






Em JavaScript, como devo usar / assíncrono / aguardar com XMLHttpRequest?

Perguntou 2 anos, 4 meses ago Ativo há 6 meses Vista 26k vezes

37.     9 

Divulgação completa: eu me qualificaria como tendo conhecimento intermediário de JavaScript. Portanto, isso está um pouco acima do meu nível de experiência no momento.

Eu tenho uma extensão do Google Chrome que faz uma solicitação AJAX para um local `file:///` assim que a página é carregada. Depois de receber a resposta novamente da solicitação, uso o código retornado em várias funções posteriormente no meu código. Na maioria das vezes, recebo a resposta antes que meu código precise ser executado. Mas às vezes eu não faço e tudo quebra.

Agora, suponho que eu poderia lançar todo o código relevante dentro do `xhr.onload` abaixo. Mas isso parece realmente ineficiente? Eu tenho muitas partes móveis que dependem da resposta e parece ruim colocá-las todas lá.

Examinei vários artigos relacionados a `async` / `wait` e estou tendo problemas para entender o conceito. Também não sou 100% positivo. Estou vendo isso da maneira certa. Eu deveria estar pensando em usar `async` / `waitit`?

Aqui está o código para minha solicitação AJAX.

```
var xhr = new XMLHttpRequest();
xhr.open("GET", url, true);
xhr.onload = function(e) {
  code = xhr.response;
};
xhr.onerror = function () {
  console.error("*** An error occurred during the XMLHttpRequest");
};
xhr.send();
```

Digamos que eu tenha várias funções que precisam ser acionadas posteriormente no meu código. No momento, eles se parecem com:

```
function doTheThing(code) {
  // I hope the response is ready.
}
```

Qual é a melhor maneira de abordar isso? Para sua informação, a `Fetch` API não é uma opção.

Aqui está uma visão de alto nível de como meu código está estruturado.

```
// AJAX request begins.

// ...

// A whole bunch of synchronous code that isn't dependant on
// the results of my AJAX request. (eg. Creating and appending
// some new DOM nodes, calculating some variables) I don't want
// to wait for the AJAX response when I could be building this stuff instead.

// ...

// Some synchronous code that is dependant on both my AJAX
// request and the previous synchronous code being complete.

// ...

// Some more synchronous code that needs the above line to
// be complete.
```


javascript espera assíncrona xmlhttprequest

editada Feb 25 '18 às 2:55

perguntou Feb 25 '18 às 1:51

 jkucpczak
2.168 ● 8 ● 27 ● 51

1 Have you considered using [Fetch](#) instead? It's Promise-based from the start. – E. Sundin Feb 25 '18 at 1:53 

Putting code into a callback has absolutely no bearing on efficiency or performance. It's just code, and a callback is just a callback. The code is either performant or not. – Pointy Feb 25 '18 at 1:54 

1 to use XMLHttpRequest with `async/await`, you'll need to make a Promise – Jaromanda X Feb 25 '18 at 1:56

1 Just call `doTheThing(code)` from inside the `onload` function. – 4castle Feb 25 '18 at 1:57 

@E.Sundin `Fetch` doesn't work with local `file:///` files which is what I need. @JaromandaX That's what I figured. Having trouble getting that to work though. – jkucpczak Feb 25 '18 at 2:02

[Report this ad](#)

4 Answers

| | | |
|--------|--------|-------|
| Active | Oldest | Votes |
|--------|--------|-------|

I usually do async/await like this:

33

```
async function doAjaxThings() {  
  // await code here  
  let result = await makeRequest("GET", url);  
  // code below here will only execute when await makeRequest() finished loading  
  console.log(result);  
}  
document.addEventListener("DOMContentLoaded", function () {  
  doAjaxThings();  
  // create and manipulate your DOM here. doAjaxThings() will run asynchronously and  
  // not block your DOM rendering  
  document.createElement("...");  
  document.getElementById("...").addEventListener(...);  
});
```

[Promisified xhr function](#) here:

```
function makeRequest(method, url) {  
  return new Promise(function (resolve, reject) {  
    let xhr = new XMLHttpRequest();  
    xhr.open(method, url);  
    xhr.onload = function () {  
      if (this.status >= 200 && this.status < 300) {  
        resolve(xhr.response);  
      } else {  
        reject({  
          status: this.status,  
          statusText: xhr.statusText  
        });  
      }  
    };  
    xhr.onerror = function () {  
      reject({  
        status: this.status,  
        statusText: xhr.statusText  
      });  
    };  
    xhr.send();  
  });  
}
```

edited Jan 9 at 14:52

 Bergi

461k ● 89 ● 745 ● 1089

answered Feb 25 '18 at 2:09

 Thắng Trần Xuân

459 ● 4 ● 12

Uncaught SyntaxError: await is only valid in async function – [jkupczak](#) Feb 25 '18 at 2:15

Like what it says, you need to put any `await` on async function. Just add `async` before the `function` text. `async function doTheThing(code) { let result = await makeRequest("GET", url); console.log(result); }` – [Thắng Trần Xuân](#) Feb 25 '18 at 2:17

Thanks for the explanation. I've edited my question above to give a brief high level view of how my code is currently structured. Would you still recommend your answer to me based on that? – [jkupczak](#) Feb 25 '18 at 2:57

I just updated my example. But if you want your 2 or more independent tasks to run in parallel (asynchronously), and run some other code only after all the tasks is done, check out Promise.all(). developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/... – [Thắng Trần Xuân](#) Feb 25 '18 at 3:55

2 @ThắngTrầnXuân If I could upvote your answer every day, I would!!! You have just ended the most frustrating development session I have experienced in years! Thank you! Thank you! Thank you! – [Shawn de Wet](#) Jun 26 '19 at 7:56

I create a promise for the XHR. Then simply use `await` inside an `async` function to call it.

