4th July 2019

File and FileReader

A File object inherits from `Blob` and is extended with filesystem-related capabilities.

There are two ways to obtain it.

First, there's a constructor, similar to `Blob`:

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

- **`fileParts`** – is an array of Blob/BufferSource/String values.
- **`fileName`** – file name string.
- **`options`** – optional object:
  - **`lastModified`** – the timestamp (integer date) of last modification.

Second, more often we get a file from `<input type="file">` or drag'n'drop or other browser interfaces. In that case, the file gets this information from OS.

As `File` inherits from `Blob`, `File` objects have the same properties, plus:

- `name` – the file name,
- `lastModified` – the timestamp of last modification.

That's how we can get a `File` object from `<input type="file">`:

```
<input type="file" onchange="showFile(this)">

<script>
function showFile(input) {
  let file = input.files[0];

  alert(`File name: ${file.name}`); // e.g my.png
  alert(`Last modified: ${file.lastModified}`); // e.g 1552830408824
}
</script>
```

Please note:

The input may select multiple files, so `input.files` is an array-like object with them. Here we have only one file, so we just take `input.files[0]`.

FileReader

FileReader is an object with the sole purpose of reading data from `Blob` (and hence `File` too) objects.

It delivers the data using events, as reading from disk may take time.

The constructor:

```
let reader = new FileReader(); // no arguments
```

The main methods:

- **`readAsArrayBuffer(blob)`** – read the data in binary format `ArrayBuffer`.
- **`readAsText(blob, [encoding])`** – read the data as a text string with the given encoding (`utf-8` by default).
- **`readAsDataURL(blob)`** – read the binary data and encode it as base64 data url.
- **`abort()`** – cancel the operation.

The choice of `read*` method depends on which format we prefer, how we're going to use the data.

- `readAsArrayBuffer` – for binary files, to do low-level binary operations. For high-level operations, like slicing, `File` inherits from `Blob`, so we can call them directly, without reading.
- `readAsText` – for text files, when we'd like to get a string.
- `readAsDataURL` – when we'd like to use this data in `src` for `img` or another tag. There's an alternative to reading a file for that, as discussed in chapter Blob: `URL.createObjectURL(file)`.

As the reading proceeds, there are events:

- `loadstart` – loading started.
- `progress` – occurs during reading.
- `load` – no errors, reading complete.
- `abort` – `abort()` called.
- `error` – error has occurred.
- `loadend` – reading finished with either success or failure.

When the reading is finished, we can access the result as:

- `reader.result` is the result (if successful)
- `reader.error` is the error (if failed).

The most widely used events are for sure `load` and `error`.

Here's an example of reading a file:

```
<input type="file" onchange="readFile(this)">

<script>
function readFile(input) {
  let file = input.files[0];

```

```
  let reader = new FileReader();

  reader.readAsText(file);

  reader.onload = function() {
    console.log(reader.result);
  };

  reader.onerror = function() {
    console.log(reader.error);
  };

}
</script>
```

`FileReader` for blobs

As mentioned in the chapter Blob, `FileReader` can read not just files, but any blobs.

We can use it to convert a blob to another format:

- `readAsArrayBuffer(blob)` – to `ArrayBuffer`,
- `readAsText(blob, [encoding])` – to string (an alternative to `TextDecoder`),
- `readAsDataURL(blob)` – to base64 data url.

`FileReaderSync` is available inside Web Workers

For Web Workers, there also exists a synchronous variant of `FileReader`, called FileReaderSync.

Its reading methods `read*` do not generate events, but rather return a result, as regular functions do.

That's only inside a Web Worker though, because delays in synchronous calls, that are possible while reading from files, in Web Workers are less important. They do not affect the page.

Summary

`File` objects inherit from `Blob`.

In addition to `Blob` methods and properties, `File` objects also have `name` and `lastModified` properties, plus the internal ability to read from filesystem. We usually get `File` objects from user input, like `<input>` or Drag'n'Drop events (`ondragend`).

`FileReader` objects can read from a file or a blob, in one of three formats:

- String (`readAsText`).
- `ArrayBuffer` (`readAsArrayBuffer`).
- Data url, base-64 encoded (`readAsDataURL`).

In many cases though, we don't have to read the file contents. Just as we did with blobs, we can create a short url with `URL.createObjectURL(file)` and assign it to `<a>` or `<img>`. This way the file can be downloaded or shown up as an image, as a part of canvas etc.

And if we're going to send a `File` over a network, that's also easy: network API like `XMLHttpRequest` or `fetch` natively accepts `File` objects.

Previous lesson Next lesson

Share

Tutorial map

Comments

- If you have suggestions what to improve - please submit a GitHub issue or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.
- To insert a few words of code, use the `<code>` tag, for several lines – use `<pre>`, for more than 10 lines – use a sandbox (plnkr, JSBin, codepen…)