НЕТОЛОГИЯ

DRAG AND DROP. FILES

АЛЕКСАНДР ШЛЕЙКО

АЛЕКСАНДР ШЛЕЙКО

Программист в Яндекс

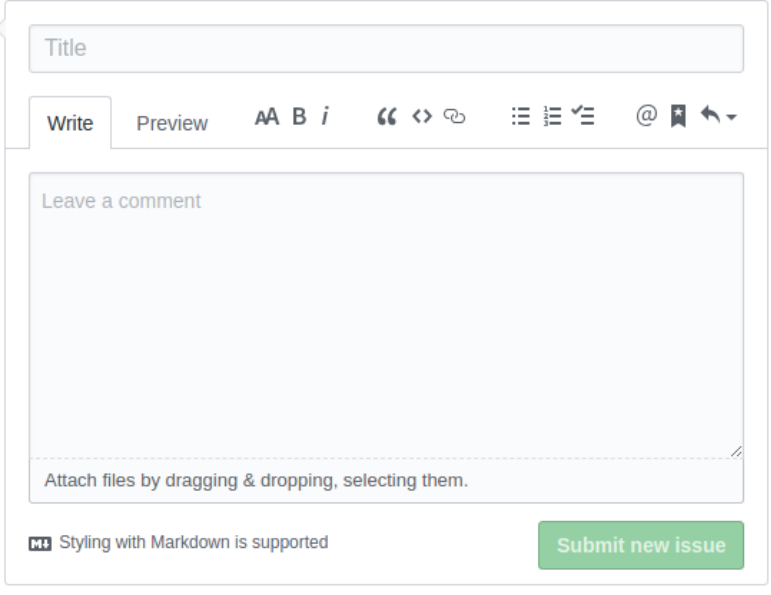
a.shleyko@yandex.ru vk.com/shleiko @dustyo_O

ПЛАН ЗАНЯТИЯ

1. `input type=file`
2. Генерация событий
3. File API
4. Drag and Drop

ЗАДАЧА

Перед нами поставили следующую задачу: реализовать виджет для загрузки картинок на наш сервис, так, чтобы загрузка была осуществима как выбором файла, так и методом Drag and Drop:



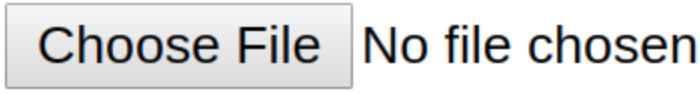
Скриншот с сайта GitHub.com
Но при этом при загрузке ещё должна отображаться превьюшка нашей загружаемой картинки.

`input type=file`

`input type=file`

Начнём с выбора файла. Для этого как вы знаете, у нас есть `<input type="file">`.

Выглядит он вот так:



Совсем не так, как хотелось бы, и, в большинстве случаев, совсем не так, как мы можем себе позволить в рамках проекта.

АТРИБУТЫ

Допустимыми являются следующие атрибуты:

- `accept` - MIME-Type или расширения файлов (`.pdf` , `.png`) через запятую, допустимы также `audio/*` , `video/*` , `image/*` ;
- `multiple` - возможность выбора нескольких файлов.

Важно: помните о том, что фронтенд не обеспечивает безопасность и то, что вы указываете допустимые типы файлов, не является гарантией того, что на сервер будут отправляться именно они.

СТИЛИЗАЦИЯ

Возможности стилизации практически отсутствуют, поэтому придётся идти обходными путями.

СКРЫТИЕ ЭЛЕМЕНТА

Чаще всего поле для ввода скрывают тем или иным способом.

Первое, что обычно приходит на ум, это:

- `display: none`;
- `visibility: false`.

Но это опасный путь, который приведётк тому, что элемент не будет участвовать в отправке формы (средствами HTML) и будет исключён из активации по Tab.

СКРЫТИЕ ЭЛЕМЕНТА

Другие варианты: позиционирование элемента таким образом, чтобы другой элемент скрывал поле ввода, либо вывод поля ввода за границы видимой области.

