Introduction to Browser Events

1. Common Events
2. How many ways to attach an event to DOM
3. Remove event listener if you no longer use
4. Bubbling (click)
5. Capturing (click)
6. target & currentTarget
7. preventDefault()
8. event load và event COMcontentLoaded
9. Sự kiện Mouse
10. pageXY & clientXY
11. Sự kiện Key
12. Sự kiện change
13. Sự kiện focus & blur
14. Sự kiện submit
15. Sự kiện scroll
16. window.pageXOffset và window.pageYOffset
17. offsetWidth và scrollWidth
18. scrollIntoView()

1. Common Events

- Sự kiện là gì?
- Thêm 1 sự kiện:

```
selector.addEventListener("evenName", handle, [option])
```

#	Event Type	Events
1	Mouse events	click, keydown, keyup, keypress, mouseover, mouseout, mousedown, mouseup, mousemove, load, ...
2	Keyboard events	keydown, keyup
3	Form element events	submit, focus
4	Document events	DOMContentLoaded
5	Window events	beforeunload

> Source: https://developer.mozilla.org/en-US/docs/Web/Events

2. How many ways to attach an event to DOM

#	Way to attach an event	Desc
1	HTML on**event** attributes	**NOT RECOMMENDED** as it makes markup complex
2	JS on**event** properties	Only able to attach **one handler**
3	JS addEventListener	**RECOMMENDED** Able to attach **many handlers**

HTML onevent attributes / Inline event handers

```
<button onclick="alert('thank you! :P')">Click me</button>
```

JS onevent properties

- this way works but only able to attach **one handler** for the event.

```
<button id="clickMeButton">Click me</button>
```

```
const clickMeButton = document.getElementById('clickMeButton');
if (clickMeButton) {
    clickMeButton.onclick = function() {
        alert('thank you! :P');
    }
}
```

JS addEventListener

RECOMMEND we can attach many handlers to an event.

```
<button id="clickMeButton">Click me</button>
```

```
const clickMeButton = document.getElementById('clickMeButton');
if (clickMeButton) {
    clickMeButton.addEventListener('click', function() {
        alert('thank you! :P');
    });
    clickMeButton.addEventListener('click', function() {
        alert('thank you again haha');
    });
}
```

3. Remove event listener if you no longer use

```
<button id="clickMeButton">Click me</button>
```

```
const clickMeButton = document.getElementById('clickMeButton');
if (clickMeButton) {
    function handleClick() {
    alert('thank you! :P');
    }
    clickMeButton.addEventListener('click', handleClick);
    // when no longer use, remove listener
    clickMeButton.removeEventListener('click', handleClick);
}
```

> Source: https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener

Tham khảo

- https://developer.mozilla.org/en-US/docs/Web/Events/Event_handlers
- https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events

4. Bubbling trong click

```
<button><span>Click</span></button>
```

```
const btn = document.querySelector("button")
const btnSpan = document.querySelector("button span")

btn.addEventListener('click', () => {
    console.log("click button")
})

// đặt ở đầu
btnSpan.addEventListener("click", (e) => {
    e.stopImmediatePropagation(); // chỉ chạy sự kiện này
    console.log("click span 2");
});

btnSpan.addEventListener("click", (e) => {
    e.stopPropagation() // hạn chế nổi bọt
    console.log("click span");
});
```

5. Capturing trong Click

Ngược lại với **Bubbling** là đi từ trong ra ngoài

```
btnSpan.addEventListener(
    "click",
    () => {
        console.log("click span 1");
    },
    { capture: true }
);
```

6. target & currentTarget

```
<button><span>Click</span></button>
```

```
const btn = document.querySelector("button")

btn.addEventListener('click', (e) => {
    console.log(e.target) // element đang được click vào
    console.log(e.currentTarget) // trỏ về element được thực hiện event click ->
btn
})
```

7. preventDefault()

```
 <a class="buttonA" href="https://myspa.vn/">
    <button><span>Click</span></button>
</a>
```

```
const btnA = document.querySelector(".buttonA")
btnA.addEventListener('click', (e) => {
    e.preventDefault() // ngăn chặn sự kiện mặc định
})
```

