JavaScript Scroll Events

Summary: in this tutorial, you will learn about JavaScript scroll events and how to handle scroll events properly.

Introduction to the JavaScript scroll events

When you scroll a document or an element, the browser fires the scroll events. You can trigger the scroll events in the following ways:

- Using the scrollbar manually
- Using the mouse wheel
- Clicking an ID link
- Calling functions in JavaScript
- etc.

To register a `scroll` event handler, you call the `addEventListener()` method on the target element, like this:

```
targetElement.addEventListener('scroll', (event) => {
    // handle the scroll event
});
```

or assign an event handler to the `onscroll` property of the target element:

```
targetElement.onscroll = (event) => {
    // handle the scroll event
};
```

Scrolling the document

Typically, you register an event handler of the `scroll` events on the `window` object to handle the scroll of the whole page.

For example, the following shows how to attach an event handler to the `scroll` event of a page:

```
window.addEventListener('scroll',(event) => {
    console.log('Scrolling...');
});
```

Alternatively, you can use the `onscroll` property on the `window` object:

```
window.onscroll = function(event) {
    //
};
```

The `onscroll` property of the `window` object is the same as `document.body.onscroll` and you can use them interchangeably, for example:

```
document.body.onscroll = null;
console.log(window.onscroll); // null
```

Scroll offsets

The `window` object has two properties related to the scroll events: `scrollX` and `scrollY` .

The `scrollX` and `scrollY` properties return the number of pixels that the document is currently scrolled horizontally and vertically. The `scrollX` and `scrollY` are double-precision floating-point values so if you need integer values, you can use the `Math.round()` to round them off.

The `scrollX` and `scrollY` are 0 if the document hasn't been scrolled at all.

The `pageXOffset` and `pageYOffset` are aliases of the `scrollX` and `scrollY` properties.

Scrolling an element

Like the `window` object, you can attach a `scroll` event handler to any HTML element. However, to track the scroll offset, you use the `scrollTop` and `scrollLeft` instead of the `scrollX` and `scrollY` .

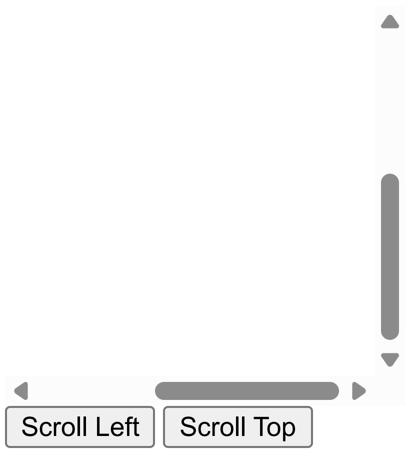
The `scrollTop` property sets or gets the number of pixels that the element's content is vertically scrolled. The `scrollLeft` property gets and sets the number of pixels that an element's content is scrolled from its left edge.

The following example shows how to handle the `scroll` event of the `div` element with the id `scrollDemo` :

```
<!DOCTYPE html>
<html>
<head>
    <title>JS Scroll Events</title>
    <style>
        #scrollDemo {
            height: 200px;
            width: 200px;
            overflow: auto;
            background-color: #f0db4f
        }

        #scrollDemo p {
            /* show the scrollbar */
            height: 300px;
            width: 300px;
        }
    </style>
</head>
<body>
    <div id="scrollDemo">
        <p>JS Scroll Event Demo</p>
    </div>

    <div id="control">
        <button id="btnScrollLeft">Scroll Left</button>
        <button id="btnScrollTop">Scroll Top</button>
    </div>

    <script>
```

```
        let control = document.querySelector('#control');

        control.addEventListener('click', function (e) {
            // get the scrollDemo
            let div = document.getElementById('scrollDemo');
            // get the target
            let target = e.target;
            // handle each button's click
            switch (target.id) {
                case 'btnScrollLeft':
                    div.scrollLeft += 20;
                    break;

                case 'btnScrollTop':
                    div.scrollTop += 20;
                    break;
            }
        });
    </script>
</body>
</html>
```

Output



The better ways to handle the scroll events

Many `scroll` events fire while you are scrolling a page or an element. If you attach an event listener to the `scroll` event, the code in the event handler needs time to execute.

This will cause an issue which is known as the scroll jank. The scroll jank effect causes a delay so that the page doesn't feel anchored to your finger.

Event throttling

It is much better to keep the `scroll` event handler lightweight and execute it every N milliseconds by using a timer. So instead of using the following code (and you should never use it):

```
window.scroll = () => {
    // place the scroll handling logic here
};
```

You should use the following code:

```
let scrolling = false;

window.scroll = () => {
    scrolling = true;
};

setInterval(() => {
    if (scrolling) {
        scrolling = false;
        // place the scroll handling logic here
    }
},300);
```

How it works:

- First, set the `scrolling` flag to `false` . If the `scroll` event fires set the `scrolling` flag to `true` inside the `scroll` event handler.

- Then, execute the `scroll` event handler using the `setInterval()` every 300 milliseconds if the `scroll` events have been fired.

This way of handling the `scroll` event is called the **event throttling** that throttles an `onscroll` 's underlying operation every 300 milliseconds. The throttling slows down the rate of execution of the scroll event handler.

Passive events

Recently, modern web browsers have supported passive events for input events like `scroll` , `touchstart` , `wheel` , etc. It allows the UI thread to handle the event immediately before passing over control to your custom event handler.

In the web browsers which support the passive events, you need to add the `passive` flag to any event listener that does not call `preventDefault()` , like this:

```
document.addEventListener(
    'scroll',
    (event) => {
        // handle scroll event
    },
    { passive: true }
);
```

Without the `passive` option, the code in the event handler will always be invoked before the UI thread carries out the scrolling.

Check out which browsers are supporting passive events here.

Summary

- The `scroll` event fires when you scroll a webpage or an element.
- For a page, the `scrollX` and `scrollY` properties return the number of pixels that the document is currently scrolled horizontally and vertically.
- For an element, the `scrollTop` and `scrollLeft` properties set or get the number of pixels that the element's content is vertically scrolled and scrolled from its left edge.

- Use the event throttling technique to better handle the scroll events. In modern web browsers, you can use passive event listeners.