выражении, возвращаемое им значение содержит все cookies, относящиеся к текущему документу. Эта строка представляет собой список пар *имя = значение*, разделенных точками с запятой и пробелами. Значение не включает какие-либо атрибуты, которые могли быть установлены для cookie. При работе со свойством document.cookie обычно приходится использовать метод split(), чтобы разбить его значение на отдельные пары имя/значение.

После извлечения значения cookie из свойства cookie его требуется интерпретировать, основываясь на том формате или кодировке, которые были указаны создателем cookie. Например, cookie можно передать функции decodeURIComponent(), а затем функции JSON.parse().

В примере 20.1 определяется функция getCookie(), которая анализирует свойство document.cookie и возвращает объект, свойства которого соответствуют парам имя/значение всех cookies в документе.

Пример 20.1. Анализ свойства document.cookie

```
// Возвращает cookies документа в виде объекта с парами имя/значение.
// Предполагается, что значения cookie кодируются с помощью
// функции encodeURIComponent().
function getCookies() {
    var cookies = {};           // Возвращаемый объект
    var all = document.cookie;  // Получить все cookies в одной строке
    if (all === "")             // Если получена пустая строка,
        return cookies;         // вернуть пустой объект
    var list = all.split("; "); // Разбить на пары имя/значение
    for(var i = 0; i < list.length; i++) { // Для каждого cookie
        var cookie = list[i];
        var p = cookie.indexOf("=");       // Отыскать первый знак =
        var name = cookie.substring(0,p);  // Получить имя cookie
        var value = cookie.substring(p+1); // Получить значение cookie
        value = decodeURIComponent(value); // Декодировать значение
        cookies[name] = value;             // Сохранить имя и значение в объекте
    }
    return cookies;
}
```

20.2.4. Ограничения cookies

Cookies предназначены для сохранения небольших объемов данных серверными сценариями, которые должны передаваться на сервер при обращении к каждому соответствующему URL-адресу. Стандарт, определяющий cookies, рекомендует производителям броузеров не ограничивать количество и размеры сохраняемых cookies, но броузеры не обязаны сохранять в сумме более 300 cookies, 20 cookies на один веб-сервер или по 4 Кбайт данных на один cookie (в этом ограничении учитываются и значение cookie, и его имя). На практике броузеры позволяют сохранять гораздо больше 300 cookies, но ограничение на размер 4 Кбайт для одного cookie в некоторых броузерах по-прежнему соблюдается.

20.2.5. Реализация хранилища на основе cookies

Пример 20.2 демонстрирует, как поверх cookies можно реализовать методы, имитирующие прикладной интерфейс объекта `Storage`. Передайте конструктору `CookieStorage()` желаемые значения атрибутов *max-age* и *path* и используйте получившийся объект подобно тому, как вы использовали бы `localStorage` или `sessionStorage`. Однако имейте в виду, что этот пример не реализует событие «storage» и не выполняет автоматическое сохранение или извлечение значений при записи или чтении свойств объекта `CookieStorage`.

