Работа с файлами в JavaScript, Часть 1: Основы

 Категория: [JavaScript](http://xdan.ru/categories/javascript)

 Опубликовано: 26 июня 2012

 Просмотров: 62864

При всем бурном развитии web, и стандартов html в частности, работа  с файлами, практически никогда не менялась. К счастью, с приходом [HTML5](http://xdan.ru/ispolzuem-datepicker-iz-jquery-ui.html) и связанных с ним API, сейчас у нас гораздо больше возможностей для работы с файлами, чем когда-либо в предыдущих версиях браузеров(iOS до сих пор нет поддержки [File API](http://xdan.ru/working-with-files-in-javascript-part-2-filereader.html)).

Тип Файл - File

Тип File определен в спецификации File API[1] и является абстрактным представлением файла. Каждый экземпляр File имеет следующие свойства:  
 name – имя файла  
 size – размер файла в байтах  
 type –  [MIME тип](http://ru.wikipedia.org/wiki/MIME" \t "_blank) файла

Объект типа File дает важную информацию о файле, не предоставляя прямой доступ к содержимому файла. Он является лишь ссылкой на файл, и получение данных из этого файла является отдельным процессом в целом.

Получение ссылок на файлы

Разумеется, доступ к пользовательским файлам строго запрещен в Интернете, потому как очевидны проблемы с безопасностью личных данных. Вы не хотели бы, чтобы Вы загружали веб-страницу, а затем она сканировала Ваш жесткий диск и выясняла, что там есть полезного. Нужно разрешение от пользователя, чтобы получить доступ к файлам с его компьютера. Тем не менее для веб-страниц чтения файлов разрешено каждый раз, когда пользователь решат что-то загрузить.  
  
 Когда вы используете элемент <input type="file">, Вы даете веб странице (и серверу) разрешение на доступ к файлу.  Так, что первое, как вы можете получить объект File, это поле <input type="file">.

HTML5 определяет файловые ссылки для всех <input type="file"> управления. Эта коллекция FileList, которая представляет собой структуру в виде массива под названием FileList содержащую объекты типа File для каждого выбранного файла в поле <input type="file">  (помните, HTML5 позволяет выбрать несколько файлов в этом элементе управления). Так что в любой момент времени, Вы можете получить доступ к файлам пользователя, которые он  выбрал, с помощью кода вроде этого:

|  |  |
| --- | --- |
| 1 | <input type="file" id="your-files" multiple> |

и

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <script>  var control = document.getElementById("your-files");  control.addEventListener("change", function(event) {      // Когда происходит изменение элементов управления, значит появились новые файлы      var i = 0,          files = control.files,          len = files.length;        for (; i < len; i++) {          console.log("Filename: " + files[i].name);          console.log("Type: " + files[i].type);          console.log("Size: " + files[i].size + " bytes");      }    }, false);  </script> |

Этот сравнительно простой код ожидает событие изменения в контроле(<input type="file"> ). Когда событие происходит, это означает, что выбор файла изменился, и код перебирает все объекты типа File и выводит информацию из них. Имейте в виду, что свойство файлов всегда доступны из JavaScript, так что вам не придется ждать следующего изменения, чтобы попытаться сделать что-то другое с ними.

Drag and drop файлов

Доступ к файлам из формы по средствам контролов по-прежнему требует действий пользователей: нахождение и выбора  интересующего файла. К счастью, HTML5 Drag and Drop  предоставляет еще один способ для пользователей, чтобы предоставить доступ к своим файлам: путем простого перетаскивания файлов с рабочего стола в веб-браузер. Все, что вам нужно сделать, чтобы это реализовать отслеживать два события.

Для того, чтобы читать файлы, которые упали на элемент страницы, вы должны отслеживать события DragOver и Drop, и отменять действия по умолчанию, в обоих. Это говорит браузеру, что вы знаете, что делаете :) и отменяет стандартные действия в таких случаях. Например, когда Вы перетаскиваете на страницу файл изображения, стандартным действием в таком случае будет открытие этого файла в этой вкладке. Это действие нужно отменить.

|  |  |
| --- | --- |
| 1 | <div id="your-files"></div> |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <script>  var target = document.getElementById("your-files");  target.addEventListener("dragover", function(event) {      event.preventDefault(); // отменяем действие по умолчанию  }, false);  target.addEventListener("drop", function(event) {      // отменяем действие по умолчанию      event.preventDefault();      var i = 0,       files = event.dataTransfer.files,       len = files.length;       for (; i < len; i++) {            console.log("Filename: " + files[i].name);            console.log("Type: " + files[i].type);            console.log("Size: " + files[i].size + " bytes");       }  }, false);  </script> |

event.dataTransfer.files другой FileList объект, через который вы можете получить доступ, к информации о файлах. Код почти такой же, как и контролами формы и объекты типа File могут быть доступны таким же образом.

AJAX pагрузка файлов

Если у вас есть ссылка на файл, то вы сможете сделать очень удобную вещь: загрузить файл с помощью [Ajax](http://xdan.ru/primeri-raboti-s-ajax-na-jquery.html). Все это возможно благодаря объекту FormData, которая определен в XMLHttpRequest . Этот объект представляет собой HTML-форму и позволяет добавлять пары ключ-значение, которые будут переданы на сервер с помощью метода append():

|  |  |
| --- | --- |
| 1  2 | var form = new FormData();  form.append("name", "Николай"); |

Самое замечательное в объекте FormData, что вы можете добавить файл непосредственно к нему, фактически имитируя загрузку файла через HTML-форму. Все, что вам нужно сделать, это добавить в файл ссылки с определенным именем, и браузер сделает все остальное. Для примера:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | // Создаем форму с несколькими значениями  var form = new FormData();  form.append("name", "Николай");  form.append("photo", control.files[0]);    // отправляем через xhr  var xhr = new XMLHttpRequest();  xhr.onload = function() {      console.log("Отправка завершена");  };  xhr.open("post", "/entrypoint", true);  xhr.send(form); |

Как только объект FormData передается в send (), надлежащие к его содержимому HTTP заголовки  устанавливаются автоматически. Вам не нужно беспокоиться об установке правильной кодировки формы при использовании файлов, сервер будет работать с полученными файлами, так как если бы  была отправлена​​ обычная HTML форма , читая данные о присланном файле из $\_FILES['photo'] и текстовыt данных из $\_POST['name']. Это дает вам универсальность, чтобы написать код обработки на стороне сервера, который может легко работать как с традиционными HTML-формами так и с формами присланными через Ajax.

И все это работает на последней версии большинства браузеров, включая Internet Explorer 10. К сожалению Internet Explorer 9 этого пока не поддерживает.

Что дальше

Теперь вы знаете два способа доступа к информации о файле в браузере: через контрол формы и через нативный 'drag and drop'. Вероятно, появятся и другие способы доступа к файлам в будущем, но сейчас Вам нужно знать только эти два. Конечно, читать информацию о файлах, это только часть проблемы.Следующим шагом будет чтение  данных из этих файлов, об этом и напишу во второй части статьи.