Рассмотрим оба варианта

ПЕРЕКРЫТИЕ ДРУГИМ ЭЛЕМЕНТОМ

```html
<div class="file-container">
  <input data-id="file" class="overlapped" type="file" accept="image/*">
  <span data-id="overlap" class="overlap">Choose file</span>
</div>
```

```css
.file-container {
  width: 200px; height: 20px;
  position: relative;
}
.overlapped {
  position: absolute;
  top: 0; left: 0; bottom: 0; right: 0; margin: 0;
  opacity: 0.1;
  width: 100%;
}
.overlap {
  position: absolute;
  top: 0; left: 0; bottom: 0; right: 0; margin: 0;
  width: 100%;
  background: #eee;
}
```

СОБЫТИЯ УКАЗАТЕЛЯ

Но при таком подходе мы сталкиваемся с проблемой: события указателя, которые возникают на элементе span не передаются нижнему элементу.

Что можно с этим сделать?

ГЕНЕРАЦИЯ СОБЫТИЙ

POINTER-EVENTS

CSS предлагает нам свойство pointer-events, которое позволяет «пропускать» события указателя через элемент нижележащим в компоновке элементам.

```
1 .overlap {
2   pointer-events: none;
3 }
```

DISPATCH EVENT

Второй вариант, основанный только на JS, позволяет нам отправлять собственные события элементам.

В частности, что мы можем сделать: мы можем подписаться на события клика на элементе `span` и отправить событие клика элементу `input`.

EventTarget

Вспоминаем интерфейс `EventTarget`:

```
interface EventTarget {
  void addEventListener(DOMString type, EventListener? callback,
       optional (AddEventListenerOptions or boolean) options);
  void removeEventListener(DOMString type, EventListener? callback,
       optional (EventListenerOptions or boolean) options);
  boolean dispatchEvent(Event event);
};
```

Dispatch Event

```
const fileEl = document.querySelector('[data-id=file]');
const overlapEl = document.querySelector('[data-id=overlap]');

overlapEl.addEventListener('click', (evt) => {
  fileEl.dispatchEvent(evt);
});
```

Но такой код работать не будет, т.к. уже однажды «отправленное» событие нельзя взять и отправить снова.

А что же можно? Можно создать своё событие.

EVENT CREATION

```
1 overlapEl.addEventListener('click', (evt) => {
2   fileEl.dispatchEvent(new Event('click', <options>));
3 });
```

Где `<options>` - объект с полями:

- `cancelable`;
- `bubbles`.

Но такой код тоже не работает, почему?

Потому что `input type="file"` требует для активации поведения по умолчанию именно `MouseEvent`, а не `Event`.

EVENT CREATION

А что если мы хотим сгененировать собственное событие (не то, которое нам предлагают создатели DOM Events)?

1. Мы можем использовать любую строку в конструкторе события (например, `app-event` вместо `click`);
2. Мы можем воспользоваться конструктором CustomEvent(`<type>`, `<init>`), где `<init>` может включать в себя поле `detail` для наших собственных, кастомных свойств.

DISPATCH

Что же получается, мы можем программно сгенерировать событие и заставить открываться тот же самый диалог выбора файлов в любой момент?

Попробуем следующий код:

```
// вне eventListener'a
fileEl.addEventListener('click', () => {
  console.log('clicked');
});
fileEl.dispatchEvent(new MouseEvent('click'));
```

`EventListener` для `click` срабатывает, но при этом окно выбора файлов не открывается.

`isTrusted`

Для событий, генерируемых браузером в результате взаимодействия с пользователем и других активностей (не программных), выставляется в `true` флаг `isTrusted`. События с таким флагом приводят к активации `default action` (если он определён и не вызван `preventDefault`).

Для событий, генерируемых программно, флаг `isTrusted` выставляется в `false` и для всех событий кроме `click` диспатчинг не приводит к активации поведения по умолчанию (`default action`).

ACTIVATION TRIGGERS & BEHAVIOR

Для событий с `isTrusted=false` в большинстве случаев, мы можем полагаться лишь на алгоритм, описанный в секции `activation`: https://html.spec.whatwg.org/#triggered-by-user-activation

Второй и третий пункт в котором гласят: реагируя на события определённого типа (`click` в их числе) мы можем в `EventListener`'е активировать поведение по умолчанию в течение времени определённого браузером (т.е. сделать `setTimeout` на `dispatch` через 10 секунд не получится).

