Headers

- Description
  - Modification restrictions
- Constructor
- Instance methods
- Examples
- Specifications
- Browser compatibility
- See also

The **`Headers`** interface of the Fetch API allows you to perform various actions on HTTP request and response headers. These actions include retrieving, setting, adding to, and removing headers from the list of the request's headers.

You can retrieve a `Headers` object via the Request.headers and Response.headers properties, and create a new `Headers` object using the Headers() constructor. Compared to using plain objects, using `Headers` objects to send requests provides some additional input sanitization. For example, it normalizes header names to lowercase, strips leading and trailing whitespace from header values, and prevents certain headers from being set.

> **NOTE:** You can find out more about the available headers by reading our HTTP headers reference.

Description

A `Headers` object has an associated header list, which is initially empty and consists of zero or more name and value pairs. You can add to this using methods like append() (see Examples.) In all methods of this interface, header names are matched by case-insensitive byte sequence.

An object implementing `Headers` can directly be used in a for...of structure, instead of entries(): `for (const p of myHeaders)` is equivalent to `for (const p of myHeaders.entries())`.

Modification restrictions

Some `Headers` objects have restrictions on whether the set(), delete(), and append() methods can mutate the header. The modification restrictions are set depending on how the `Headers` object is created.

- For headers created with Headers() constructor, there are no modification restrictions.
- For headers of Request objects:
  - If the request's mode is `no-cors`, you can modify any CORS-safelisted request header name/value.
  - Otherwise, you can modify any non-forbidden request header name/value.
- For headers of Response objects:
  - If the response is created using Response.error() or Response.redirect(), or received from a fetch() call, the headers are immutable and cannot be modified.
  - Otherwise, if the response is created using Response() or Response.json(), you can modify any non-forbidden response header name/value.

All of the Headers methods will throw a TypeError if you try to pass in a reference to a name that isn't a valid HTTP Header name. The mutation operations will throw a `TypeError` if the header is immutable. In any other failure case they fail silently.

Constructor

- Headers()
  - : Creates a new `Headers` object.

Instance methods

- Headers.append()
  - : Appends a new value onto an existing header inside a `Headers` object, or adds the header if it does not already exist.
- Headers.delete()

- : Deletes a header from a `Headers` object.

- Headers.entries()
  - : Returns an iterator allowing you to go through all key/value pairs contained in this object.
- Headers.forEach()
  - : Executes a provided function once for each key/value pair in this `Headers` object.
- Headers.get()
  - : Returns a String sequence of all the values of a header within a `Headers` object with a given name.
- Headers.getSetCookie()
  - : Returns an array containing the values of all Set-Cookie headers associated with a response.
- Headers.has()
  - : Returns a boolean stating whether a `Headers` object contains a certain header.
- Headers.keys()
  - : Returns an iterator allowing you to go through all keys of the key/value pairs contained in this object.
- Headers.set()
  - : Sets a new value for an existing header inside a `Headers` object, or adds the header if it does not already exist.
- Headers.values()
  - : Returns an iterator allowing you to go through all values of the key/value pairs contained in this object.

> **NOTE:** To be clear, the difference between Headers.set() and Headers.append() is that if the specified header does already exist and does accept multiple values, Headers.set() will overwrite the existing value with the new one, whereas Headers.append() will append the new value onto the end of the set of values. See their dedicated pages for example code.

> **NOTE:** When Header values are iterated over, they are automatically sorted in lexicographical order, and values from duplicate header names are combined.

Examples

In the following snippet, we create a new header using the `Headers()` constructor, add a new header to it using `append()`, then return that header value using `get()`:

```
const myHeaders = new Headers();

myHeaders.append("Content-Type", "text/xml");
myHeaders.get("Content-Type"); // should return 'text/xml'
```

The same can be achieved by passing an array of arrays or an object literal to the constructor:

```
let myHeaders = new Headers({
  "Content-Type": "text/xml",
});

// or, using an array of arrays:
myHeaders = new Headers([["Content-Type", "text/xml"]]);

myHeaders.get("Content-Type"); // should return 'text/xml'
```

Specifications

Browser compatibility

See also

- ServiceWorker API
- HTTP access control (CORS)
- HTTP

□□□ #3
□ □□□ □□□ 2026-10-31 03:27:53 UTC
□ cocobook-sync □□□ 2026-11-04 17:34:44 UTC