Ссылки по теме:

[File API спецификация](http://dev.w3.org/2006/webapi/FileAPI/" \t "_blank)

[HTML5 Drag and Drop](http://www.whatwg.org/specs/web-apps/current-work/multipage/dnd.html" \l "dnd" \t "_blank)

[XMLHttpRequest Level 2](http://www.w3.org/TR/XMLHttpRequest/" \t "_blank)

Связанные статьи

[Работа с файлами в JavaScript, Часть 2: FileReader](http://xdan.ru/Working-with-files-in-JavaScript-Part-2-FileReader.html)  
[Работа с файлами в JavaScript, Часть 3:Событие прогресса и ошибки](http://xdan.ru/Working-with-files-in-JavaScript-Part-3-Progress-events-and-errors.html)  
[Работа с файлами в JavaScript, Часть 4:Объект URL](http://xdan.ru/working-with-files-in-javascript-part-4-object-urls.html)

Работа с файлами в JavaScript, Часть 2: FileReader

 Категория: [JavaScript](http://xdan.ru/categories/javascript)

 Опубликовано: 29 июня 2012

 Просмотров: 53765

В моем предыдущем посте, я затронул тему использования файлов в [JavaScript](http://xdan.ru/uluchshaem-juzabiliti-textarea-s-pomoshhju-ace-js.html), с особым акцентом на том, как получить доступ к объектам File. Эти объекты, содержащие блок метаданных, можно получить только тогда, когда пользователь либо собирается загрузить файл через контрол формы или перетаскивает его методом Drag&Drop на веб-страницу. Итак у Вас есть эти метаданные, следующим шагом является чтение данных из них.

Тип FileReader

FileReader  имеет одно назначение: чтение данных из файла и сохранение их в переменной JavaScript. API намеренно разработан так, чтобы быть похожим на XMLHttpRequest, так как оба, по сути являются методом загрузки данных из внешних (вне браузера) ресурсов.Чтение осуществляется асинхронно, чтобы не блокировать браузер.

Есть несколько форматов, в которые  FileReader может представлять данные из файла,  формат должен быть задан, когда файл открывается для чтения. Чтение осуществляется с помощью вызова одного из следующих методов:

readAsText()  – возвращает содержимое файла как plain text  
readAsBinaryString() – возвращает содержимое файла в виде строки закодированных двоичных данных (устарело – вместо него используйте readAsArrayBuffer() )  
readAsArrayBuffer() – возвращает содержимое файла как ArrayBuffer (хорошо для двоичных данных, например, изображения)  
readAsDataURL() – возвращает содержимое файла как data URL

Каждый из этих методов инициирует чтение файла и похож на метод send ()  объекта XHR,  инициирующий HTTP запрос. Таким образом, вы должны установить обработчик загрузки событие onload, прежде чем начать читать. Результат чтения всегда представлены как event.target.result. Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | var reader = new FileReader();  reader.onload = function(event) {      var contents = event.target.result;      console.log("Содержимое файла: " + contents);  };    reader.onerror = function(event) {      console.error("Файл не может быть прочитан! код " + event.target.error.code);  };    reader.readAsText(file); |

Этот пример просто читает содержимое файла и выводит его в виде обычного текста в консоль. Обработчик события onload  вызывается, когда файл успешно прочитан в то время, как OnError вызывается, если файл не был прочитан по каким-то причинам. Объект типа FileReader доступен внутри обработчика события через event.target. В случае успеха чтения данных, в поле result, будет содержимое файла, иначе информацию об ошибках.

Чтение в data URIs

Вы можете использовать тот же код для чтения в data URI. Data URI (иногда называемый data URLs) представляют собой интересный вариант, если вы хотите, например, вывести изображение только, что прочтенное с диска. Вы можете сделать это, используя следующий код:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | var reader = new FileReader();  reader.onload = function(event) {      var dataUri = event.target.result,          img     = document.createElement("img");        img.src = dataUri;      document.body.appendChild(img);  };    reader.onerror = function(event) {      console.error("Файл не может быть прочитан! код " + event.target.error.code);  };    reader.readAsDataURL(file); |

Этот код просто вставляет изображение, которое было прочитано с диска на страницу. Поскольку data URI содержит все изображения, оно может быть передано непосредственно в атрибут src тега  <img> и отображено на странице. Как альтернативу данному методу, Вы могли бы , загружать изображение и рисовать его на <canvas>:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | var reader = new FileReader();  reader.onload = function(event) {      var dataUri = event.target.result,          context = document.getElementById("mycanvas").getContext("2d"),          img     = new Image();        // ждать, пока изображение не будет полностью обработано      img.onload = function() {          context.drawImage(img, 100, 100);      };      img.src = dataUri;  };    reader.onerror = function(event) {      console.error("Файл не может быть прочитан! код " + event.target.error.code);  };    reader.readAsDataURL(file); |

Этот код загружает изображение в новый объект Image, а затем использует его, чтобы сделать изображение на Canvas'е (с указанием  ширины и высоты  100).  
  
Data URIs , как правило, используются для этой цели, но может быть использован на любом другом типе файлов. Наиболее распространенный вариант использования для чтения файлов в data URI  для отображения содержимого файлов сразу на веб-странице.

Чтение в ArrayBuffers

Тип ArrayBuffer впервые был введен как часть WebGL. ArrayBuffer представляет собой конечное число байтов, которые могут быть использованы для хранения данных любого размера. Данные, которые записываются в ArrayBuffer являются типизированным массивом, и не могут содержать разнотипные данные, так как это могут делать традиционные JavaScript массивы.

Вы можете использовать ArrayBuffer в первую очередь при работе с бинарными файлами, чтобы иметь более точный контроль над данными.  Вы можете передать ArrayBuffer непосредственно в метод send ()  объекта XHR для передачи исходных данных на сервер (на стороне сервера, данные принимаются и обрабатываются, как двоичные данные ).

Что дальше

Читать данные из файла с помощью FileReader довольно просто. Если вы знаете, как использовать XMLHttpRequest, то чтение данных из файлов для Вас далось также легко. В следующей части этой серии, вы узнаете больше об использовании событий  FileReader  и возможных ошибках.

Ссылки по теме:

[ArrayBuffer](https://developer.mozilla.org/en/JavaScript_typed_arrays/ArrayBuffer" \t "_blank)  
[Спецификация по типизированным массивам](http://www.khronos.org/registry/typedarray/specs/latest/" \t "_blank)  
[XMLHttpRequest Level 2](http://www.w3.org/TR/XMLHttpRequest/" \t "_blank)

Связанные статьи

[Работа с файлами в JavaScript, Часть 1: Основы](http://xdan.ru/Working-with-files-in-JavaScript-Part-1-The-Basics.html)  
[Работа с файлами в JavaScript, Часть 3:Событие прогресса и ошибки](http://xdan.ru/Working-with-files-in-JavaScript-Part-3-Progress-events-and-errors.html)  
[Работа с файлами в JavaScript, Часть 4:Объект URL](http://xdan.ru/working-with-files-in-javascript-part-4-object-urls.html)

Работа с файлами в JavaScript, Часть 3:Событие прогресса и ошибки

Объект FileReader используется для чтения данных из файлов, которые доступны через браузер. В моей предыдущей статье вы узнали, как используя объект FileReader читать данные из файла в различных форматах. FileReader во многом очень похож на XMLHttpRequest.

Событие прогресса (Progress events)

События показывающее прогресс какого-либо процесса очень распространены. Эти события спроектированы для отображения прогресса передачи данных. Такая передача происходит как при запросе данных с сервера, так и при запросе данных с диска, что FileReader и делает.

Есть шесть событий прогресса:

* loadstart – указывает что процесс загрузки данных начался. Это событие всегда срабатывает первым
* progress – срабатывает несколько раз по мере загрузки данных, дает доступ к промежуточным данным
* error –  срабатывает когда загрузка окончилась неудачей
* abort – срабатывает когда загрузка данных была отменена вызовом метода abort() (такой метод есть и у XMLHttpRequest).
* load – срабатывает только тогда, когда все данные были успешно загружены
* loadend – срабатывает, когда объект завершил передачу данных. Всегда срабатывает после error, abort или load.