ИТОГИ

Таким образом, перехватывая сгенерированное пользователем событие с `isTrusted` мы можем вызвать окно выбора файлов выполнив `dispatchEvent` или вызвав метод `click` на `<input type="file">`.

RESEARCH

Кстати, можете подумать, почему в песочнице JSBin вот такой код будет работать (не в `EventListenere`, а просто в скрипте):

```
document.querySelector('[data-id=file]').dispatchEvent(new MouseEvent('click'));
```

А вот такой уже нет:

```
etTimeout(() => {
  document.querySelector('[data-id=file]').dispatchEvent(new MouseEvent('click'));
}, 10000);
```

АБСОЛЮТНОЕ ПОЗИЦИОНИРОВАНИЕ

При абсолютном позиционировании элемент чаще всего выносится за пределы видимой области браузера (отрицательный `top`).

Обработка при этом не изменится, нужно будет также `dispatch`ить событие в обработчике с `isTrusted=true`.

PREVIEW

Осталось нам научиться выводить `preview` картинки. Разместим для этого тег `img`:

```
<img data-id="preview">
```

```
const previewEl = document.querySelector('[data-id=preview]');
```

CHANGE EVENT

При выборе пользователем файла генерируется событие `change` с одной важной особенностью: если пользователь выбирает один и тот же файл два раза подряд, то повторно оно не генерируется.

ВЫБРАННЫЕ ФАЙЛЫ

Выбранные файлы (или файл в условиях отсутствия атрибута `multiple`) будет храниться в поле `files`:

```
fileEl.addEventListener('change', (evt) => {
  const files = Array.from(evt.currentTarget.files);
});
```

FILE API

FILE API

Браузер предоставляет нам FileAPI, в котором определены ключевые интерфейсы:

- `FileList` - индексируемый список файлов
- `File` - наследник интерфейса `Blob`, добавляющий атрибут name и `lastModified`:

```
interface File : Blob {
  readonly attribute DOMString name;
  readonly attribute long long lastModified;
};
```

FILE API

— `Blob` - интерфейс, описывающий последовательность байт с атрибутами `size` и `type` (Mime-Type):

```
interface Blob {
  readonly attribute unsigned long long size;
  readonly attribute DOMString type;

  // slice Blob into byte-ranged chunks
  Blob slice(optional [Clamp] long long start,
             optional [Clamp] long long end,
             optional DOMString contentType);
};
```

BLOB URL

Хорошо, мы можем получить доступ к мета-данным файла, но как же получить доступ к контенту?

У нас есть возможность генерировать URL, которые мы можем использовать в качестве атрибута `src`:

- `URL.createObjectURL(Blob blob)`;
- `URL.revokeObjectURL(DOMString url)`.

```
fileEl.addEventListener('change', (evt) => {
   const files = Array.from(evt.currentTarget.files);
  previewEl.src = URL.createObjectURL(files[0]);
  previewEl.addEventListener('load', () => {
    URL.revokeObjectURL(previewEl.src);
  });
});
```

BLOB URL

Blob URL формируется в формате: `blob:<scheme>://<host>:<port>/<uuid>`. Его можно использовать в ссылках, в том числе в `location.href=<blob-url>`, но ровно до тех пор, пока мы не вызвали `revokeObjectURL`.

`revokeObjectURL`

Q: Зачем вызывать `revokeObjectURL`?

A: Пока он не будет вызван, объект `Blob` так и будет висеть в памяти.

Q: Почему он вызывается в `load`?

A: После того, как он загружен в изображение, уже нет в нём необходимости (если мы не собираемся его повторно использовать).