Пример 20.2. Реализация интерфейса объекта Storage на основе cookies

```
/*
 * CookieStorage.js
 * Этот класс реализует прикладной интерфейс объекта Storage, на который ссылаются
 * свойства localStorage и sessionStorage, но поверх HTTP Cookies.
 */
function CookieStorage(maxage, path) { // Аргументы определяют срок хранения
                                       // и область видимости

    // Получить объект, хранящий все cookies
    var cookies = (function() {        // Функция getCookies(), реализованная выше
        var cookies = {};              // Возвращаемый объект
        var all = document.cookie;     // Получить все cookies в одной строке
        if (all === "")                // Если получена пустая строка
            return cookies;            // вернуть пустой объект
        var list = all.split("; ");    // Разбить на пары имя/значение
        for(var i = 0; i < list.length; i++) { // Для каждого cookie
            var cookie = list[i];
            var p = cookie.indexOf("=");       // Отыскать первый знак =
            var name = cookie.substring(0,p);  // Получить имя cookie
            var value = cookie.substring(p+1); // Получить значение cookie
            value = decodeURIComponent(value); // Декодировать значение
            cookies[name] = value;             // Сохранить имя и значение
        }
        return cookies;
    }());

    // Собрать имена cookies в массиве
    var keys = [];
    for(var key in cookies) keys.push(key);
```

```
// Определить общедоступные свойства и методы Storage API

// Количество хранящихся cookies
this.length = keys.length;

// Возвращает имя n-го cookie или null, если n вышло за диапазон индексов
this.key = function(n) {
    if (n < 0 || n >= keys.length) return null;
    return keys[n];
};

// Возвращает значение указанного cookie или null.
this.getItem = function(name) { return cookies[name] || null; };

// Сохраняет значение
this.setItem = function(key, value) {
    if (!(key in cookies)) { // Если cookie с таким именем не существует
        keys.push(key);      // Добавить ключ в массив ключей
        this.length++;       // И увеличить значение length
    }

    // Сохранить пару имя/значение в множестве cookies.
    cookies[key] = value;

    // Установить cookie.
    // Предварительно декодировать значение и создать строку
    // имя=кодированное-значение
    var cookie = key + "=" + encodeURIComponent(value);

    // Добавить в строку атрибуты cookie
    if (maxage) cookie += "; max-age=" + maxage;
    if (path) cookie += "; path=" + path;

    // Установить cookie с помощью свойства document.cookie
    document.cookie = cookie;
};

// Удаляет указанный cookie
this.removeItem = function(key) {
    if (!(key in cookies)) return; // Если не существует, ничего не делать

    // Удалить cookie из внутреннего множества cookies
    delete cookies[key];

    // И удалить ключ из массива имен.
    // Это легко можно было бы сделать с помощью метода indexOf() массивов,
    // определяемого стандартом ES5.
    for(var i = 0; i < keys.length; i++) { // Цикл по всем ключам
        if (keys[i] === key) {             // При обнаружении ключа
            keys.splice(i,1);              // Удалить его из массива.
            break;
        }
    }
    this.length--;                         // Уменьшить значение length

    // Наконец фактически удалить cookie, присвоив ему пустое значение
    // и истекший срок хранения.
    document.cookie = key + "=; max-age=0";
};
```

```
    // Удаляет все cookies
    this.clear = function() {
        // Обойти все ключи и удалить cookies
        for(var i = 0; i < keys.length; i++)
            document.cookie = keys[i] + "=; max-age=0";
        // Установить внутренние переменные в начальное состояние
        cookies = {};
        keys = [];
        this.length = 0;
    };
}
```

20.3. Механизм сохранения userData в IE

В IE версии 5 и ниже поддерживается механизм сохранения данных на стороне клиента, доступный в виде нестандартного свойства `behavior` элемента документа. Использовать его можно следующим образом:

```
var memory = document.createElement("div");          // Создать элемент
memory.id = "_memory";                               // Дать ему имя
memory.style.display = "none";                       // Не отображать его
memory.style.behavior = "url('#default#userData')"; // Присоединить свойство
document.body.appendChild(memory);                   // Добавить в документ
```

После того как для элемента будет определено поведение «userData», он получает новые методы `load()` и `save()`. Вызов метода `load()` загружает сохраненные данные. Этому методу можно передать строку, напоминающую имя файла и идентифицирующую конкретный пакет хранящихся данных. После загрузки данных пары имя/значение становятся доступны в виде атрибутов элемента, получить которые можно с помощью метода `getAttribute()`. Чтобы сохранить новые данные, необходимо установить атрибуты вызовом метода `setAttribute()` и затем вызвать метод `save()`. Удалить значение можно с помощью методов `removeAttribute()` и `save()`. Ниже приводится пример использования элемента `memory`, инициализированного выше:

```
memory.load("myStoredData");                 // Загрузить сохраненные данные
var name = memory.getAttribute("username"); // Получить элемент данных
if (!name) {                                 // Если он не был определен,
    name = prompt("Как вас зовут?");         // запросить у пользователя
    memory.setAtttribute("username", name); // Установить как атрибут
    memory.save("myStoredData");             // И сохранить до следующего раза
}
```

По умолчанию данные, сохраняемые с помощью механизма userData, имеют неограниченный срок хранения и сохраняются до тех пор, пока явно не будут удалены. Однако с помощью свойства `expires` можно определить дату, когда истечет срок хранения данных. Например, в предыдущий фрагмент можно было бы добавить следующие строки, чтобы указать, что срок хранения данных истечет через 100 дней:

```
var now = (new Date()).getTime();              // Текущее время в миллисекундах
var expires = now + 100 * 24 * 60 * 60 * 1000; // + 100 дней в миллисекундах
expires = new Date(expires).toUTCString();     // Преобразовать в строку
memory.expires = expires;                      // Установить срок хранения
```

Данные, сохраняемые с помощью механизма userData в IE, доступны только для документов, хранящихся в том же каталоге, что и оригинальный документ, сохранивший их. Это более ограниченная область видимости, чем у cookies, которые также доступны документам в подкаталогах оригинального каталога. В механизме userData отсутствует какой-либо эквивалент атрибутов *path* и *domain* в cookies, позволяющий расширить область видимости данных.