8. event load và event COMcontentLoaded

Sự kiện load

Sự kiện load được kích hoạt khi toàn bộ trang web và tất cả các tài nguyên liên kết của nó (như hình ảnh, tệp CSS, và tệp JavaScript) đã được tải xong. Đây là sự kiện cuối cùng trong quá trình tải trang.

```
<h1>Hello World</h1>
<img src="https://source.unsplash.com/random" alt="Example Image">
```

```
window.addEventListener('load', function() {
    console.log('Tất cả nội dung của trang đã được tải.');
});
```

- Khi sử dụng sự kiện load trên đối tượng window, đảm bảo rằng tất cả các phần của trang đã được tải xong, bao gồm các tài nguyên như hình ảnh và CSS.
- load: Thích hợp cho các tác vụ yêu cầu tất cả tài nguyên trang web đã được tải, chẳng hạn như các thao tác liên quan đến hình ảnh.

Sự kiện DOMContentLoaded

DOMContentLoaded: Sau khi tài liệu HTML đã được tải và phân tích cú pháp hoàn tất, trước khi các tài nguyên như hình ảnh hoàn tất tải.

```
 document.addEventListener('DOMContentLoaded', function() {
    console.log('DOM content loaded and parsed.');
});
```

DOMContentLoaded: Thích hợp cho các thao tác DOM không phụ thuộc vào các tài nguyên khác như hình ảnh, giúp trang web có thể tương tác nhanh hơn.

9. Event Mouse

1. click:

Được kích hoạt khi người dùng nhấp chuột (ấn và thả nút chuột) trên một phần tử.

```
element.addEventListener('click', function(event) {
    console.log('Element was clicked');
});
```

2. dblclick:

Được kích hoạt khi người dùng nhấp đúp chuột trên một phần tử.

```
element.addEventListener('dblclick', function(event) {
    console.log('Element was double-clicked');
});
```

3. mousedown:

Được kích hoạt khi người dùng nhấn một nút chuột xuống trên một phần tử.

```
element.addEventListener('mousedown', function(event) {
    console.log('Mouse button pressed down');
});
```

4. mouseup:

Được kích hoạt khi người dùng thả một nút chuột đã được nhấn xuống trên một phần tử.

```
element.addEventListener('mouseup', function(event) {
    console.log('Mouse button released');
});
```

5. mousemove:

Được kích hoạt khi người dùng di chuyển chuột qua một phần tử.

```
element.addEventListener('mousemove', function(event) {
    console.log('Mouse is moving over the element');
});
```

6. mouseover:

Được kích hoạt khi con trỏ chuột di chuyển vào một phần tử.

```
element.addEventListener('mouseover', function(event) {
    console.log('Mouse moved over the element');
});
```

7. mouseout:

Được kích hoạt khi con trỏ chuột di chuyển ra khỏi một phần tử.

```
element.addEventListener('mouseout', function(event) {
    console.log('Mouse moved out of the element');
});
```

8. mouseenter:

Được kích hoạt khi con trỏ chuột di chuyển vào một phần tử. Không bị kích hoạt khi di chuyển vào các phần tử con bên trong phần tử.

```
```

```
element.addEventListener('mouseenter', function(event) {
    console.log('Mouse entered the element');
});
```

9. mouseleave:

Được kích hoạt khi con trỏ chuột di chuyển ra khỏi một phần tử. Không bị kích hoạt khi di chuyển vào các phần tử con bên trong phần tử.

```
element.addEventListener('mouseleave', function(event) {
    console.log('Mouse left the element');
});
```

10. contextmenu:

Được kích hoạt khi người dùng nhấp chuột phải để mở menu ngữ cảnh.

```
element.addEventListener('contextmenu', function(event) {
    console.log('Context menu opened');
    event.preventDefault(); // Ngăn chặn menu ngữ cảnh mặc định
});
```