ШАГ В СТОРОНУ

Blob URL'ы можно использовать также для скачивания объекта. Для этого достаточно создать ссылку с атрибутом `download=<name>`, `rel=noopener` и кликнуть по ней:

```
fileEl.addEventListener('change', (evt) => {
  const files = Array.from(evt.currentTarget.files);

  const file = files[0];
  const a = document.createElement('a');
  a.download = file.name;
  a.href = URL.createObjectURL(file);
  a.rel = 'noopener';
  setTimeout(() => URL.revokeObjectURL(a.href), 60000);
  setTimeout(() => a.dispatchEvent(new MouseEvent('click')));
});
```

Это «урезанный» вариант, не учитывающий многих особенностей. Полный вариант приведён в библиотеке `FileSaver.js`.

CONTENT

Это хорошо, что мы можем создавать ссылки, но как получить сам контент?

Допустим, мы хотим прочитать содержимое текстового файла и отобразить его в `textarea`?

`FileReader`

Для этого нам предоставляется объект FileReader, который и позволяет читать файлы:

- `readAsDataURL(Blob blob)`;
- `readAsText(Blob blob)`;
- `readAsArrayBuffer(Blob blob)`;
- `readAsBinaryString(Blob blob)`.

FileReader

```
function readFile(file) {
  return new Promise((resolve, reject) => {
    const reader = new FileReader();
    reader.addEventListener('load', (evt) => {
      resolve(evt.target.result);
    });
    reader.addEventListener('error', (evt) => {
      reject(evt.target.error);
    });

    reader.readAsArrayBuffer(file);
  });
}
```

DataURL

Специальный формат в виде `data:<mediatype>;<base64>,<data>` позволяющий использовать бинарные данные в виде строки. Строка закодирова в base64 (функции `atob`, `btoa`).

Поскольку это валидный URL, мы можем его использовать так же, как и Blob URL (для отображения и скачивания).

`Text`

Здесь нет ничего особенного, достаточно установить в `textarea.value` прочитанную строку.

`ArrayBuffer`

С `ArrayBuffer` вы уже знакомы, стоит лишь отметить, что конструктор `Blob` позволяет использовать `ArrayBuffer` (`DataView`, `*Array`) в конструкторе.