Механизм userData позволяет сохранять гораздо большие объемы данных, чем cookies, но меньшие, чем объекты `localStorage` и `sessionStorage`.

Пример 20.3 реализует методы `getItem()`, `setItem()` и `removeItem()` интерфейса `Storage` поверх механизма userData в IE. (Он не реализует такие методы, как `key()` и `clear()`, потому что механизм userData не предоставляет возможность выполнять итерации по всем хранящимся элементам.)

Пример 20.3. Частичная реализация интерфейса Storage на основе механизма userData в IE

```
function UserDataStorage(maxage) {
    // Создать элемент документа и установить в нем специальное
    // свойство behavior механизма userData, чтобы получить доступ
    // к методам save() и load().
    var memory = document.createElement("div");        // Создать элемент
    memory.style.display = "none";                     // Не отображать его
    memory.style.behavior = "url('#default#userData')"; // Присоединить свойство behavior
    document.body.appendChild(memory);                 // Добавить в документ

    // Если указано значение параметра maxage, хранить данные maxage секунд
    if (maxage) {
        var now = new Date().getTime();    // Текущее время
        var expires = now + maxage * 1000; // maxage секунд от текущего времени
        memory.expires = new Date(expires).toUTCString();
    }

    // Инициализировать хранилище, загрузив сохраненные значения.
    // Значение аргумента выбирается произвольно, но оно должно совпадать
    // со значением, переданным методу save()
    memory.load("UserDataStorage");     // Загрузить сохраненные данные

    this.getItem = function(key) {      // Загрузить значения атрибутов
        return memory.getAttribute(key) || null;
    };
    this.setItem = function(key, value) {
        memory.setAttribute(key,value); // Сохранить значения как атрибуты
        memory.save("UserDataStorage"); // Сохранять после любых изменений
    };

    this.removeItem = function(key) {
        memory.removeAttribute(key);    // Удалить сохраненные значения
        memory.save("UserDataStorage"); // Сохранить новое состояние
    };
}
```

Поскольку программный код из примера 20.3 будет работать только в IE, можно воспользоваться условными комментариями IE, чтобы предотвратить его загрузку в броузерах, отличных от IE:

```
<!--[if IE]>
<script src="UserDataStorage.js"></script>
<![endif]-->
```

20.4. Хранилище приложений и автономные веб–приложения

Стандарт HTML5 определяет новую особенность «кэш приложений» (application cache), которая может использоваться веб-приложениями для сохранения самих себя локально в броузере пользователя. Объекты `localStorage` и `sessionStorage` позволяют сохранять данные веб-приложений, тогда как кэш приложений позволяет сохранять сами приложения – все файлы (HTML, CSS, JavaScript, изображения и т. д.), необходимые для работы приложения. Кэш приложений отличается от обычного кэша веб-броузера: он не очищается, когда пользователь очищает обычный кэш. И кэшированные приложения не очищаются по признаку LRU (least-recently used – давно не используемые), как это может происходить в обычном кэше фиксированного размера. Приложения сохраняются в кэше не временно: они устанавливаются и могут оставаться в нем, пока не удалят себя сами или не будут удалены пользователем. В действительности, кэш приложений вообще не является кэшем – для него больше подошло бы название «хранилище приложений» (application storage).

Основная причина необходимости установки веб-приложений локально заключается в обеспечении их доступности при работе в автономном режиме (например, в самолете или когда сотовый телефон находится вне доступа к сети). Веб-приложения, способные работать автономно, устанавливают себя в кэш приложений, используют `localStorage` для сохранения своих данных и реализуют механизм синхронизации для передачи сохраненных данных при подключении к сети. Пример автономного веб-приложения мы увидим в разделе 20.4.3, но сначала нам необходимо узнать, как приложение может установить себя в кэш приложений.

