

[🏠](#) → [Network requests](#)

📅 19th November 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:

```
1 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:

```
1 <form id="formElem">
2   <input type="text" name="name" value="John">
3   <input type="text" name="surname" value="Smith">
4   <input type="submit">
5 </form>
6
7 <script>
8   formElem.onsubmit = async (e) => {
9     e.preventDefault();
10
11     let response = await fetch('/article/formdata/post/user', {
12       method: 'POST',
13       body: new FormData(formElem)
14     });
15
16     let result = await response.json();
17
18     alert(result.message);
19   };
20 </script>
```



<input type="text" value="John"/>	<input type="text" value="Smith"/>	<input type="button" value="Submit"/>
-----------------------------------	------------------------------------	---------------------------------------

In this example, the server code is not presented, as it's beyond 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 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 one 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:

```
1 let formData = new FormData();
2 formData.append('key1', 'value1');
3 formData.append('key2', 'value2');
4
5 // List key/value pairs
6 for(let [name, value] of formData) {
7   alert(`${name} = ${value}`); // key1=value1, then key2=value2
8 }
```



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:

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



```

6
7 <script>
8   formElem.onSubmit = async (e) => {
9     e.preventDefault();
10
11     let response = await fetch('/article/formdata/post/user-avatar', {
12       method: 'POST',
13       body: new FormData(formElem)
14     });
15
16     let result = await response.json();
17
18     alert(result.message);
19   };
20 </script>

```

Picture:

No file chosen

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.

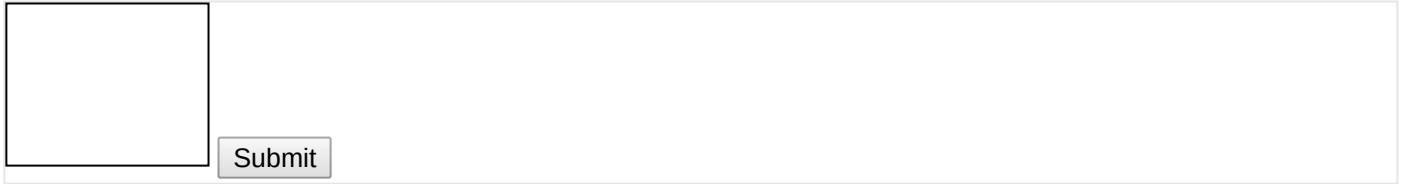
This example submits an image from `<canvas>` , along with some other fields, as a form, using `FormData` :

```

1 <body style="margin:0">
2   <canvas id="canvasElem" width="100" height="80" style="border:1px solid"></
3
4   <input type="button" value="Submit" onclick="submit()">
5
6   <script>
7     canvasElem.onmousemove = function(e) {
8       let ctx = canvasElem.getContext('2d');
9       ctx.lineTo(e.clientX, e.clientY);
10      ctx.stroke();
11    };
12
13    async function submit() {
14      let imageBlob = await new Promise(resolve => canvasElem.toBlob(resolve,
15
16      let formData = new FormData();
17      formData.append("firstName", "John");
18      formData.append("image", imageBlob, "image.png");
19
20      let response = await fetch('/article/formdata/post/image-form', {
21        method: 'POST',
22        body: formData
23      });
24      let result = await response.json();
25      alert(result.message);
26    }
27
28

```

```
29     </script>
    </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



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](#)...)