January 26, 2021

How to serialize form data with vanilla JS

***UPDATE:** This article was updated to account for forms with multiple fields that have the same name.*

Historically, getting data from a form into a format you could send to an API was a bit difficult, but modern techniques make it a lot easier.

The FormData object

The `FormData` object provides an easy way to serialize form fields into key/value pairs.

You can use the `new FormData()` constructor to create a new FormData object, passing in the form to serialize as an argument. Form fields must have a `name` attribute to be included object. Otherwise, they're skipped. The `id` property doesn't count.

For example, if you have a form that looked like this…

```
<form id="post">

  <label for="title">Title</label>
  <input type="text" name="title" id="title" value="Go to the beac
h">

  <label for="body">Body</label>
  <textarea id="body" name="body">Soak up the sun and swim in the o
cean.</textarea>

  <input type="hidden" name="userId" value="1">

  <button>Submit</button>

</form>
```

You would serialize it like this...

```
// Get the form
let form = document.querySelector('#post');

// Get all field data from the form
// returns a FormData object
let data = new FormData(form);
```

Looping through FormData

The `FormData` object is an iterable, which means you can loop through it using a `for...of` loop (https://gomakethings.com/the-for...of-loop-in-vanilla-js/) (*you'll notice that a trend with many of the newer JS methods*).

Each `entry` in the loop is an array of key/value pairs.

```
// logs...
// ["title", "Go to the beach"]
// ["body", "Soak up the sun and swim in the ocean."]
// ["userId", "1"]
for (let entry of data) {
  console.log(entry);
}
```

Because the `entry` values are arrays, you can also use array destructuring (https://gomakethings.com/destructing-in-vanilla-js/#array-destructuring) to assign the `key` and `value` to their own variables within the `for...of` loop.

```
// logs "title", "Go to the beach", etc.
for (let [key, value] of data) {
  console.log(key);
  console.log(value);
}
```

Adding, updating, and removing items from a FormData object

The `FormData` object has several methods that you can use to add, remove, and update items.

Use the `FormData.set()` method to replace an existing entry, or add a new one if an entry with that key doesn't exist. Pass in the `key` and `value` as arguments.

```
// Updates the userId field with a new value
data.set('userId', '3');

// Creates a new key, "date", with a value of "4"
data.set('date', 'July 4');
```

Use the `FormData.append()` method to add a new entry, passing in the `key` and `value` as arguments. If an item with that `key` already exists, another one is added and the existing one is unaffected.

```
// Add a second "body" key to the data FormData object
data.append('body', 'Eat ice cream');
```

Use the `FormData.delete()` method to delete an entry, passing in the `key` as an argument. If more than one item with that `key` exist, all of them are deleted.

```
// Delete items with the "body" key
data.delete('body');
```

How to convert a FormData object into an object or query string

The FormData object *can* be submitted as-is to some endpoints with a `content-type` header of `multipart/form-data`, not not all APIs support that.

To serialize a `FormData` object into a query string, pass it into the `new URLSearchParams()` constructor. This will create a `URLSearchParams` object of encoded query string values.

Then, call the `URLSearchParams.toString()` method on it to convert it into a query string.

```
// Get the form
let form = document.querySelector('#post');

// Get all field data from the form
let data = new FormData(form);

// Convert to a query string
let queryString = new URLSearchParams(data).toString();
```

To serialize a `FormData` object into a plain object, we need to loop through each entry with a `for...of` loop and add it to an object.

```
let obj = {};
for (let [key, value] of data) {
  obj[key] = value;
}
```

But, if there's more one form field with the same name, the original value will get overwritten. To account for this, we need to check if the `key` already exists in the `obj`. If it does, we want to convert it to an array and `Array.push()` the new `value` into it.

```
let obj = {};
for (let [key, value] of data) {
  if (obj[key] !== undefined) {
    if (!Array.isArray(obj[key])) {
      obj[key] = [obj[key]];
    }
    obj[key].push(value);
  } else {
    obj[key] = value;
  }
}
```

Here's a helper function you can use to convert a `FormData` object into a plain object. Pass in the `FormData` object as an argument.

```
function serialize (data) {
  let obj = {};
  for (let [key, value] of data) {
    if (obj[key] !== undefined) {
      if (!Array.isArray(obj[key])) {
        obj[key] = [obj[key]];
      }
      obj[key].push(value);
    } else {
      obj[key] = value;
    }
  }
  return obj;
}
```

And here’s how you would use it.

```
// Get the form
let form = document.querySelector('#post');

// Get all field data from the form
let data = new FormData(form);

// Convert to an object
let formObj = serialize(data);
```

⏰🦉 ***Early Bird Sale!*** *Today through Monday, get 40% off registration in the next session of the Vanilla JS Academy (https://vanillajsacademy.com).*

Hate the complexity of modern front-end web development? *I send out a short email each weekday on how to build a simpler, more resilient web. Join over 13k others.*