

# Drag&Drop a Web Worker

05.05.2022

Marek Nagy

# Drag & Drop

# D&D ťahanie

- elementu nastaviť atribút **draggable** na **true**
  - ťahaním vznikne aktuálne „foto“ elementu (ghost obrázok), t.j. neanimuje sa,...

Obrázky sa už defaultne dajú ťahať. Vec prehliadača.

```
<img src='moj.png' draggable='true'>
```

```
<div class='box' draggable='true'>Janko Hraško</div>
```

```
let elem = document.createElement ('div');  
elem.innerHTML = 'Janko Hraško';  
elem.classList.add ('box');  
  
elem.setAttribute ('draggable', 'true');  
elem.draggable = true;
```

Atribút je aj „namapovaný“ na vlastnosť DOM elementu.

# Udalosti **presúvaného** elementu

- trieda **DragEvent** rozširuje triedu **MouseEvent**

## 'dragstart'

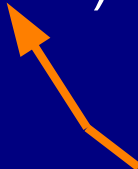
- pri chytení elementu myškou

## 'drag'

- chytený „obraz elementu“ sa myškou posúva
- postupne posiela pozíciu (event.clientX, event.clientY)
- (Firefox nedáva súradnice!)

## 'dragend'

- „obraz elementu“ sa pustil



Už by mali byť  
v MouseEvent-e  
podporované aliasy  
event.x, event.y.

# Udalosti **cieľového** elementu

- t.j. kam sa púšťa ťahaný element

## **'dragenter'**

- nad element vošiel presúvaný „obraz elementu“

## **'dragleave'**

- z elementu odišiel presúvaný „obraz elementu“

## **'dragover'**

- nad elementom je presúvaný „obraz elementu“
- viacnásobná udalosť s aktuálnou pozíciou (event.x, event.y)

## **'drop'**

- do elementu bol pustený „obraz elementu“
- aby fungoval, treba e.**preventDefault()** v udalosti **dragover**!
- vhodné dať sem e.**preventDefault()**, aby prehliadač nereagoval



# Trieda DragEvent

- je rozšírením triedy MouseEvent
- inštanciu dostanú funkcie zavesené na udalosti

## dataTransfer

- inštancia triedy DataTransfer
- iba na čítanie
- vytvorený objekt v rámci celej „akcie ťahania“
- obsahuje údaje, ktoré sa „ťahajú“
- **zdieľaný** objekt pre všetky D&D udalosti

# Trieda **DataTransfer**

## vzhľad kurzora

### **effectAllowed**

- signalizuje akciu, ktorá sa vykoná
- vzhľad presúvacieho kurzora myši (**copy**, **move**, **link**, ...)
- nastaviť iba pri udalosti **dragstart**

### **dropEffect**

- signalizuje aká akcia sa vykoná pri pustení
- vzhľad kurzora myši (**none**, **copy**, **move**, **link**, ...)
- nastaviť pri udalosti **dragover** alebo **dragenter**



# Trieda **DataTransfer**

## prenášané údaje

Pre každý typ  
(**mimetype**) je len  
jeden záznam!

### **setData (type, data)**

- nastaví „dragovacie“ dáta (**reťazec**) pre daný typ
- napr. url linka súboru, id elementu, ...

```
e.dataTransfer.setData ('text/plain', 'Môj text');
```

### **getData (type)**

- získa nastavený reťazec pre daný typ

```
let txt = e.dataTransfer.getData('text/plain');
```

### **clearData (type)**

- zruší dáta k danému typu. Ak sa neuvedie typ, zruší všetko!

# Trieda **DataTransfer**

prenášané údaje

**types**


- pole nastavených typov, ktoré sa „dragujú“

# Trieda **DataTransfer**

## prenášané súbory

### files

- zoznam dragovaných súborov do prehliadača
  - z disku klienta
- objekty triedy **File**
- načítanie pomocou **FileReader-u**
  - je zabezpečené, že prehliadač nemôže zadať ľubovoľnú cestu na disk klienta!
- súbor formou objektu **File** príde iba z:
  - **dragovania**
  - **formulára: <input type='file'>**
  - t.j. užívateľ vie o tom, čo posiela do prehliadača na spracovanie

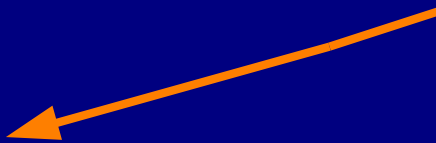


Starší prístup.  
Súbory sa dajú  
získať aj z **items**.

# Trieda **DataTransfer**

prenášané údaje a súbory

Kompaktný  
zoznam textov a  
súborov. Novší  
prístup.



## items

- inštancia **DataTransferItemList**:

**length** - počet prvkov

**add (data, type)** - pridaj údaj pod daný typ

**remove (index)** - odstráň konkrétny prvok

**clear ()** - vymaž všetko

- prvky sú inštancie **DataTransferItem**:

**kind** - string / file

**type** - mimetype

**getAsString()** - daj údaj

**getAsFile()** - daj File objekt

# D&D medzi oknami

- D&D presahuje hranice prehliadača
- z prehliadača možno vytiahnuť do inej aplikácie
  - obrázky, texty, ... (čo je v **items**)
  - aplikácia dostane obsah `dataTransfer` objektu

```
let url = e.dataTransfer.getData('text/plain');  
// url == "http://localhost:9000/obrazok.gif"
```

- väčšinou je to url linka objektu na stiahnutie
- do prehliadača možno vložiť:
  - obrázky, súbory, texty, ... (čo bude v **items**)
  - súbory (objekty triedy **File**) sú separátne aj vo **files**