Два события, error и load, были обсуждены в моем предыдущем посте. Другие события дают более узкий контроль над процессом передачи данных.

Отслеживание прогресса

Когда вы хотите отследить прогресс чтения файлов, используйте событие progress. Объект [event](http://xdan.ru/kak-napisat-plagin-na-jquery.html) который является параметром этого события содержит 3 поля, для контроля передаваемых данных

* lengthComputable – тип boolean указывает может ли браузер определить размер файла
* loaded – число байтов которые уже загружены
* total – общее количество байтов которые нужно прочитать

Эти данные позволяют создать прогресс бар, который будет показывать информацию от прогрессе загрузки . Например, вы можете использовать элемент HTML5 <progress> для мониторинга прогресса чтения файла. Вы можете связать уровень прогресса с фактическими данными, используя следующий код:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | var reader = new FileReader(),  progressNode = document.getElementById("my-progress");    reader.onprogress = function(event) {    if (event.lengthComputable) {      progressNode.max = event.total;      progressNode.value = event.loaded;    }  };    reader.onloadend = function(event) {    var contents = event.target.result,    error = event.target.error;    if (error != null) {      console.error("File could not be read! Code " + error.code);    } else {      progressNode.max = 1;      progressNode.value = 1;      console.log("Contents: " + contents);    }  };    reader.readAsText(file); |

Это похоже на подход, который использует Gmail при реализации "drag and drop" загрузки файла, где вы видите прогрессбар сразу после добавления файла к электронному письму. Этот прогрессбар показывает, насколько файл уже передан на сервер.

Работа с ошибками

Даже если Вы читате локальный файл, это может привести к краху чтения. Спецификация [File API](http://xdan.ru/working-with-files-in-javascript-part-1-the-basics.html) определяет 4 типа ошибок:

* NotFoundError – файл не может быть найден.
* SecurityError – чтение файла не безопасно либо запрещено. Если файл слишком большой то Вы тоже увидите эту ошибку.
* NotReadableError – файл существует, но не может быть прочитан, скорее всего, из-за проблемы с правами доступа.
* EncodingError – возникает к примеру когда читаете файл как dataURI и длинна его выходит за пределы поддерживаемые браузером

При возникновении [ошибки](http://xdan.ru/ne-rabotaet-denwer-na-windows-8.html) во время чтения файла, полю объекта FileReader error присваивается экземпляр одной из вышеупомянутых ошибок. По крайней мере, именно так написано в спецификации. На самом деле, браузеры реализовывают это как объект FileError, который имеет поле код, указывающий тип ошибки, которая произошла. Каждый тип ошибки представляет собой целочисленную константу:

* FileError.NOT\_FOUND\_ERR для ошибки NotFoundError.
* FileError.SECURITY\_ERR для SecurityError.
* FileError.NOT\_READABLE\_ERR для NotReadableError.
* FileError.ENCODING\_ERR для EncodingError.
* FileError.ABORT\_ERR когда вызван abort() в процессе чтения файла.

Вы можете проверить тип ошибки либо во время события error или во время события loadend:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | var reader = new FileReader();    reader.onloadend = function(event) {    var contents = event.target.result,      error = event.target.error;    if (error != null) {      switch (error.code) {        case error.ENCODING\_ERR:          console.error("Проблемы кодировки!");          break;          case error.NOT\_FOUND\_ERR:          console.error("Файл не найден!");          break;          case error.NOT\_READABLE\_ERR:          console.error("Файл не может быть прочитан!");          break;          case error.SECURITY\_ERR:          console.error("Проблема безопасности в файл!");          break;          default:          console.error("Я понятия не имею, что случилось!");      }    } else {      progressNode.max = 1;      progressNode.value = 1;      console.log("Contents: " + contents);    }  };    reader.readAsText(file); |

Что дальше

Объект FileReader это полнофункциональный объект с большим количеством методов сходных с методами XMLHttpRequest. из прошлых постов, Вы должны уметь считывать данные из файла с помощью JavaScript и отправлять данные на сервер, если это необходимо. Тем не менее, экосистема File API гораздо больше, чем описано  в этой серии статей, и в следующей части вы узнаете о новых мощных функциях, предназначенных для работы с файлами.

Чтение файлов на JavaScript вообще штука интересная, Вы можете сохранить данные на карту памяти SD/SDHC/Secure Digital любых объемов? а затем прочитать их при помощи этого же FileAPI, что несомненно очень удобно. С учетом того, что онлайн приложения развиваются семимильными шагами покупаем карту [SD SDHC](http://www.sotmarket.ru/category/karti_pamiti/sd_sdhc.html" \t "_blank)  в каком нибудь SotMarket и экспериментируем с чтением файла через JavaScript FileAPI уже сейчас.

Ссылки по теме

[Progress Events](http://www.w3.org/TR/progress-events/" \t "_blank)  
[File API](http://www.w3.org/TR/FileAPI/" \t "_blank)

Связанные статьи

[Работа с файлами в JavaScript, Часть 1: Основы](http://xdan.ru/Working-with-files-in-JavaScript-Part-1-The-Basics.html)  
[Работа с файлами в JavaScript, Часть 2: FileReader](http://xdan.ru/Working-with-files-in-JavaScript-Part-2-FileReader.html)  
[Работа с файлами в JavaScript, Часть 4:Объект URL](http://xdan.ru/working-with-files-in-javascript-part-4-object-urls.html)

Рассказать друзьям

[Facebook](http://xdan.ru/" \l "facebook" \t "_blank) [Twitter](http://xdan.ru/" \l "twitter" \t "_blank) [Google+](http://xdan.ru/" \l "google_plus" \t "_blank) [VK](http://xdan.ru/" \l "vk" \t "_blank) [Ресурс](https://www.addtoany.com/share" \l "url=http%3A%2F%2Fxdan.ru%2Fworking-with-files-in-javascript-part-3-progress-events-and-errors.html&title=%D0%A0%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%D1%81%20%D1%84%D0%B0%D0%B9%D0%BB%D0%B0%D0%BC%D0%B8%20%D0%B2%20JavaScript%2C%20%D0%A7%D0%B0%D1%81%D1%82%D1%8C%203%3A%D0%A1%D0%BE%D0%B1%D1%8B%D1%82%D0%B8%D0%B5%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B5%D1%81%D1%81%D0%B0%20%D0%B8%20%D0%BE%D1%88%D0%B8%D0%B1%D0%BA%D0%B8)

Работа с файлами в JavaScript, Часть 4:Объект URL

Ранее в блоге в серии  Вы узнали, как использовать файлы традиционным путем. Вы можете загрузить файлы на сервер, и Вы можете читать содержимое файла с диска. Это наиболее распространенный способ работы с файлами. Тем не менее, это совершенно новый метод, может упростить некоторые общие задачи. Новый способ заключается в использовании объекта URL.

Что такое объект URL?

Объект Url - это адрес (идентификатор), который указывают на файл на диске. Предположим, что Вы хотите вывести изображение с из диска пользователя на веб-страницу. Сервер ничего не должен знать о файле, поэтому нет необходимости загружать его туда. Вы просто хотите отобразить файл в странице. Можно, как показано в предыдущих постах, получить ссылку на объект File, считать данные в data URI, а затем назначить data URI  обычному <img>. Но думаю, это не лучший подход: образ уже существует на диске, читать изображения в другой формат для того, чтобы использовать его? Если вы создаете объект URL, вы можете связать его с  <img> и тот получит доступ к этому локальному файлу напрямую.