13

```
function getHTML(url) {  
  return new Promise(function (resolve, reject) {  
    var xhr = new XMLHttpRequest();  
    xhr.open('get', url, true);  
    xhr.responseType = 'document';  
    xhr.onload = function () {
```

Ao usar nosso site, você reconhece que leu e entendeu nossa [Política de Cookies](#) , [Política de Privacidade](#) e nossos [Termos de Serviço](#) .

```

    } else {
      reject(status);
    }
  };
  xhr.send();
});
}

async function schemaPageHandler(){
  try {
    var parser = new window.DOMParser();
    var remoteCode = await getHTML('https://schema.org/docs/full.html');
    var sourceDoc = parser.parseFromString(remoteCode, 'text/html');
    var thingList = sourceDoc.getElementById("C.Thing");
    document.getElementById("structured-data-types").appendChild(thingList);
  } catch(error) {
    console.log("Error fetching remote HTML: ", error);
  }
}

```

answered Dec 22 '18 at 16:58



Ronnie Royston

10.2k ● 4 ● 43 ● 64

thanks for the comprehensive example. I am not so sure about the reject() inside onload method. What is for sure, if you really want to handle all errors there should be onerror handler. But I didn't find definitive answer for if onload is fired for cases other than those ending with 2xx status (OK). [There are tests saying that it does not happen](#), but on the other hand in first example [in this docs, they do check status](#) – [papo](#) Dec 15 '19 at 4:30

Why using DOMParser here? When you used `xhr.responseType = 'document'`; the `xhr.response` is already of type HTMLDocument. If you instead resolve with `resolve(xhr.response)` then you'll get directly what you're now getting to `sourceDoc`. Seems to me now you're DOM parsing in XHR then stringifying (innerHTML) the result and again DOM parsing it. Or am I missing something? – [papo](#) Dec 16 '19 at 18:58

You get two options,

first is to use newer `fetch` api which is promise based, with which you can do

```

let response = await fetch(url);
response = await response.json(); // or text etc..
// do what you wanna do with response

```

Other option if you really want to use XMLHttpRequest is to promisify it

```

let response = await new Promise(resolve => {
  var xhr = new XMLHttpRequest();
  xhr.open("GET", url, true);
  xhr.onload = function(e) {
    resolve(xhr.response);
  };
  xhr.onerror = function () {
    resolve(undefined);
    console.error("*** An error occurred during the XMLHttpRequest");
  };
  xhr.send();
});
// do what you wanna do with response

```

possible full solution

```

(async () => {
  let response = await new Promise(resolve => {
    var xhr = new XMLHttpRequest();
    xhr.open("GET", url, true);
    xhr.onload = function(e) {
      resolve(xhr.response);
    };
    xhr.onerror = function () {
      resolve(undefined);
      console.error("*** An error occurred during the XMLHttpRequest");
    };
    xhr.send();
  });
  doTheThing(response)
})();

```

edited Feb 25 '18 at 2:23

answered Feb 25 '18 at 2:04



Jibby Jose

2,460 ● 2 ● 18 ● 29

Thanks for the answer. Unfortunately, `fetch` doesn't allow `file:///` which is what I need. I'll try your `XMLHttpRequest` option and see if that works out for me. – [jkupczak](#) Feb 25 '18 at 2:05

I'm getting this error `Uncaught SyntaxError: await is only valid in async function` – [jkupczak](#) Feb 25 '18 at 2:08

@jkupczak you need to have this bits inside an async function eg: `async function fn () {}`, if you are doing this in the top level, you can put the code into a async IIFE `(async function fn() {})() – Jibby Jose` Feb 25 '18 at 2:11

When you say `// do what you wanna do with response`, how do I do that? I've got a function later on that I only want to fire once the response comes back. I don't know how to code that – [jkupczak](#) Feb 25 '18 at 2:19

Ao usar nosso site, você reconhece que leu e entendeu nossa [Política de Cookies](#), [Política de Privacidade](#) e nossos [Termos de Serviço](#).



You can for example create an asynchronous class to use instead of the original one. It lacks some methods but it can serve as an example.

```
(function() {
    "use strict";

    var xhr = Symbol();

    class XMLHttpRequestAsync {
        constructor() {
            this[xhr] = new XMLHttpRequest();
        }
        open(method, url, username, password) {
            this[xhr].open(method, url, true, username, password);
        }
        send(data) {
            var sxhr = this[xhr];
            return new Promise(function(resolve, reject) {
                var errorCallback;
                var loadCallback;

                function cleanup() {
                    sxhr.removeEventListener("load", loadCallback);
                    sxhr.removeEventListener("error", errorCallback);
                }

                errorCallback = function(err) {
                    cleanup();
                    reject(err);
                };

                loadCallback = function() {
                    resolve(xhr.response);
                };

                sxhr.addEventListener("load", loadCallback);
                sxhr.addEventListener("error", errorCallback);


                sxhr.addEventListener("load", function load() {
                    sxhr.removeEventListener("load", load);
                    resolve(sxhr.response);
                });
                sxhr.send(data);
            });
        }
        set responseType(value) {
            {
                this[xhr].responseType = value;
            }
        }
        setRequestHeader(header, value) {
            this[xhr].setRequestHeader(header, value);
        }
    }

    addEventListener("load", async function main() {
        removeEventListener("load", main);

        var xhr = new XMLHttpRequestAsync();
        xhr.responseType = "json";
        xhr.open("GET", "appserver/main.php/" + window.location.hash.substring(1));
        console.log(await xhr.send(null));
    });
})();
```

[Run code snippet](#)[Expand snippet](#)

answered Sep 17 '18 at 18:06

 [user2704215](#)
11 ● 1