10th August 2019

Fetch API

So far, we know quite a bit about `fetch`.

Let's see the rest of API, to cover all its abilities.

Please note:

Please note: most of these options are used rarely. You may skip this chapter and still use `fetch` well.

Still, it's good to know what `fetch` can do, so if the need arises, you can return and read the details.

Here's the full list of all possible `fetch` options with their default values (alternatives in comments):

```
let promise = fetch(url, {
  method: "GET", // POST, PUT, DELETE, etc.
  headers: {
    // the content type header value is usually auto-set
    // depending on the request body
    "Content-Type": "text/plain;charset=UTF-8"
  },
  body: undefined // string, FormData, Blob, BufferSource, or URLSearchParams
  referrer: "about:client", // or "" to send no Referer header,
  // or an url from the current origin
  referrerPolicy: "no-referrer-when-downgrade", // no-referrer, origin, same-
  mode: "cors", // same-origin, no-cors
  credentials: "same-origin", // omit, include
  cache: "default", // no-store, reload, no-cache, force-cache, or only-if-ca
  redirect: "follow", // manual, error
  integrity: "", // a hash, like "sha256-abcdef1234567890"
  keepalive: false, // true
  signal: undefined, // AbortController to abort request
  window: window // null
});
```

An impressive list, right?

We fully covered `method`, `headers` and `body` in the chapter Fetch.

The `signal` option is covered in Fetch: Abort.

Now let's explore the rest of capabilities.

referrer, referrerPolicy

These options govern how `fetch` sets HTTP `Referer` header.

Usually that header is set automatically and contains the url of the page that made the request. In most scenarios, it's not important at all, sometimes, for security purposes, it makes sense to remove or shorten it.

The `referrer` option allows to set any `Referer` within the current origin) or remove it.

To send no referer, set an empty string:

```
fetch('/page', {
  referrer: "" // no Referer header
});
```

To set another url within the current origin:

```
fetch('/page', {
  // assuming we're on https://javascript.info
  // we can set any Referer header, but only within the current origin
  referrer: "https://javascript.info/anotherpage"
});
```

The `referrerPolicy` option sets general rules for `Referer`.

Requests are split into 3 types:

1. Request to the same origin.
2. Request to another origin.
3. Request from HTTPS to HTTP (from safe to unsafe protocol).

Unlike `referrer` option that allows to set the exact `Referer` value, `referrerPolicy` tells the browser general rules for each request type.

Possible values are described in the Referrer Policy specification:

- **`"no-referrer-when-downgrade"`** – the default value: full `Referer` is sent always, unless we send a request from HTTPS to HTTP (to less secure protocol).
- **`"no-referrer"`** – never send `Referer`.
- **`"origin"`** – only send the origin in `Referer`, not the full page URL, e.g. only `http://site.com` instead of `http://site.com/path`.
- **`"origin-when-cross-origin"`** – send full `Referer` to the same origin, but only the origin part for cross-origin requests (as above).
- **`"same-origin"`** – send full `Referer` to the same origin, but no referer for for cross-origin requests.
- **`"strict-origin"`** – send only origin, don't send `Referer` for HTTPS→HTTP requests.
- **`"strict-origin-when-cross-origin"`** – for same-origin send full `Referer`, for cross-origin send only origin, unless it's HTTPS→HTTP request, then send nothing.
- **`"unsafe-url"`** – always send full url in `Referer`, even for HTTPS→HTTP requests.

Here's a table with all combinations:

Value	To same origin	To another origin	HTTPS→HTTP

Value	To same origin	To another origin	HTTPS→HTTP
`"no-referrer"`	-	-	-
`"no-referrer-when-downgrade"` or `""` (default)	full	full	-
`"origin"`	origin	origin	origin
`"origin-when-cross-origin"`	full	origin	origin
`"same-origin"`	full	-	-
`"strict-origin"`	origin	origin	-
`"strict-origin-when-cross-origin"`	full	origin	-
`"unsafe-url"`	full	full	full

Let's say we have an admin zone with URL structure that shouldn't be known from outside of the site.

If we send a `fetch`, then by default it always sends the `Referer` header with the full url of our page (except when we request from HTTPS to HTTP, then no `Referer`).

E.g. `Referer: https://javascript.info/admin/secret/paths`.

If we'd like other websites know only the origin part, not URL-path, we can set the option:

```
1  fetch('https://another.com/page', {
2    // ...
3    referrerPolicy: "origin-when-cross-origin" // Referer: https://javascript.i
4  });
```

We can put it to all `fetch` calls, maybe integrate into JavaScript library of our project that does all requests and uses `fetch` inside.