Как это работает?

[File API](http://xdan.ru/working-with-files-in-javascript-part-1-the-basics.html) определяет глобальный объект, называемый URL, который имеет два метода. Первый createObjectURL(), который принимает ссылку на файл и возвращает объект URL. Это позволяет браузеру управлять через URL локальным файлом. Второй способ revokeObjectURL (), который указывает браузеру, как уничтожить URL, который передается в него, фактически освобождая память. Конечно, все объекты URL уничтожаются, как только веб-страницы выгружается, но хорошей практикой будет освободить их, когда они больше не нужны.

Поддержки объекта URL, как и для других частей File API, не так хороша. На момент написания поста, Internet Explorer, Firefox,  поддерживают глобальный объект URL. Chrome поддерживает его в форме webkitURL в то время как Safari и Опера не имеют поддержки.

Пример использования

Вывести изображение, которое пользователь выбрал на веб страницу::

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | var URL = window.URL || window.webkitURL,    imageUrl,    image;    if (URL) {    imageUrl = URL.createObjectURL(file);    image = document.createElement("img");    image.onload = function() {      URL.revokeObjectURL(imageUrl);    };    image.src = imageUrl;    document.body.appendChild(image);  } |

В примере объявляется переменная URL, которая совмещает различные реализации браузеров. Предполагая, что URL поддерживается браузером, код продолжает создавать объект URL прямо из файла и сохраняет его в ImageUrl. Новый элемент <img> создается и на обработчик события onload, вешается уничтожение объекта URL (подробнее об этом чуть позже). Затем, поле src изображения связывается с URL и  элемент добавляется в страницу(это не обязательно, можно использовать уже существующий на странице элемент <img>).

Почему мы уничтожили объект URL после загрузки изображения? После загрузки изображения, URL больше не нужен, если вы не собираетесь использовать его с другим элементом. В этом примере, изображение загружается в один элемент, и после того, как изображение было полностью загружены, URL не несет никакой полезной цели. Самое время, чтобы освободить  память, связанную с ним.

Безопасность и другие факторы

На первый взгляд, это возможность немного опасна. Вы загружаете файл непосредственно с компьютера пользователя через URL. Однако URL сам по себе не является большой проблемой безопасности, потому что это URL-адрес, который назначается динамически в браузере и был бы бесполезен на любом другом компьютере. А как насчет кросс-домменного использования?

File API запрещает использование объекта URL из разных доменов. Когда объект URL создан, он связан со страницей, в которой выполняется JavaScript, поэтому вы не можете использовать объект URL из www.wrox.com на странице p2p.wrox.com. Тем не менее, две страницы из www.wrox.com, где одна из них встроена в другую при помощи iframe, способны обмениваться объектами URL.  
  
Объект URL, существует лишь постольку, поскольку документ, в котором он был создан открыт. Когда документ выгружается, все URL объекты уничтожаются. Таким образом, не имеет смысла хранить объект URL на стороне клиента  для дальнейшего использования, он являются бесполезным после того, как страница была выгружена.  
  
Вы можете использовать объект URL везде, где браузер создает GET запрос, это изображения, сскрипты, web workers, таблиц стилей, audio, и video. Вы не сможете использовать объект URL  при POST запросе.

Что дальше?

Возможность создания URL ссылающихся непосредственно на локальный файлам является мощным инструментом. Вместо того, чтобы читать локальный файл в JavaScript для того, чтобы отобразить его на странице, вы можете просто создать URL и указать элементу страницы использовать его, как адрес. Этот процесс значительно упрощает использование локальных файлов на странице. Тем не менее, удовольствие от работы с файлами в JavaScript только начинается. В следующей статье Вы узнаете некоторые интересные способы работы с данными из файла.

Самый первый способ применения, который лезет в голову это загрузка аватара на страницу. Очень удобно будет выбирать себе аву, редактировать ее, а лишь потом отсылать на сервер. Таким образом вы повысите юзабилити своего ресурса, после этого повыситься  посещаемость, которая перерастет в новых клиентов. Однако перед этим, советую провести seo [аудит сайта](http://www.ldi.ru/audit-saita-v-poiskovuh-sistemah.html" \t "_blank), дабы выяснить каким-образом он индексируется поисковыми системами.

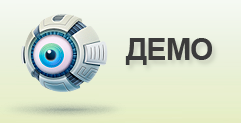
Ссылки по теме

[File API](http://www.w3.org/TR/FileAPI/" \t "_blank)

Связанные статьи

[Работа с файлами в JavaScript, Часть 1: Основы](http://xdan.ru/Working-with-files-in-JavaScript-Part-1-The-Basics.html)  
[Работа с файлами в JavaScript, Часть 2: FileReader](http://xdan.ru/Working-with-files-in-JavaScript-Part-2-FileReader.html)  
[Работа с файлами в JavaScript, Часть 3:Событие прогресса и ошибки](http://xdan.ru/Working-with-files-in-JavaScript-Part-3-Progress-events-and-errors.html)

# HTML5 File API для взаимодействия с файлами в браузере

[](http://codepen.io/stanislas-prime/pen/QwLPqd)

Недавно я столкнулся с определённой задачей: необходимо, чтобы пользователь загрузил файл через браузер, сделал несколько преобразований и затем сохранил результат.

Данную задачу можно было бы реализовать старым способом:

* Загрузить изображение через AJAX
* Рендеринг загруженного изображения в браузере
* Преобразуем изображение через JavaScript
* Делаем дополнительный запрос для преобразования изображения и
* Сохраняем преобразованное изображение на сервере

Слишком много серверных операций. Данный способ не очень эффективен, так что мы нашли другой способ - [HTML5 File API](http://dev.w3.org/2006/webapi/FileAPI/).

## Что такое HTML5 File API?

HTML5 File API позволяет локально преобразовывать файлы, без предварительной загрузки на сервер.

## 3 основных HTML5 File объекта

Для работы с данным API вам нужно ознакомиться с тремя объектами:

[File](https://developer.mozilla.org/en-US/docs/Web/API/File) – простой объект файла с некоторыми метаданными.

[FileList](https://developer.mozilla.org/en-US/docs/Web/API/FileList) – список из объектов типа файл.

[FileReader](https://developer.mozilla.org/en-US/docs/Web/API/FileReader) – объект для считывания файла и некоторыми метода обработки.

## Получение доступа к файлу через JavaScript

Список файлов можно получить после их выбора через HTML элемент input. Вот небольшой пример по взаимодействию с загружаемым файлом. Мы воспользуемся console.log() для вывода промежуточной информации.

