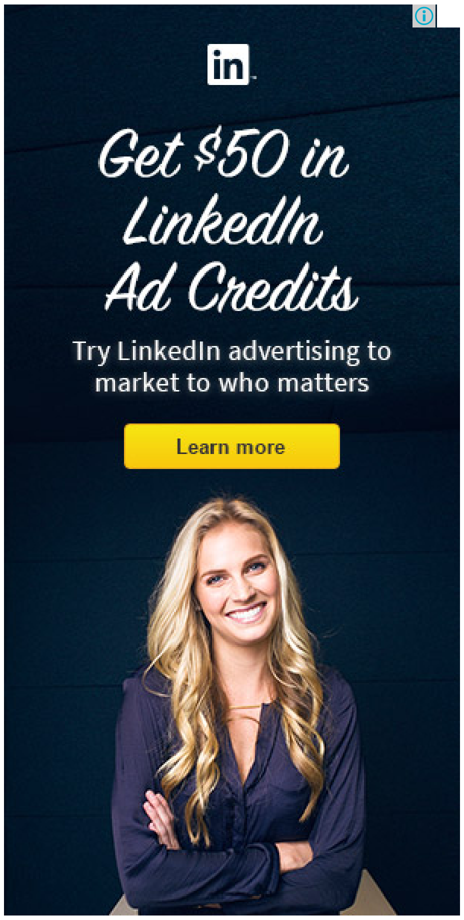
Understanding Synchronous and Asynchronous in JavaScript – Part 2

Published by Preethi Ranjit, in Coding


in
Get $50 in LinkedIn Ad Credits
Try LinkedIn advertising to market to who matters
Learn more

In the first part of this post, we saw how the **concepts of synchronous and asynchronous** are perceived **in JavaScript**. In this second part, Mr X appears again to help us understand **how the setTimeout and AJAX APIs work**.

Understanding Synchronous and Asynchronous JavaScript – Part 1

Synchronous and asynchronous are confusing concepts in JavaScript, especially for beginners. Two or more things are synchronous when…

Read more

You might also like

An odd request

Let's rewind back to the story of Mr X and the movie you want to leave for. Say you leave a task for Mr X before the outing, and tell him that he can only begin to work on this task **five hours *after* he got your message**.

He isn't happy about it, remember, he doesn't take a new message until he is done with the current one, and if he takes yours, he **has to wait for *five* hours** to even start on the task. So, to not be wasteful of time, he **brings in a helper**, Mr H.

Instead of waiting, he asks Mr H to **leave a new message for the task in the queue** after the given hours had passed, and moves on to the next message.

Five hours past; Mr H **updates the queue** with a new message. After he's done processing all the accrued messages prior to Mr H's, Mr X **carries out your requested task**. So, this way, you can leave a request to be **complied upon at a later time**, and not wait until it's fulfilled.

But why does Mr H leave a message in the queue instead of directly contacting Mr X? Because as I mentioned in the first part, the *only* way to contact Mr X is **by leaving a message to him** via phone call — no exceptions.

1. The `setTimeout()` method

.Net AngularJS Jobs

Looking for a Job or Seeking a New Career? Head to www.monsterindia.com to Find Better.

monsterindia.com

Suppose you have a set of code that you want to **execute after a certain time**. In order to do that, you just **wrap it in a function**, and **add it to a `setTimeout()` method** along with the delay time. The syntax of `setTimeout()` is as follows:

```
setTimeout(function, delay-time, arg...)
```

Report a bug

Fresh Resources for Web Developers – January 2017
Thoriq Firdaus

Create Beautiful Gradient Transitions with Granim.js
Jake Rocheleau

CMS.js – The Newest Free JavaScript Site Generator
Jake Rocheleau

Fresh Resources for Web Developers – December 2016
Thoriq Firdaus

10 Sites to Download High Quality Stock Videos for Free
Rean

Meet Yarn: The Newest JavaScript Package Manager Developed By Facebook
Jake Rocheleau

