

Debouncing and throttling

Debouncing and Throttling in JavaScript — Explained in Detail

Debouncing and **throttling** are two performance optimization techniques used to control how often a function is executed, especially in response to high-frequency events like `scroll`, `resize`, `input`, or `keypress`.

Why Are They Needed?

When events like `scroll` or `keyup` fire rapidly, they can trigger a function **hundreds of times per second**, leading to:

- Performance issues
- Laggy UI
- High CPU usage

To avoid this, we **control** how often a function is executed using **debouncing** or **throttling**.

Debouncing

Debouncing ensures a function is called only once after a specified delay after the last event.

How It Works:

- An event is triggered → start/reset a timer.
- If another event is triggered within the delay → reset the timer.
- Function only runs **after no event occurs for the delay duration**.

Use Case:

- Auto-saving text input
- Search-as-you-type

- Form validation

Example:

```
function debounce(func, delay) {  
  let timeout;  
  return function (...args) {  
    clearTimeout(timeout);  
    timeout = setTimeout(() => func.apply(this, args), delay);  
  };  
}  
  
const handleInput = debounce(() => {  
  console.log("API call after user stops typing");  
}, 300);  
  
document.getElementById("search").addEventListener("input", handleInput);
```

Throttling

Throttling ensures a function is called at most once every specified interval, no matter how many events occur.

How It Works:

- On the first event → allow the function to run.
- For subsequent events within the delay → ignore them.
- After delay → function can run again.

Use Case:

- `scroll` or `resize` event handlers
- Button spamming prevention
- Game loops or animation triggers

Example:

```
function throttle(func, delay) {  
  let lastCall = 0;  
  return function (...args) {  
    const now = new Date().getTime();  
    if (now - lastCall >= delay) {  
      lastCall = now;  
      func.apply(this, args);  
    }  
  };  
}  
  
const handleScroll = throttle(() => {  
  console.log("Scroll event handled");  
}, 500);  
  
window.addEventListener("scroll", handleScroll);
```

Debounce vs Throttle

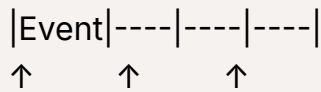
Feature	Debounce	Throttle
Frequency	Runs only after event stops	Runs at regular intervals
Use case	Input fields, auto-save	Scrolling, resizing
Behavior	Delays execution until inactivity	Limits execution to once per interval

Visualization

Debounce:

|Event|----|----|----| → Execution after all events
 ↑ (only 1 execution after no input)

Throttle:



→ Execution every 500ms

✓ Real-World Analogy

- **Debounce:** Think of someone who waits until you stop talking before replying.
- **Throttle:** Think of someone who listens but responds only every 5 seconds no matter how much you say.

← Summary

Concept	Trigger Timing	Use Case
Debounce	After a pause in event firing	Input/search fields
Throttle	At a fixed rate during firing	Scroll/resize/button spam

Let me know if you want reusable `debounce` and `throttle` utility functions for your React/Vue/Vanilla JS projects!