### Выбор одного файла

|  |  |  |
| --- | --- | --- |
| 1 | // отслеживаем использование элемента input с id = “the-file-input” | |
| 2 | $("#the-file-input").change(function() { |

|  |  |  |
| --- | --- | --- |
| 3 | // выводим объект FileList | |
| 4 | console.log(this.files); |

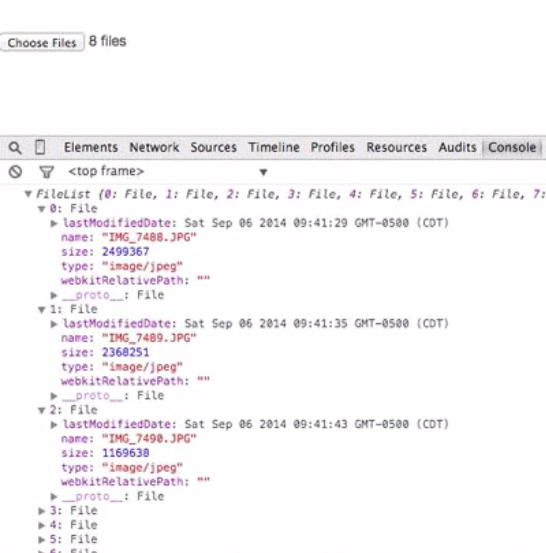
|  |  |
| --- | --- |
| 5 | }); |
| 1 | <!-- получаем доступ через input --> | |

|  |  |
| --- | --- |
| 2 | <input id="the-file-input" type="file"> |



### Выбор нескольких файлов

|  |  |  |
| --- | --- | --- |
| 1 | <!-- для выбора нескольких файлов, добавляем атрибут multiple --> | |
| 2 | <input id="the-file-input" type="file" multiple> |



## Рендеринг файла в браузере

Теперь когда мы знаем как получить доступ к объекту FileList, осталось узнать как осуществить рендеринг изображения в браузере. Для этого нам нужно скормить объект File, объекту FileReader для генерации локального url, который будет доступен через атрибут src элемента image.

|  |  |
| --- | --- |
| 01 | // рендеринг изображения |
| 02 | function renderImage(file) { | |

|  |  |
| --- | --- |
| 03 |  |
| 04 | // генерация нового объекта FileReader | |

|  |  |  |
| --- | --- | --- |
| 05 | var reader = new FileReader(); | |
| 06 |  |

|  |  |  |
| --- | --- | --- |
| 07 | // подстановка изображения в атрибут src | |
| 08 | reader.onload = function(event) { |

|  |  |
| --- | --- |
| 09 | the\_url = event.target.result |
| 10 | $('#some\_container\_div').html("<img src="" + the\_url + "" alt="">") | |

|  |  |  |
| --- | --- | --- |
| 11 | } | |
| 12 |  |

|  |  |  |
| --- | --- | --- |
| 13 | // при считке файла, вызывается метод, описанный выше | |
| 14 | reader.readAsDataURL(file); |

|  |  |  |
| --- | --- | --- |
| 15 | } | |
| 16 |  |

|  |  |
| --- | --- |
| 17 | // обработка элемента input |
| 18 | $("#the-file-input").change(function() { | |

|  |  |  |
| --- | --- | --- |
| 19 | console.log(this.files) | |
| 20 |  |

|  |  |  |
| --- | --- | --- |
| 21 | // выбор первого изображения из FileList и передача в функцию | |
| 22 | renderImage(this.files[0]) |

|  |  |
| --- | --- |
| 23 | }); |

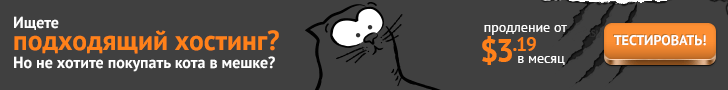
В данном примере мы продемонстрировали как можно отобразить изображение в браузере без загрузки на сервер! Это отличный вариант, особенно если на нашем сайте огромное количество пользователей, который совершают данную операцию.



## Заключение

Метод, который мы показали может снизить нагрузку на сервер. Так же это новый полигон для создания приложений по обработке фото.

Данный урок подготовлен для вас командой сайта [ruseller.com](http://ruseller.com/)  
Источник урока: <http://scotch.io/tutorials/use-the-html5-file-api-to-work-with-files-locally-in-the-browser>  
Перевел: Станислав Протасевич  
Урок создан: 11 Ноября 2014  
Просмотров: 11195  
[Правила перепечатки](http://ruseller.com/page.php?doc=reprint)

[](https://hostiq.com.ua/virtual-hosting/?utm_source=ruseller.com&utm_medium=sponsored&utm_campaign=ruseller-banner-flash)

## 5 последних уроков рубрики "HTML5"

* http://ruseller.com/lessons/les2786/thumbnail.png

#### [Определение доступности атрибута HTML5 с помощью JavaScript](http://ruseller.com/lessons.php?rub=43&id=2786)

Сегодня мы посмотрим, как можно организовать проверку доступности атрибута HTML5 с помощью JavaScript. Проверять будем работу элементов details и summary.

* http://ruseller.com/lessons/les2726/thumbnail.png

#### [HTML5: API работы с вибрацией](http://ruseller.com/lessons.php?rub=43&id=2726)

HTML5 - глоток свежего воздуха в современном вебе. Она повлиял не только на классический веб, каким мы знаем его сейчас. HTML5 предоставляет разработчикам ряд API для создания и улучшения сайтов с ориентацией на мобильные устройства. В этой статье мы рассмотрим API для работы с вибрацией.

* http://ruseller.com/lessons/les2675/thumbnail.jpg

#### [Создание форм с помощью Webix Framework — 4 практических примера](http://ruseller.com/lessons.php?rub=43&id=2675)

Веб дизайнеры частенько сталкиваются с необходимостью создания форм. Данная задача не простая, и может вызвать головную боль (особенно если вы делаете что-то не стандартное, к примеру, много-страничную форму). Для упрощения жизни можно воспользоваться фрэймворком. В этой статье я покажу вам несколько практических приёмов для создания форм с помощью фрэймворка Webix.

* http://ruseller.com/lessons/les2672/thumbnail.jpg

#### [Знакомство с фрэймворком Webix](http://ruseller.com/lessons.php?rub=43&id=2672)

В этой статье мы бы хотели познакомить вас с фрэймворком Webix. Для демонстрации возможностей данного инструмента мы создадим интерфейс online аудио плеера. Не обольщайтесь — это всего лишь модель интерфейса. Исходный код доступен в демо и на странице GitHub.

* http://ruseller.com/lessons/les2662/thumbnail.png

#### [Отключение HTML5 валидации](http://ruseller.com/lessons.php?rub=43&id=2662)

Бесит валидация, которая добавляется к полям формы, если вы пользуетесь HTML5? Её можно без проблем отключить.

# Файловая система и JavaScript. Обработка файлов в браузере

[Владимир Дашукевич](https://xbsoftware.ru/author/vladimir/) |Апрель 25, 2014

Существует множество задач, связанных с созданием, парсингом, сохранением и удалением файлов. Большинство из них выполняется на стороне сервера. Прежде всего потому, что Java, C# или любой другой язык, который вы используете на сервере, может легко справиться со всеми этими задачами.

Однако, в этом случае существует большая проблема: у всех ваших пользователей имеется в распоряжении только один сервер, но каждый из них хочет выполнить какие-то действия. Это может вызвать увеличение расхода ресурсов сервера или даже вывести ваш сервер из строя из-за огромного количества запросов.

Вторая причина, по которой вы все еще выполняете вышеуказанные действия на сервере, заключается в том, что вы просто еще не прочитали данную статью и не знаете всех возможностей современных браузеров. Не волнуйтесь! Мы откроем вам все их секреты и тайны.

Данную тему можно разделить на три части: прошлое, настоящее и будущее.

## Прошлое. ActiveX

В прошлом существовало только два браузера: Netscape и IE. Однажды Microsoft решил порадовать пользователей IE и добавил невероятную для того времени функциональность, которая позволяла коду, написанному на языке JavaScript, обмениваться информацией с операционной системой Windows. Данный способ получил название ActiveX.