20.4.1. Объявление кэшируемого приложения

Чтобы установить приложение в кэш приложений, необходимо создать файл объявления: файл, перечисляющий URL всех ресурсов, необходимых приложению. Затем нужно просто добавить ссылку на файл объявления в основную HTML-страницу приложения, определив атрибут `manifest` в теге `<html>`:

```
<!DOCTYPE HTML>
<html manifest="myapp.appcache">
<head>...</head>
<body>...</body>
</html>
```

Файлы объявлений должны начинаться со строки «CACHE MANIFEST». В следующих строках должны перечисляться URL-адреса кэшируемых ресурсов. Относительные URL-адреса откладываются относительно URL-адреса файла объявления. Пустые строки игнорируются. Строки, начинающиеся с символа #, являются комментариями и также игнорируются. Перед комментариями могут быть пробелы, но они не могут следовать в строке за какими-либо непробельными символами. Ниже приводится пример простого файла объявления:

```
CACHE MANIFEST
# Строка выше определяет тип файла. Данная строка является комментарием

# Следующие строки определяют ресурсы, необходимые для работы приложения
myapp.html
myapp.js
myapp.css
images/background.png
```

Этот файл объявления служит признаком приложения, устанавливаемого в кэш. Если веб-приложение содержит более одной веб-страницы (более одного HTML-файла, которые могут быть открыты пользователем), в каждой из этих страниц должен быть определен атрибут `<html manifest=>`, ссылающийся на файл объявления. Факт наличия во всех страницах ссылок на один и тот же файл объявления недвусмысленно говорит о том, что все они должны кэшироваться вместе как части одного и того же веб-приложения. Если в приложении имеется всего несколько HTML-страниц, их обычно перечисляют непосредственно в файле объявления. Однако это совершенно необязательно: все файлы, ссылающиеся на файл объявления, будут считаться частью веб-приложения и вместе с ним будут установлены в кэш.

Простой файл объявления, подобный тому, что показан выше, должен перечислять *все* ресурсы, необходимые веб-приложению. После загрузки веб-приложения в первый раз и установки его в кэш при последующих обращениях к нему оно будет загружаться из кэша. Когда приложение загружается из кэша, все необходимые ему ресурсы должны быть перечислены в файле объявления. Ресурсы, которые не были перечислены, не загружаются. Эта политика имитирует работу в автономном режиме. Если простое кэшированное приложение сможет запускаться из кэша, оно точно так же сможет запускаться, когда броузер работает в автономном режиме. Более сложные веб-приложения в общем случае не могут кэшировать каждый необходимый им ресурс отдельно. Но они тем не менее могут использовать кэш приложений, если они имеют более сложные объявления.

MIME–тип объявления кэшируемого приложения

По соглашению файлам объявлений кэшируемых приложений даются имена с расширением *.appcache*. Однако это всего лишь соглашение, а для фактической идентификации типа файла веб-сервер *должен* отправлять файл объявления с MIME-типом «text/cache-manifest». Если при отправке файла объявления сервер установит в заголовке «Content-Type» любой другой MIME-тип, приложение не будет установлено в кэш. Вам может потребоваться специально настроить свой веб-сервер на использование нужного MIME-типа, например, создав в Apache файл *.htaccess* в каталоге веб-приложения.

20.4.1.1. Сложные объявления

При запуске приложения из кэша загружаются только ресурсы, перечисленные в файле объявления. В примере файла объявления, представленном выше, URL-

адреса ресурсов перечисляются по одному. В действительности, файлы объявлений имеют более сложный синтаксис, чем было показано в этом примере, и существует еще два способа перечисления ресурсов в файлах объявлений. Для идентификации типов записей в объявлении используются специальные строки-заголовки разделов. Простые записи, как те, что были показаны выше, помещаются в раздел «CACHE:», который является разделом по умолчанию. Два других раздела начинаются с заголовков «NETWORK:» и «FALLBACK:». (В файле объявления может быть любое количество разделов, и они могут следовать в любом порядке.)

Раздел «NETWORK:» определяет ресурсы, которые никогда не должны кэшироваться и всегда должны загружаться из сети. Здесь можно перечислить, например, URL-адреса серверных сценариев. URL-адреса в разделе «NETWORK:» в действительности являются префиксами URL-адресов. Все ресурсы, URL-адреса которых начинаются с этих префиксов, будут загружаться только из сети. Если броузер работает в автономном режиме, то попытки обратиться к таким ресурсам будут оканчиваться неудачей. В разделе «NETWORK:» допускается использовать шаблонный URL-адрес «*». В этом случае броузер будет пытаться загружать из сети все ресурсы, не упомянутые в объявлении. Это фактически отменяет правило, которое требует явно перечислять в файле объявления все ресурсы, необходимые кэшируемому приложению.

