

Раздел

[Бинарные данные и файлы](#)


Навигация по уроку

FileReader

Итого

Комментарии

Поделиться

[Редактировать на GitHub](#) → [Бинарные данные и файлы](#) 21-го августа 2019

File и FileReader

Объект **File** наследуется от объекта **Blob** и обладает возможностями по взаимодействию с файловой системой.

Есть два способа его получить.

Во-первых, есть конструктор, похожий на **Blob** :

```
1 new File(fileParts, fileName, [options])
```

- **fileParts** – массив значений **Blob** / **BufferSource** / строки.
- **fileName** – имя файла, строка.
- **options** – необязательный объект со свойством:
 - **lastModified** – дата последнего изменения в формате таймстемп (целое число).

Во-вторых, чаще всего мы получаем файл из `<input type="file">` или через перетаскивание с помощью мыши, или из других интерфейсов браузера. В этом случае файл получает эту информацию из ОС.

Так как **File** наследует от **Blob**, у объектов **File** есть те же свойства плюс:

- **name** – имя файла,
- **lastModified** – таймстемп для даты последнего изменения.

В этом примере мы получаем объект **File** из `<input type="file">`:

```
1 <input type="file" onchange="showFile(this)">
2
3 <script>
4 function showFile(input) {
5   let file = input.files[0];
6
7   alert(`File name: ${file.name}`); // например, my.png
8   alert(`Last modified: ${file.lastModified}`); // напр
9 }
10 </script>
```

На заметку:

Через `<input>` можно выбрать несколько файлов, поэтому `input.files` – псевдомассив выбранных файлов. Здесь у нас только один файл, поэтому мы просто берём `input.files[0]`.

FileReader

FileReader объект, цель которого читать данные из **Blob** (и, следовательно, из **File** тоже).

Данные передаются при помощи событий, так как чтение с диска может занять время.

Конструктор:

```
1 let reader = new FileReader(); // без аргументов
```

Основные методы:

Раздел

Бинарные данные и файлы

Навигация по уроку

FileReader

Итого

Комментарии

Поделиться



Редактировать на GitHub



- `readAsArrayBuffer(blob)` – считать данные как `ArrayBuffer`
- `readAsText(blob, [encoding])` – считать данные как строку (кодировка по умолчанию: `utf-8`)
- `readAsDataURL(blob)` – считать данные как base64-кодированный URL.
- `abort()` – отменить операцию.

Выбор метода для чтения зависит от того, какой формат мы предпочитаем, как мы хотим далее использовать данные.

- `readAsArrayBuffer` – для бинарных файлов, для низкоуровневой побайтовой работы с бинарными данными. Для высокоуровневых операций у `File` есть свои методы, унаследованные от `Blob`, например, `slice`, мы можем вызвать их напрямую.
- `readAsText` – для текстовых файлов, когда мы хотим получить строку.
- `readAsDataURL` – когда мы хотим использовать данные в `src` для `img` или другого тега. Есть альтернатива – можно не читать файл, а вызвать `URL.createObjectURL(file)`, детали в главе [Blob](#).

В процессе чтения происходят следующие события:

- `loadstart` – чтение начато.
- `progress` – срабатывает во время чтения данных.
- `load` – нет ошибок, чтение окончено.
- `abort` – вызван `abort()`.
- `error` – произошла ошибка.
- `loadend` – чтение завершено (успешно или нет).

Когда чтение закончено, мы сможем получить доступ к его результату следующим образом:

- `reader.result` результат чтения (если оно успешно)
- `reader.error` объект ошибки (при неудаче).



Наиболее часто используемые события – это, конечно же, `load` и `error`.



Вот пример чтения файла:

```
1 <input type="file" onchange="readFile(this)">
2
3 <script>
4 function readFile(input) {
5   let file = input.files[0];
6
7   let reader = new FileReader();
8
9   reader.readAsText(file);
10
11   reader.onload = function() {
12     console.log(reader.result);
13   };
14
15   reader.onerror = function() {
16     console.log(reader.error);
17   };
18
19 }
20 </script>
```



Раздел

[Бинарные данные и файлы](#)

Навигация по уроку

FileReader

Итого

Комментарии

Поделиться



[Редактировать на GitHub](#)



FileReader для Blob

Как упоминалось в главе [Blob](#), `FileReader` работает для любых объектов `Blob`, а не только для файлов.

Поэтому мы можем использовать его для преобразования `Blob` в другой формат:

- `readAsArrayBuffer(blob)` – в `ArrayBuffer`,
- `readAsText(blob, [encoding])` – в строку (альтернатива `TextDecoder`),
- `readAsDataURL(blob)` – в формат base64-кодированного URL.

Для Web Worker также доступен FileReaderSync

Для веб-воркеров доступен синхронный вариант `FileReader`, именуемый [FileReaderSync](#).

Его методы считывания `read*` не генерируют события, а возвращают результат, как это делают обычные функции.

Но это только внутри веб-воркера, поскольку задержки в синхронных вызовах, которые возможны при чтении из файла, в веб-воркерах менее важны. Они не влияют на страницу.

Итого

`File` объекты наследуют от `Blob`.

Помимо методов и свойств `Blob`, объекты `File` также имеют свойства `name` и `lastModified` плюс внутреннюю возможность чтения из файловой системы. Обычно мы получаем объекты `File` из пользовательского ввода, например, через `<input>` или перетаскиванием с помощью мыши, в событии `dragend`.

Объекты `FileReader` могут читать из файла или `Blob` в одном из трёх форматов:

- Строка (`readAsText`).
- `ArrayBuffer` (`readAsArrayBuffer`).
- URL в формате base64 (`readAsDataURL`).

Однако, во многих случаях нам не нужно читать содержимое файла. Как и в случае с `Blob`, мы можем создать короткий URL с помощью `URL.createObjectURL(file)` и использовать его в теге `<a>` или ``. Таким образом, файл может быть загружен или показан в виде изображения, как часть `canvas` и т.д.

А если мы собираемся отправить `File` по сети, то это также легко, поскольку в сетевые методы, такие как `XMLHttpRequest` или `fetch`, встроена возможность отсылки `File`.

Проводим [курсы по JavaScript и фреймворкам](#).



Комментарии

перед тем как писать...