

# FILE API



МИХАИЛ КУЗНЕЦОВ / ING



# МИХАИЛ КУЗНЕЦОВ

Developer ING



[@mkuznetcov](https://www.telegram.me/mkuznetcov)

# ПЛАН ЗАНЯТИЯ

- Выбор файла с компьютера
- FileList
- File
- Перенос файлов в окно браузера или event.dataTransfer
- URL.createObjectURL() / URL.revokeObjectURL()
- FileReader
- Отправка файлов на удаленный сервер



# **ВЫБОР ФАЙЛА С КОМПЬЮТЕРА**

---

# HTML-ТЕГ

```
<input type="file" id="fileInput">
```

Этот элемент ввода отображает кнопку, после нажатия на которую появляется окно выбора файла.

- Атрибут `multiple` разрешает выбор нескольких файлов.
- Атрибут `accept` ограничивает типы файлов, которые можно выбрать.

# АТРИБУТ АССЕРТ

Возможно задавать ограничения несколькими способами:

- Один тип - `image/jpeg` - для выбора доступны только jpeg файлы.
- Несколько типов (через запятую) - `image/jpeg, image/png`.
- Маска - `image/*` - доступны изображения любых типов.
- Комбинация всего, что было выше:

`text/plain, image/*, video/avi, audio/mp3`

# ОТСЛЕЖИВАЕМ МОМЕНТ ВЫБОРА

Чтобы отследить момент выбора файлов, мы можем добавить обработчик события `change` на этот элемент ввода:

```
1 document.querySelector('#fileInput')
2   .addEventListener('change', onSelectFiles);
3
4 function onSelectFiles(event) {
5     const files = event.currentTarget.files;
6
7     console.log(files);
8 }
```

Функция `onSelectFiles` будет вызвана, когда меняются выбранные файлы. Если один и тот же файл будет выбран дважды, то функция будет вызвана лишь единожды.

# currentTarget

`currentTarget` — объект, представляющий наш элемент ввода, у него есть свойство `files`, по которому мы можем получить файлы, выбранные пользователем.

Но что представляет собой это свойство?





# FileList



*`files` – не обычный массив и не имеет методов массива (таких, как `forEach`, `map` и других). С массивом его роднит только свойство `length`, отвечающее за количество выбранных файлов.*



# СПЕЦИАЛЬНЫЙ ТИП

`files` имеет тип `FileList`. Это специальный тип, который используется для работы с пользовательскими файлами. В `FileList` определен `Symbol.iterator`, что позволяет нам:

1. Использовать оператор `for-of`.
2. Использовать функцию `Array.from` для преобразования в массив.

# ПРОСМАТРИВАЕМ ФАЙЛЫ

Создадим массив файлов из свойства `event.target.files`. Затем выведем в консоль информацию о каждом файле:

```
1 function onSelectFiles(event) {  
2   const files = Array.from(event.target.files);  
3  
4   files.forEach(console.log);  
5 }
```



File

## ОБЪЕКТ ТИПА `File`

Каждый элемент объекта `FileList` является объектом типа `File`. Он не имеет каких-либо специальных методов, но имеет полезные свойства:

- `name` – название файла;
- `size` – размер файла в байтах;
- `type` – MIME-тип файла, позволяет понять, что это за файл – изображение, текст, аудиозапись и так далее.

## ВЫВЕДЕМ ИНФОРМАЦИЮ В DOM

```
<div id="filesInfo"></div>
```

```
1 function onSelectFiles(event) {  
2   const files = Array.from(event.target.files);  
3  
4   updateFilesInfo(files);  
5 }
```

# ФУНКЦИЯ С ИНФОРМАЦИЕЙ

```
1 function updateFileInfo(files) {
2   const fileInfo = document.querySelector('#filesInfo');
3   const fragment = document.createDocumentFragment();
4   fileInfo.innerHTML = '';
5
6   files.forEach(file => {
7     const fileDescription = `
8       Название: ${file.name},
9       Размер: ${file.size},
10      Тип: ${file.type}
11    `;
12     const p = document.createElement('p');
13
14     p.innerText = fileDescription;
15     fragment.appendChild(p);
16   });
17
18   fileInfo.appendChild(fragment);
19 }
```



# ПРИНЦИП РАБОТЫ

Функция `updateFilesInfo` находит `div`, в котором мы будем хранить информацию о файлах. Она очищает его содержимое, на случай, если пользователь захочет выбрать файлы повторно, и проходит по файлам, добавляя новые узлы `p` с информацией о файле.