# Trieda **DataTransfer**

## ghost obrázok

### **setDragImage (elem, xOffset, yOffset)**

- nastaví **ghost obrázok** podľa elementu
  - defaultne je to podľa dragnutého elementu
  - ak je to img, prevezme, inak „odfotí“
- pri udalosti **dragstart**
- xOffset, yOffset je posun ghost obrázka vzhľadom na kurzor



# Trieda File

- reprezentuje aktuálny stav súboru na disku
  - t.j. súbor nie je načítaný
  - ak sa súbor zmenil, už sa nepodarí načítať
- vlastnosti objektu sa dajú iba čítať

## name

- meno súboru bez cesty

## size

- veľkosť v bajtoch

## type

- mime type súboru



# Trieda FileReader

- treba vytvoriť `new FileReader()`
- metódy realizujú načítanie obsahu (do vlastnosti `result`)

`readAsArrayBuffer (fileobj)`

- načítanie ako bloku bajtov (ArrayBuffer)

`readAsDataURL (fileobj)`

- načítanie ako špeciálne URL, ktoré obsahuje v sebe aj dáta

`readAsText (fileobj)`

- načíta ako reťazec

`abort ()`

- ukončí načítavanie, bez výsledku

`result`

- obsahuje dáta načítaného súboru

# FileReader udalosti

'load'

- po načítaní

'error'

- s chybou pri načítaní

'progress'

- postupné informovanie o načítaní

```
let fr = new FileReader ();
fr.addEventListener ('load', e => {
  let img = document.createElement ('img');
  img.src = this.result;
  document.body.appendChild (img);
});

fr.readAsDataURL (fileObj);
```

# FileReader async metóda

```
function readAsDataURLAsync (file) {  
  return new Promise ((resolve, reject) => {  
    let reader = new FileReader ();  
  
    reader.onload = () => {  
      resolve (reader.result);  
    };  
  
    reader.onerror = () => {  
      reader.abort ();  
      reject (new Error ('readAsDataURL problem.'));  
    };  
  
    reader.readAsDataURL (file);  
  });  
}
```



# Web Worker

# Web Worker

- zdĺhavý výpočet "zasekne" funkčnosť celej stránky
  - hlavné vlákno
- HTML5
- Worker - skript, ktorý beží na pozadí
  - ďalšie vlákno
  - hlavné vlákno môže ďalej pokračovať v obhospodarovaní stránky

# Web Worker

- nemá prístup do objektu window
- nevie manipulovať s DOM-om
- vie vytvárať WebSockets

# Vytvorenie Workera

- separátny JavaScript súbor

```
w = new Worker ('mojWorker.js')
```

- **same-origin policy** (zásada rovnakého pôvodu)
  - protocol, host a port musia byť rovnaké
- google-chrome --**allow-file-access-from-files**
  - pri lokálnom načítaní stránky (nie cez server)



# Ukončenie Workera

- okamžite ukončí vlákno
  - worker už nič nespraví

```
w.terminate ();
```

# Prijímanie a posielanie dát

- zavesenie callback funkcie na udalosť

```
w.onmessage = event => {  
  ... event.data ...  
}
```

- udalosť obsahuje prijaté **data** z workera

- poslanie správy workeru

```
w.postMessage (data);
```

- **data** sa kopírujú (nie zdieľajú) t.j. sú na oboch stranách komunikácie

# Posielanie dát bez kopírovania

- iba zmena vlastníctva dát → rýchlejšie

```
w.postMessage (data, [data]);
```

- odosielateľ bude mať prázdnu hodnotu a prijímateľ dostane dáta

- iba pre transferable objects (ArrayBuffer,...)

```
d = new Float64Array (100);  
w.postMessage (d.buffer, [d.buffer]);
```

# Chybová udalosť

- zavesenie callback funkcie

```
w.onerror = event => {  
  
    event.message // Správa  
    event.filename // súbor  
    event.lineno  // číslo riadku, kde je chyba  
  
}
```

# Vytvorený Worker

- automaticky referencuje seba ako globálny objekt
  - netreba písať **this**
  - podobne ako netreba písať referenciu **window** pre globálne premenné v hlavnom vlákne

# Posielanie a prijímanie dát

- zavesenie callback funkcie

```
onmessage = event => {  
  ... event.data ...  
}
```

- posielanie správy hlavnému vláknu

```
postMessage (data);
```

# Ukončenie Workera

- samoukončenie

```
close ();
```

# Udalosti na strane Workera

- pri chybe

```
onerror = event => {  
}
```

- podľa zmeny stavu pripojenia k internetu
  - príliš nepodporované

```
ononline = event => {  
}  
onoffline = event => {  
}
```





# Meranie času

- cez Date
  - celé číslo [ms] od 1.1.1970

```
t = Date.now () // počet milisekúnd od 1.1.1970
```

- cez Performance
  - desatinné číslo [ms] od otvorenia stránky / workera

Presnejšie meranie.  
Hardvérový čas, ktorý  
sa nesynchronizuje  
so skutočným časom  
hodiniek.

```
t = performance.now () // milisekundy od otvorenia stránky

performance.mark ('start'); //timestamp1
//...kód
performance.mark ('end'); //timestamp2
performance.measure ('moja', 'start', 'end'); //rozdiel

t = performance.getEntriesByName ('moja'); //pole záznamov
console.log (t[0].duration);

performance.clearMarks (); performance.clearMeasures ();
```



Ďakujem za pozornosť