# Java Script

NEXT

GENIUS space

# План

Події

форми

event propagation

Drag'n'Drop

Подія – це сигнал від браузера у тому, що щось сталося.
Події використовуються для реакції на дії відвідувача та виконання коду.
Події стають у чергу та обробляються у порядку надходження, асинхронно, незалежно.
Існує багато видів подій, розглянемо деякі з них.

click - відбувається, коли клацнули на елемент лівою кнопкою миші
submit - відвідувач надіслав форму
focus - відвідувач фокусується на елементі, наприклад, натискає на input
contextmenu – відбувається, коли клацнули на елемент правою кнопкою миші.
mouseover / mouseout – коли миша наводиться на / залишає елемент.
mousedown / mouseup – коли натиснули / відпустили кнопку миші на елементі.
mousemove – під час руху миші.

всі доступні події https://developer.mozilla.org/en-US/docs/Web/Events

Події клавіатури:

keydown та keyup – коли користувач натискає / відпускає клавішу.
Події елементів форми:

submit – користувач надіслав форму <form>.
focus – користувач фокусується на елементі, наприклад, натискає на <input>.
Події документа:

DOMContentLoaded – коли HTML завантажено й оброблено, DOM документа повністю побудований і доступний.
CSS події:

transitionend – коли CSS-анімацію завершено.
I

# Обробники подій

Події можна призначити обробник, тобто функцію, яка спрацює, щойно подія сталася.

Саме завдяки обробникам JavaScript код може реагувати на дії користувача.

Є кілька способів призначити події обробник. Зараз ми їх розглянемо, починаючи з найпростішого.

Використання атрибута HTML
Обробник може бути призначений прямо в розмітці, атрибуті, який називається on<event>.

Наприклад, щоб призначити обробник події click на елементі input, можна використовувати атрибут onclick, ось так:

<input value="Натисни мене" onclick="alert('Клік!')" type="button">
При натисканні мишкою на кнопці виконається код, вказаний в атрибуті onclick.

## Порядок спрацювання подій

Одна дія може викликати кілька подій. Наприклад, клік викликає спочатку стрілку вниз, а потім натисніть і клацніть. У випадках, коли одна дія генерує кілька подій, їх порядок фіксований. Тобто, обробники викликатимуться в порядку mousedown → mouseup → click.
Кожна подія опрацьовується незалежно. Наприклад, при натисканні, події мишіup і click виникають одночасно, але обробляються послідовно. Спочатку повністю завершується обробка mouseup, потім запускається click.

## Слухачі подій

Для того, щоб елемент реагував на дії користувача, на нього необхідно повісити слухача (обробник) події. Тобто функцію, яка спрацює, як тільки подія відбулася. Саме завдяки слухачам подій скрипт може реагувати на дії користувача.
Методи elem.addEventListener() і elem.removeEventListener() це сучасний спосіб призначити або видалити обробник, при цьому можна використовувати скільки завгодно обробників на одному типі події.

addEventListener

Синтаксис додавання обробника:
element.addEventListener(event, handler, [options]);

event
Назва події, наприклад "click".

handler
Посилання на функцію-обробник.

options
Додатковий об’єкт із властивостями:

- once: якщо true, тоді обробник буде автоматично вилучений після виконання.
- capture: фаза, на якій повинен спрацювати обробник, докладніше про це буде розказано у розділі Bubbling and capturing. Так історично склалося, що options може бути false/true, це те саме, що {capture: false/true}.
- passive: якщо true, тоді обробник ніколи не викличе preventDefault(), докладніше про це буде розказано у розділі Типові дії браузера.

Для видалення обробника слід використовувати removeEventListener:

element.removeEventListener(event, handler, [options]);

Метод addEventListener дозволяє додавати кілька обробників на одну подію одного елемента, наприклад:

```
<input id="elem" type="button" value="Натисни мене"/>
<script>
function handler1() {
alert('Дякую!');
};
function handler2() {
alert('Ще раз дякую!');}  elem.onclick = () => alert("Привіт");
  elem.addEventListener("click", handler1); // Дякую!
  elem.addEventListener("click", handler2); // Ще раз дякую!
</script>
```

Форми: подія та метод submit

Подія submit ініціюється, коли форма надсилається. Зазвичай це використовується для перевірки форми перед відправкою на сервер або щоб запобігти її відправленню та обробці в JavaScript.

Метод form.submit() дозволяє ініціювати відправку форми за допомогою JavaScript. Ми можемо використовувати його для динамічного створення та надсилання власних форм на сервер.

