02.12.2022 JavaScript Events, Callback, Promises, Fetch

Content

customevents.js

- HMTL: /1_customevents.html
- Javascript: /js/1_customevents.js
- online info: HOW TO: addEventListener
- online info: get innerhtml from document

A little script to show the functionality on input/output functionality build inbetween the HTML and linked JS scripts with events.
Basically a textfield is provided by the HTML, beneath it is a empty container. The written text is just a small instruction to use the HTML in a browser correctly.
Now let'S look at the Javascript-script(pleonasm, i know...).
We construct a new Event called 'myEvent'. Then we put an eventListener on the whole document, which is triggers when a key is pressed and if the key is 'Enter', then it takes the written text from the input field and sends it back to the HTML to be written beneath the input field.

How do events work? CustomEvents are linked to an eventlistener, which takes the CustomEvent as Parameter for input Data example to use Events as an abstraction.
Of course the same problems can be soved using functions with proper return values.
But Events are asynchronous in comparison to singlethreaded functions. Ergo costly code should be put in Events to enhance performance.

eventloop.js

- HMTL: /2_eventloop.html
- Javascript: /js/2_eventloop.js

- online info: HOW TO: addEventListener
- online info: get innerhtml from document

A HTML with 3 buttons, the instructions are in German, but kept short and simple for anyone capable of German easy to follow. The Javascript file behind it is a little more complex to that.
Firstly we have a resource demanding function with has a small but high number loop which calculates '0' or '1' in a random occurence. It works like a coinflip and feeds a counter how many '0's or '1's got calculated.
It doesn't really feed any real purpose other than demonstrate that 'random' really means random.
It is interpreted as such that the result numbers would be similar or even equal (for example alsways '0') it wouldn't be random.

To further demonstrate the queue handling in Javascript Button1 is set up with different operations, including triggering the event for Button2 and Button3.
Javascript puts all operations in queue as soon as the interpreter reads the operation. That's the Default. But is it a function then it excludes the content and interprets it when finished with the first go through.
That logic is recursiv how deep the funcion is embedded in other functions. Is an interval or timeout set to trigger the execution of the wrapped code, then it is executed as soon as the interval or timeout calls for the function.
The timer for interval and timeout starts as soon as the Javascript interpreter comes across the line. (so be aware the timer doesn't necessary start when the file is loaded).
PS: Look out for Jim!

async-testing.js

- HMTL: /async-testing.html
- Javascript: /js/async-testing.js
- online info: HOW TO: addEventListener
- online info: get innerhtml from document
- online info: get multithreaded code to wait
- online info: use of callback
- online info: use of promise
- online info: use of fetch

In this exercise the HTML document is completely empty except for the declaration of head and body. All visible items are generated through the Javascript document. We have the same demanding operation from the previous exercise, but run in in a callback function. A callback is asynchronous, which means the rest of the script will be processed while the callback is outsourced to another core. We also get a template to load very demanding images efficiently. We fetch at the very first smaller versions, less detailed versions of the images to quickly present the webpage for a user. With a promise we preload the bigger sized

versions in the background, when IF the page needs to access the new images they are already loaded. Now the main difference between Callback and Promise is the errorhandling.
A Callback gets his task to operate, when its finished it immediatly throws you the result, but it will never give you a response if there are problems. As example: “I will Call you back WHEN I get the result!”
A Promise gives you a result no matter what. The constructor works in 2 pathways => return valid and return invalid. Either way with the abstract instance of both results can be further worked on.
In the working script the successfull operation has to be finished within 5 seconds AND work without error.
All images are loaded with “fetch” fetch is asynchornous and written in cascading steps, to first finish a step before proceeding with the next step to avoid null referencing(variable otherwise might be used before its value is loaded).
The funcions asynch/await are implemented to use the function “fetchImage” within another function.
PS: function “readJson” wasn’t required and was included on accident.

progress.js

- HMTL: /async-testing.html
- Javascript: /js/async-testing.js
- online info: HOW TO: use ajax
- online info: get innerhtml from document
- online info: custom events

Hereby the HTML is also empty and we generate the needed tags in the javascript file.
We create a progress bar with grey background and a red loading bar(inspiration youtube).
It is linked to a custom event to handle the current stage of the loading progress. With the onprogress function we get nearly realtime feedback about the progress of a file.
If the bar reaches 100% width it will be removed.
We load a Json file with ajax syntax. The process is handles with a promise to handle errors with the connection.
The connection builds up with ‘ajax’.open and will be deconstruted after ‘ajax’.send. A XMLHttpRequest/Ajax gives while deployed feedback on its status. We check beforehand if the connection is “200 - OK” stop prematurely if there is an error. With promise we can wait with the next step till the whole Json is loaded, then we dispatch the function for errorhandling or the function when successful. This function futhermore registers the output as “entries” and we can access it like an array.

storage.js

cookie.js

watertracking.js

Author Benjamin Lamprecht, 02.12.2022