[Live Demo](#)

---

**event.dataTransfer**

---

# DRAG & DROP

Окно выбора файлов выполняет свою задачу, но иногда хочется большей интерактивности. Например, хочется перенести файлы прямо в окно браузера, вместо того, чтобы путаться в директориях.

В данной ситуации нам поможет интерфейс Drag & Drop. Событие `drop` происходит, когда один или несколько элементов переносятся в специальную зону, которая ожидает этого переноса.

# ВЫДЕЛЯЕМ МЕСТО

Обновим разметку, чтобы выделить место, в которое ожидается перенос файлов:

```
1 <div id="filesInfo">
2   Перенесите файлы сюда!
3 </div>
```

Добавим стилей:

```
1 #filesInfo {
2   border: 2px dashed gray;
3   padding: 8px 16px;
4 }
```

# ОБРАБОТЧИК СОБЫТИЙ

Так же добавим `EventListener` на события `dragover` и `drop`:

```
1  const fileInfo = document.querySelector('#filesInfo');
2
3  fileInfo.addEventListener('drop', onFilesDrop);
4  fileInfo.addEventListener('dragover', event => event
5    .preventDefault());
6
7  function onFilesDrop(event) {
8    event.preventDefault();
9
10   const files = Array.from(event.dataTransfer.files);
11
12   updateFilesInfo(files);
13 }
```

# ПРЕДОТВРАЩАЕМ ОТКРЫТИЕ

`EventListener` на событие `dragover` дает понять браузеру, что элемент ожидает переноса элементов, а на событие `drop` — поймать момент «сбрасывания» файлов.

Нужно учесть, что при переносе файлов браузер может попытаться открыть файл для чтения в той же вкладке.

Чтобы помешать ему, необходимо сделать `event.preventDefault()`. `event.dataTransfer` — это специальный объект, содержащий данные, которые переносятся во время операций Drag & Drop. Он может содержать один или более элементов разных типов. В этом объекте нас интересует свойство `files`, знакомого нам типа `FileList`.

[Live Demo](#)

---

`URL.createObjectURL()` и

`URL.revokeObjectURL()`

---

# ПРЕВЬЮ ИЗОБРАЖЕНИЙ

Сделаем работу с файлами еще чуть более интерактивной — будем показывать превью изображений.

Сначала нам нужно понять, что обрабатываемый файл — изображение. Затем нам нужно создать элемент `img` и каким-то образом установить в значение атрибута `src` ссылку на нужное нам изображение.



## URL.createObjectURL()

Этот метод принимает один аргумент – `File` или `Blob`, а возвращает URL.

Браузер будет хранить в памяти `File` или `Blob`, на который указывает возвращенная url до тех пор, пока не будет вызван `URL.revokeObjectURL()`

`URL.revokeObjectURL()` освобождает память.

# ВАЖНЫЕ ПРАВИЛА

При работе с `createObjectURL` важно помнить два правила:

1. `createObjectURL` всегда создает новый URL, даже если вызывается дважды для одного и того же файла.
2. Всегда используйте `revokeObjectURL`, иначе случится утечка памяти.

