JS

Debouncing vs Throttling in JavaScript

MASTER EFFICIENT EVENT HANDLING

Event Handling in JavaScript

- In web development, managing how often functions are executed in response to events is crucial for performance.
- Two powerful techniques for this are Debouncing and Throttling.

Debouncing

- Debouncing is a technique used to limit the rate at which a function is executed. It ensures that the
- function is triggered only after a specified time has elapsed since the last time it was invoked. Example: Waiting until the user stops typing to send an API request for search results.

Real-World Use Case - Debouncing

Search Input Fields: When a user types in a search bar, you don't want to make an API call after every keystroke. Instead, debounce the function to send the request after the user has stopped typing.

Implementation

Implementation Example - Debouncing

```
function debounce(func, delay) {
  let debounceTimer;
  return function(...args) {
    const context = this;
    clearTimeout(debounceTimer);
    debounceTimer = setTimeout(()
                    => func.apply(context, args), delay);
  };
}

// Usage
const handleSearch = debounce(() => {
  // API call for search
}, 300);
```

JS

Throttling

- Throttling limits the number of times a function can be called within a specific timeframe.
- This technique ensures that a function is called at regular intervals, regardless of how many times an event is triggered.

Example: Limiting the number of times a scroll event handler is fired.

JS

Real-World Use Case - Throttling

Scroll Event Handlers: When a user scrolls a page, throttling the scroll event handler ensures the function is executed at a fixed rate, improving performance.

Implementation

Implementation Example - Throttling

```
function throttle(func, limit) {
  let lastFunc;
  let lastRan;
  return function(...args) {
    const context = this;
    if (!lastRan) {
      func.apply(context, args);
      lastRan = Date.now();
    } else {
      clearTimeout(lastFunc);
      lastFunc = setTimeout(function() {
        if ((Date.now() - lastRan) >= limit) {
          func.apply(context, args);
          lastRan = Date.now();
        }
      }, limit - (Date.now() - lastRan));
    }
  };
}

// Usage
const handleScroll = throttle(() => {
  // Handle scroll event
}, 200);
```

JS

Best Practices

- Debounce when you need to wait for a user's action to complete (e.g., input fields). Throttle when you
- need to control the rate of execution for continuous events (e.g., scrolling). Both techniques
- help optimize performance and improve user experience.

Did you find it useful?

LIKE, SHARE, AND FOLLOW FOR MORE JAVASCRIPT TIPS!