How to Add Styles to TinyMCE & Markdown Content
Jake Rocheleau

Build Powerful 3D Browser Games with BabylonJS
Jake Rocheleau

React.js: 14 Tools & Resources for Web Developers
Anna Monus

The `arg...` parameter stands for any argument the function takes, and `delay-time` is to be added in milliseconds. Below you can see a simple code example, that outputs “hey” in the console after 3 seconds.

```
setTimeout( function() { console.log('hey') }, 3000 );
```

Ready to move in Prestige Homes at
3.99%
Interest*
For more information, call:
Toll Free 1800 313 0080
or Log on to www.prestigeconstructions.com/offers
Projects Approved by
HDFC HOME LOANS
OFFER 1ST FEB TO 12TH FEB

Once `setTimeout()` starts running, **instead of blocking the call stack** until the indicated delay time is over, a **timer is triggered**, and the call stack is gradually emptied for the next message (similarly to the correspondence between Mr X and Mr H).

When the timer expires, a new message **joins the queue**, and the event loop picks it up when the call stack is free after processing all the messages before it — thus the code runs asynchronously.

2. AJAX

AJAX (Asynchronous JavaScript and XML) is a concept that uses the `XMLHttpRequest` (XHR) API to **make server requests** and **handle the responses**.

When browsers make server requests without using XMLHttpRequest, the **page refreshes** and **reloads its UI**. When the processing of requests and responses are handled by the XHR API, and **UI remains unaffected**.

Report a bug

So, basically the goal is to **make requests without page reloads**. Now, where is the "asynchronous" in this? Just using XHR code (which we'll see in a moment) doesn't mean it's AJAX, because the XHR API can **work in both synchronous and asynchronous ways**.

XHR **by default** is set to **work asynchronously**; when a function makes a request using XHR, it **returns without waiting for the response**.

If XHR is configured to **be synchronous**, then the function waits until the **response is received and processed** before returning.

Code Example 1

This example presents an `XMLHttpRequest` **object creation**. The `open()` method, validates the request URL, and the `send()` method sends the request.

```
Asynchronous XHR
var xhr = new XMLHttpRequest();
xhr.open("GET", url);
xhr.send();
```

Any direct access to the response data after `send()` will be in vain, because `send()` **doesn't wait** until the request is completed. Remember, XMLHTTPRequest is set to work asynchronously by default.

Code Example 2

The `hello.txt` file in this example is a simple text file containing the text 'hello'. The `response` property of XHR is invalid, since it didn't output the text 'hello'.

```
Asynchronous XHR
var xhr = new XMLHttpRequest();
xhr.open("GET", "he
```

Report a bug

```
xhr.send();
document.write(xhr.response);
// empty string
```

XHR implements a micro-routine that **keeps checking for response** in every millisecond, and **triggers complimentary events** for the different states a request goes through. When the response is loaded, **a load event is triggered by XHR**, which can deliver a valid response.

```
Asynchronous XHR
var xhr = new XMLHttpRequest();
xhr.open("GET", "hello.txt");
xhr.send();
xhr.onload = function(){ document.write(this.response) }
// writes 'hello' to the document
```

The response inside the load event **outputs 'hello'**, the correct text.

Going the asynchronous way is preferred, as it doesn't block other scripts until the request is completed.

If the response has to be processed synchronously, we pass `false` as the last argument of `open`, which **flags the XHR API** saying it **has to be synchronous** (by default the last argument of `open` is `true`, which you needn't explicitly specify).

```
Synchronous XHR
var xhr = new XMLHttpRequest();
xhr.open("GET", "hello.txt", false);
xhr.send();
document.write(xhr.response);
// writes 'hello' to document
```

Why learn all this?

Almost all beginners make some mistakes with asynchronous concepts such as `setTimeout()` and AJAX, for example by assuming `setTimeout()` executes code after the delay time, or by processing response directly inside a function making an AJAX request.

If you know how the puzzle fits, you can **avoid such confusion**. You know that the delay time in `setTimeout()` does not indicate the time **when the code execution starts**, but the time **when the timer expires** and a new message is queued, which will only be processed when the call stack is free to do so.

Report a bug

0 Comments

Sort by Newest

Add a comment...

Facebook Comments Plugin

Report a bug