DRAG AND DROP

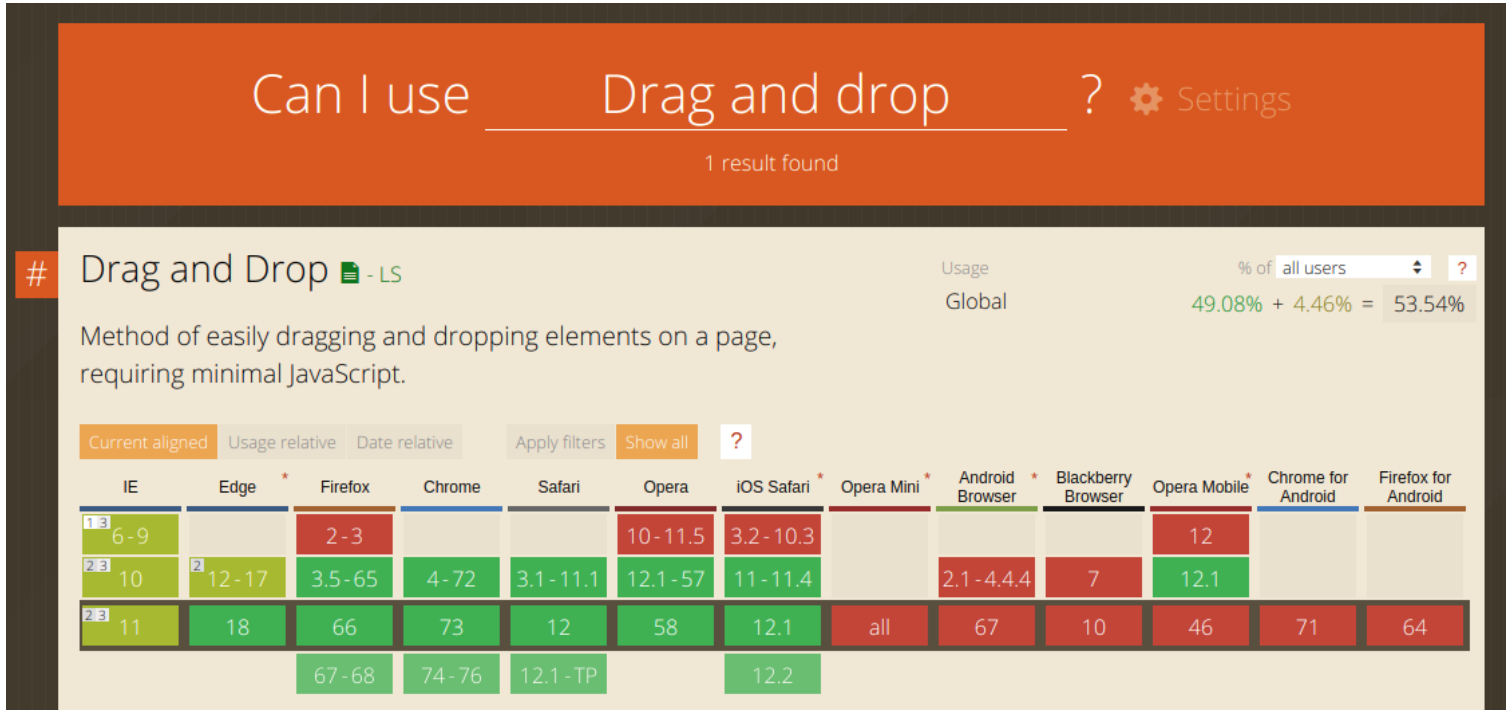
DRAG AND DROP

Drag and Drop в Web традиционно делится на две реализации:

- имитация с помощью событий мыши/touch'а;
- Drag and Drop API (далее - DnD).

СОВМЕСТИМОСТЬ

1. Мобильные браузеры плохо поддерживают DnD
2. DnD обладает рядом ограничений
3. Но перенос файлов с рабочего стола в браузер нужно реализовать именно с помощью DnD



MOUSE & TOUCH EVENTS

Элемент устанавливается `child` для `body` с абсолютным позиционированием:

- `mousedown` / `touchstart` - фиксация начала перемещения элемента;
- `mousemove` / `touchmove` - установка `top` / `left` элемента;
- `mouseup` / `touchend` - принятие решение о итоговом местоположении.

СПИСОК ЭЛЕМЕНТОВ

Отвлечёмся от нашей задачи с файлами и попробуем сделать список элементов, которые можно перемещать (изменять порядок) с помощью Drag and Drop.

Разметка:

```
<ul class="items">
  <li class="items-item">First Item</li>
  <li class="items-item">Second Item</li>
  <li class="items-item">Third Item</li>
  <li class="items-item">Fourth Item</li>
</ul>
```

СТИЛИ

```
.items {
  list-style-type: none;
  padding-left: 0;
}
.items-item {
  border: 1px solid #999;
  padding: 10px;
  margin-bottom: -1px;;
  cursor: pointer;
}
.items-item:first-child {
  border-top-left-radius: 10px;
  border-top-right-radius: 10px;
}
.items-item:last-child {
border-bottom-left-radius: 10px;
  border-bottom-right-radius: 10px;
}
.items-item.dragged {
  border-radius: 10px;
  position: absolute;
  z-index: 999;
  pointer-events: none;
  background: #fff;
}
```

ОБЩИЙ ПОДХОД

При mousedown клонируем элемент и привязываем к указателю мыши. При `mouseup` определяем, куда «кинули элемент» и устанавливаем родителя.

```
let draggedEl = null;
let ghostEl = null;
const itemsEl = document.querySelector('.items');
itemsEl.addEventListener('mousedown', (evt) => {
  evt.preventDefault();
  if (!evt.target.classList.contains('items-item')) {
    return;
  }
  draggedEl = evt.target;
  ghostEl = evt.target.cloneNode(true);
  ghostEl.classList.add('dragged');
  document.body.appendChild(ghostEl);
  ghostEl.style.left = `${evt.pageX - ghostEl.offsetWidth / 2}px`;
  ghostEl.style.top = `${evt.pageY - ghostEl.offsetHeight / 2}px`;
});
```