Подія submit

Виникає під час відправлення форми. Його застосовують для валідації форми форми перед відправкою. Щоб відправити форму, відвідувач має два способи:

Натиснути кнопку з type="submit"

Натиснути клавішу Enter, перебуваючи в якомусь полі форми

Хоч би який спосіб вибрав відвідувач – буде згенеровано подію submit. В обробнику цієї події можна перевірити дані та виконати дії за результатами перевірки.

Метод: submit
Щоб надіслати форму на сервер вручну, ми можемо викликати form.submit().

Тоді подія submit не генерується. Передбачається, що якщо програміст викликає form.submit(), то сценарій вже здійснив всю пов’язану обробку.

Іноді це використовується для створення та надсилання форми вручну, наприклад:

```
let form = document.createElement('form');
form.action = 'https://google.com/search';
form.method = 'GET';

form.innerHTML = '<input name="q" value="Тест">';

// форма повинна бути в документі, щоб її надіслати
document.body.append(form);
form.submit();
```

## Клавіатура: keydown та keyup

Є три основні події клавіатури: keydown, keypress і keyup. При натисканні клавіші спочатку відбувається keydown, після чого keypress, і тільки потім keyup, коли клавішу віджали.

Події keydown та keyup спрацьовують при натисканні будь-якої клавіші, включаючи службові. А ось keypress спрацьовує лише якщо натиснута символьна клавіша, тобто натискання призводить до появи символу. Клавіші, що управляють, такі як Ctrl, Shift, Alt та інші, не генерують подію keypress.

Властивість KeyboardEvent.key доступна для читання і повертає значення клавіші, натиснутої користувачем, беручи до уваги стан клавіш модифікаторів, таких як shiftKey, а також поточну мову та модель клавіатури.

Властивість KeyboardEvent.code є фізичною клавішею на клавіатурі (на відміну від символу, згенерованого натисканням клавіші). Іншими словами, ця властивість повертає значення, яке не змінюється за допомогою клавіатури або стану клавіш-модифікаторів.

## Фокусування

Елемент отримує фокус, натиснувши на ньому мишкою, клавіші Tab або вибравши на планшеті. Момент отримання фокусу та втрати дуже важливий, при отриманні фокусу ми можемо підвантажити дані для автодоповнення, почати відстежувати зміни. При втраті фокусу перевірити введені дані.

При фокусуванні на елемент відбувається подія focus, а коли фокус зникає, наприклад, відвідувач клікає в іншому місці екрана, відбувається подія blur.
За замовчуванням багато елементів не можуть отримати фокусування. Наприклад, якщо натиснути на div, то фокусування на ньому не відбудеться. До речі, фокус може бути тільки на одному елементі в одиницю часу, а поточний елемент, на якому фокус доступний як document.activeElement.
Активувати або скасувати фокус можна програмно, викликавши в коді однойменні методи elem.focus() та elem.blur() елемента.

## Подія change

Відбувається після зміни елемента форми, коли зміна зафіксована. Для input:text або textarea подія відбудеться не при кожному введенні, а при втраті фокусу, що не завжди зручно.

Наприклад, поки що ви набираєте щось у текстовому полі — події немає. Але щойно фокус зник, відбудеться подія change. Для інших елементів, наприклад select, input:checkbox і input:radio, воно спрацьовує відразу при виборі значення.

## Подія input

Спрацьовує лише на текстових елементах, input:text і textarea, за зміни значення елемента. Не чекає втрати фокусу, на відміну від зміни.
У сучасних браузерах input – найголовніша подія для роботи з текстовим елементом форми. Саме його, а не keydown або keypress, слід використовувати

# Поширення подій

Поширення подій (event propagation) - важлива, але незрозуміла тема, коли йдеться про події. Це всеосяжний термін, який включає три різні етапи життя події: затоплення, націлення і спливання.

Поширення події двонаправлене - воно починається на window, йде до цільового елементу і закінчується на window. Поширення часто неправильно використовується як синонім стадії спливу. Щоразу, коли відбувається подія, відбувається її поширення.

При настанні події вона проходить через три обов'язкові фази:
Capture phase - подія починається на window і тоне (проходить через всі вузли-предки) до найглибшого цільового елемента, де сталася подія.
Target phase - подія дійшла до найглибшого цільового елемента. Цей етап включає лише повідомлення вузла, у якому сталася подія.
Bubbling phase - заключна фаза, подія спливає від найглибшого, цільового елемента, через усі вузли-предки, до window.

## Приклад делегування: дії в розмітці

Скажімо, ми хочемо створити меню з кнопками «Зберегти», «Завантажити», «Пошук» і так далі. А ще є об’єкт з методами save, load, search... Як їх поєднати?

