目录

[总结 2](#_Toc508646703)

[fetch函数 2](#_Toc508646704)

[使用 2](#_Toc508646705)

[.then写法 3](#_Toc508646706)

[headers 3](#_Toc508646707)

[创建 4](#_Toc508646708)

[append 4](#_Toc508646709)

[set／get／getAll/delete 4](#_Toc508646710)

[has 4](#_Toc508646711)

[body 4](#_Toc508646712)

[Body.bodyUsed 4](#_Toc508646713)

[Body.arrayBuffer() 5](#_Toc508646714)

[Body.blob() 5](#_Toc508646715)

[Body.formData() 5](#_Toc508646716)

[Body.json() 5](#_Toc508646717)

[Body.text() 5](#_Toc508646718)

[request 5](#_Toc508646719)

[构造拷贝 6](#_Toc508646720)

[跨域mode 6](#_Toc508646721)

[属性 7](#_Toc508646722)

[Request.method 7](#_Toc508646723)

[Request.url 7](#_Toc508646724)

[Request.headers 7](#_Toc508646725)

[Request.context 8](#_Toc508646726)

[Request.referrer 8](#_Toc508646727)

[Request.mode 8](#_Toc508646728)

[Request.credentials 8](#_Toc508646729)

[Request.integrity 8](#_Toc508646730)

[Request.cache 9](#_Toc508646731)

[方法 9](#_Toc508646732)

[Request.clone() 9](#_Toc508646733)

[response 9](#_Toc508646734)

[构造 9](#_Toc508646735)

[属性 9](#_Toc508646736)

[Response.type 9](#_Toc508646737)

[Response.url 10](#_Toc508646738)

[Response.useFinalURL 10](#_Toc508646739)

[Response.status 10](#_Toc508646740)

[Response.ok 10](#_Toc508646741)

[Response.statusText 10](#_Toc508646742)

[Response.headers 11](#_Toc508646743)

[方法 11](#_Toc508646744)

[Response.clone() 11](#_Toc508646745)

[Response.error() 11](#_Toc508646746)

[Response.redirect() 11](#_Toc508646747)

[axios 11](#_Toc508646748)

# 总结

发送请求或者获取资源，需要使用 [GlobalFetch.fetch](https://developer.mozilla.org/zh-CN/docs/Web/API/GlobalFetch/fetch) 方法。它在很多接口中都被实现了，比如 [Window](https://developer.mozilla.org/zh-CN/docs/Web/API/Window) 和 [WorkerGlobalScope](https://developer.mozilla.org/zh-CN/docs/Web/API/WorkerGlobalScope)。所以在各种环境中都可以用这个方法获取到资源。

用async/await, 避免一直.then

Fetch引入了3个接口，它们分别是 Headers,Request 以及 Response 。他们直接对应了相应的HTTP概念，但是基于安全考虑，有些区别，例如支持CORS规则以及保证cookies不能被第三方获取。

# fetch函数

fetch()函数的参数和传给Request()构造函数的参数保持完全一致

## 使用

fetch(myRequest) .then(function(response) {

## .then写法

fetch("http://www.example.org/submit.php", {

method: "POST",

headers: {

"Content-Type": "application/x-www-form-urlencoded"

},

body: "firstName=Nikhil&favColor=blue&password=easytoguess"

}).then(**function**(res) {

**if** (res.ok) {

alert("Perfect! Your settings are saved.");

} **else** **if** (res.status == 401) {

alert("Oops! You are not authorized.");

}

}, **function**(e) {

alert("Error submitting form!");

});

# headers

Headers对象有一个特殊的 guard 属性。这个属性没有暴露给Web，但是它影响到哪些内容可以在Headers对象中被改变。

如果使用了一个不合法的HTTP Header属性名，那么Headers的方法通常都抛出 TypeError 异常。如果不小心写入了一个不可写的属性，也会抛出一个 TypeError 异常。除此以外的情况，失败了并不抛出异常。

## 创建

**var** reqHeaders = **new** Headers();

## append

reqHeaders.append("Content-Length", content.length.toString());

## set／get／getAll/delete

reqHeaders.**set**("Content-Type", "text/html");

console.log(reqHeaders.getAll("X-Custom-Header")); *// ["ProcessThisImmediately", "AnotherValue"]*

reqHeaders.delete("X-Custom-Header");

### has

# body

### Body.bodyUsed

只读

包含一个指示body是否被读取过的 Boolean 值。

### Body.arrayBuffer()

使用一个buffer数组来读取 Response流中的数据，并将bodyUsed状态改为已使用。

### Body.blob()

使用一个Blob对象来读取 Response流中的数据，并将bodyUsed状态改为已使用。

### Body.formData()

使用一个 FormData 对象来读取 Response流中的数据，并将bodyUsed状态改为已使用。

### Body.json()

使用一个 JSON 对象来读取 Response流中的数据，并将bodyUsed状态改为已使用。

### Body.text()

使用一个USVString (文本) 对象来读取 Response流中的数据，并将bodyUsed状态改为已使用。

# request

url + json参数。

参数需要method和headers，同时Request也接受一个特定的body，mode，credentials以及cache hints.