## ОБНОВЛЯЕМ ФУНКЦИЮ

Перепишем функцию `updateFileInfo` с использованием `createObjectURL` \ `revokeObjectURL`:

```
1 function updateFileInfo(files) {
2     const imageTypeRegExp = /^image\\/;
3     const fileInfo = document
4         .querySelector('#fileInfo');
5     const fragment = document.createDocumentFragment();
6     fileInfo.innerHTML = '';
7
8     // Информация о файле
9
10    fileInfo.appendChild(fragment);
11 }
```

# ИНФОРМАЦИЯ О ФАЙЛЕ

```
1 files.forEach(file => {
2     const fileDescription = `
3         Название: ${file.name},
4         Размер: ${file.size},
5         Тип: ${file.type}
6     `;
7     const fileContent = document.createElement('div');
8     const p = document.createElement('p');
9
10    p.innerText = fileDescription;
11
12    // Выводим превью для изображений
13
14    fileContent.appendChild(p);
15    fragment.appendChild(fileContent);
16 });
```

## ВЫВОДИМ ПРЕВЬЮ ДЛЯ ИЗОБРАЖЕНИЙ

```
1  if (imageTypeRegExp.test(file.type)) {
2      const img = document.createElement('img');
3
4      img.width = 300;
5      img.height = 300;
6      img.src = URL.createObjectURL(file);
7      img.addEventListener('load', event => {
8          URL.revokeObjectURL(event.target.src);
9      });
10
11     fileContent.appendChild(img);
12 }
```

[Live Demo](#)

## СМОТРИМ НА ТИП ФАЙЛА

Мы создаем `regexr`, чтобы по MIME-типу файла определить, возможно ли получить его превью. Если превью получить можно, то мы используем `URL.createObjectURL`, чтобы получить ссылку на изображение.



# FileReader

# ТЕКСТОВЫЙ РЕДАКТОР

Предположим, что нам нужно реализовать функционал простого текстового редактора. Пользователь выбирает файл с компьютера и мы должны отобразить его содержимое.

Создадим элемент ввода, позволяющий выбрать только текстовый файл, а также создадим многострочное текстовое поле:

```
<input type="file" accept="text/plain">  
<textarea id="fileContent" rows="20" cols="80">  
</textarea>
```



# ЧТЕНИЕ ФАЙЛА

Теперь прочитаем содержимое файла и запишем его в `textarea`:

```
1 document.querySelector('input[type=file]')
2   .addEventListener('change', handleFileChange)
3
4 function handleFileChange(event) {
5   setTextareaContent(event.currentTarget.files[0]);
6 }
7
8 function setTextareaContent(file) {
9   const fileContent = document
10     .querySelector('#fileContent');
11   const reader = new FileReader();
12
13   fileContent.value = '';
14
15   reader.addEventListener('load', event => {
16     fileContent.value = event.target.result;
17   });
18
19   reader.readAsText(file);
20 }
```

# ПРОЦЕСС ЧТЕНИЯ И ОТОБРАЖЕНИЯ

Метод `readAsText` используется, чтобы прочитать контент текстового файла. Он принимает сам файл и кодировку, в которой хранит результат чтения. По умолчанию используется кодировка **UTF-8**. Когда файл будет прочтен, мы записываем в `textarea` результат чтения — текст.


# МЕТОДЫ `FileReader`

`FileReader` содержит и другие методы:

- Метод `readAsArrayBuffer` читает переданный файл и сохраняет результат чтения в объект типа `ArrayBuffer`. Это может быть полезно, чтобы в дальнейшем типизировать данные и обрабатывать их определенным образом.
- Метод `readAsDataURL` используется для чтения переданного файла и использования результата чтения в качестве URL (например, для элемента `audio`).