`move`, `leave`

```
itemsEl.addEventListener('mousemove', (evt) => {
  evt.preventDefault(); // не даём выделять элементы
  if (!draggedEl) {
    return;
  }
  ghostEl.style.left = `${evt.pageX - ghostEl.offsetWidth / 2}px`;
  ghostEl.style.top = `${evt.pageY - ghostEl.offsetHeight / 2}px`;
});
itemsEl.addEventListener('mouseleave', (evt) => {
  // при уходе курсора за границы контейнера - отменяем перенос
  if (!draggedEl) {
    return;
  }
  document.body.removeChild(ghostEl);
  ghostEl = null;
  draggedEl = null;
});
```

up

```
itemsEl.addEventListener('mouseup', (evt) => {
  if (!draggedEl) {
    return;
  }
  const closest = document.elementFromPoint(evt.clientX, evt.clientY);
  evt.currentTarget.insertBefore(draggedEl, closest);

  document.body.removeChild(ghostEl);
  ghostEl = null;
  draggedEl = null;
});
```

Метод `elementFromPoint` позволяет найти элемент, по конкретной позиции в документе.

УПРОЩЕНИЯ

Реализация приведена достаточно упрощённая, в частности не учитывается:

- наличие вложенных элементов в `li` (тогда нужно искать ближайшего предка);
- размеры элементов (при клонировании);
- позиция курсора при перетаскивании (он выше середины элемента или ниже).

Данные детали вы вполне способны реализовать самостоятельно, используя `DOM Traversing` и знания о метриках элементов.

РЕШЕНИЕ С ПОЗИЦИЕЙ КУРСОРА (УЧЁТ СЕРЕДИНЫ)

```
itemsEl.addEventListener('mouseup', (evt) => {
  if (!draggedEl) {
    return;
  }
  const closest = document.elementFromPoint(evt.clientX, evt.clientY);

  const { top } = closest.getBoundingClientRect();
  if (evt.pageY > window.scrollY + top + closest.offsetHeight / 2) {
    evt.currentTarget.insertBefore(draggedEl, closest.nextElementSibling);
  } else {
    evt.currentTarget.insertBefore(draggedEl, closest);
  }

  document.body.removeChild(ghostEl);
  ghostEl = null;
  draggedEl = null;
});
```

TOUCH

Для Touch устройств координаты прикосновения можно получить через `evt.changedTouches[0].pageX/pageY`.

DND

Вернёмся к DnD: мы его будем использовать только для переноса файлов.

Интересующие нас события:

- `dragover` - переносимый объект находится над `drop target`'ом;
- `drop` - переносимый объект `drop`'ается (сбрасывается) в `drop target`.

Обрабатывать эти события нужно на элементе, в который разрешён “сброс” объекта:

```
<div data-id="drop-area" class="drop-area"></div>
```

```
.drop-area {
  height: 200px;
  width: 100%;
  border: 1px dashed red;
}
```

DND

Для разрешения переноса файла в определённую область необходимо сделать `preventDefault` в двух этих событиях:

```
const dropEl = document.querySelector('[data-id=drop-area]');

dropEl.addEventListener('dragover', (evt) => {
  evt.preventDefault();
});

dropEl.addEventListener('drop', (evt) => {
  evt.preventDefault();
  // evt.dataTransfer.files -> FileList
  const files = Array.from(evt.dataTransfer.files);
});
```

DND

Таким образом, мы вполне можем использовать функциональность, которую реализовали до этого для `<input type="file>`, чтобы прочитать нужные нам файлы*.

Задача совмещения `<input type="file">` и DnD вам будет предложена в качестве домашней.

Важно: если пользователь не перенёс ни одного файла, а переносит элемент со страницы, то `files` будет пустым (это нужно проверять, как и количество, а также тип перемещаемых файлов).

ИТОГИ

Сегодня у нас была достаточно большая лекция:

- мы обсудили работу с файлами;
- поговорили о генерируемых событиях;
- разобрали Drag and Drop.

НЕТОЛОГИЯ

Задавайте вопросы и напишите отзыв о лекции!

АЛЕКСАНДР ШЛЕЙКО

 a.shleyko@yandex.ru

 vk.com/shleiko

 @dustyo_0