Описанная технология может применяться даже сегодня в последней версии браузера IE (по умолчанию, ActiveX отключен). Не будем долго останавливаться на этой теме, поскольку ActiveX актуален только для IE, и пользователю необходимо проделать много необычных действий со своим браузером для того, чтобы запустить сценарий, который содержит ActiveXObjects.

В разделе, посвященном настоящему времени, мы поговорим об API для работы с файлами и Drag-and-Drop, а также о некоторых интересных примерах.

Самая же увлекательная часть содержится в разделе о “будущем”, где будут раскрыты все секреты файловой системы в браузере. Давайте приступим к изучению возможностей браузеров.

## Настоящее. Управление добавленными файлами

Существует два способа передать файл в современный браузер (IE10+):

1) Input тег с типом “file”;  
2) Перетянуть файлы на какой-либо элемент DOM-модели.

### Старые добрые времена с элементом Input

Первый вариант может использоваться даже в старых браузерах. Единственное отличие в том, что мы не можем получить содержимое файлов в IE9 и более ранних версий. Это означает, что для выполнения некоторых действий нам все еще необходимо использовать сервер. Чтобы получить файлы, выбранные пользователем, нам нужно добавить обработчик события “change” и проверять свойство “files” данного элемента при вызове функции-обработчика.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | var template = "<div><span>Name: </span><span>{{Name}}</span></div><div><span>Size: </span><span>{{Size}}</span></div><div><span>Data: </span><span>{{Data}}</span></div>",         data = document.getElementById("fileData");           document.getElementById("fileElement").addEventListener("change", function(e){                   var file = this.files ? this.files[0] : {                                     name: this.value                             },                             fileReader = window.FileReader ? new FileReader() : null;                               if (file){                                     if (fileReader){                                               fileReader.addEventListener("loadend", function(e){                                                         data.innerHTML = template.replace("{{Name}}", file.name).replace("{{Size}}", file.size).replace("{{Data}}", e.target.result.substring(0, 10));                                               }, false);                                               fileReader.readAsText(file);                                     } else {                                               data.innerHTML = template.replace("{{Name}}", file.name).replace("{{Size}}", "Don't know").replace("{{Data}}", "This browser isn't smart enough!");                                     }                             }         }, false); |

Как видите, здесь мы использовали метод “readAsText” объекта “FileReader”, чтобы получить содержимое файла. У него также есть методы, которые читают файл(файлы) и возвращают результаты в различных форматах.

Вы можете использовать метод „readAsArrayBuffer“, который читает файл, и возвращает файлы в виде массива данных в двоичном формате; „readAsDataURL“, который возвращает URL в виде “data:image/png,base64:”. Вы можете поместить этот URL в атрибуте src тега IMG, например (смотрите демо #2).

Будьте внимательны, используя метод “readAsBinaryString”, поскольку он не поддерживается браузером IE версий 10-11.

Возможно, вы также хотите контролировать процесс чтения файла, отобразить индикаторы прогресса или ожидания, или прервать его, если файлы слишком объемные. Специально для вас у FileReader есть такие события как onprogress, onabort, onloadstart, onloadend, onerror, а также метод abort, который останавливает процесс чтения и выдает ошибку, которая может быть обработана.

В данном примере мы немного исправили функциональность для IE9, у которого нет возможности чтения файлов. Из информации о выбранном файле (множественный выбор не поддерживается) мы можем получить только его имя, которое хранится в свойстве “value” элемента input.

Выберите картинку:

|  |  |
| --- | --- |
| 1  2  3  4 | fileReader.addEventListener("loadend", function(e){         img.src = e.target.result;  }, false);  fileReader.readAsDataURL(file); |

### Перетащи меня в браузер!

Это был старый способ добавления файлов в браузер. Он мог использоваться даже в IE6 (addEventListener поддерживается в IE9+), с некоторыми изменениями. В современных браузерах (IE10+) существует другой способ добавления файлов. Все, что нужно сделать, это повесить обработчик события «drop” на любой элемент веб-страницы. Это может быть как специальный элемент с текстом внутри, так и просто body. После этого пользователь может перетащить какой-либо файл в элемент, на котором висит обработчик события “drop”. Выбранный файл можно найти в свойстве “dataTransfer” объекта события. Давайте посмотрим на демо:

Перетащите свой файл сюда

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | var template = "<div><span>Name: </span><span>{{Name}}</span></div><div><span>Size: </span><span>{{Size}}</span></div><div><span>Data: </span><span>{{Data}}</span></div>",                   data = document.getElementById("fileData"),          preventDefault = function(e){                             e.preventDefault();          },          highlight = {            add: function(e){                   preventDefault(e);                   e.target.classList.add("hoverClass");            },            remove: function(e){                   preventDefault(e);                   e.target.classList.remove("hoverClass");            }          },          uploader = document.getElementById("fileElement");           if (window.FileReader){          uploader.addEventListener("dragover", highlight.add, false);          uploader.addEventListener("mouseout", highlight.remove, false);                   uploader.addEventListener("drop", function(e){                             var file = e.dataTransfer.files[0],                                     fileReader = new FileReader();                               if (file) {                                     if (fileReader){                                               fileReader.addEventListener("loadend", function(e){                                                         data.innerHTML = template.replace("{{Name}}", file.name).replace("{{Size}}", file.size).replace("{{Data}}", e.target.result.substring(0, 10));                                               }, false);                                               fileReader.readAsText(file);                                     } else {                                               alert("Sorry, but this browser isn't smart enough( Choose another one");                                     }                             }                             preventDefault(e);                   }, false);         } else {                   alert("Sorry, but this browser isn't smart enough( Choose another one");         } |

Несколько важных замечаний по этому демо:

1) Прежде всего, вам не стоит проверять, поддерживает ли ваш браузер событие “ondrop”, поскольку используется IE9. Он поддерживает данное событие, но не имеет свойства dataTransfer.

2) Если вы хотите поймать событие “ondrop”, вам необходимо добавить обработчик события “ondragover” и вызвать функцию preventDefault. По умолчанию, браузер попытается загрузить этот файл или отобразить его на странице (если это рисунок или PDF файл), и вам нужно его остановить.

Поддержка старых версий браузеров: в таких браузерах, как IE9, вы можете поместить элемент input из предыдущих примеров на текущем элементе, растянуть его, установить для него прозрачность равную 100%, и изменить текст на следующий: “Кликните и выберите файл”. Когда пользователь кликнет на этот элемент, он увидит обычный диалог для выбора файлов. Подобное решение подходит также для планшетов и телефонов, у которых нет возможности разделить экраны и перетягивать элементы между ними.

### Перетащи меня из браузера!

Предыдущий пример мог вызвать следующий вопрос о функциональности drag-and-drop: “Можем ли мы перетащить файлы из браузера на компьютер?” Ответ будет “Да”, но работает это только в браузере Chrome. Тем не менее, это отличная функция для вашей веб страницы, и мы должны ее испробовать. В следующем примере вам нужно перетянуть картинку в формате PDF на свой компьютер, и она будет сохранена как PDF файл.

Перетащите эту картинку из браузера на ваш рабочий стол

|  |  |
| --- | --- |
| 1  2  3 | document.getElementById("fileData").addEventListener("dragstart", function(e){                   e.dataTransfer.setData("DownloadURL", "application/pdf:doc.pdf:file:///C:/personal/publications/demo/doc.pdf");         }, false); |

Код в данном примере достаточно прост. Все, что вам нужно сделать — это добавить обработчик события “dragstart” и задать путь к данным, которые необходимо переместить. Это может быть ссылка на какой-либо документ на сервере или двоичные данные. Если вы хотите получить файл с сервера, нужно передать специальную строку, состоящую из типа содержимого, имени файла, который будет сохранен, и полного пути к нему. Например: “application/pdf:newDoc.pdf:https://mywebsite.com”.