# FILEREADER + PROMISE

```
1  function readTextFile(blob) {
2      return new Promise((resolve, reject) => {
3          const reader = new FileReader();
4
5          reader.addEventListener('load', (event) => {
6              resolve(event.target.result);
7          });
8          reader.addEventListener('error', reject);
9
10         reader.readAsText(blob);
11     });
12 }
```



# ОТПРАВКА ФАЙЛОВ НА УДАЛЕННЫЙ СЕРВЕР

# ОТПРАВКА НЕПОСРЕДСТВЕННО FILE ИЛИ BLOB

Для отправки файлов на сервер будем использовать XMLHttpRequest.

```
1 function sendFile(file) {  
2   const xhr = new XMLHttpRequest();  
3  
4   xhr.open('POST', '/api/files');  
5   xhr.addEventListener('load', () => {  
6     if (xhr.status === 200) {  
7       console.log(`Файл ${file.name} сохранен.`);  
8     }  
9   });  
10  xhr.send(file);  
11 }
```

## ВАРИАНТ ОТПРАВКИ С FormData

Если кроме файла необходимо отправить его название и/или еще какую-то дополнительную информацию, то больше подойдет объект `FormData`:

```
1 function sendFile(file) {  
2   const formData = new FormData();  
3   formData.append('file', file);  
4  
5   const xhr = new XMLHttpRequest();  
6   xhr.open('POST', '/api/files');  
7   xhr.addEventListener('load', () => {  
8     if (xhr.status === 200){  
9       console.log(`Файл ${file.name} сохранен.`);  
10    }  
11  });  
12  
13  xhr.send(formData);  
14 }
```

# ЗАГРУЗКА ФАЙЛОВ НА СЕРВЕР

`FormData` позволяет создать точно такую же форму, которая создается при отправке формы через HTML (используя `<button type="submit">`). Метод `append` записывает переданные данные по переданному ключу.


```
1  -----WebKitFormBoundaryeAgNm7qDauCFdp6m
2      Content-Disposition: form-data; name="file"; filename="textfile.txt"
3      Content-Type: text/plain
4
5
6  -----WebKitFormBoundaryeAgNm7qDauCFdp6m--
```





# ИТОГИ

# ВЫБОР ФАЙЛА

-  — стандартный элемент выбора файлов.
- Атрибут `multiple` разрешает загрузку нескольких файлов.
- В атрибуте `ассерт` указывается, какой тип файлов может быть загружен.
- Есть событие `change`.

# ОБРАБОТКА ФАЙЛОВ

- Через свойство `files` у объекта `currentTarget` можно получить доступ к загруженным файлам.
- `files` – неполноценный массив. Есть только свойство `length`.
- У `files` специальный тип `FileList`.
- Доступны `for-of` и `Array.from`.
- У каждого элемента объекта `FileList` тип `File` со свойствами:
  - `name` – имя файла;
  - `size` – размер в байтах;
  - `type` – MIME-тип файла.

# DRAG & DROP

- Событие `dragover` указывает браузеру, что ожидается загрузка файлов путем переноса.
- Событие `drop` срабатывает, когда файлы переносятся в определенную область окна браузера.
- Не забываем о `.preventDefault()`, чтобы браузер не открывал загружаемые файлы.

## ПОЛУЧАЕМ URL И ОЧИЩАЕМ ПАМЯТЬ

- `URL.createObjectURL()` – статический метод объекта `URL`.
- Принимает один аргумент и возвращает URL.
- Создает новый URL, даже если файл один и тот же.
- Пока жив документ – жив и URL.
- `URL.revokeObjectURL()` очищает память.
- Очищаем память всегда, чтобы избежать утечки памяти.

# ЧТЕНИЕ ФАЙЛОВ

- Для чтения содержимого файла используем объект `FileReader`.
- Метод `readAsText` читает содержимое текстового файла.
- По умолчанию используется кодировка UTF-8.
- Метод `readAsArrayBuffer` для чтения бинарных файлов.
- `readAsDataURL` читает файл в data url.



## ОТПРАВКА ФАЙЛОВ НА СЕРВЕР

- Файлы отправляются на сервер с помощью XMLHttpRequest
- Можно отправлять непосредственно объекты File или Blob на сервер
- Для отправки содержимого файла и мета информации удобно использовать FormData



**НЕТОЛОГИЯ**  
университет интернет-профессий

**Задавайте вопросы и напишите отзыв о лекции!**

**МИХАИЛ КУЗНЕЦОВ**



[@mkuznetcov](https://www.instagram.com/mkuznetcov)