JS Fetch API

Fetch API is an interface for working with HTTP requests and responses, providing the ability to interact with web services and network resources directly from the **JavaScript** node.

The Fetch API interface includes the following objects:

- `Headers` : Request/response headers
- `Request` : A network resource request
- `Response` : A response to a request
- `fetch()` : A function used to retrieve network resources.

Before use, the above objects must be imported from the built-in `builtIn/Fetch` module:

```
import {fetch, Request, Response, Headers} from "builtIn/Fetch";
```

> **Note**: The `Request` and `Response` objects lack some methods and properties that are standard in browsers. For a complete list, see the description below. For details, see Full API Description.

Object Descriptions

Headers

An iterable collection of HTTP headers. The object's methods allow you to get, set, add, and remove headers from the request's header collection.

Constructor

To create a `Headers` object, use the following line:

```
let headers = new Headers([init]);
```

where `init` takes an object of type `HeadersInit` . Optional parameter.

Methods

▼ append

`append(name, value)`

- `name` : The name of the header to add. Takes a `string` value. Required parameter.
- `value` : The value of the header. Takes a `string` value. Required parameter.

Adds a new value to an existing header within the `Headers` object, or adds the header if it doesn't exist. The method returns `undefined` .

▼ delete

`delete(name)`

- `name` : The name of the header to delete. Takes a `string` value. Required parameter. Deletes a header from the current `Headers` object. The method returns `undefined` .

▼ entries

`entries()`

The method returns an iterable collection of name/value pairs of the headers contained in the `Headers` object. The name and value are strings.

▼ forEach

`forEach(callbackfn(value, key, parent)[, thisArg])`

- `callbackfn` : The function to be applied to each header. Required parameter. The following parameters are passed to `callbackfn` in the specified order:
  - `value` : The value of the header; of `string` type.
  - `key` : The name of the header. Takes a `string` value.
  - `parent` : The `Headers` object
- `thisArg` : The value to use as `this` when calling `callbackfn` . Optional parameter.

The method iterates over the collection of headers in the `Headers` object and executes the provided function for each header.

▼ get

`get(name)`

- `name` : The name of the header. Takes a `string` value. Required parameter.

Returns a string representing the value of the header, or `null` if this header is not set.

▼ set

`set(name, value)`

- `name` : The name of the header. Takes a `string` value. Required parameter.
- `value` : The value of the header, `string` . Required parameter.

Sets a new value for an existing header within the `Headers` object or adds the header if it doesn't exist. The method returns `undefined` .

▼ has

`has(name)`

- `name` : The name of the header. Takes a `string` value. Required parameter.

Returns `true` or `false` depending on whether the `Headers` object contains a header with the specified name.

▼ keys

`keys()`

Returns an iterable collection of the names of the headers in the `Headers` object.

▼ values

`values()`

Returns an iterable collection of the values of the headers in the `Headers` object.

Request

An HTTP request.

Constructor

To create a `Request` object, write:

```
let request = new Request(input[, init]);
```

where:

- `input` — an object of type `RequestInfo` . Required parameter. Takes the URL of the requested resource or an object that implements the `Request` interface.
- `init` — an object that implements the `RequestInit` interface. Optional parameter. Takes the parameters of the HTTP request:
  - `body` — the body of the HTTP request. An object of type `BodyInit` . The `body` parameter can be a string or an object of type `ArrayBuffer` , `ArrayBufferView` .

- `headers` — the headers of the HTTP request. A `Headers` object.
- `method` — a string containing the HTTP request method (`get` , `post` , etc).
- `redirect` — a string containing the redirect handling mode (`follow` , `error` , or `manual`).

Implementation features

Setting `Cookie` and `Cookie2` HTTP headers is allowed.

Properties

▼ bodyUsed

`bodyUsed`

Contains a boolean value indicating whether the request body has been read. (The request body can only be read once.) Returns `true` or `false` . Read-only.

▼ headers

`headers`

Contains the HTTP request headers (a `Headers` object). Read-only.

▼ method

`method`

Contains the request method (`GET` , `POST` , etc.). Returns a `string` value. Read-only.

▼ redirect

`redirect`

Contains the redirect handling mode. Returns one of the following `string` values:

- `follow`
- `error`
- `manual`

If the property is not specified when creating the request, it defaults to `follow` . Read-only.

▼ url

`url`

> Contains the URL of the request. Returns a `string` value. Read-only.

Methods

▼ arrayBuffer

> `arrayBuffer()`
>
> Reads the request body and returns a promise of an `ArrayBuffer` value.

▼ json

> `json()`
>
> Returns a promise with an object obtained by parsing the request body as JSON text.

▼ text

> `text()`
>
> Returns a promise with a string obtained by interpreting the request body as text in UTF-8 encoding.

▼ clone

> `clone()`
>
> Creates a copy of the current `Request` object.

Response

A response to an HTTP request.

Constructor

You can create a new `Response` object using the constructor as follows:

```
let response = new Response([body][, init]);
```

where:

- `body` takes an object of type `BodyInit`, or a `null` value. Optional parameter.
- `init` takes an object that implements the `ResponseInit` interface. Optional parameter.