Как уже было сказано, перетащить файл можно только из браузера Chrome. Однако, проблемой это не является, поскольку это не основная функциональность вашего веб-приложения, а просто отличное дополнение для пользователей Chrome, которая может быть благополучно заменена обычной ссылкой для загрузки нужного файла.

**“Все течет, все изменяется … и … дважды в одну и ту же реку войти невозможно”. (Гераклит)**

Теперь мы можем получить файлы и их содержимое в формате JavaScript. Но что мы можем с ними делать? Как мы можем их изменить? Вот в чем вопрос.

Прежде всего, мы можем работать с текстом. Когда мы прочитаем файл как текстовую строку, мы получим все его текстовое содержимое и сможем его парсить. Например, пользователь может перетянуть новый CSS файл, и вы сможете сразу применить его на своей странице. Или, если это файл со статьей, которую нужно опубликовать, вы можете его прочитать, применить какие-либо стили к заголовкам и поместить его в текстовый редактор на вашей странице.

Примером также может быть парсинг файла в формате excel. Вам не нужно посылать его на сервер, конвертировать в JSON и отправлять обратно. Это можно сделать на клиентской стороне, не тратя времени на запрос и ожидание ответа сервера. Уже есть библиотеки для парсинга xls и xlsx файлов, которые вы можете найти здесь: [XLS](https://github.com/SheetJS/js-xlsx) и [XLSX](https://github.com/SheetJS/js-xlsx).

Еще один способ использования файлов и их содержимого в JavaScript — это работа с двоичными данными. Это могут быть рисунки, видео или какие-либо другие файлы. Если мы хотим обрабатывать двоичные данные, нам нужно использовать новые свойства JavaScript, такие как объект Blob и типизированные массивы. Мы не будем углубляться в эту тему, потому что она слишком большая и заслуживает отдельной статьи.

Объект Blob — это объект, который выглядит, как обычный файл на JavaScript, и содержит необработанные данные. Конструктор такого объекта принимает два параметра — данные и объект с его опциями. В состав данных могут входить ArrayBuffer, типизированный массив, объект Blob или строки. Это все те данные, которые будут храниться в новом объекте Blob. Опции включают в себя тип данных.

|  |  |
| --- | --- |
| 1 | new Blob(["<div>Hello world</div>"], {type: "text/html"}) |

У каждого объекта Blob есть тип и свойства, которые задают его размеры, а также метод “slice”. Все данные, которые содержатся в данном объекте Blob, можно разделить с помощью метода “slice”. Этот метод возвращает новый объект Blob, который содержит в себе отделенную часть данных.

Типизированные массивы и буферы используются для хранения двоичных данных фиксированной длины и определенного типа ([Int8Array](https://developer.mozilla.org/en/JavaScript_typed_arrays/Int8Array) или [Int32Array](https://developer.mozilla.org/en/JavaScript_typed_arrays/Int8Array)).

Все это позволяет нам делать занимательные вещи на клиентской стороне, например, сжимать файлы. [Эта библиотека](https://github.com/tlrobinson/zipjs) позволяет применить алгоритм Deflate ко всем выбранным файлам, и собрать их в один zip файл. Кроме того, она использует Web Workers, чтобы описанный процесс протекал в фоновом режиме.

Еще один пример обработки файла, содержащего двоичные данные, это [PDF.js](https://github.com/mozilla/pdf.js" \l "readme" \o "читать о pdf.js" \t "_blank). Это проект, который находится в открытом доступе и используется для отображения на веб-странице документов в формате PDF. Все, что вам нужно сделать, это передать ему массив двоичных данных, и он отрисует содержимое PDF-документа на странице.

## Будущее. Создайте свою файловую систему с блэкджеком и… всеми причитающимися атрибутами

Теперь мы знаем, как получать файлы и изменять их. Настало время раскрыть секрет хранения файлов. Да-да, речь идет именно о создании и хранении папок и файлов внутри браузера.

Во время написания этой статьи данная функциональность поддерживалась только браузерами Chrome и Opera. Опять-таки, это не такая уж важная проблема, поскольку данная возможность может использоваться как дополнительный инструмент для кэширования файлов в вашем браузере. Давайте более детально рассмотрим ее API.  
Прежде всего, нам нужно запросить объект файловой системы с помощью метода webkitRequestFileSystem. Он принимает 4 параметра: тип, размер, функция, которая будет вызвана в случае успешной обработки запроса, и функция, которая будет вызвана при возникновении ошибки.

В браузере имеются два типа файловой системы: временная и постоянная. Первая может быть создана без запроса разрешения пользователя. Вы просто ее создаете и затем используете. Однако, если ваш браузер столкнется с какой-либо проблемой, как, например, недостаток памяти, он удалит все хранящиеся в нем данные. Поэтому данный тип хранения информации может быть использован только для кэширования данных, которые можно восстановить.

Еще один тип файловой системы — постоянный. Браузер никогда не удалит файлы, хранящиеся в ней. Тем не менее, если вы запросите данный тип файловой системы без разрешения пользователя, она не будет работать.

Для того, чтобы запросить разрешение пользователя на использование постоянного хранилища, нам нужно вызвать метод navigator.webkitPersistentStorage.requestQuota.

Константы обоих типов файловых систем хранятся в объекте window: window.TEMPORARY или window.PERSISTENT. Второй параметр — это нужный нам размер хранилища в байтах.

Теперь мы можем создать наше первое хранилище вместимостью 1Гб:  
Временная файловая система:

|  |  |
| --- | --- |
| 1  2  3 | window.webkitRequestFileSystem(window.TEMPORARY, 1000\*1024\*1024, function(fs){         alert("Storage is ready");  }, errorHandler); |

Постоянная файловая система:

|  |  |
| --- | --- |
| 1  2  3  4  5 | navigator.webkitPersistentStorage.requestQuota(1000\*1024\*1024, function(bytes) {           window.webkitRequestFileSystem(window.PERSISTENT, bytes, function(fs){                alert("storage is ready");           }, errorHandler);         }, errorHandler); |

Как видите, мы используем много обработчиков ошибок, которые должны отлавливать ошибки, связанные с созданием хранилищ. Объект FileError содержит полный список типов ошибок:

Итак, теперь у нас есть доступ к хранилищу, и мы можем создать нашу первую папку и файл, и сохранить их. После того, как запрос на использование постоянного хранилища (так же, как и в случае с временным) одобрен, мы получаем объект файловой системы в качестве первого аргумента функции, возвращаемой при успешном выполнении запроса.

Данный объект имеет интерфейс следующего вида:

Как видите, для создания, перемещения, удаления и получения файлов и директорий используются специальные методы. Полученный нами объект “fs” ссылается на корневой каталог файловой системы браузера.

Данный каталог и все другие каталоги, которые мы будем создавать, представлены интерфейсом[DirectoryEntry](http://dev.w3.org/2009/dap/file-system/pub/FileSystem/#the-directoryentry-interface). Все файлы данной файловой системы представлены интерфейсом [FileEntry](http://dev.w3.org/2009/dap/file-system/pub/FileSystem/" \l "the-fileentry-interface" \o "читать" \t "_blank). Мы будем использовать методы getFile и getDirectory для того, чтобы создать файлы и папки и получить их.

