|  |
| --- |
| EPAM Systems |
| Filsesystem |
| [Document subtitle] |

|  |
| --- |
| Oksana Spolnyk  [Date] |

# Определение

Наконец появилась возможность взаимодействия с локальными файлами с помощью HTML5. Для этого служит спецификация API файлов. В частности, API файлов можно использовать для создания уменьшенных изображений при передаче соответствующих картинок на сервер или для того, чтобы приложение могло сохранить ссылку на файл, если пользователя нет в сети. Кроме того, с помощью кода на стороне клиента можно проверить, отвечает ли тип MIME загружаемого файла его расширению, или установить ограничения по размеру.

В спецификации указаны некоторые интерфейсы для доступа к файлам в локальной файловой системе.

1. File – отдельный файл. Такая информация, как название, размер файла, тип MIME и ссылка на обработчик, доступна только для чтения.
2. FileList – последовательность объектов File в виде массива, позволяющая, например, реализовать функцию <input type="file" multiple> или перетаскивание папки с файлами с рабочего стола.
3. Blob – позволяет разделить файл на фрагменты заданной величины.

В сочетании с описанными выше структурами данных интерфейс FileReader можно использовать для асинхронного чтения файла с помощью знакомых функций обработки событий в JavaScript. Это позволяет отслеживать процесс чтения, обнаруживать ошибки и определять, что загрузка завершена. API во многом напоминают модель событий XMLHttpRequest.

## Выбор файлов с помощью формы ввода данных

Самый простой способ загрузки файлов – использование стандартного элемента <input type="file">. JavaScript возвращает список выбранных объектов File в виде объекта FileList. Вот пример использования атрибута multiple для выбора сразу нескольких файлов:

<input type="file" id="files" name="files[]" multiple />  
<output id="list"></output>  
  
<script>  
 **function** handleFileSelect(evt) {  
 **var** files = evt.target.files; // FileList object  
  
 // files is a FileList of File objects. List some properties.  
 **var** output = [];  
 **for** (**var** i = 0, f; f = files[i]; i++) {  
 output.push('<li><strong>',

escape(f.name), '</strong> (', f.type || 'n/a', ') - ',  
 f.size, ' bytes, last modified: ',  
 f.lastModifiedDate.toLocaleDateString(), '</li>');  
 }  
 document.getElementById('list').innerHTML = '<ul>' + output.join('') + '</ul>';  
 }  
  
 document.getElementById('files').addEventListener('change', handleFileSelect, **false**);  
</script>

## Выбор файлов с помощью перетаскивания

Еще одним способом загрузки файлов является их перетаскивание с рабочего стола в браузер. Чтобы включить поддержку перетаскивания, достаточно внести небольшие изменения в код из предыдущего примера.

<div id="drop\_zone">Drop files here</div>  
<output id="list"></output>  
  
<script>  
 **function** handleFileSelect(evt) {  
 evt.stopPropagation();  
 evt.preventDefault();  
  
 **var** files = evt.dataTransfer.files; // FileList object.  
  
 // files is a FileList of File objects. List some properties.  
 **var** output = [];  
 **for** (**var** i = 0, f; f = files[i]; i++) {  
 output.push('<li><strong>', escape(f.name), '</strong> (', f.type || 'n/a', ') - ',  
 f.size, ' bytes, last modified: ',  
 f.lastModifiedDate.toLocaleDateString(), '</li>');  
 }  
 document.getElementById('list').innerHTML = '<ul>' + output.join('') + '</ul>';  
 }  
  
 **function** handleDragOver(evt) {  
 evt.stopPropagation();  
 evt.preventDefault();  
 evt.dataTransfer.dropEffect = 'copy'; // Explicitly show this is a copy.  
 }  
  
 // Setup the dnd listeners.  
 **var** dropZone = document.getElementById('drop\_zone');  
 dropZone.addEventListener('dragover', handleDragOver, **false**);  
 dropZone.addEventListener('drop', handleFileSelect, **false**);  
</script>

## Чтение файлов

Получив ссылку на объект File, создайте экземпляр объекта FileReader, который сохранит его содержание в память. Как только завершается загрузка, на стороне пользователя вызывается событие onload. Его атрибут result можно использовать для доступа к данным в файле.

Существует четыре метода асинхронного чтения файлов с помощью объекта FileReader.

1. FileReader.readAsBinaryString(Blob|File) – свойство result содержит данные о файле или его фрагменте в виде строки бинарного кода. Каждый байт представлен целым числом от 0 до 255.
2. FileReader.readAsText(Blob|File, opt\_encoding) – свойство result содержит данные о файле или его фрагменте в виде текстовой строки. По умолчанию используется кодировка UTF-8. Чтобы задать другой формат, используйте необязательный параметр кодировки opt\_encoding.
3. FileReader.readAsDataURL(Blob|File) – свойство result содержит данные о файле или его фрагменте в виде схемы data:URL.
4. FileReader.readAsArrayBuffer(Blob|File) – свойство result содержит данные о файле или его фрагменте в виде объекта ArrayBuffer.

Если для объекта FileReader вызывается один из этих методов, то ход его обработки можно отслеживать с помощью атрибутов onloadstart, onprogress, onload, onabort, onerror и onloadend.

В примере ниже из выделенных пользователем файлов выбираются картинки, вызывается функция reader.readAsDataURL() и выводится уменьшенное изображение путем установки для атрибута src значения, представляющего URL данных.

<input type="file" id="files" name="files[]" multiple />  
<output id="list"></output>  
  
<script>  
 **function** handleFileSelect(evt) {  
 **var** files = evt.target.files; // FileList object  
  
 // Loop through the FileList and render image files as thumbnails.  
 **for** (**var** i = 0, f; f = files[i]; i++) {  
  
 // Only process image files.  
 **if** (!f.type.match('image.\*')) {  
 **continue**;  
 }  
  
 **var** reader = **new** FileReader();  
  
 // Closure to capture the file information.  
 reader.onload = (**function**(theFile) {  
 **return function**(e) {  
 // Render thumbnail.  
 **var** span = document.createElement('span');  
 span.innerHTML = ['<img class="thumb" src="', e.target.result,  
 '" title="', escape(theFile.name), '"/>'].join('');  
 document.getElementById('list').insertBefore(span, **null**);  
 };  
 })(f);  
  
 // Read in the image file as a data URL.  
 reader.readAsDataURL(f);  
 }  
 }  
  
 document.getElementById('files').addEventListener('change', handleFileSelect, **false**);  
</script>

## Разделение файла на фрагменты

В некоторых случаях считывание всего файла в память не является оптимальным вариантом. Предположим, вы решили создать инструмент для асинхронной загрузки файлов. Чтобы увеличить ее скорость, можно разделить файл на фрагменты одинаковой величины, которые будут считываться и отправляться по отдельности. На сервере фрагменты будут снова объединяться в нужном порядке.

Интерфейс File поддерживает метод slice для разбиения файла на фрагменты. Первым аргументом этого метода является начальный байт файла, вторым – его последний байт, а третьим – необязательная строка с описанием типа содержания. Семантика этого метода недавно была изменена, и теперь необходимо использовать префикс браузера:

**if** (file.webkitSlice) {  
 **var** blob = file.webkitSlice(startingByte, endindByte);  
} **else if** (file.mozSlice) {  
 **var** blob = file.mozSlice(startingByte, endindByte);  
}  
reader.readAsBinaryString(blob);

# Источники

http://www.html5rocks.com/ru/tutorials/file/dndfiles/