However, in practice, you are more likely to encounter a `Response` object returned by the `fetch()` function.

Implementation features

The `Set-Cookie` and `Set-Cookie2` headers are allowed for use with the request's `Cookie` and `Cookie2` headers.

Properties

▼ bodyUsed

`bodyUsed`

Contains a `Boolean` value indicating whether the response body has been read (the response body can only be read once). Returns `true` or `false`. Read-only.

▼ headers

`headers`

Contains the HTTP response headers (a `Headers` object). Read-only.

▼ ok

`ok`

Contains a `Boolean` value indicating whether the response was successful (status in the range of `2xx`) or not. Returns `true` or `false`. Read-only.

▼ redirected

`redirected`

Contains a `Boolean` value indicating whether the response is the result of a redirected request. Returns `true` or `false`. Read-only.

▼ status

`status`

Contains the HTTP response status code. Returns a `number` value. Read-only.

▼ statusText

`statusText`

> Contains a message corresponding to the HTTP response status code. For example, `OK` corresponds to status code `200`, `Continue` to `100`, `Not Found` to `404`. Returns a `string` value. Read-only.

▼ **url**

> **url**
>
> Contains the URL of the response. The value of the `url` property will be the final URL obtained after any redirects. Returns a `string` value. Read-only.

Methods

▼ **arrayBuffer**

> `arrayBuffer()`
>
> Reads the response body and returns a promise of an `ArrayBuffer` value.

▼ **json**

> `json()`
>
> Returns a promise with an object obtained by parsing the response body as JSON text.

▼ **text**

> `text()`
>
> Returns a promise with a string obtained by interpreting the request body as text in UTF-8 encoding.

▼ **clone**

> `clone()`
>
> Creates a copy of the current `Response` object.

fetch()

The asynchronous fetch function starts the process of fetching a resource from the network, returning a promise of a `Response` object.

`fetch(resource[, init])`, where

- `resource` takes an object that implements the `Request` interface, or a string containing the request URL. Required parameter.
- `init` takes an object that implements the `RequestInit`. Optional parameter.

Examples

Using Fetch API

```
import {fetch, Request, Headers} from "builtIn/Fetch";

// Create a request headers object:
let headers = new Headers({"Content-Type": "text/html", "Custom-Header":
"delete me"})
// Log the header value
console.log("Custom-Header: ", headers.get("Custom-Header"))
// Delete the header
headers.delete("Custom-Header")
// Check if the header exists
console.log(headers.has('Custom-Header'));
// Add a new header
headers.append("Accept-Charset", "utf-8")
// Change the header value
headers.set("Content-Type", "application/json")

// Create a request object:
let request = new Request("http://httpbin.org/post", {
    method: "post",
    headers: headers,
    body: "{ \"str\": message }",
    redirect: "follow"
});

// Log the request parameters:
console.log("url: " + request.url);
console.log("bodyUsed: " + request.bodyUsed);
console.log("redirect: " + request.redirect);
console.log("method: " + request.method);
for (let header of headers.entries()) {
    console.log(header[0]+ ': '+ header[1]);
}

// Call the httpbin.org service and log the response parameters:
fetch(request)
    .then(response => {
        console.log("ok: " + response.ok);
```

```
        console.log("status: " + response.status);
        console.log("statusText: " + response.statusText);
        console.log("redirected: " + response.redirected);
        console.log("url: " + response.url);
        headers.forEach(function(value, key) {
            console.log("Name : ", key, " Value : ", value)
        });
        response.text().then(text => console.log(text));
    }).catch(e => console.log(e));
```

Obtaining exchange rates from the European Central Bank

This example uses the free `frankfurter.app` service, which provides currency exchange rates published by the European Central Bank.

```
import { fetch } from "builtIn/Fetch";

// Request exchange rates with USD as the base currency
fetch("https://api.frankfurter.app/latest?from=USD")
    .then(response => {
        if (!response.ok) {
            throw new Error(`HTTP error! status: ${response.status},
statusText: ${response.statusText}`);
        }
        return response.json();
    })
    .then(data => console.log(JSON.stringify(data)))
    .catch(e => console.log(e));
```

Sequential execution of requests

```
import {fetch, Headers} from "builtIn/Fetch";

(async() => {
    try {
        // Request the 1st service
        let response1 = await
fetch('https://jsonplaceholder.typicode.com/posts', {
            method: 'POST',
            body: JSON.stringify({
                title: "foo",
                body: "bar",
                userId: 1,
            }),
            headers: new Headers({"Content-type": "application/json;
charset=UTF-8"}),
        });
        let payload = await response1.arrayBuffer();

        // Request the 2nd service
        let response2 = await fetch('http://httpbin.org/post', {
            method: 'POST',
            headers: new Headers({"Content-Type": "application/json",
                                  "Accept-Charset": "utf-8"}),
            body: payload,
        });
        console.log(await response2.text())
    } catch(e) {
        console.log(e);
    }
})()
```

Read on: