Fetch API IMPORTANT

[**Supplying request options**](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch#supplying_request_options)

The fetch() method can optionally accept a second parameter, an init object that allows you to control a number of different settings:

See [fetch()](https://developer.mozilla.org/en-US/docs/Web/API/fetch) for the full options available, and more details.

JS

Copy to Clipboard

// Example POST method implementation:

async function postData(url = "", data = {}) {

// Default options are marked with \*

const response = await fetch(url, {

method: "POST", // \*GET, POST, PUT, DELETE, etc.

mode: "cors", // no-cors, \*cors, same-origin

cache: "no-cache", // \*default, no-cache, reload, force-cache, only-if-cached

credentials: "same-origin", // include, \*same-origin, omit

headers: {

"Content-Type": "application/json",

// 'Content-Type': 'application/x-www-form-urlencoded',

},

redirect: "follow", // manual, \*follow, error

referrerPolicy: "no-referrer", // no-referrer, \*no-referrer-when-downgrade, origin, origin-when-cross-origin, same-origin, strict-origin, strict-origin-when-cross-origin, unsafe-url

body: JSON.stringify(data), // body data type must match "Content-Type" header

});

return response.json(); // parses JSON response into native JavaScript objects

}

postData("https://example.com/answer", { answer: 42 }).then((data) => {

console.log(data); // JSON data parsed by `data.json()` call

});

Note that mode: "no-cors" only allows a limited set of headers in the request:

* Accept
* Accept-Language
* Content-Language
* Content-Type with a value of application/x-www-form-urlencoded, multipart/form-data, or text/plain

[**Aborting a fetch**](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch#aborting_a_fetch)

To abort incomplete fetch() operations, use the [AbortController](https://developer.mozilla.org/en-US/docs/Web/API/AbortController) and [AbortSignal](https://developer.mozilla.org/en-US/docs/Web/API/AbortSignal) interfaces.

JS

const controller = new AbortController();

const signal = controller.signal;

const url = "video.mp4";

const downloadBtn = document.querySelector("#download");

const abortBtn = document.querySelector("#abort");

downloadBtn.addEventListener("click", async () => {

try {

const response = await fetch(url, { signal });

console.log("Download complete", response);

} catch (error) {

console.error(`Download error: ${error.message}`);

}

});

abortBtn.addEventListener("click", () => {

controller.abort();

console.log("Download aborted");

});

[**Sending a request with credentials included**](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch#sending_a_request_with_credentials_included)

To cause browsers to send a request with credentials included on both same-origin and cross-origin calls, add credentials: 'include' to the init object you pass to the fetch() method.

JS

Copy Clipboard

fetch("https://example.com", {

credentials: "include",

});

**Note:** Access-Control-Allow-Origin is prohibited from using a wildcard for requests with credentials: 'include'. In such cases, the exact origin must be provided; even if you are using a CORS unblocker extension, the requests will still fail.

**Note:** Browsers should not send credentials in *preflight requests* irrespective of this setting. For more information see: [CORS Requests with credentials](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS#requests_with_credentials).

If you only want to send credentials if the request URL is on the same origin as the calling script, add credentials: 'same-origin'.

JS

// The calling script is on the origin 'https://example.com'

fetch("https://example.com", {

credentials: "same-origin",

});

To instead ensure browsers don't include credentials in the request, use credentials: 'omit'.

JSCopy to Clipboard

fetch("https://example.com", {

credentials: "omit",

});

[**Uploading JSON data**](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch#uploading_json_data)

Use [fetch()](https://developer.mozilla.org/en-US/docs/Web/API/fetch) to POST JSON-encoded data.

JS

async function postJSON(data) {

try {

const response = await fetch("https://example.com/profile", {

method: "POST", // or 'PUT'

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify(data),

});

const result = await response.json();

console.log("Success:", result);

} catch (error) {

console.error("Error:", error);

}

}

const data = { username: "example" };

postJSON(data);

[**Uploading a file**](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch#uploading_a_file)

Files can be uploaded using an HTML <input type="file" /> input element, [FormData()](https://developer.mozilla.org/en-US/docs/Web/API/FormData/FormData) and [fetch()](https://developer.mozilla.org/en-US/docs/Web/API/fetch).