Записи в разделе «FALLBACK:» включают два URL-адреса в каждой строке. Ресурс, указанный во втором URL, загружается и сохраняется в кэше. Первый URL используется как префикс. Все URL-адреса, соответствующие этому префиксу, не кэшируются и при возможности загружаются из сети. Если попытка загрузить ресурс с таким URL-адресом терпит неудачу, вместо него будет использоваться кэшированный ресурс, определяемый вторым URL-адресом. Представьте веб-приложение, включающее несколько видеоруководств. Поскольку видеоролики имеют большой объем, они не подходят для сохранения в локальном кэше. Для работы в автономном режиме файл объявления мог бы предусматривать отображение вместо них текстовой справки.

Ниже приводится более сложный файл объявления кэшируемого приложения:

```
CACHE MANIFEST

CACHE:
myapp.html
myapp.css
myapp.js

FALLBACK:
videos/ offline_help.html

NETWORK:
cgi/
```

20.4.2. Обновление кэша

При запуске кэшированного веб-приложения все его файлы загружаются непосредственно из кэша. Если броузер подключен к сети, он также асинхронно проверит наличие изменений в файле объявления. Если он изменился, будут загружены и установлены в кэш приложения новый файл объявления и все файлы, на которые он ссылается. Обратите внимание, что броузер не проверяет наличие

изменений в кэшированных файлах – проверяется только файл объявления. Например, если вы изменили файл сценария на языке JavaScript и вам необходимо, чтобы ваше веб-приложение обновило свой кэш, вам следует обновить файл объявления. Поскольку список файлов, необходимых приложению, при этом не изменяется, проще всего добиться требуемого результата, изменив номер версии:

```
CACHE MANIFEST
# MyApp версия 1 (изменяйте этот номер, чтобы заставить броузеры повторно
# загрузить следующие файлы)
MyApp.html
MyApp.js
```

Аналогично, если потребуется, чтобы веб-приложение удалило себя из кэша приложений, следует удалить файл объявления на сервере, чтобы на запрос этого файла возвращался бы HTTP-ответ 404 «Not Found», и изменить HTML-файл или файлы, удалив из них ссылки на файл объявления.

Обратите внимание, что броузеры проверяют файл объявления и обновляют кэш асинхронно, после (или во время) загрузки копии приложения из кэша. Для простых веб-приложений это означает, что после обновления файла объявления пользователь должен дважды загрузить приложение, чтобы получить обновленную версию: в первый раз будет загружена старая версия из кэша, после чего произойдет обновление файлов в кэше, а во второй раз из кэша будет загружена новая версия.

В ходе обновления кэша броузер запускает множество событий, что дает возможность зарегистрировать их обработчики и извещать пользователя. Например:

```
applicationCache.onupdateready = function() {
    var reload = confirm("Доступна новая версия приложения, которая\n" +
                         "будет использована при следующем запуске.\n" +
                         "Хотите ли перезапустить ее сейчас?");
    if (reload) location.reload();
}
```

Обратите внимание, что этот обработчик событий регистрируется в объекте `ApplicationCache`, на который ссылается свойство `applicationCache` объекта `Window`. Броузеры, поддерживающие кэш приложений, определяют это свойство. Помимо события «updateready», показанного выше, существует еще семь различных событий, имеющих отношение к кэшу приложений. В примере 20.4 демонстрируются простые обработчики, которые выводят сообщения, информирующие пользователя о ходе обновления кэша и о его текущем состоянии.

Пример 20.4. Обработка событий кэша приложений

```
// Эту функцию используют все обработчики событий, реализованные ниже, и выводят
// с ее помощью сообщения, информирующие о состоянии кэша приложений.
// Поскольку все обработчики отображают сообщения таким способом, они
// возвращают false, чтобы отменить дальнейшее распространение события
// и предотвратить вывод сообщений самим броузером.
function status(msg) {
    // Вывести сообщение в элементе документа с id="statusline"
    document.getElementById("statusline").innerHTML = msg;
    console.log(msg);      // А также в консоли для отладки
}
```