deeecode

DEBOUNCING IN JAVASCRIPT

JS

swipe left

JS

DEBOUNCING IN JAVASCRIPT

Debouncing is a technique used to improve the performance of applications by **reducing the rate at which functions are called**.

This technique can be implemented across different languages, but I'll use JavaScript in this post.

Some functions are expensive, and are evoked too many times than necessary. A good example, which I will focus on, is a search input that searches results when a user types.

The function for handling this search, say **searchFunction**, is evoked every time the user enters an input.

swipe left

JS

DEBOUNCING IN JAVASCRIPT

If this function is not expensive or slow, it may not be a problem. But if this function is, it would be wise to optimize it, by applying **the debouncing technique**.

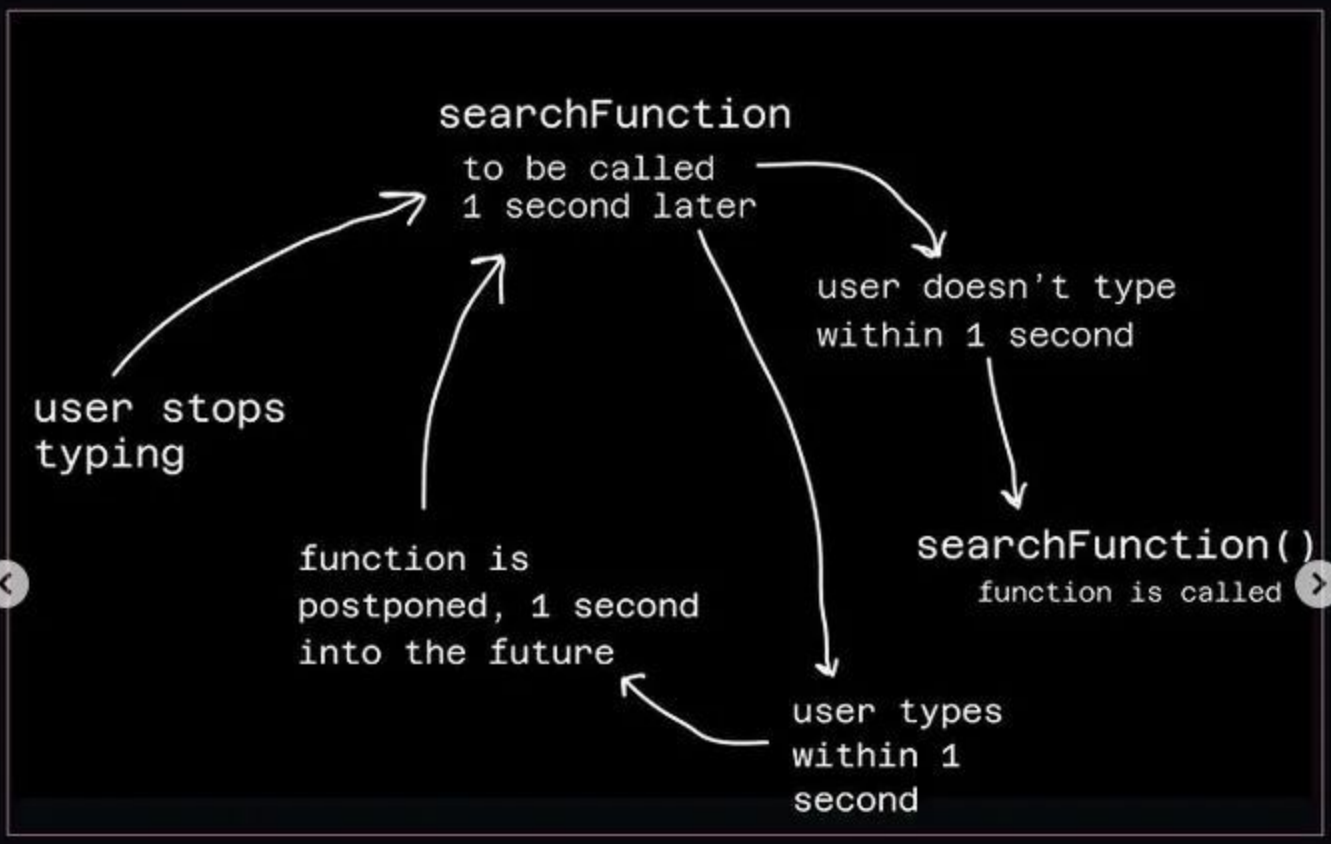
The goal of this technique is to reduce the rate at which the function is called. It does this by "**waiting**" as the user types. And when the user stops typing for some time, you can evoke the function.

This way, when the user types "iphone (waiting) XR", instead of **9** function calls "**i-p-h-o-n-e- -X-R**", it will be **2** function calls(the first time the user waits, and when the users finishes typing)

swipe left

JS

DEBOUNCING IN JAVASCRIPT



In this illustration, you see how the debouncing technique works. For a wait period of 1 second, if the user stops typing, the wait starts counting, if the user types again during that wait period, the wait timer restarts again.

swipe left

JS

DEBOUNCING IN JAVASCRIPT

So how do you implement this in JavaScript?

```
function debounce(func, delay = 1000) {
  let timeoutId

  return function(...arguments) {
    console.log("function called")
    clearTimeout(timeoutId)

    timeoutId = setTimeout(() => {
      func(...arguments)
    }, delay)
  }
}

function searchFunction() {
  console.log("I am doing something")
}

const debounced = debounce(searchFunction)
```

There might be different ways to implement this but the one I share here is using **setTimeout** and **clearTimeout**. I'll explain what's happening in this code in the next image.

swipe left

JS

DEBOUNCING IN JAVASCRIPT

The debounce function receives two arguments: the **function to be debounced**, and the **wait period** called delay. The function starts by declaring a **timeoutId** variable without a value.

Then this function returns another function. In this second function, you log “function called” and you cancel the timeoutId if one has been set.

And next, you assign a new timeoutId by declaring a setTimeout expression of 1 second. In this setTimemout, you execute the func argument.

swipe left

JS

DEBOUNCING IN JAVASCRIPT

When you attach this to an input, you will get the following when typing:

When the user types **h**, the returned function in the debounce function logs "**function called**". The **timeoutId** is null yet, so the **clearTimeout** does nothing. Then the **setTimeout** expression to execute the **searchFunction** in 1 second is declared, and the **timeoutId** updated. When the user types **e**, "function called" is logged, and the updated **timeoutId** is cancelled and a new setTimeout declared. It continues like that until the user waits.

swipe left

DEBOUNCING IN JAVASCRIPT

That is the idea of debouncing. There are many advanced features you can add to the debounce method but it's beyond the scope of this post.

Debouncing helps improve performance but reducing the rate at which a function is called. We've seen how to do this using a wait-and-call approach. It waits for the user to stop typing for a particular period and calls the function.

If the user types again before the period completes, it postpones the function and waits again. That is, you are debouncing the function.

swipe left