JS

async function upload(formData) {

try {

const response = await fetch("https://example.com/profile/avatar", {

method: "PUT",

body: formData,

});

const result = await response.json();

console.log("Success:", result);

} catch (error) {

console.error("Error:", error);

}

}

const formData = new FormData();

const fileField = document.querySelector('input[type="file"]');

formData.append("username", "abc123");

formData.append("avatar", fileField.files[0]);

upload(formData);

[**Uploading multiple files**](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch#uploading_multiple_files)

Files can be uploaded using an HTML <input type="file" multiple /> input element, [FormData()](https://developer.mozilla.org/en-US/docs/Web/API/FormData/FormData) and [fetch()](https://developer.mozilla.org/en-US/docs/Web/API/fetch).

JS

async function uploadMultiple(formData) {

try {

const response = await fetch("https://example.com/posts", {

method: "POST",

body: formData,

});

const result = await response.json();

console.log("Success:", result);

} catch (error) {

console.error("Error:", error);

}

}

const photos = document.querySelector('input[type="file"][multiple]');

const formData = new FormData();

formData.append("title", "My Vegas Vacation");

for (const [i, photo] of Array.from(photos.files).entries()) {

formData.append(`photos\_${i}`, photo);

}

uploadMultiple(formData);

[**Processing a text file line by line**](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch#processing_a_text_file_line_by_line)

The chunks that are read from a response are not broken neatly at line boundaries and are Uint8Arrays, not strings. If you want to fetch a text file and process it line by line, it is up to you to handle these complications. The following example shows one way to do this by creating a line iterator (for simplicity, it assumes the text is UTF-8, and doesn't handle fetch errors).

JS

async function\* makeTextFileLineIterator(fileURL) {

const utf8Decoder = new TextDecoder("utf-8");

const response = await fetch(fileURL);

const reader = response.body.getReader();

let { value: chunk, done: readerDone } = await reader.read();

chunk = chunk ? utf8Decoder.decode(chunk) : "";

const newline = /\r?\n/gm;

let startIndex = 0;

let result;

while (true) {

const result = newline.exec(chunk);

if (!result) {

if (readerDone) break;

const remainder = chunk.substr(startIndex);

({ value: chunk, done: readerDone } = await reader.read());

chunk = remainder + (chunk ? utf8Decoder.decode(chunk) : "");

startIndex = newline.lastIndex = 0;

continue;

}

yield chunk.substring(startIndex, result.index);

startIndex = newline.lastIndex;

}

if (startIndex < chunk.length) {

// Last line didn't end in a newline char

yield chunk.substr(startIndex);

}

}

async function run() {

for await (const line of makeTextFileLineIterator(urlOfFile)) {

processLine(line);

}

}

run();

[**Checking that the fetch was successful**](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch#checking_that_the_fetch_was_successful)

A [fetch()](https://developer.mozilla.org/en-US/docs/Web/API/fetch) promise will reject with a [TypeError](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/TypeError) when a network error is encountered or CORS is misconfigured on the server-side, although this usually means permission issues or similar — a 404 does not constitute a network error, for example. An accurate check for a successful fetch() would include checking that the promise resolved, then checking that the [Response.ok](https://developer.mozilla.org/en-US/docs/Web/API/Response/ok) property has a value of true. The code would look something like this:

JS

async function fetchImage() {

try {

const response = await fetch("flowers.jpg");

if (!response.ok) {

throw new Error("Network response was not OK");

}

const myBlob = await response.blob();

myImage.src = URL.createObjectURL(myBlob);

} catch (error) {

console.error("There has been a problem with your fetch operation:", error);

}

}

[**Supplying your own request object**](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch#supplying_your_own_request_object)

Instead of passing a path to the resource you want to request into the fetch() call, you can create a request object using the [Request()](https://developer.mozilla.org/en-US/docs/Web/API/Request/Request) constructor, and pass that in as a fetch() method argument:

JSCopy to Clipboard

async function fetchImage(request) {

try {

const response = await fetch(request);

if (!response.ok) {

throw new Error("Network response was not OK");

}

const myBlob = await response.blob();

myImage.src = URL.createObjectURL(myBlob);

} catch (error) {

console.error("Error:", error);

}

}

const myHeaders = new Headers();

const myRequest = new Request("flowers.jpg", {

method: "GET",

headers: myHeaders,

mode: "cors",

cache: "default",

});

fetchImage(myRequest);

Request() accepts exactly the same parameters as the fetch() method. You can even pass in an existing request object to create a copy of it:

JS

const anotherRequest = new Request(myRequest, myInit);

This is pretty useful, as request and response bodies can only be used once. Making a copy like this allows you to effectively use the request/response again while varying the init options if desired. The copy must be made before the body is read.