9th August 2019

FormData

This chapter is about sending HTML forms: with or without files, with additional fields and so on.

FormData objects can help with that. As you might have guessed, it's the object to represent HTML form data.

The constructor is:

```
let formData = new FormData([form]);
```

If HTML `form` element is provided, it automatically captures its fields.

The special thing about `FormData` is that network methods, such as `fetch`, can accept a `FormData` object as a body. It's encoded and sent out with `Content-Type: form/multipart`.

From the server point of view, that looks like a usual form submission.

Sending a simple form

Let's send a simple form first.

As you can see, that's almost one-liner:

```
<form id="formElem">
  <input type="text" name="name" value="John">
  <input type="text" name="surname" value="Smith">
  <input type="submit">
</form>

<script>
  formElem.onsubmit = async (e) => {
    e.preventDefault();

    let response = await fetch('/article/formdata/post/user', {
      method: 'POST',
      body: new FormData(formElem)
    });

    let result = await response.json();

    alert(result.message);
  };
</script>
```

John Smith Submit

In this example, the server code is not presented, as it's beyound our scope. The server accepts the POST request and replies "User saved".

FormData Methods

We can modify fields in `FormData` with methods:

- `formData.append(name, value)` – add a form field with the given `name` and `value`,
- `formData.append(name, blob, fileName)` – add a field as if it were `<input type="file">`, the third argument `fileName` sets file name (not form field name), as it it were a name of the file in user's filesystem,
- `formData.delete(name)` – remove the field with the given `name`,
- `formData.get(name)` – get the value of the field with the given `name`,
- `formData.has(name)` – if there exists a field with the given `name`, returns `true`, otherwise `false`

A form is technically allowed to have many fields with the same `name`, so multiple calls to `append` add more same-named fields.

There's also method `set`, with the same syntax as `append`. The difference is that `.set` removes all fields with the given `name`, and then appends a new field. So it makes sure there's only field with such `name`, the rest is just like `append`:

- `formData.set(name, value)`,
- `formData.set(name, blob, fileName)`.

Also we can iterate over formData fields using `for..of` loop:

```
let formData = new FormData();
formData.append('key1', 'value1');
formData.append('key2', 'value2');

// List key/value pairs
for(let [name, value] of formData) {
  alert(`${name} = ${value}`); // key1=value1, then key2=value2
}
```

Sending a form with a file

The form is always sent as `Content-Type: form/multipart`, this encoding allows to send files. So, `<input type="file">` fields are sent also, similar to a usual form submission.

Here's an example with such form:

```
<form id="formElem">
  <input type="text" name="firstName" value="John">
  Picture: <input type="file" name="picture" accept="image/*">
  <input type="submit">
</form>
```

```
<script>
  formElem.onsubmit = async (e) => {
    e.preventDefault();

    let response = await fetch('/article/formdata/post/user-avatar', {
      method: 'POST',
      body: new FormData(formElem)
    });

    let result = await response.json();

    alert(result.message);
  };
</script>
```

John Picture: Choose File No file chosen Submit

Sending a form with Blob data

As we've seen in the chapter Fetch, it's easy to send dynamically generated binary data e.g. an image, as `Blob`. We can supply it directly as `fetch` parameter `body`.

In practice though, it's often convenient to send an image not separately, but as a part of the form, with additional fields, such as "name" and other metadata.

Also, servers are usually more suited to accept multipart-encoded forms, rather than raw binary data.

This example submits an image from `<canvas>`, along with some other fields, as a form, using `FormData`:

```
<body style="margin:0">
  <canvas id="canvasElem" width="100" height="80" style="border:1px solid"></canvas>

  <input type="button" value="Submit" onclick="submit()">

  <script>
    canvasElem.onmousemove = function(e) {
      let ctx = canvasElem.getContext('2d');
      ctx.lineTo(e.clientX, e.clientY);
      ctx.stroke();
    };

    async function submit() {
      let imageBlob = await new Promise(resolve => canvasElem.toBlob(resolve, 'image/

      let formData = new FormData();
      formData.append("firstName", "John");
      formData.append("image", imageBlob, "image.png");

      let response = await fetch('/article/formdata/post/image-form', {
        method: 'POST',
        body: formData
      });
      let result = await response.json();
      alert(result.message);
    }

```

```
28    </script>
29  </body>
```

Please note how the image `Blob` is added:

```
1  formData.append("image", imageBlob, "image.png");
```

That's same as if there were `<input type="file" name="image">` in the form, and the visitor submitted a file named `"image.png"` (3rd argument) with the data `imageBlob` (2nd argument) from their filesystem.

The server reads form data and the file, as if it were a regular form submission.

Summary

FormData objects are used to capture HTML form and submit it using `fetch` or another network method.

We can either create `new FormData(form)` from an HTML form, or create a object without a form at all, and then append fields with methods:

- `formData.append(name, value)`
- `formData.append(name, blob, fileName)`
- `formData.set(name, value)`
- `formData.set(name, blob, fileName)`

Let's note two peculiarities here:

1. The `set` method removes fields with the same name, `append` doesn't. That's the only difference between them.
2. To send a file, 3-argument syntax is needed, the last argument is a file name, that normally is taken from user filesystem for `<input type="file">`.

Other methods are:

- `formData.delete(name)`
- `formData.get(name)`
- `formData.has(name)`

That's it!

Previous lesson

Next lesson