Ví dụ:

```
<div class="box" id="box">Hover me!</div>
```

```
.box {
    width: 200px;
    height: 200px;
    background-color: #3498db;
    color: white;
    display: flex;
    align-items: center;
    justify-content: center;
    font-size: 1.5em;
    margin: 50px;
    user-select: none;
}
```

```
const box = document.getElementById('box');
```

```
box.addEventListener('click', function(event) {
    console.log('Click event');
});

box.addEventListener('dblclick', function(event) {
    console.log('Double click event');
});

box.addEventListener('mousedown', function(event) {
    console.log('Mouse down event');
});

box.addEventListener('mouseup', function(event) {
    console.log('Mouse up event');
});

box.addEventListener('mousemove', function(event) {
    console.log('Mouse move event');
});

box.addEventListener('mouseover', function(event) {
    console.log('Mouse over event');
});

box.addEventListener('mouseout', function(event) {
    console.log('Mouse out event');
});

box.addEventListener('mouseenter', function(event) {
    console.log('Mouse enter event');
});

box.addEventListener('mouseleave', function(event) {
    console.log('Mouse leave event');
});

box.addEventListener('contextmenu', function(event) {
    console.log('Context menu event');
    event.preventDefault(); // Ngăn chặn menu ngữ cảnh mặc định
});
```

10. pageXY & clientXY

clientX và clientY

- clientX: Trả về tọa độ X của con trỏ chuột tính từ góc trên bên trái của vùng hiển thị (viewport) hiện tại của trình duyệt. Giá trị này không bao gồm phần trang web bị cuộn.
- clientY: Trả về tọa độ Y của con trỏ chuột tính từ góc trên bên trái của vùng hiển thị (viewport) hiện tại của trình duyệt. Giá trị này cũng không bao gồm phần trang web bị cuộn.

pageX và pageY

- pageX: Trả về tọa độ X của con trỏ chuột tính từ góc trên bên trái của tài liệu (document), bao gồm cả phần bị cuộn.
- pageY: Trả về tọa độ Y của con trỏ chuột tính từ góc trên bên trái của tài liệu (document), bao gồm cả phần bị cuộn.

So sánh clientX/Y và pageX/Y

Thuộc tính	Tọa độ từ đâu	Bao gồm phần cuộn
clientX	Vùng hiển thị (viewport)	Không
clientY	Vùng hiển thị (viewport)	Không
pageX	Tài liệu (document)	Có
pageY	Tài liệu (document)	Có

Ví dụ

```
<div class="box" id="box">Move your mouse!</div>
```

```
body {
    height: 2000px; /* Tạo trang dài để có thể cuộn */
    font-family: Arial, sans-serif;
}
.box {
    width: 300px;
    height: 300px;
    background-color: #3498db;
    color: white;
    display: flex;
    align-items: center;
    justify-content: center;
    font-size: 1.5em;
    margin: 50px;
    user-select: none;
    position: fixed;
    top: 10px;
    left: 10px;
}
```

```
const box = document.getElementById('box');
document.addEventListener('mousemove', function(event) {
    const clientX = event.clientX;
    const clientY = event.clientY;
    const pageX = event.pageX;
    const pageY = event.pageY;
```

```
    box.textContent = `client: (${clientX}, ${clientY}) page: (${pageX},
${pageY})`;
});
```

11. Sự kiện Key

1. keydown:

Được kích hoạt khi một phím trên bàn phím được nhấn xuống

```
document.addEventListener('keydown', function(event) {
    console.log('Key down:', event.key);
});
```

2. keypress:

Được kích hoạt khi một phím ký tự được nhấn xuống. Lưu ý rằng sự kiện này không còn được khuyến nghị sử dụng, thay vào đó nên sử dụng keydown và keyup.

```
document.addEventListener('keypress', function(event) {
    console.log('Key press:', event.key);
});
```

3. keyup:

Được kích hoạt khi một phím trên bàn phím được thả ra.

```
document.addEventListener('keyup', function(event) {
    console.log('Key up:', event.key);
});
```

Ví dụ:

```
<h1>Press any key</h1>
<div id="output"></div>
```

```
 const output = document.getElementById('output');

document.addEventListener('keydown', function(event) {
    output.textContent = `Key down: ${event.key} (code: ${event.code})`;
});

document.addEventListener('keyup', function(event) {
```

```
    output.textContent = `Key up: ${event.key} (code: ${event.code})`;
});
```

