SNS COLLEGE OF ENGINEERING

An Autonomous Institution

Coimbatore-107

19TS601-FULL STACK DEVELOPMENT

UNIT-1

JAVASCRIPT AND BASICS OF MERN STACK

Mutation observer - Event loop: microtasks and macrotasks

Mutation Observer

- A built-in object that observes a DOM element, firing a callback in case of modifications is known as MutationObserver.
- The first step should be creating an observer using a callback-function, like this:
  - let observer = new MutationObserver(callback);
- The next step is attaching it to a DOM node, as follows:
  - observer.observe(node, config);

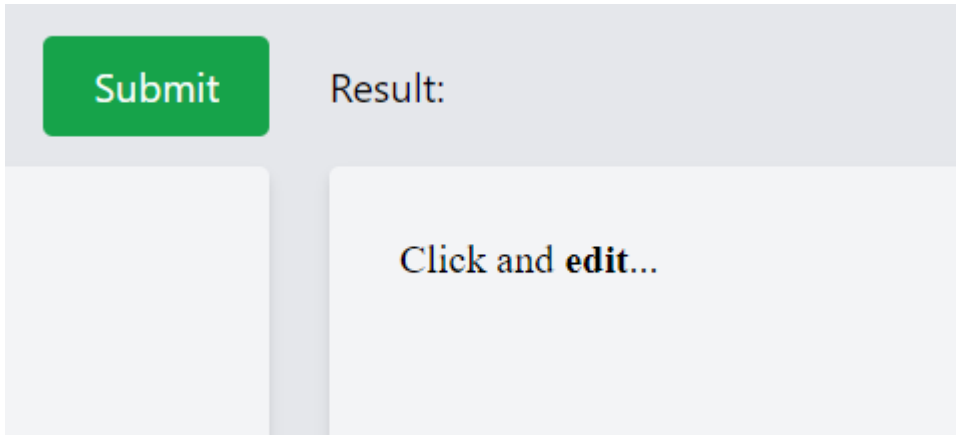
- Config is an object that has options specifying the changes to respond to:
  - childList: modifications in the direct children of the node.
  - subtree: inside all the node descendants.
  - attributes: the node attributes.
  - attributeFilter: an array of attribute names for observing only the selected ones.
  - characterData: to observe the node.data or not.

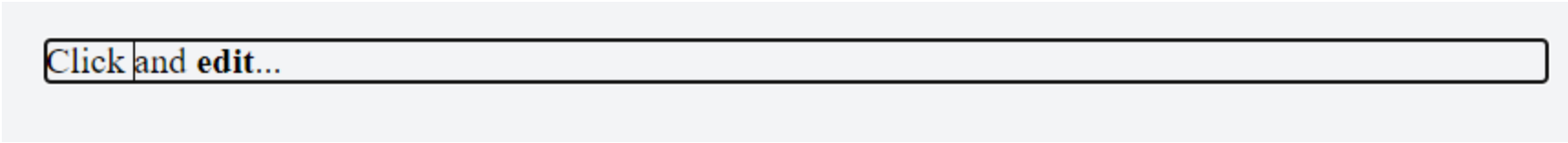
- The callback is run after any changes. The changes are transferred to the first argument as a list of MutationRecord objects, and the observer becomes the second object.
- The MutationRecord object includes the following properties:
  - type: the type of mutation. It is one of "attributes", "characterData", and "childList".
  - target: that's where the change happens.

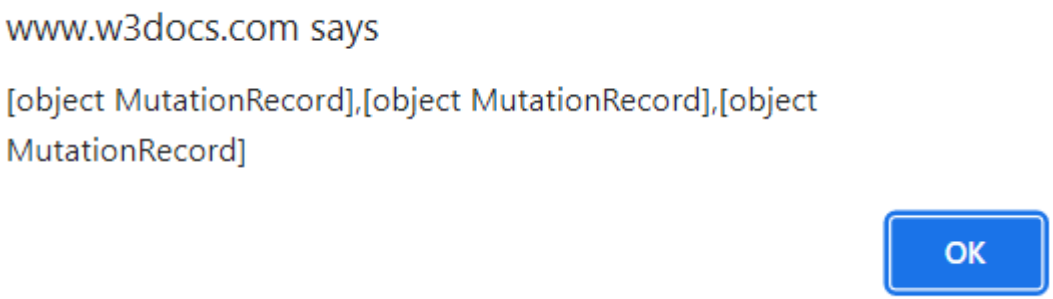
- addedNodes/removedNodes: the added or removed nodes.
- previousSibling/nextSibling: the previous/next sibling to added or removed nodes.
- attributeName/attributeNamespace: the changed attribute name or namespace.
- oldValue: the previous value merely for text or attribute changes, in case the matching option is set attributeOldValue/characterDataOldValue.

