hillel IT school

Lesson 19

План заняття

- Додавання обробників-подій DOM-елементам;
- Можливі події;
- Фази обробки подій;
- Делегування подій;
- Об'єкт Event;

Події

Подія – це сигнал від браузера, що щось сталося. Всі DOM-вузли подають такі сигнали (хоча події бувають не тільки в DOM).

Події миші:

- click – відбувається, коли клацнули на елемент лівою кнопкою миші (на пристроях із сенсорними екранами воно відбувається при торканні).
- 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.

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.

Використання атрибута HTML

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

Атрибут HTML-тега – не найзручніше місце для написання великої кількості коду, тому краще створити окрему JavaScript-функцію та викликати її там.

Використання властивостей DOM

Можемо призначати обробник, використовуючи властивість DOM-елемента on<event>.

Якщо обробник заданий через атрибут, то браузер читає HTML-розмітку, створює нову функцію із вмісту атрибута та записує у властивість.

Оскільки в елемента DOM може бути тільки одна властивість з ім'ям onclick, то призначити більше одного обробника таким чином не можна.

Щоб видалити обробник – установіть elem.onclick = null.

Не використовуйте setAttribute для обробників.

Регістр DOM-властивості має значення…

this

Усередині обробника події this посилається на поточний елемент, тобто на той, на якому, як кажуть, «висить» (тобто призначений) обробник.

addEventListener

Фундаментальний недолік описаних вище способів присвоєння обробника – неможливість повісити кілька обробників для однієї події.

Синтаксис додавання обробника:

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

1. event - Назва події, наприклад "click".
2. handler - Посилання на функцію-обробник.
3. 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 Бульбашковий механізм (спливання та занурення). 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.

addEventListener

Метод addEventListener дозволяє додавати кілька обробників на одну подію одного елемента.

Ми можемо призначити обробником не лише функцію, а й об’єкт за допомогою addEventListener. У такому разі, коли відбувається подія, викликається метод об’єкта handleEvent.

Обробники на тому ж елементі та в тій же фазі запускаються у встановленому порядку

removeEventListener

Видалення підписника на подію вимагає саме ту ж функцію

removeEventListener потребує тієї ж фази події.

Об’єкт події

Коли відбувається подія, браузер створює об’єкт події, записує в нього деталі та передає його як аргумент функції-обробнику.

Спливання

Принцип Спливання простий.

Коли подія відбувається на елементі, спочатку запускаються обробники на ньому, потім на його батькові, потім на інших предках і так до самого верху.

При кліку на внутрішній <input> спочатку виконується onclick:

1. У самого <input>.
2. Потім зовнішнього .
3. Потім зовнішнього <form>.
4. І так далі вгору до об’єкта document.

Процес називається “булькання (спливання)”, тому що події спливають від внутрішнього елемента вгору через батьків, подібно до бульбашки у воді. Майже всі події спливають (але це більш вийняток).

event.target

Обробник батьківського елемента завжди може отримати детальну інформацію про те, де це насправді сталося.

Найбільш глибоко вкладений елемент, що викликав подію, називається цільовим елементом, та доступний як event.target.

Відмінності від this (=event.currentTarget):

- event.target – “цільовий” елемент, який ініціював подію, він не змінюється в процесі спливання.
- this – “поточний” елемент, той, на якому в даний момент виконується обробник.

Припинення спливання

Бульбашкова подія іде від цільового елемента прямо вгору. Зазвичай вона спливає вгору до елемента <html>, а потім до об’єкта document, і деякі події навіть досягають вікна, викликаючи всі обробники на своєму шляху.

Але будь-який обробник може вирішити, що подію повністю оброблено і зупинити її спливання.

Для цього існує метод – event.stopPropagation().

event.stopPropagation() зупиняє подальше рухання вгору, але на поточному елементі виконуються всі інші обробники.

Щоб зупинити спливання та запобігти виконанню обробників на поточному елементі, є метод event.stopImmediatePropagation(). Після його виклику інші обробники не виконуються.

Не зупиняйте спливання без потреби!