Thuộc tính của đối tượng sự kiện Key

```
document.addEventListener('keydown', function(event) {
    let message = `Key down: ${event.key} (code: ${event.code})`;

    if (event.altKey) {
        message += ' + Alt';
    }
    if (event.ctrlKey) {
        message += ' + Ctrl';
    }
    if (event.shiftKey) {
        message += ' + Shift';
    }
    if (event.metaKey) {
        message += ' + Meta';
    }

    output.textContent = message;
});
```

12. Sự kiện change

Ví dụ 1

```
<label for="inputText">Enter text:</label>
<input type="text" id="inputText">
<div id="output"></div>
```

```
const inputText = document.getElementById('inputText');
const output = document.getElementById('output');

inputText.addEventListener('change', function(event) {
    output.textContent = `You entered: ${event.target.value}`;
});
```

Ví dụ 2

```
<label for="selectMenu">Choose an option:</label>
<select id="selectMenu">
    <option value="option1">Option 1</option>
    <option value="option2">Option 2</option>
```

```
    <option value="option3">Option 3</option>
</select>
<div id="output"></div>
```

```
const selectMenu = document.getElementById('selectMenu');
const output = document.getElementById('output');

selectMenu.addEventListener('change', function(event) {
    output.textContent = `You selected: ${event.target.value}`;
});
```

13. Sự kiện focus & blur

Sự kiện focus

Sự kiện focus được kích hoạt khi một phần tử nhận tiêu điểm, thường là khi người dùng nhấp vào phần tử đó hoặc sử dụng phím Tab để di chuyển đến phần tử.

```
<h1>Focus Event Example</h1>
<input type="text" id="input1" placeholder="Click to focus">
<input type="text" id="input2" placeholder="Click to focus">
```

```
input {
    display: block;
    margin-bottom: 10px;
}
.focus {
    border: 2px solid blue;
}
```

```
const inputs = document.querySelectorAll('input');

inputs.forEach(input => {
    input.addEventListener('focus', function(event) {
        event.target.classList.add('focus');
        console.log('Focused:', event.target.id);
    });
});
```

Sự kiện blur

Sự kiện blur được kích hoạt khi một phần tử mất tiêu điểm, thường là khi người dùng nhấp ra ngoài phần tử đó hoặc sử dụng phím Tab để di chuyển đến phần tử khác.

```
<h1>Blur Event Example</h1>
<input type="text" id="input1" placeholder="Click to focus">
<input type="text" id="input2" placeholder="Click to focus">
```

```
input {
    display: block;
    margin-bottom: 10px;
}
.focus {
    border: 2px solid blue;
}
```

```
const inputs = document.querySelectorAll('input');

inputs.forEach(input => {
    input.addEventListener('blur', function(event) {
        event.target.classList.remove('focus');
        console.log('Blurred:', event.target.id);
    });
});
```

Kết hợp cả **focus** và **blur**

```
const inputs = document.querySelectorAll('input');

inputs.forEach(input => {
    input.addEventListener('focus', function(event) {
        event.target.classList.add('focus');
        console.log('Focused:', event.target.id);
    });

    input.addEventListener('blur', function(event) {
        event.target.classList.remove('focus');
        console.log('Blurred:', event.target.id);
    });
});
```

14. Sự kiện submit

- Sự kiện submit trong JavaScript được kích hoạt khi một biểu mẫu (form) được gửi.
- Sự kiện này thường được sử dụng để xử lý và kiểm tra dữ liệu đầu vào của biểu mẫu trước khi gửi dữ liệu tới máy chủ.
- Bằng cách sử dụng sự kiện này, bạn có thể ngăn chặn việc gửi biểu mẫu nếu dữ liệu không hợp lệ hoặc thực hiện các hành động khác trước khi gửi biểu mẫu.

```
<h1>Submit Event Example</h1>
<form id="myForm">
    <input type="text" id="username" name="username" placeholder="Enter your
username" required>
    <input type="password" id="password" name="password" placeholder="Enter your
password" required>
    <button type="submit">Submit</button>
</form>
<div id="output"></div>
```

```
form {
    max-width: 400px;
    margin: auto;
}
input, button {
    display: block;
    width: 100%;
    margin-bottom: 10px;
    padding: 10px;
    font-size: 1em;
}
```

```
const form = document.getElementById('myForm');
const output = document.getElementById('output');

form.addEventListener('submit', function(event) {
    event.preventDefault(); // Ngăn chặn việc gửi biểu mẫu mặc định

    const username = form.elements['username'].value;
    const password = form.elements['password'].value;

    // Kiểm tra dữ liệu đầu vào
    if (username === '' || password === '') {
        output.textContent = 'Please fill out all fields.';
        return;
    }

    output.textContent = `Username: ${username}, Password: ${password}`;
});
```

