função global fetch()

O método global `fetch()` inicia o processo de busca de um recurso na rede, retornando uma promessa que é cumprida assim que a resposta estiver disponível.

A promessa é resolvida para o `Response` objeto que representa a resposta à sua solicitação.

Uma `fetch()` promessa é rejeitada apenas quando um erro de rede é encontrado (o que geralmente ocorre quando há um problema de permissão ou algo semelhante). Uma `fetch()` promessa *não* rejeita erros de HTTP (`404` , etc.). Em vez disso, um `then()` manipulador deve verificar as propriedades `Response.ok` e/ou `Response.status` .

`WindowOrWorkerGlobalScope` é implementado por `Window` e `WorkerGlobalScope` , o que significa que o `fetch()` método está disponível em praticamente qualquer contexto no qual você queira buscar recursos.

O `fetch()` método é controlado pela `connect-src` diretiva da Política de segurança de conteúdo , e não pela diretiva dos recursos que está recuperando.

> **Observação:** os `fetch()` parâmetros do método são idênticos aos do `Request()` construtor.

Sintaxe

JS

```
fetch(resource)
fetch(resource, options)
```

Parâmetros

`resource`

Isso define o recurso que você deseja buscar. Isso pode ser:

- Uma string ou qualquer outro objeto com um stringifier — incluindo um `URL` objeto — que fornece a URL do recurso que você deseja buscar.
- Um `Request` objeto.

`options` Opcional

Um objeto que contém as configurações personalizadas que você deseja aplicar à solicitação. As opções possíveis são:

`method`

O método de solicitação, por exemplo, `"GET"` , `"POST"` . O padrão é `"GET"` . Observe que o `Origin` cabeçalho não é definido em solicitações Fetch com um método de `HEAD` ou `GET` . (Este comportamento foi corrigido no Firefox 65 — consulte o bug 1508661 do Firefox .) Qualquer string que não diferencie maiúsculas de minúsculas para um dos métodos no RFC 9110 será automaticamente maiúscula. Se você quiser usar um método personalizado (como `PATCH`), você mesmo deve colocar em letras maiúsculas.

`headers`

Quaisquer cabeçalhos que você deseja adicionar à sua solicitação, contidos em um `Headers` objeto ou um objeto literal com `String` valores. Observe que alguns nomes são proibidos .

> **Observação:** o `Authorization` cabeçalho HTTP pode ser adicionado a uma solicitação, mas será removido se a solicitação for redirecionada entre origens.

`body`

Qualquer corpo que você deseja adicionar à sua solicitação: pode ser a `Blob` , an `ArrayBuffer` , a `TypedArray` , a `DataView` , a `FormData` , a `URLSearchParams` , objeto de string ou literal ou um `ReadableStream` objeto. Esta última possibilidade ainda é experimental; verifique as informações de compatibilidade para verificar se você pode usá-lo. Observe que uma solicitação usando o método `GET` or `HEAD` não pode ter um corpo.

`mode`

O modo que você deseja usar para a solicitação, por exemplo, `cors` , `no-cors` ou `same-origin` .

`credentials`

Controla o que os navegadores fazem com as credenciais (cookies , entradas de autenticação HTTP e certificados de cliente TLS). Deve ser uma das seguintes strings:

`omit`

Informa aos navegadores para excluir credenciais da solicitação e ignorar quaisquer credenciais enviadas de volta na resposta (por exemplo, qualquer `Set-Cookie` cabeçalho).

`same-origin`

Diz aos navegadores para incluir credenciais com solicitações para URLs de mesma origem e usar quaisquer credenciais enviadas de volta em respostas de URLs de mesma origem. **Este é o valor padrão.**

`include`

Diz aos navegadores para incluir credenciais em solicitações de mesma origem e entre origens e sempre usar quaisquer credenciais enviadas de volta nas respostas.

> **Observação:** as credenciais podem ser incluídas em solicitações de origem cruzada simples e "finais", mas não devem ser incluídas em solicitações de comprovação CORS .

`cache`

Uma string indicando como a solicitação irá interagir com o cache HTTP do navegador . Os valores possíveis, `default` , `no-store` , `reload` , `no-cache` , `force-cache` e `only-if-cached` , estão documentados no artigo para a `cache` propriedade do `Request` objeto.

`redirect`

Como lidar com uma `redirect` resposta:

- `follow` : Siga redirecionamentos automaticamente. Salvo indicação em contrário, o modo de redirecionamento é definido como `follow` .
- `error` : Abortar com um erro se ocorrer um redirecionamento.
- `manual` : O chamador pretende processar a resposta em outro contexto. Consulte o padrão de busca WHATWG para obter mais informações.

`referrer`

Uma string especificando o referenciador da solicitação. Pode ser um URL de mesma origem, `about:client` ou uma string vazia.

`referrerPolicy`

Especifica a política do referenciador a ser usada para a solicitação. Pode ser um de `no-referrer`, `no-referrer-when-downgrade`, `same-origin`, `origin`, `strict-origin`, `origin-when-cross-origin`, `strict-origin-when-cross-origin` ou `unsafe-url`.

`integrity`

Contém o valor de integridade do sub-recurso da solicitação (por exemplo, `sha256-BpfBw7ivV8q2jLiT13fxDYAe2tJllusRSZ273h2nFSE=`).

`keepalive`

A `keepalive` opção pode ser usada para permitir que a solicitação sobreviva à página. Fetch com o `keepalive` sinalizador é um substituto para a `Navigator.sendBeacon()` API.

`signal`

Uma `AbortSignal` instância de objeto; permite que você se comunique com uma solicitação de busca e a interrompa, se desejar, por meio de um arquivo `AbortController`.