继承至body。

## 构造拷贝

**var** req = **new** Request("/index.html");

**var** copy = **new** Request(req);

**var** uploadReq = **new** Request("/uploadImage", {

method: "POST",

headers: {

"Content-Type": "image/png",

},

body: "image data"

});

## 跨域mode

mode 属性用来决定是否允许跨域请求，以及哪些response属性可读。可选的mode属性值为 same-origin ， no-cors （默认）以及 cors 。

same-origin 模式很简单，如果一个请求是跨域的，那么返回一个简单的error，这样确保所有的请求遵守同源策略。

no-cors 模式允许来自CDN的脚本、其他域的图片和其他一些跨域资源，但是首先有个前提条件，就是请求的method只能是"HEAD","GET"或者"POST"。此外，任何 ServiceWorkers 拦截了这些请求，它不能随意添加或者改写任何headers，除了 [***这些***](https://fetch.spec.whatwg.org/#simple-header)。第三，JavaScript不能访问Response中的任何属性，这保证了 ServiceWorkers 不会导致任何跨域下的安全问题而隐私信息泄漏。

cors 模式我们通常用作跨域请求来从第三方提供的API获取数据。这个模式遵守 [***CORS协议***](https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS) 。只有有限的 [***一些***](https://fetch.spec.whatwg.org/#concept-filtered-response-cors) headers被暴露给Response对象，但是body是可读的。

## 属性

所有属性都只读。

### Request.method

请求使用的方法 (GET, POST, 等.)

### Request.url

请求使用的 URL。

### Request.headers

请求所关联的 Headers 对象。

### Request.context

请求的上下文 例如：(例如：audio, image, iframe, 等)

是一个非常大的枚举集合定义了获得的资源的种类，它可能是 image 比如请求来自于img标签，可能是 worker 如果是一个worker脚本，等等。如果使用fetch()函数，这个值是 fetch 。

### Request.referrer

请求的来源 (例如：client).

### Request.mode

请求的模式 (例如： cors, no-cors, same-origin).

### Request.credentials

决定了cookies是否能跨域得到。这个属性与XHR的withCredentials标志相同，但是只有三个值，分别是"omit"（默认）,"same-origin"以及"include"。Request.redirect

如何处理重定向模式 (例如： follow, error, or manual)

### Request.integrity

请求内容的 subresource integrity 值 (例如： sha256-BpfBw7ivV8q2jLiT13fxDYAe2tJllusRSZ273h2nFSE=).

### Request.cache

请求的缓存模式 (例如： default, reload, no-cache).

Request implements Body, so it also has the following property available to it:

## 方法

### Request.clone()

创建当前request的副本。

# response

通常在fetch()的回调中获得。但是它们也可以用JS构造，不过通常这招只用于ServiceWorkers。

## 构造

Response有个接收两个可选参数的构造器。第一个参数是返回的body，第二个参数是一个json，设置status、statusText以及headers。

**new** Response("Response body", {

headers: { "Content-Type" : "text/plain" }

});

## 属性

### Response.type

包含Response的类型 (例如, basic, cors).

* "basic": 正常的，同域的请求，包含所有的headers除开"Set-Cookie"和"Set-Cookie2"。
* "cors": Response从一个合法的跨域请求获得， [***一部分header和body***](https://fetch.spec.whatwg.org/#concept-filtered-response-cors) 可读。
* "error": 网络错误。Response的status是0，Headers是空的并且不可写。当Response是从Response.error()中得到时，就是这种类型。
* "opaque": Response从"no-cors"请求了跨域资源。依靠Server端来做限制。

"error"类型会导致fetch()函数的Promise被reject并回调出一个TypeError。

### Response.url

包含Response的URL.

### Response.useFinalURL

包含了一个布尔值来标示这是否是该Response的最终URL.

### Response.status

包含Response的状态码 (例如, 200 成功).

### Response.ok

包含了一个布尔值来标示该Response成功(状态码200-299) 还是失败.

### Response.statusText

包含了与该Response状态码一致的状态信息 (例如, OK对应200).

### Response.headers

包含此Response所关联的Headers 对象.

它是只读的(with guard "response")，url属性是当前Response的来源URL。

## 方法

### Response.clone()

创建一个Response对象的克隆

### Response.error()

返回一个绑定了网络错误的新的Response对象

### Response.redirect()

用另一个URL创建一个新的 response.

# axios

**const** response = **await** axios.post(urls.insert, data, {  
 headers: { 'Cube-Authorization': token },  
});  
payload = response.data.payload;