Ví dụ nâng cao: Gửi dữ liệu bằng AJAX

```
form.addEventListener('submit', function(event) {
    event.preventDefault(); // Ngăn chặn việc gửi biểu mẫu mặc định

    const formData = new FormData(form);
```

```
    fetch('/submit', {
        method: 'POST',
        body: formData
    })
    .then(response => response.json())
    .then(data => {
        output.textContent = `Server response: ${JSON.stringify(data)}`;
    })
    .catch(error => {
        console.error('Error:', error);
        output.textContent = 'An error occurred. Please try again.';
    });
});
```

15. Sự kiện scroll

- Sự kiện scroll trong JavaScript được kích hoạt khi người dùng cuộn trang web hoặc một phần tử có thể cuộn (scrollable element).
- Sự kiện này thường được sử dụng để thực hiện các tác vụ liên quan đến cuộn, chẳng hạn như tải thêm nội dung khi người dùng cuộn xuống cuối trang, hiển thị hoặc ẩn các phần tử dựa trên vị trí cuộn, hoặc thực hiện các hiệu ứng cuộn.

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 20px;
    height: 2000px; /* Tăng chiều cao của trang để có thể cuộn */
}
#scrollPosition {
    position: fixed;
    top: 10px;
    right: 10px;
    background: rgba(0, 0, 0, 0.7);
    color: white;
    padding: 5px;
}
```

```
<h1>Scroll Event Example</h1>
<p>Scroll down to see the effect.</p>
<div id="scrollPosition">Scroll Position: 0</div>
```

```
const scrollPosition = document.getElementById('scrollPosition');

window.addEventListener('scroll', function() {
    const scrollTop = window.scrollY || document.documentElement.scrollTop;
```

```
    scrollPosition.textContent = `Scroll Position: ${scrollTop}`;
});
```

16. window.pageXOffset và window.pageYOffset

`window.pageXOffset` và `window.pageYOffset` là các thuộc tính của đối tượng window trong JavaScript, được sử dụng để lấy giá trị cuộn ngang và cuộn dọc của cửa sổ trình duyệt.

window.pageXOffset `window.pageXOffset` trả về giá trị số pixel mà tài liệu đã được cuộn theo chiều ngang (từ trái sang phải). Giá trị này thường được sử dụng khi bạn muốn biết vị trí hiện tại của cuộn ngang trong tài liệu.

window.pageYOffset `window.pageYOffset` trả về giá trị số pixel mà tài liệu đã được cuộn theo chiều dọc (từ trên xuống dưới). Giá trị này thường được sử dụng khi bạn muốn biết vị trí hiện tại của cuộn dọc trong tài liệu.

```
<div class="content">
<h1>Welcome to the Scroll Example</h1>
<p>Scroll down to see the "Back to Top" button.</p>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit...</p>
<!-- Add more content to make the page scrollable -->
<p style="margin-top: 1000px;">End of content.</p>
</div>
<button id="backToTopButton" onclick="scrollToTop()">Back to Top</button>
```

```
#backToTopButton {
  display: none; /* Initially hidden */
  position: fixed;
  bottom: 20px;
  right: 20px;
  padding: 10px 20px;
  background-color: #007bff;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  font-size: 16px;
}

#backToTopButton:hover {
  background-color: #0056b3;
}
```

```
window.addEventListener('scroll', function() {
    console.log('Horizontal scroll: ', window.pageXOffset);
    console.log('Vertical scroll: ', window.pageYOffset);
```

