How to use pointer events in JavaScript

JavaScript has a newer standard called pointer events. It handles both mouse and touch operations, so you don't have to worry about implementing each separately.

Why only care about mouse and touchscreen input? In fact, you can handle both at the same time using **pointer events** .

Many web applications assume that the user's pointing device will be a mouse, so they use mouse events to handle the interaction. However, with the rise of touchscreen devices, users no longer need a mouse to interact with the web. Therefore, it is necessary to support a variety of input devices to have the largest possible audience.

JavaScript has a newer standard called pointer events. It handles both mouse and touch operations, so you don't have to worry about implementing each separately.

What is a pointer event?

Pointer event - The pointer event is a web standard that defines a unified way to handle different input methods in web browsers, including mouse, touch, and pen. These events provide a consistent and platform-independent way to interact with web content across different devices.

In short, pointer events help you handle user interactions across all sources.

Pointer event types

Pointer events are named similarly to the mouse events you may be familiar with. For each mouseEvent in JavaScript, you have a corresponding pointerEvent. That means you can revisit the old code and switch 'mouse' to 'pointer' to start supporting touch and pen input.

The following table shows how pointer events differ from mouse events:

Pointer events	Mouse events
pointerdown	mousedown
pointerup	mouseup
pointermove	mousemove
pointerleave	mouseleave
pointerover	mouseover
pointerenter	mouesenter
pointerout	mouseout
pointercancel	none
gotpointercapture	none
lostpointercapture	none

You can see that there are 3 more pointer events without corresponding mouse events. You will learn more about these events later.

Using pointer events in JavaScript

You can use pointer events the same way you use mouse events. Like most event handling, this process typically follows this sample flow:

1. **Use a DOM** selector to fetch an element.
2. **Use the addEventListener** method , specify the event of interest and a callback function.
3. **Use the callback** argument's properties and methods , an **Event** object .

An example of how to use the pointermove event is as follows:

```
// Nh?n thành ph?n m?
c tiêu const target = document.getElementById('target'); // Thêm m?
t trình nghe s? ki?n vào thành ph?n m?
c tiêu target.addEventListener('pointermove', handlePointerMove); // Hàm ??
 x? lý s? ki?n di chuy?n con tr? function handlePointerMove(ev) { // X?
 lý s? ki?n theo cách b?n mu?
n target.textContent = `Moved at x: ${ev.clientX}, y: ${ev.clientY}`; }
```

This code block defines a target element and a JavaScript function to handle the **pointermove** event , then it uses a JavaScript event listener to attach the pointer event and function to the target element.

Using this code, an element with ID 'target' will display control coordinates when you move the cursor, finger or touch the pen over it.

You can use other pointer events in the same way.

Pointercancel event

The pointercancel event is fired when the pointer event is interrupted before it completes its originally intended operation. Normally, this browser cancels any pointer events that may have previously been executed. There are many reasons why **pointercancel** might trigger, for example:

1. When a user receives a call or other interruptive notification while dragging an element on the screen.
2. When receiving device orientation change.
3. When the browser window loses focus.
4. When the user switches to another tab or application.

With the pointercancel event , you can handle these situations in the desired way. For example:

```
const target = document.getElementById('target'); target.addEventListener('pointe
? lý tình hu?ng t?i n?i h?y con tr?. Ví d?, b?n có th? kích ho?
```

```
t ghi hình chu?t. target.releasePointerCapture(event.pointerId); });
```

Record cursor

Pointer capture is a feature that allows a particular HTML element to capture and respond to all pointer events for a particular pointer, even if those events occur outside the boundaries of that element.

You can set pointer capture for an element using the **setpointercapture()** method and enable it with **releasepointercapture()** .

The pointer events **gotpointercapture** and **lostpointercapture** are useful in capturing pointer images.

Gotpointercapture event

The gotpointercapture event is fired whenever an element acquires a pointer. You can use this event to initialize a state for an HTML element that handles pointer events. For example, in a painting application, you can use the **gotpointercapture** event to set the initial position of the cursor.

lostpointercapture event

The lostpointercapture event is fired when a component loses the ability to capture the pointer. You can use it to clean up or remove any state created when the component achieves cursor rotation. In a painting application, you may want to hide the cursor.

Hope the article is useful to you.