`priority`

Especifica a prioridade da solicitação de busca em relação a outras solicitações do mesmo tipo. Deve ser uma das seguintes strings:

- `high`: uma solicitação de busca de alta prioridade em relação a outras solicitações do mesmo tipo.
- `low`: uma solicitação de busca de baixa prioridade em relação a outras solicitações do mesmo tipo.
- `auto`: determina automaticamente a prioridade da solicitação de busca em relação a outras solicitações do mesmo tipo (padrão).

Valor de retorno

A `Promise` que resolve para um `Response` objeto.

Exceções

`AbortError` `DOMException`

A solicitação foi abortada devido a uma chamada ao método. `AbortController` `abort()`

`TypeError`

Pode ocorrer pelos seguintes motivos:

Razão	Exemplos de falha

Nome de cabeçalho inválido.	`// space in "C ontent-Type"`
	`const headers = {`
	`  'C ontent-Type': 'text/xml',`
	`  'Breaking-Bad': '<3',`
	`};`
	`fetch('https://example.com/', { headers });`

	`const headers = [`
	`  ['Content-Type', 'text/html', 'extra'],`
	`];`
	`fetch('https://example.com/', { headers });`

Razão	Exemplos de falha
Invalid URL or scheme, or using a scheme that fetch does not support, or using a scheme that is not supported for a particular request mode.	`fetch('blob://example.com/', { mode: 'cors' });`
URL includes credentials.	`fetch('https://user:password@example.com/');`
Invalid referrer URL.	`fetch('https://example.com/', { referrer: './abc\u0000df' });`
Invalid modes (`navigate` and `websocket`).	`fetch('https://example.com/', { mode: 'navigate' });`
If the request cache mode is "only-if-cached" and the request mode is other than "same-origin".	`fetch('https://example.com/', { cache: 'only-if-cached', mode: 'no-cors', });`
If the request method is an invalid name token or one of forbidden headers (`'CONNECT'` , `'TRACE'` or `'TRACK'`).	`fetch('https://example.com/', { method: 'CONNECT' });`
If the request mode is "no-cors" and the request method is not a CORS-safe-listed method (`'GET'` , `'HEAD'` , or `'POST'`).	`fetch('https://example.com/', { method: 'CONNECT', mode: 'no-cors', });`
If the request method is `'GET'` or `'HEAD'` and the body is non-null or not undefined.	`fetch('https://example.com/', { method: 'GET', body: new FormData(), });`
If fetch throws a network error.	

Examples

In our Fetch Request example (see Fetch Request live) we create a new `Request` object using the relevant constructor, then fetch it using a `fetch()` call. Since we are fetching an image, we run `Response.blob()` on the response to give it the proper MIME type so it will be handled properly, then create an Object URL of it and display it in an `<img>` element.

JS

```
const myImage = document.querySelector("img");
```

```
const myRequest = new Request("flowers.jpg");

fetch(myRequest)
  .then((response) => {
    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }

    return response.blob();
  })
  .then((response) => {
    myImage.src = URL.createObjectURL(response);
  });
```

In the Fetch with init then Request example (see Fetch Request init live), we do the same thing except that we pass in an `init` object when we invoke `fetch()`:

JS

```
const myImage = document.querySelector("img");

const myHeaders = new Headers();
myHeaders.append("Accept", "image/jpeg");

const myInit = {
  method: "GET",
  headers: myHeaders,
  mode: "cors",
  cache: "default",
};

const myRequest = new Request("flowers.jpg");

fetch(myRequest, myInit).then((response) => {
  // …
});
```

You could also pass the `init` object in with the `Request` constructor to get the same effect:

JS

```
const myRequest = new Request("flowers.jpg", myInit);
```

You can also use an object literal as `headers` in `init`.

JS

```
const myInit = {
  method: "GET",
  headers: {
    Accept: "image/jpeg",
  },
  mode: "cors",
  cache: "default",
};

const myRequest = new Request("flowers.jpg", myInit);
```

Specifications

Specification
Fetch Standard # fetch-method

Browser compatibility

Report problems with this compatibility data on GitHub

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	
`fetch`	Chrome 42	Edge 14	Firefox 39	Opera 29	Safari 10.1	Chrome 42 Android	Firefox 39 for Android	O Ar
`Authorization` header removed from cross-origin redirects	Chrome No	Edge No	Firefox 111	Opera No	Safari 16.1	Chrome No Android	Firefox 111 for Android	O Ar
Support for blob: and data:	Chrome 48	Edge 79	Firefox 39	Opera 35	Safari 10.1	Chrome 48 Android	Firefox 39 for Android	O Ar
`init.keepalive` parameter	Chrome 66	Edge 15	Firefox No	Opera 53	Safari 13	Chrome 66 Android	Firefox No for Android	O Ar
`init.priority` parameter	Chrome 101	Edge 101	Firefox No	Opera 87	Safari No	Chrome 101 Android	Firefox No for Android	O Ar
`init.referrerPolicy` parameter	Chrome 52	Edge 79	Firefox 52	Opera 39	Safari 11.1	Chrome 52 Android	Firefox 52 for Android	O Ar
`init.signal` parameter	Chrome 66	Edge 16	Firefox 57	Opera 53	Safari 11.1	Chrome 66 Android	Firefox 57 for Android	O Ar
Available in workers	Chrome 42	Edge 14	Firefox 39	Opera 29	Safari 10.1	Chrome 42 Android	Firefox 39 for Android	O Ar

Tip: you can click/tap on a cell for more information.

Full support No support Experimental. Expect behavior to change in the future. See implementation notes.

User must explicitly enable this feature. Has more compatibility info.

See also

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

This page was last modified on Jul 17, 2023 by MDN contributors.