Перше, що спадає на думку – це призначити окремий обробник кожній кнопці. Але є більш елегантне рішення. Ми можемо додати один обробник до всього меню та атрибути data-action до кожної кнопки відповідно до методів, які вони викликають:

```
<button data-action="save">
Клікніть, щоб Зберегти
</button>
```

Обробник читає атрибут і виконує відповідний метод. Подивіться на робочий приклад:

```
<div id="menu">
<button data-action="save">Зберегти</button>
<button data-action="load">Завантажити</button>
<button data-action="search">Пошук</button>
</div>
<script>
class Menu {
constructor(elem) {
this._elem = elem;
elem.onclick = this.onClick.bind(this);  // (*)}
save() {
alert('збереження'); }
load() {
alert('завантаження'); }
search() {
alert('пошук'); }
onClick(event) {
let action = event.target.dataset.action;
if (action) {
this[action]();
} }; }
new Menu(menu);
</script>
```

Зауважте, що this.onClick прив’язаний до this у (*). Це важливо, тому що інакше this в ньому посилатиметься на елемент DOM (elem), а не на об’єкт Menu, і this[action] буде не тим, який нам потрібен.
Отже, які переваги дає нам тут делегування?
Нам не потрібно писати код, щоб призначити обробник кожній кнопці. Достатньо створити один метод і помістити його в розмітку.

Структура HTML гнучка, ми можемо в будь-який момент додати/видалити кнопки.
Ми також можемо використовувати класи .action-save, .action-load, але підхід з використанням атрибутів data-action вважається семантично кращим. Крім того, його можна використовувати в правилах CSS.
Делегування подій можна також використовувати для додавання певної «поведінки» елементам декларативно, за допомогою спеціальних атрибутів та класів.
Шаблон складається з двох частин:

1. Ми додаємо спеціальний атрибут до елемента, який описує його поведінку.
2. За допомогою делегування ставиться один обробник на документ, що відстежує усі події і, якщо елемент має атрибут, виконує відповідну дію.

## Drag'n'Drop з подіями миші

Drag’n’Drop – відмінний спосіб поліпшити інтерфейс. Захоплення елементу мишкою і його перенесення візуально спростять що завгодно: від копіювання і переміщення документів (як у файлових менеджерах) до оформлення замовлення (“покласти до кошику”).

У сучасному стандарті HTML5 є розділ про Drag and Drop – який містить спеціальні події саме для Drag’n’Drop перенесення, такі як dragstart, dragend та інші.

Ці події дозволяють нам підтримувати спеціальні види drag’n’drop, наприклад, обробляти перенесення файлу з файлового менеджера ОС у вікно браузеру. Після чого JavaScript може отримати доступ до вмісту таких файлів.

Але у браузерних подій Drag Events є обмеження. Наприклад, ми не можемо заборонити перенесення з певної області. Також ми не можемо зробити перенесення тільки “горизонтальним” або тільки “вертикальним”. І є багато інших завдань по перетяганню, які не можуть бути виконані за їх допомогою. Крім того, підтримка таких подій на мобільних пристроях дуже низька.

## Drag’n’Drop алгоритм

Наш алгоритм Drag’n’Drop виглядає таким чином:

На mousedown – підготувати елемент до переміщення, якщо це необхідно (наприклад, створити його клон, додати до нього клас або щось ще).

Потім, на mousemove перемістити його, змінивши значення left/top при позиціюванні position: absolute.

На mouseup – виконати усі дії, пов’язані із завершенням перенесення.

# Додаткові матеріали

- https://developer.mozilla.org/en-US/docs/Web/Events
- https://uk.javascript.info/introduction-browser-events
- https://uk.javascript.info/bubbling-and-capturing
- https://uk.javascript.info/default-browser-action
- https://uk.javascript.info/dispatch-events
- https://uk.javascript.info/onscroll
- https://uk.javascript.info/forms-submit
- https://uk.javascript.info/keyboard-events
- https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/key
- https://uk.javascript.info/events-change-input
- https://uk.javascript.info/form-elements
- https://uk.javascript.info/event-delegation
- https://uk.javascript.info/mouse-events-basics
- https://uk.javascript.info/mouse-drag-and-drop

# Домашнє завдання

-----1----

Напишіть такий JavaScript, щоб після натискання на кнопку button, елемент  зникав.

-----2----

Напишіть такий код, щоб після натискання на кнопку, вона зникала.

-----3----

Створіть дерево, яке показує/приховує дочірні вузли при кліці