```
    if (window.pageYOffset > 100) {
        document.getElementById('backToTopButton').style.display = 'block';
    } else {
        document.getElementById('backToTopButton').style.display = 'none';
    }
});

function scrollToTop() {
    window.scrollTo({
        top: 0,
        behavior: 'smooth'
    });
}
```

17. offsetWidth và scrollWidth

offsetWidth và **scrollWidth** là hai thuộc tính của các phần tử DOM trong JavaScript, được sử dụng để đo lường kích thước của một phần tử theo các cách khác nhau.

offsetWidth

- **offsetWidth** Trả về chiều rộng của phần tử bao gồm cả padding, border và thanh cuộn (nếu có). Nó không bao gồm margin.
- Công dụng: Sử dụng để lấy chiều rộng thực tế của phần tử, bao gồm cả viền và padding.

scrollWidth

- **scrollWidth** Trả về chiều rộng của nội dung phần tử, bao gồm cả nội dung không hiển thị do tràn (overflow). Nếu nội dung tràn ra ngoài vùng nhìn thấy của phần tử, **scrollWidth** sẽ lớn hơn hoặc bằng **offsetWidth**.
- Công dụng: Sử dụng để lấy chiều rộng của toàn bộ nội dung, bao gồm cả nội dung không hiển thị.

Ví dụ:

```
<div class="container">
<div class="content">
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur lacinia,
nunc nec luctus interdum, velit risus tincidunt justo, at interdum metus mi eget
sapien. Donec vel dolor nulla.
</div>
</div>
<p id="result"></p>
```

```
.container {
  width: 300px;
  height: 100px;
```

```
  overflow: auto;
  border: 1px solid #000;
  padding: 10px;
}

.content {
  width: 600px; /* Make content wider than container */
}
```

```
window.addEventListener('load', function() {
    const container = document.querySelector('.container');
    const result = document.getElementById('result');

    const offsetWidth = container.offsetWidth;
    const scrollWidth = container.scrollWidth;

    result.textContent = `offsetWidth: ${offsetWidth}px, scrollWidth:
${scrollWidth}px`;
});
```

18. scrollIntoView()

- **scrollIntoView** là một phương thức của các phần tử DOM trong JavaScript, được sử dụng để cuộn phần tử đó vào vùng hiển thị của cửa sổ trình duyệt.
- Phương thức này rất hữu ích khi muốn đảm bảo rằng một phần tử cụ thể nằm trong vùng nhìn thấy của người dùng mà không cần họ phải cuộn thủ công.

Cú pháp

```
element.scrollIntoView([options]);
```

Tham số

boolean:

- true (hoặc không cung cấp giá trị) sẽ cuộn phần tử vào vùng nhìn thấy sao cho phần tử đó nằm trên cùng của khung nhìn (nếu có thể).
- false sẽ cuộn phần tử vào vùng nhìn thấy sao cho phần tử đó nằm dưới cùng của khung nhìn (nếu có thể).

object: Cho phép cung cấp các tùy chọn chi tiết hơn dưới dạng một đối tượng với các thuộc tính sau:

- behavior: Xác định kiểu cuộn, có thể là "auto" (mặc định), "smooth", "instant".
- block: Xác định căn chỉnh dọc của phần tử, có thể là "start", "center", "end", "nearest".
- inline: Xác định căn chỉnh ngang của phần tử, có thể là "start", "center", "end", "nearest".

Ví dụ:

```
#myDIV {
    height: 250px;
    width: 250px;
    overflow: auto;
    background: coral;
}

#content {
    margin:500px;
    height: 800px;
    width: 2000px;
    background: coral;
    position: relative;
}
```

```
<h1>The Element Object</h1>
<h2>The scrollIntoView() Method</h2>

<p>Click the buttons to scroll to the top or to the bottom of the element with
id="content".</p>

<p>
    <button onclick="scrollToTop()">Scroll to the top</button>
    <button onclick="scrollToBottom()">Scroll to the bottom</button>
</p>

<div id="myDIV">
    <div id="content">
        <div style="position:absolute;top:0;">Some text at the top</div>
        <div style="position:absolute;bottom:0">Some text at the bottom</div>
    </div>
</div>
```

```
const element = document.getElementById("content");

function scrollToTop() {
    element.scrollIntoView({ behavior: "smooth", block: "start", inline: "nearest"
});
}

function scrollToBottom() {
    element.scrollIntoView({ behavior: "smooth", block: "end", inline: "start" });
}
```