- MutationObserver can respond to changes inside DOM: added and removed elements, text content, and attributes.
- It can be used for tracking changes represented by other parts of the code, as well as for integrating with third-party scripts.
- MutationObserver is capable of tracking any changes.

```
<html>
 <body>
  <div contentEditable id="elemId">Click and <b>edit</b>...</div>
  <script>
   let observer = new MutationObserver(mutationRecords => {
    alert(mutationRecords); // alert(the changes)
   });
   // observe everything except attributes
   observer.observe(elemId, {
    childList: true, // observe direct children
    subtree: true, // lower descendants too
    characterDataOldValue: true, // pass old data to callback
   });
  </script>
 </body>
</html>
```


Submit
Result:
Click and **edit**...

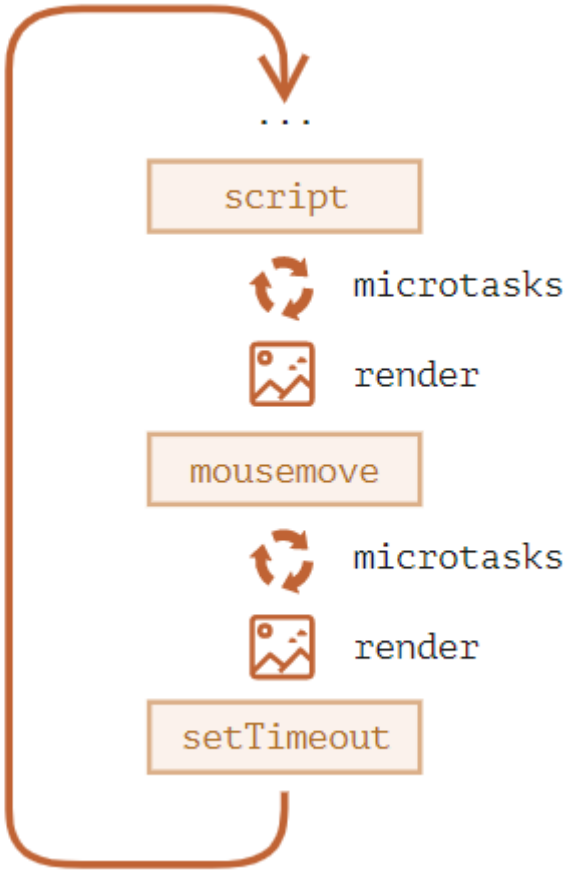

Click and **edit**...


www.w3docs.com says
[object MutationRecord],[object MutationRecord],[object MutationRecord]
OK

JavaScript Event Loop: microtasks and macrotasl

- A macro task is a collection of distinct and independent tasks.
- Microtasks are minor tasks that update the state of an application and should be completed before the browser moves on to other activities, such as re-rendering the user interface.
- Promise callbacks and DOM modification changes are examples of microtasks.

event loop


...
script
microtasks
render
mousemove
microtasks
render
setTimeout

- A more detailed event loop algorithm (though still simplified compared to the specification):
  - Dequeue and run the oldest task from the macrotask queue (e.g. “script”).
  - Execute all microtasks:
    - While the microtask queue is not empty:
    - Dequeue and run the oldest microtask.
  - Render changes if any.
  - If the macrotask queue is empty, wait till a macrotask appears.
  - Go to step 1.

To schedule a new macrotask:

- Use zero delayed setTimeout(f).

- That may be used to split a big calculation-heavy task into pieces, for the browser to be able to react to user events and show progress between them.
- Also, used in event handlers to schedule an action after the event is fully handled (bubbling done).

- To schedule a new microtask
  - Use queueMicrotask(f).
  - Also promise handlers go through the microtask queue.
- There's no UI or network event handling between microtasks: they run immediately one after another.
- So one may want to queueMicrotask to execute a function asynchronously, but within the environment state.

Thank You