Первый параметр данных методов — это файл или путь к нему. Этот путь может быть как абсолютным, так и относительным. Второй параметр — это объект с действиями, которые выполняются в случае, если в указанной директории файл отсутствует. Данный параметр может иметь свойства “create” и “exclusive”.  
Если свойству “create” задано значение “true”, будет создан новый файл или директория, а старый файл будет переписан. Если же мы для данного свойства зададим значение “false”, методы вернут объект [FileEntry](http://dev.w3.org/2009/dap/file-system/pub/FileSystem/" \l "the-fileentry-interface" \o "читать" \t "_blank) или выдадут ошибку, если файла не существует. Если задано свойство “exclusive” со значением “true”, файлы не будут переписаны.

Для того, чтобы различить файлы и папки, у каждого из двух интерфейсов есть флаги: isDirectory и isFile. Кроме того, существует специальный класс FileWriter, который служит для добавления содержимого в файл. Его необходимо запрашивать у браузера для каждого файла, который нужно изменить. Давайте добавим несколько файлов:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | // создадим папку “files” в корневом каталоге                fs.root.getDirectory("files", {create: true}, function(directory){                          //в папке "files" создадим файл “newFile.txt”                          directory.getFile('newFile.txt', {create: true}, function(file) {                          //добавим текст в файл “newFile.txt”                               file.createWriter(function(writer) {                                writer.onwriteend = function(e) {                                  alert("done");                                };                                // создадим новый объект  Blob и запишем его в log.txt.                                var blob = new Blob(['hello world!!!!'], {type: 'text/plain'});                        writer.write(blob);                   });                     }, errorHandler);                }); |

Все прошло успешно и, вроде бы, без ошибок. Давайте проверим наши файл и папку. Необходимо запустить некоторые экспериментальные функции в DevTool браузера Chrome, чтобы увидеть содержимое файловой системы браузера. Поэтому, вам нужно перейти на страницу “chrome://flags/#enable-devtools-experiments”. После перезагрузки Chrome, в настройках DevTool вы увидите вкладку “Эксперименты”. Здесь вам нужно включить опцию “File System inspection” и перезапустить DevTool.

После этого вы увидите опцию “FileSystem” на вкладке “Resources” раздела DevTool. Здесь отображаются все созданные файлы и хранилища.

Данные файлы мы также можем получить по специальному внутреннему URL браузера, который может быть установлен в качестве атрибута “src” тега IMG. Он начинается с слова “filesystem:” и выглядит вот так: “filesystem:{URL of site}/persistent/files/newFile.txt”.  
Чтобы получить URL, мы использовали метод “toURL”.  
Теперь у нас есть вся необходимая информация для того, чтобы создать веб-приложение для кэширования аудио и видео файлов, которое позволяет слушать и смотреть их без подключения к интернету. Прежде всего, загрузим двоичный файл. Затем, мы сохраним его в постоянном хранилище и запустим.

# Урок 15: Чтение из текстового файла

В предыдущем уроке мы научились использовать РНР для доступа к файловой системе сервера. В этом уроке используем эту информацию для чтения из простого текстового файла.

Текстовые файлы отлично подходят для хранения разного рода данных. Они не так гибки, как базы данных, но обычно не требую такого количества памяти. Более того, текстовые файлы имеют формат, который читается на большинстве систем.

## Открыть текстовый файл

Для открытия текстового файла используем функцию [[документация](http://php.net/manual/en/function.fopen.php)fopen](http://php.net/manual/en/function.fopen.php). Вот её синтаксис:

fopen(filename, mode)

**filename**

Имя открываемого файла.

**mode**

Mode/Режим может быть "r" (reading/чтение), "w" (writing/запись) или "a" (appending/присоединение). В этом уроке мы будем только читать из файла и, соответственно, используем "r". В следующем уроке мы научимся записывать и присоединять текст.

Примеры этого урока используют файл [unitednations.txt](http://ru.html.net/tutorials/php/unitednations.txt). Это простой список программ и фондов ООН и их доменов. Можете загрузить этот файл или создать свой и использовать его в примерах.

Сначала попробуем открыть unitednations.txt:

<?php

// Открыть текстовый файл

$f = fopen("unitednations.txt", "r");

// Закрыть текстовый файл

fclose($f);

?>

## Пример 1: Чтение строки из текстового файла

С помощью функции [[документация](http://php.net/manual/en/function.fgets.php)fgets](http://php.net/manual/en/function.fgets.php) можно читать строку из текстового файла. Этот метод читает до первого символа переноса строки (но не включая символ переноса строки).

<html>

<head>

<title>Чтение из текстовых файлов</title>

</head>

<body>

<?php

$f = fopen("unitednations.txt", "r");

// Читать строку их текстового файла и записать содержимое клиенту

echo fgets($f);

fclose($f);

?>

</body>

</html>

 -  [Отображение примера](http://ru.html.net/tutorials/php/lesson15_ex1.php)

## Пример 2: Чтение всех строк текстового файла

<html>

<head>

<title>Чтение из текстовых файлов</title>

</head>

<body>

<?php

$f = fopen("unitednations.txt", "r");

// Читать построчно до конца файла

while(!feof($f)) {

echo fgets($f) . "<br />";

}

fclose($f);

?>

</body>

</html>

 -  [Отображение примера](http://ru.html.net/tutorials/php/lesson15_ex2.php)

В этом примере мы выполняем цикл по всем строкам и используем функцию [[документация](http://php.net/manual/en/function.feof.php)feof](http://php.net/manual/en/function.feof.php) (for end-of-file/до конца файла) для проверки достижения конца файла. Если конец не достигнуть ("!" - см. Урок 6), строка записывается.

Вместо цикличного прохода по всем строкам мы можем получить тот же результат функцией [fread](http://php.net/manual/en/function.fread.php). При работе с очень большими текстовыми файлами помните, что [fread](http://php.net/manual/en/function.fread.php) использует больше ресурсов, чем [[документация](http://php.net/manual/en/function.fgets.php)fgets](http://php.net/manual/en/function.fgets.php). Для маленьких файлов разница в работе несущественна.

## Пример 3: Простая директория ссылок

Как сказано в начале этого урока, текстовые файлы могут отлично подойти для хранения данных. Это показано на следующем примере, где создаётся простая директория ссылок из содержимого файла [unitednations.txt](http://ru.html.net/tutorials/php/unitednations.txt).

В файле систематизированно записаны: название программы, запятая, домен. Как вы, вероятно, могли предположить, в файле с разделением запятыми можно записать куда больше информации.

Для получения информации из каждой строки используем массив. См. в [Уроке 8](http://ru.html.net/tutorials/php/lesson8.php) о массивах.

<html>

<head>

<title>Чтение из текстовых файлов</title>

</head>

<body>

<?php

$f = fopen("unitednations.txt", "r");

// Читать построчно до конца файла

while (!feof($f)) {

// Создать массив с запятой-разделителем

$arrM = explode(",",fgets($f));

// Записать ссылки (получить данные из массива)

echo "<li><a href='http://" . $arrM[1] . "'>" . $arrM[0]. "</a></li>";

}

fclose($f);

?>

</body>

</html>

 -  [Отображение примера](http://ru.html.net/tutorials/php/lesson15_ex3.php)

Весьма удобно, правда? В принципе вы можете расширить этот файл сотнями ссылок или расширить директорию, включив также адреса.

В следующем уроке мы мы посмотрим, как записать в текстовый файл.