Its only difference compared to the default behavior is that for requests to another origin `fetch` sends only the origin part of the URL (e.g. `https://javascript.info`, without path). For requests to our origin we still get the full `Referer` (maybe useful for debugging purposes).

Referrer policy is not only for `fetch`

Referrer policy, described in the specification, is not just for `fetch`, but more global.

In particular, it's possible to set the default policy for the whole page using `Referrer-Policy` HTTP header, or per-link, with `<a rel="noreferrer">`.

mode

The `mode` option is a safe-guard that prevents occasional cross-origin requests:

- **`"cors"`** – the default, cross-origin requests are allowed, as described in Fetch: Cross-Origin Requests,
- **`"same-origin"`** – cross-origin requests are forbidden,
- **`"no-cors"`** – only simple cross-origin requests are allowed.

This option may be useful when the URL for `fetch` comes from a 3rd-party, and we want a "power off switch" to limit cross-origin capabilities.

credentials

The `credentials` option specifies whether `fetch` should send cookies and HTTP-Authorization headers with the request.

- **`"same-origin"`** – the default, don't send for cross-origin requests,
- **`"include"`** – always send, requires `Accept-Control-Allow-Credentials` from cross-origin server in order for JavaScript to access the response, that was covered in the chapter Fetch: Cross-Origin Requests,
- **`"omit"`** – never send, even for same-origin requests.

cache

By default, `fetch` requests make use of standard HTTP-caching. That is, it honors `Expires`, `Cache-Control` headers, sends `If-Modified-Since`, and so on. Just like regular HTTP-requests do.

The `cache` options allows to ignore HTTP-cache or fine-tune its usage:

- **`"default"`** – `fetch` uses standard HTTP-cache rules and headers,
- **`"no-store"`** – totally ignore HTTP-cache, this mode becomes the default if we set a header `If-Modified-Since`, `If-None-Match`, `If-Unmodified-Since`, `If-Match`, or `If-Range`,
- **`"reload"`** – don't take the result from HTTP-cache (if any), but populate cache with the response (if response headers allow),
- **`"no-cache"`** – create a conditional request if there is a cached response, and a normal request otherwise. Populate HTTP-cache with the response,
- **`"force-cache"`** – use a response from HTTP-cache, even if it's stale. If there's no response in HTTP-cache, make a regular HTTP-request, behave normally,
- **`"only-if-cached"`** – use a response from HTTP-cache, even if it's stale. If there's no response in HTTP-cache, then error. Only works when `mode` is `"same-origin"`.

redirect

Normally, `fetch` transparently follows HTTP-redirects, like 301, 302 etc.

The `redirect` option allows to change that:

- **`"follow"`** – the default, follow HTTP-redirects,
- **`"error"`** – error in case of HTTP-redirect,
- **`"manual"`** – don't follow HTTP-redirect, but `response.url` will be the new URL, and `response.redirected` will be `true`, so that we can perform the redirect manually to the new URL (if needed).

integrity

The `integrity` option allows to check if the response matches the known-ahead checksum.

As described in the specification, supported hash-functions are SHA-256, SHA-384, and SHA-512, there might be others depending on a browser.

For example, we're downloading a file, and we know that it's SHA-256 checksum is "abcdef" (a real checksum is longer, of course).

We can put it in the `integrity` option, like this:

```
fetch('http://site.com/file', {
  integrity: 'sha256-abcdef'
});
```

Then `fetch` will calculate SHA-256 on its own and compare it with our string. In case of a mismatch, an error is triggered.

keepalive

The `keepalive` option indicates that the request may "outlive" the webpage that initiated it.

For example, we gather statistics about how the current visitor uses our page (mouse clicks, page fragments he views), to analyze and improve user experience.

When the visitor leaves our page – we'd like to save the data at our server.

We can use `window.onunload` event for that:

```
window.onunload = function() {
  fetch('/analytics', {
    method: 'POST',
    body: "statistics",
    keepalive: true
  });
};
```

Normally, when a document is unloaded, all associated network requests are aborted. But `keepalive` option tells the browser to perform the request in background, even after it leaves the page. So this option is essential for our request to succeed.

It has few limitations:

- We can't send megabytes: the body limit for `keepalive` requests is 64kb.
  - If gather more data, we can send it out regularly in packets, so that there won't be a lot left for the last `onunload` request.
  - The limit is for all currently ongoing requests. So we can't cheat it by creating 100 requests, each 64kb.
- We can't handle the server response if the request is made in `onunload` , because the document is already unloaded at that time, functions won't work.
  - Usually, the server sends empty response to such requests, so it's not a problem.

Previous lesson Next lesson

Share

Tutorial map

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…)