Занурення

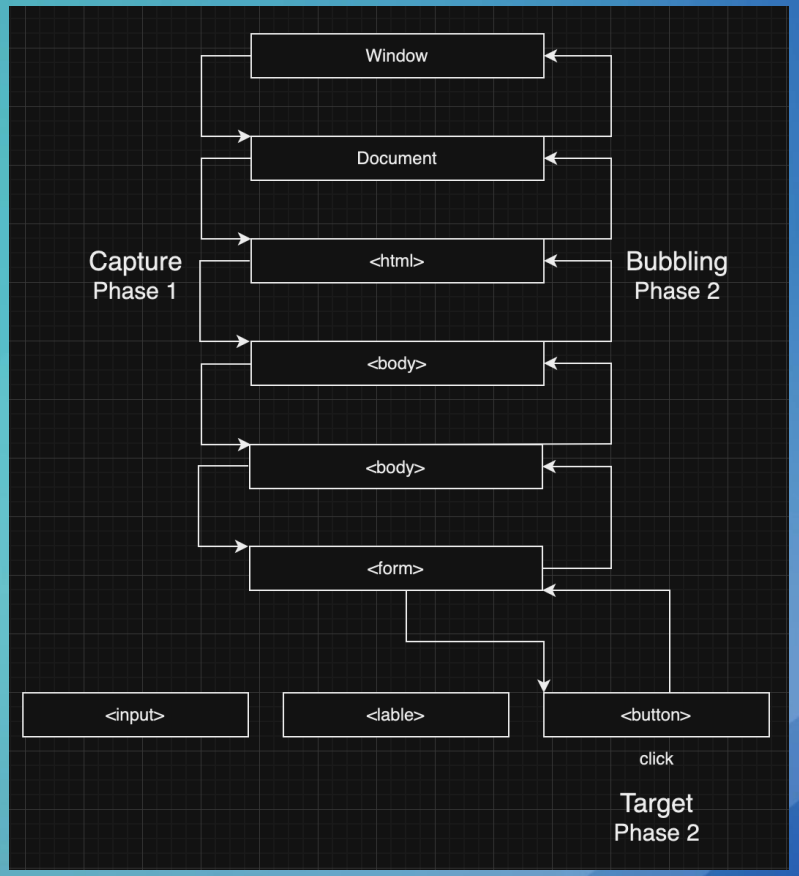
Цей механізм рідко використовується у реальному коді, але іноді може бути корисним.

Стандарт DOM Events описує 3 фази поширення подій:

- Фаза занурення (capturing phase) – подія спускається до елемента.
- Фаза цілі (target phase) – подія досягає цільового елемента.
- Фаза спливання (bubbling phase) – подія “спливає” від елемента вгору.

Щоб перехопити подію на фазі занурення (capturing phase), нам потрібно встановити опцію capture обробника на значення true

event.stopPropagation() під час занурення також перешкоджає спливанню події



Делегування подій

Ідея в тому, що якщо у нас є багато елементів, які обробляються подібним чином, то замість того, щоб призначати обробник кожному з них, ми ставимо один обробник на їхнього спільного предка.

У обробнику ми отримуємо event.target, щоб побачити, де насправді сталася подія і обробити її.

Шаблон “поведінки”

Ідея в тому, що якщо у нас є багато елементів, які обробляються подібним чином, то замість того, щоб призначати обробник кожному з них, ми ставимо один обробник на їхнього спільного предка.

У обробнику ми отримуємо event.target, щоб побачити, де насправді сталася подія і обробити її.

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.

Завжди використовуйте метод addEventListener для обробників на рівні документу, а не document.on<event>

Типові дії браузера

Багато подій автоматично призводять до певних дій, які виконує браузер. Якщо ми обробляємо подію в JavaScript, ми можемо не захотіти, щоб відбулась типова дія браузера, і замість цього захочемо реалізувати іншу поведінку.

Є два способи запобігти діям браузера:

1. event.preventDefault().
2. Якщо обробник призначено за допомогою on<event> (а не addEventListener), потрібно повернути false. Повернення false з обробника є винятком

Певні події перетікають одна в іншу. Якщо запобігти першій події, то не буде і другої.

event.defaultPrevented

Властивість event.defaultPrevented має значення true, якщо типову дію було скасовано, і false в іншому випадку.

Конструктор подій

Ми можемо створювати об’єкти Event вони базуються на інтерфейсі Event.

Базовий синтаксис: new Event(typeArg, eventInit);

- typeArg– тип події, назва.
- eventInit– об’єкт з трьома необов’язковими властивостями

1. "bubbles": (Необов'язковий) логічне значення (Boolean), що вказує – чи буде подія спливаючою. За промовчанням false.
2. "cancelable": (Необов'язковий) логічне значення (Boolean) вказує, чи може бути подія скасовано. За промовчанням false.
3. "composed": (Необов'язковий) логічне значення Boolean вказівне – чи буде подія випливати назовні за межі shadow root. За промовчанням false.

dispatchEvent

Після створення об’єкта події ми повинні “запустити” її на елементі за допомогою виклику elem.dispatchEvent(event)

Подій інтерфейсу

Усі події інтерфейсу описані у документації

Короткий список:

- UIEvent
- FocusEvent
- MouseEvent
- WheelEvent
- KeyboardEvent
- …

Правильний конструктор дозволяє вказати стандартні властивості для цього типу події.

Користувацькі події

Для наших власних, абсолютно нових типів подій, ми повинні використовувати new CustomEvent. Технічно CustomEvent – це те ж саме, що й Event, за одним винятком. Ми можемо додати додаткову властивість detail для будь-якої спеціальної інформації, яку ми хочемо передати разом із подією.

Властивість detail може містити будь-які дані. Технічно ми могли б жити і без них, оскільки ми можемо призначити будь-які властивості звичайному об’єкту new Event після його створення. Але CustomEvent забезпечує спеціальне поле detail, щоб уникнути конфліктів з іншими властивостями події.

Дякую за увагу